Dynamic Secrets: Unleashing the Thor's Hammer ⚒️ of FOSS Security

- Sakshi Nasha

About me

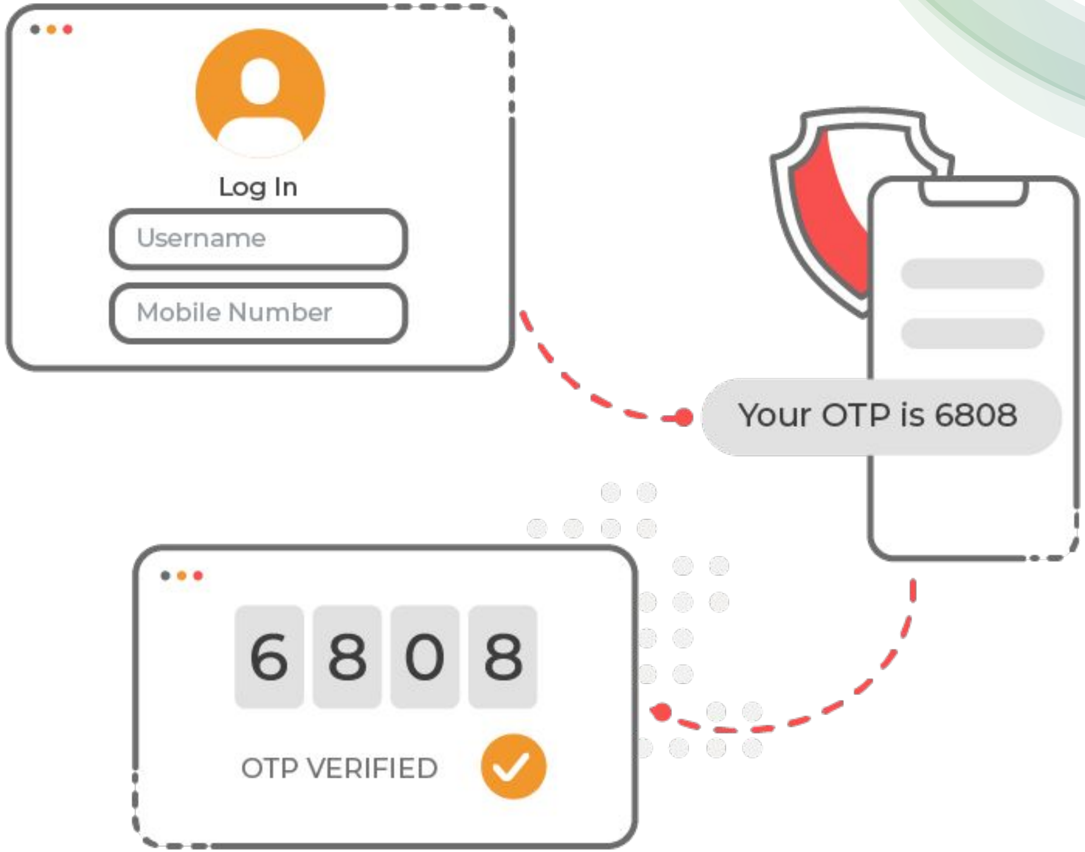
Let’s connect over LinkedIn :

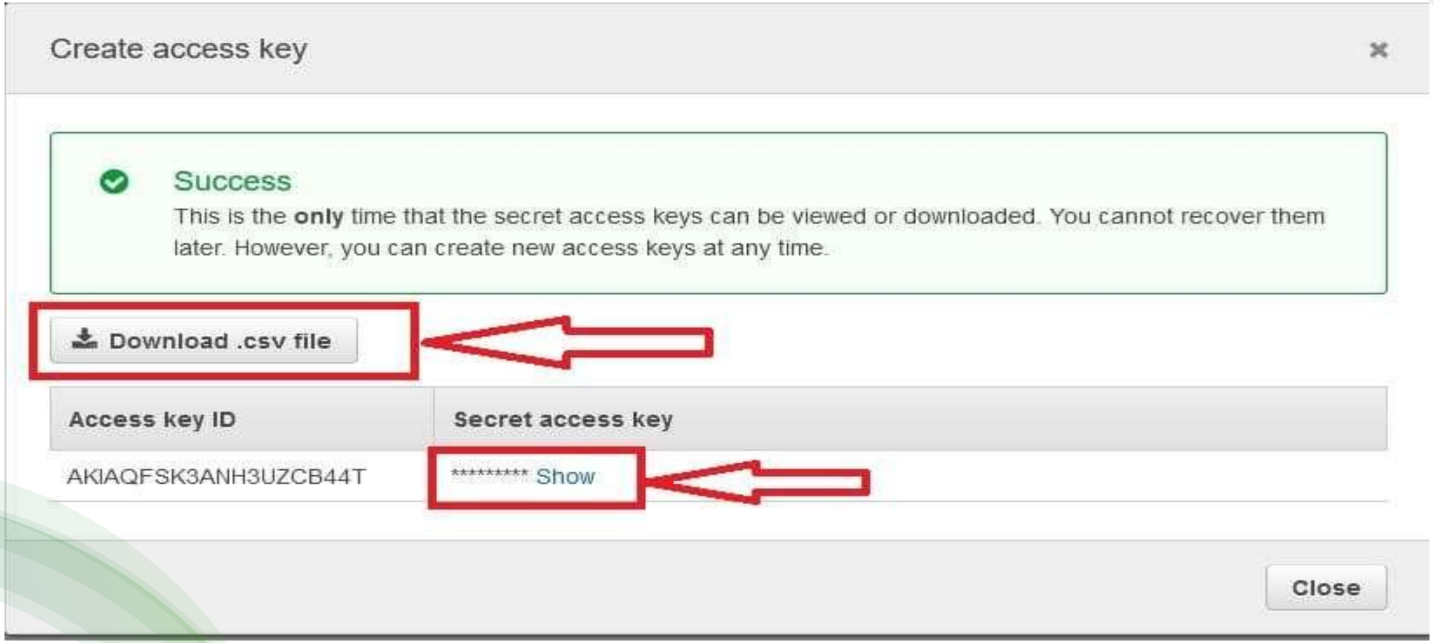
How much security is too much security for you ?

