



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

- Sakshi Nasha



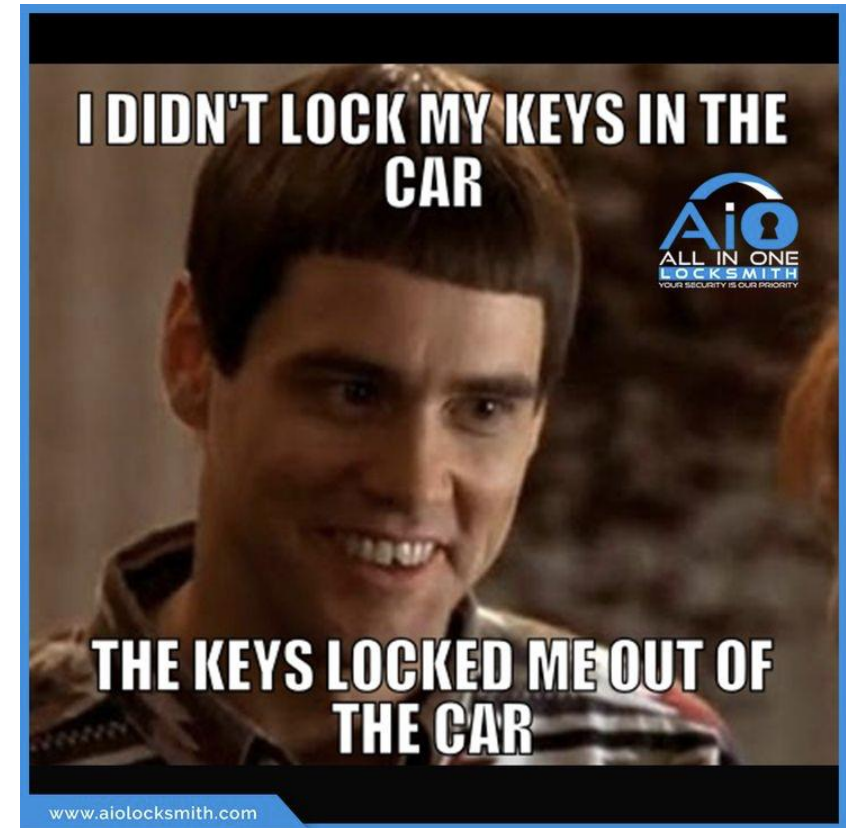
About me



**Let's connect over
LinkedIn :**



How much security is too much security for you ?



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

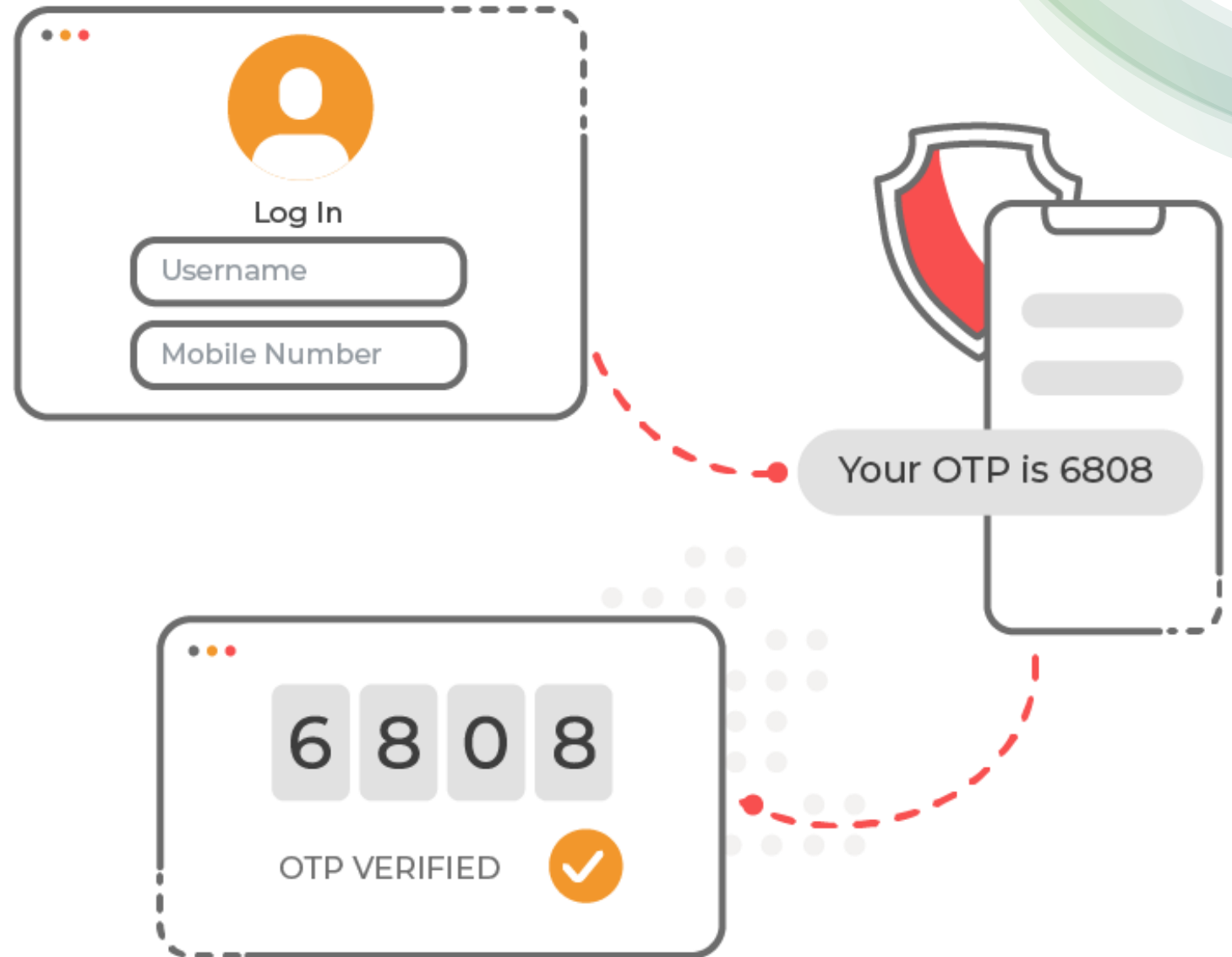





Diagram1: Creating a Credential (Access key and secret) with the AWS S3 bucket (IAM ROLE)

Hardcoding secrets in the code

```
main.go x
1 package main
2
3 import (
4     "fmt"
5     "os"
6 )
7
8 func main() {
9     databaseName := "53CR3TD4T4B453"
10    secretKey := "5UP3R53CR3T"
11    secretPhrase := "Always know where your towel is. - Douglas Adams, The Hitchhiker's Guide to the Galaxy"
12
13    var dbName string
14    var dbPass string
15
16    fmt.Println("Please enter database name:")
17    fmt.Scanf("%s", &dbName)
18
19    fmt.Println("Please enter database password:")
20    fmt.Scanf("%s", &dbPass)
21
22    if dbName == databaseName && dbPass == secretKey {
23        fmt.Println("Welcome to the database!")
24        fmt.Println("Your secret phrase is: ", secretPhrase)
25        os.Exit(0)
26    }
27    fmt.Println("Sorry, wrong database name or password")
28 }
29
```



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)