I DIDN'T LOCK MY KEYS IN THE CAR
AiO
ALL IN ONE
LOCKSMITH
YOUR SECURITY IS OUR PRIORITY
THE KEYS LOCKED ME OUT OF THE CAR
www.aiolocksmith.com

Real Life Example

1) Login Authentication : Bank Website
2) AWS account verification 2MFA
3) Transaction Confirmation : Credit card / Debit card
4) Email Verification
5) Microsoft Authenticator – Company Portal





Create access key

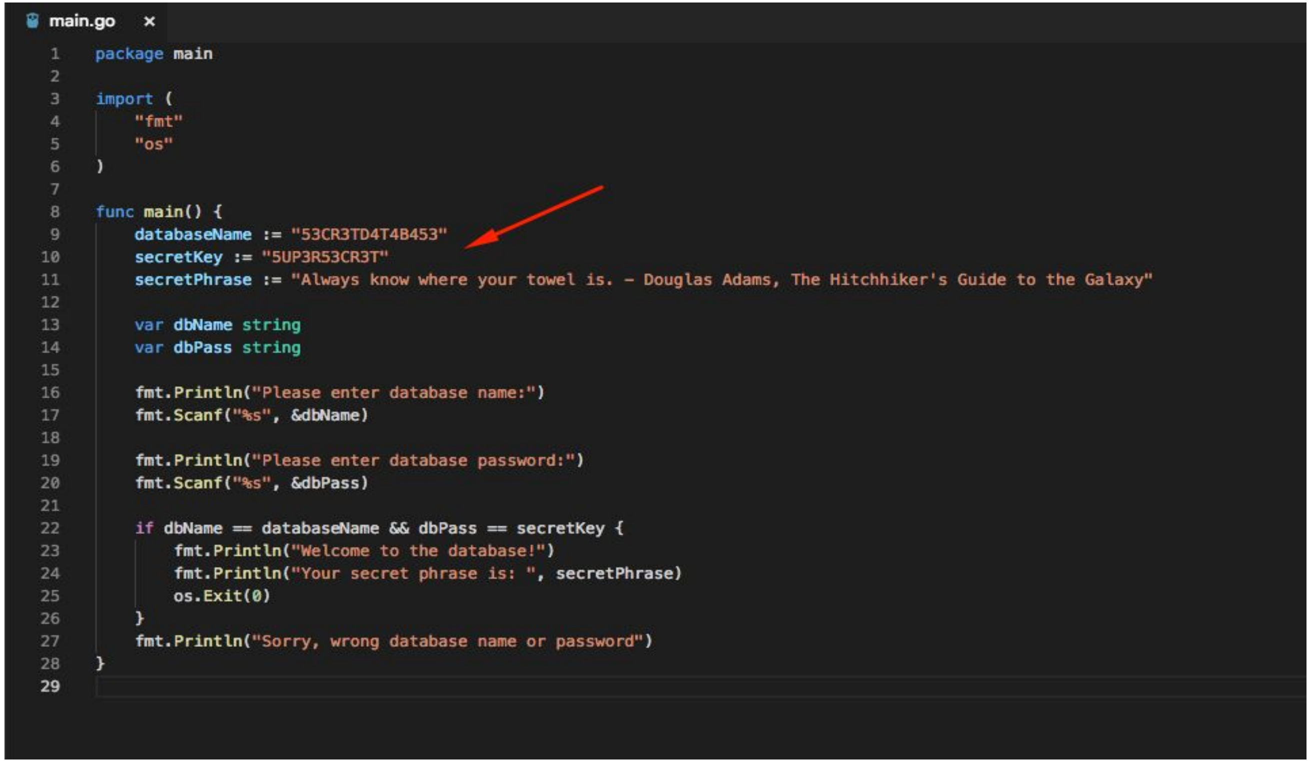
Success

This is the **only** time that the secret access keys can be viewed or downloaded. You cannot recover them later. However, you can create new access keys at any time.

Access key ID	Secret access key
AKIAQFSK3ANH3UZCB44T	********* Show

Diagram1: Creating a Credential (Access key and secret) with the AWS S3 bucket (IAM ROLE)

Hardcoding secrets in the code



main.go

```go
package main

import (
	"fmt"
	"os"
)

func main() {
	databaseName := "53CR3TD4T4B453"
	secretKey := "5UP3R53CR3T"
	secretPhrase := "Always know where your towel is. - Douglas Adams, The Hitchhiker's Guide to the Galaxy"

	var dbName string
	var dbPass string

	fmt.Println("Please enter database name:")
	fmt.Scanf("%s", &dbName)

	fmt.Println("Please enter database password:")
	fmt.Scanf("%s", &dbPass)

	if dbName == databaseName && dbPass == secretKey {
		fmt.Println("Welcome to the database!")
		fmt.Println("Your secret phrase is: ", secretPhrase)
		os.Exit(0)
	}
	fmt.Println("Sorry, wrong database name or password")
}
```

Static Secrets v/s Dynamic Secrets

Static secrets are secrets that are pre-defined and do not change unless explicitly modified by an administrator or authorized user.

- Examples: API keys, database passwords, or encryption keys that remain constant over time unless intentionally rotated.

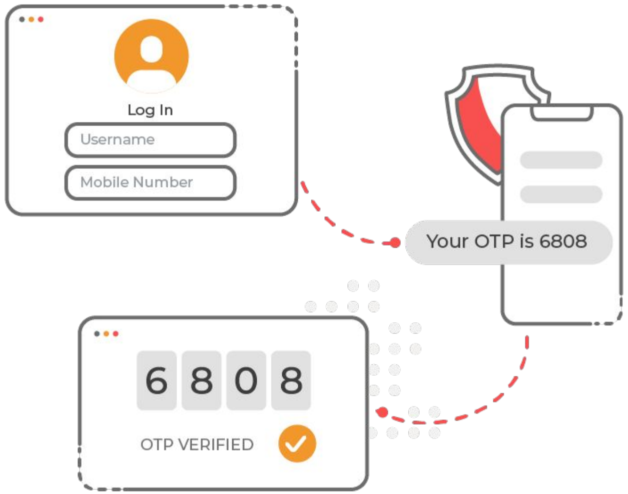
Dynamic secrets are credentials or tokens that are generated programmatically and have a short lifespan. They are typically generated on-demand and automatically revoked after a specified time period or usage.

- Examples : Temporary Access, Session Token or Short-lived tokens , Temporary Service Accounts(created from automation scripts or microservices)

Current key Challenges faced with managing and storing secrets

- **Security Risks**: Leakage or unauthorized access to secrets such as passwords, API keys, and tokens can lead to data breaches and compromise sensitive information.
- **Integration Complexity**: Seamlessly integrating secrets management solutions across diverse environments, including on-premises and multi-cloud setups, can be complex and challenging.
- **Balancing Security and Flexibility**: Finding a balance between rc security measures and maintaining operational flexibility to support agile development and deployment practices.
- **Cost Management**: High costs associated with proprietary secrets management solutions (licensing fees, operational expenses)
- **Manual Processes**:
  Tedious manual processes for key rotation, encryption, and distribution of secrets.
- **Compliance and Governance**:
  Ensuring compliance with industry regulations and internal governance policies
  while managing secrets securely.
- **Scalability**: Scaling secrets management solutions to accommodate growing volumes of credentials and applications in dynamic cloud-nativ environments

TODAY, we will generate a short-lived token for our own application



Agenda of Workshop

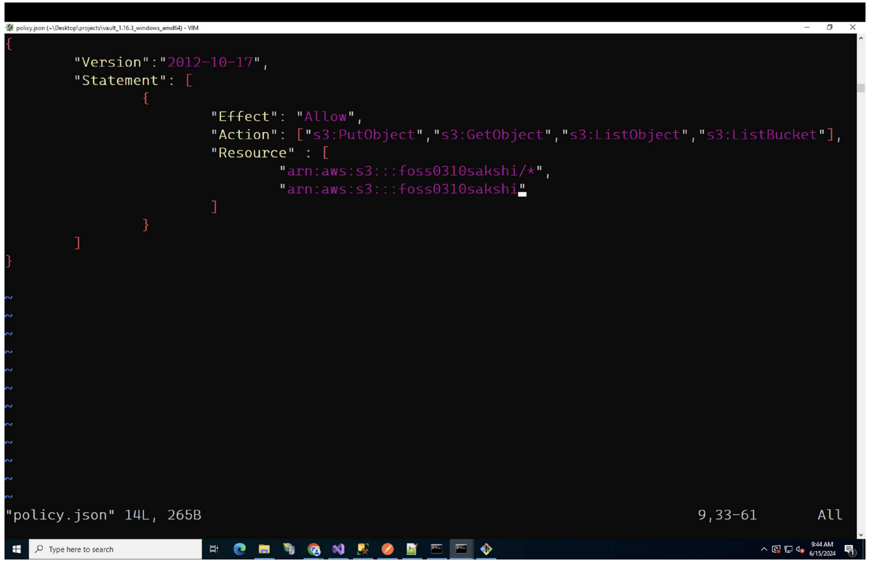
1. Introduction
2. Installation
3. Basic CRUD Operations
4. Using AWS secret Engine
5. Generating Dynamic Secrets : harnessing the POWER of FOSS tool
6. Operations by Using AWS Dynamic Secrets
7. Lease : renewal, revocation and inspection
8. AddingTTL (Expiry)
9. Disabling AWS Secret Engine
10. Auto Rotation Policy
11. Extension to Databases
12. Other FOSS tools to manage secrets
13. Summary

Prerequisites

- AWS Bucket
- AWS IAM USER and AWS IAM Policy
- What is HashiCorp Vault
- Client Server Architecture
- What is Secret Engine

Prerequistics

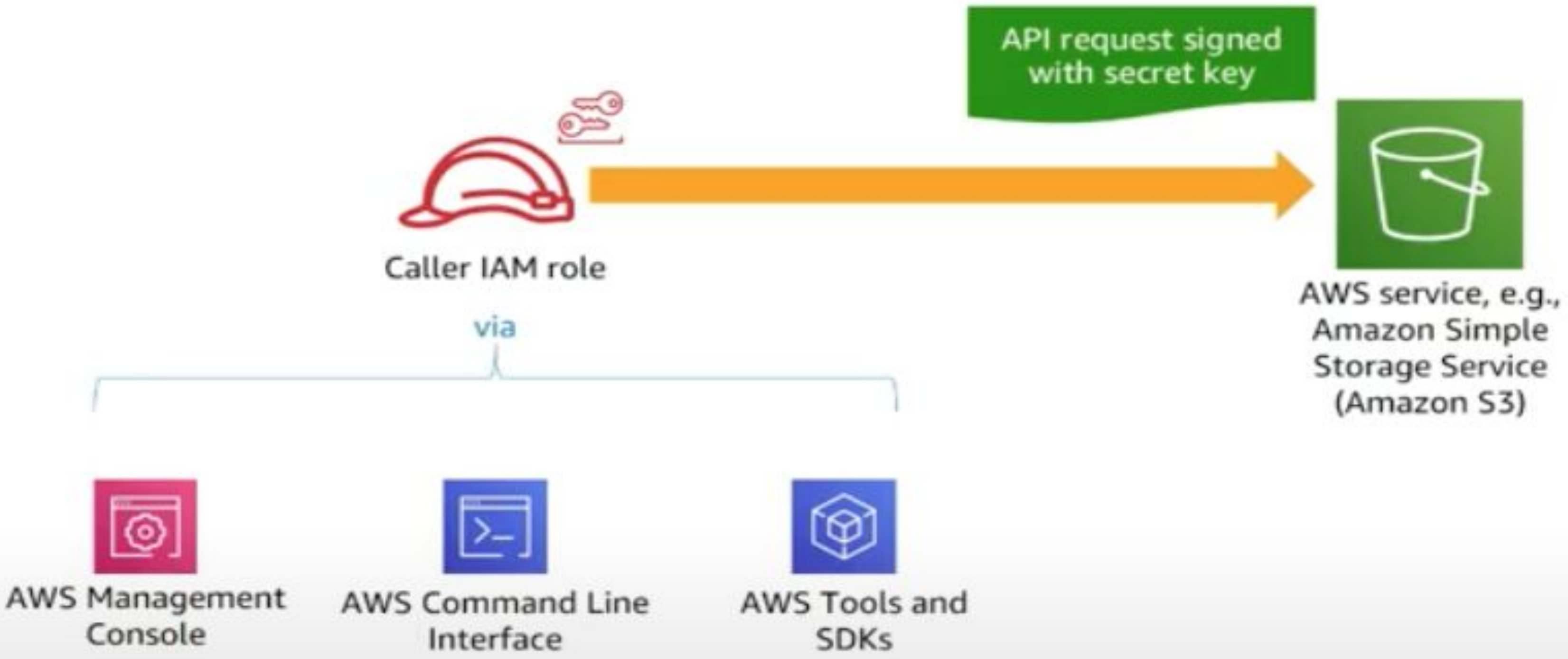
- Amazon Simple Storage Service (Amazon S3) is an object storage service. To know more
- AWS IAM (securely control access to AWS resources)
  - IAM USER : entity that you create in AWS (human user or workload)
  - IAM Policy : You manage access/permissions in AWS by creating policies and attaching them to IAM identities (users, groups of users, or roles) or AWS resources. AWS evaluates these policies when an IAM principal (user or role) makes a request.
  - **IAM Credentials** : keys used to access the AWS resources



policy.json (~\Desktop\projects\vault_1.16.3_windows_amd64) - VIM

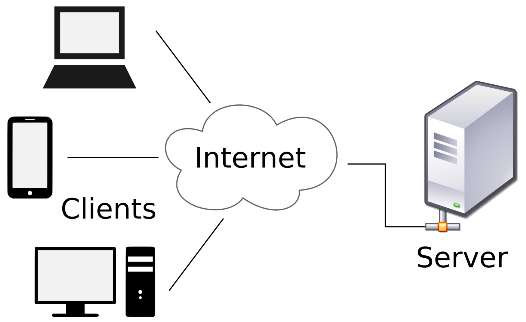
```
{
        "Version":"2012-10-17",
        "Statement": [
                {
                        "Effect": "Allow",
                        "Action": ["s3:PutObject","s3:GetObject","s3:ListObject","s3:ListBucket"],
                        "Resource" : [
                                "arn:aws:s3:::foss0310sakshi/*",
                                "arn:aws:s3:::foss0310sakshi"
                        ]
                }
        ]
}
~
~
~
~
~
~
~
~
~
~
~
~
"policy.json" 14L, 265B                                               9,33-61       All
```

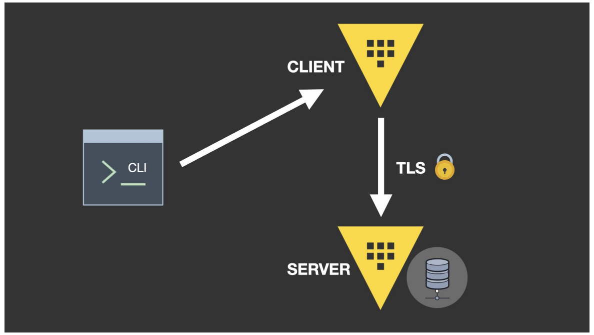
How an authentication works in AWS


API request signed with secret key
Caller IAM role
via
AWS Management Console
AWS Command Line Interface
AWS Tools and SDKs
AWS service, e.g., Amazon Simple Storage Service (Amazon S3)

Prerequisites

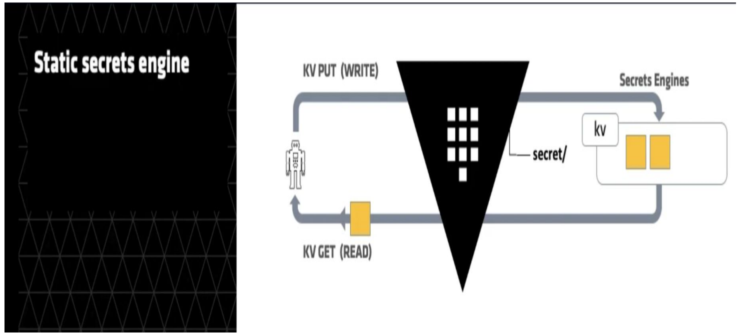
- **HashiCorp Vault** is a tool for managing secrets and protecting sensitive data.
- The **client-server model** involves a server that provides resources or services and a client that requests and uses those services. The server listens for and responds to client requests, while clients initiate requests and receive responses.
- **Server**: The Vault server is the central component that stores and manages secrets. It handles requests from clients, performs authentication, and provides secret data based on policies.
- **Client**: Clients interact with the Vault server to request secrets, perform authentication, or store new secrets. This can include applications, infrastructure services, or even human users.

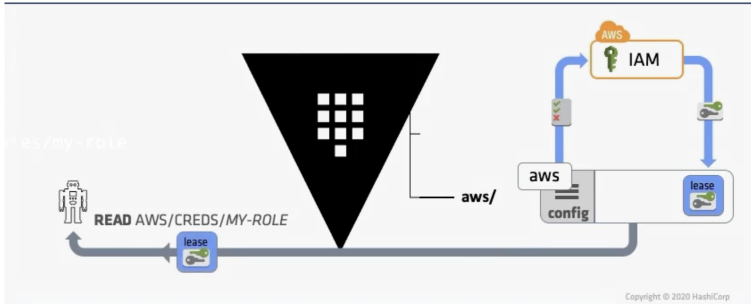




Prerequisites

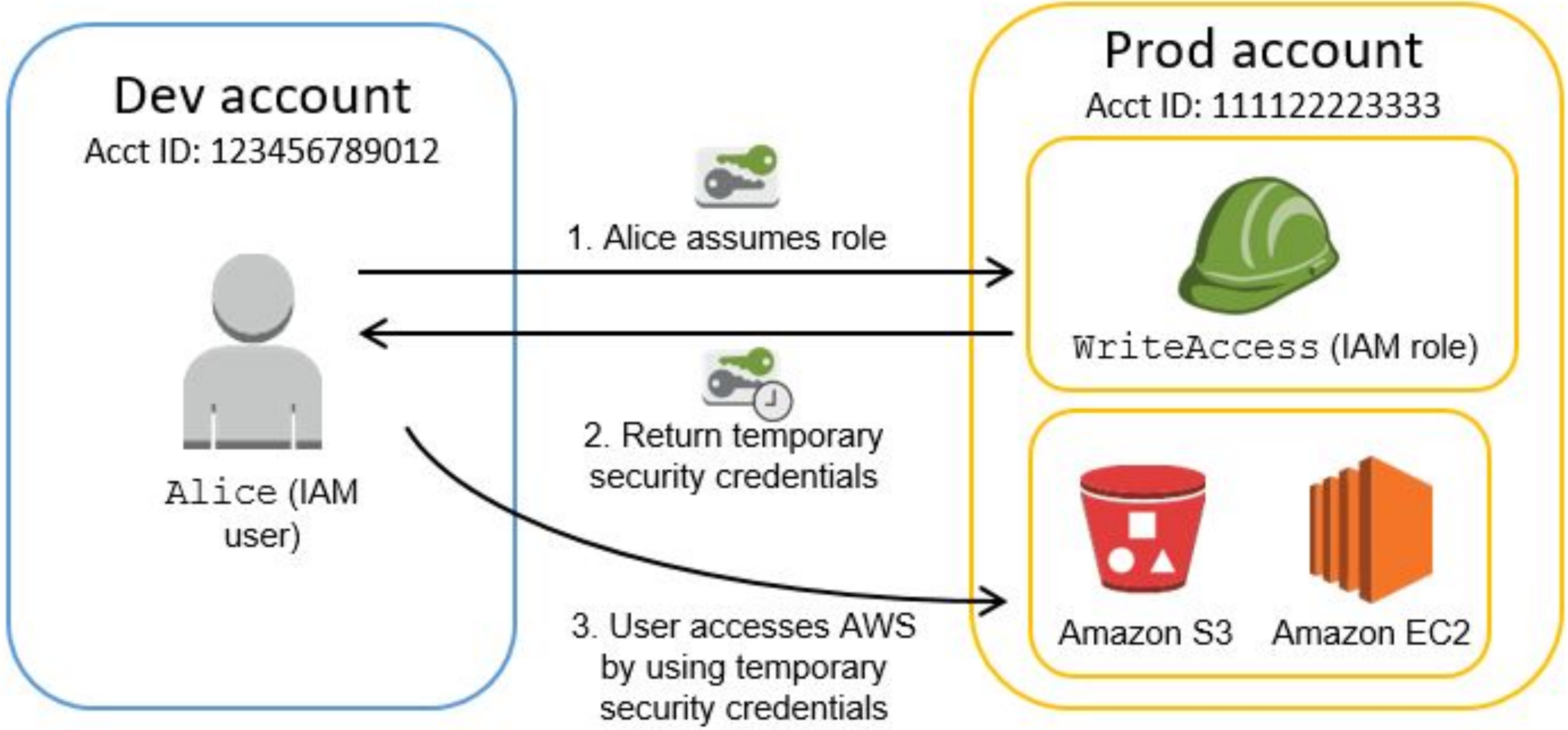
- **Secret Engine :** is a component that enables the storage, management, and retrieval of specific types of secrets. Each secret engine provides different functionalities, such as generating dynamic credentials, encrypting data, or managing sensitive information, tailored to various use cases like databases, APIs, or key management.
- AWS secrets engine : The AWS secrets engine generates AWS access credentials dynamically based on IAM policies.
- The AWS secrets engine supports the concept of static credentials as well as dynamic secrets.

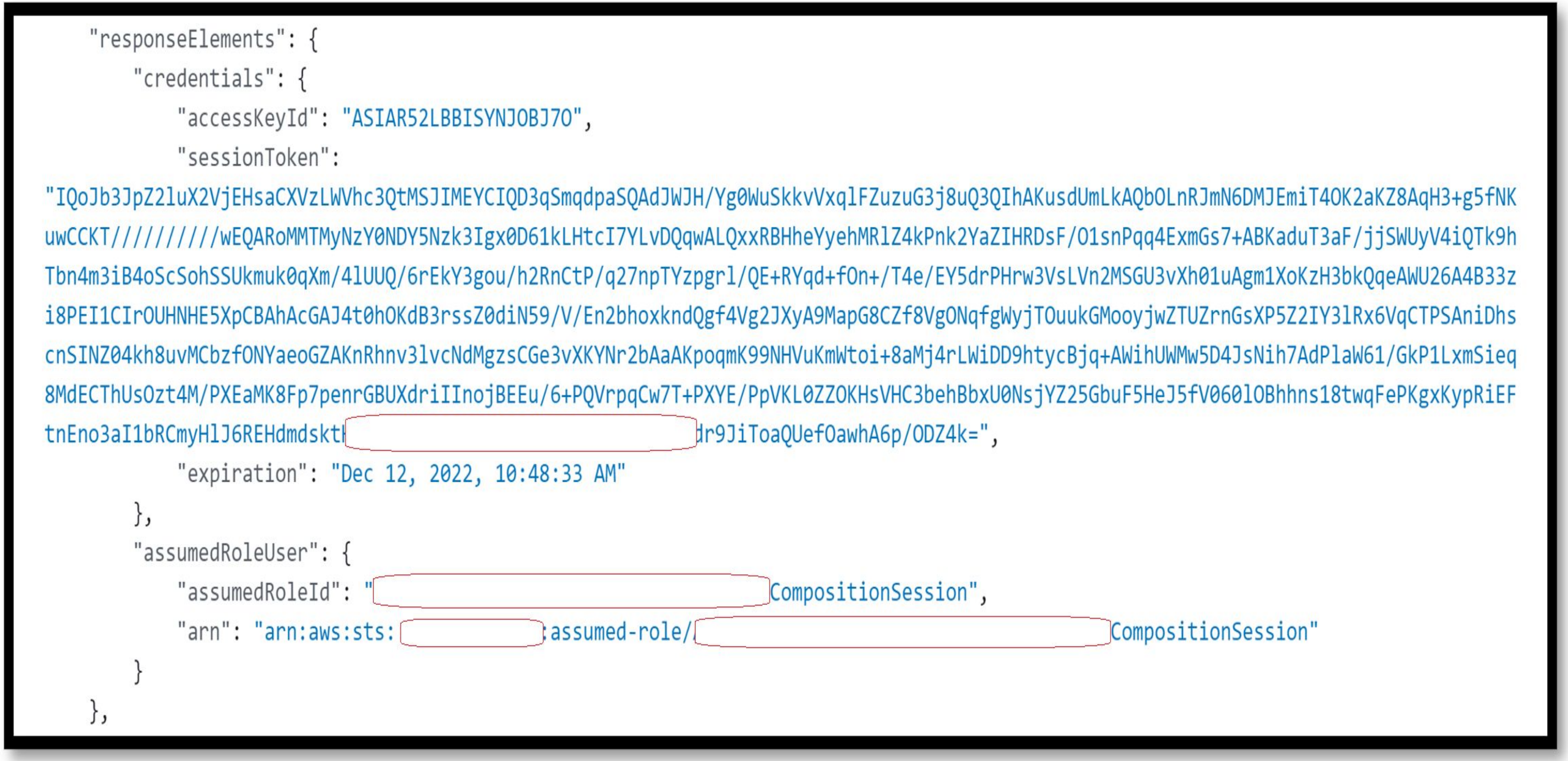




Introducing short lived token-based authentication.

- Short lived tokens are **JWT** (JSON Web Token) with **session time**
- Short-term credentials consist of three parts: **Access key ID, secret access key, and a security token.**
- The users once register their credentials, receive a unique encrypted token that is valid for a specified session time. During this session, users can directly access the website or application without login requirements.
- You can create these tokens using AWS Security Token Service (**STS**) , Google Short lived creds for service account (GCP) , Azure Storage resources using shared access signatures (**SAS**)
- While creating the tokens, you can specify the **duration** : How long the STS/SAS is valid.
- Examples : **Financial applications (ICICI Bank, HDFC Bank), Sensitive Multi Cloud applications**


Dev account
Acct ID: 123456789012
Alice (IAM user)
1. Alice assumes role
2. Return temporary security credentials
3. User accesses AWS by using temporary security credentials
Prod account
Acct ID: 111122223333
WriteAccess (IAM role)
Amazon S3
Amazon EC2



```
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAR52LBBISYNJOBJ7O",
            "sessionToken":
"IQoJb3JpZ2luX2VjEHsaCXVzLWVhc3QtMSJIMEYCIQD3qSmqdpaSQAdJWJH/Yg0WuSkkvVxqlFZuzuG3j8uQ3QIhAKusdUmLkAQbOLnRJmN6DMJEmiT4OK2aKZ8AqH3+g5fNK
uwCCKT//////////wEQARoMMTMyNzY0NDY5Nzk3Igx0D61kLHtcI7YLvDQqwALQxxRBHheYyehMRlZ4kPnk2YaZIHRDsF/O1snPqq4ExmGs7+ABKaduT3aF/jjSWUyV4iQTk9h
Tbn4m3iB4oScSohSSUkmuk0qXm/4lUUQ/6rEkY3gou/h2RnCtP/q27npTYzpgrl/QE+RYqd+fOn+/T4e/EY5drPHrw3VsLVn2MSGU3vXh01uAgm1XoKzH3bkQqeAWU26A4B33z
i8PEI1CIrOUHNHE5XpCBAhAcGAJ4t0hOKdB3rssZ0diN59/V/En2bhoxkndQgf4Vg2JXyA9MapG8CZf8VgONqfgWyjTOuukGMooyjwZTUZrnGsXP5Z2IY3lRx6VqCTPSAniDhs
cnSINZ04kh8uvMCbzfONYaeoGZAKnRhnv3lvcNdMgzsCGe3vXKYNr2bAaAKpoqmK99NHVuKmWtoi+8aMj4rLWiDD9htycBjq+AWihUWMw5D4JsNih7AdPlaW61/GkP1LxmSieq
8MdECThUsOzt4M/PXEaMK8Fp7penrGBUXdriIInojBEEu/6+PQVrpqCw7T+PXYE/PpVKL0ZZOKHsVHC3behBbxU0NsjYZ25GbuF5HeJ5fV060lOBhhns18twqFePKgxKypRiEF
tnEno3aI1bRCmyHlJ6REHdmdskt[redacted]dr9JiToaQUefOawhA6p/ODZ4k=",
            "expiration": "Dec 12, 2022, 10:48:33 AM"
        },
        "assumedRoleUser": {
            "assumedRoleId": "[redacted]CompositionSession",
            "arn": "arn:aws:sts:[redacted]:assumed-role/[redacted]CompositionSession"
        }
    },
```

let the
adventure
BEGIN

Demo

- Link : https://www.youtube.com/watch?v=wNUGV_sdm0Y

Perks of using Short lived Tokens

1 Enhancing Security

- Short-lived tokens have a **limited lifespan**, reducing the exposure window for potential attacks.
- If a token is compromised, its validity period is short, minimizing the risk of unauthorized access.
- Regular token expiration forces users to **re-authenticate**, ensuring better security.

2 Mitigating Token Abuse

- Tokens are often used to authorize access to resources.
- By making tokens short lived, we limit the time an attacker can use to abuse a stolen token.
- Thus, **minimizing** the **risk window** significantly

3 Least Privilege Principle

- A user should only have access to what they absolutely **need** and not what they want.
- When permissions **change** (e.g., user roles or access levels), short-lived tokens automatically reflect the updates upon renewal.
- Long-lived tokens may retain outdated permissions, leading to security risks.

Auto Rotation Policy

- Auto rotation Policy

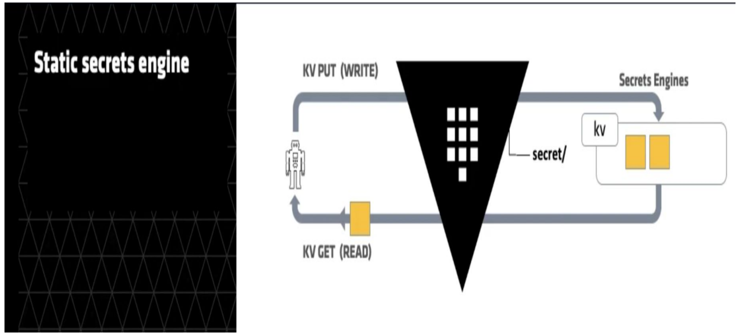
Extension to Databases

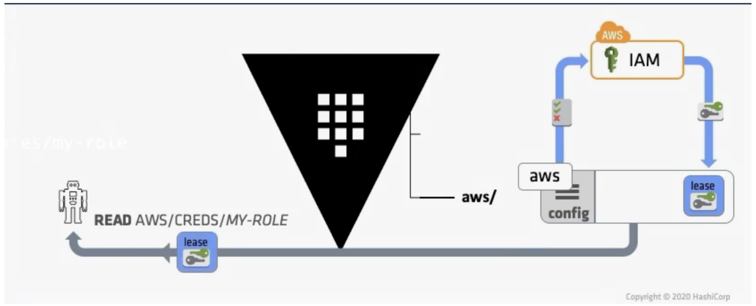
- Dynamic secrets: database secrets engine | Vault | HashiCorp Developer

FOSS Tool

- HashiCorp Vault
- Keycloak
- Seaked Secrets

Free and Open Source Software (FOSS) tools provide robust solutions for securely managing and storing credentials in applications.





“You got the incredible power of FOSS through the Thor's Hammer to protect your
the Asgard (Applications) from potential threats ”

USE WISELY

Future References :

aws

Microsoft Azure

Future References :

ORACLE

My VOX

Thank you

FOSS UNITED

until next time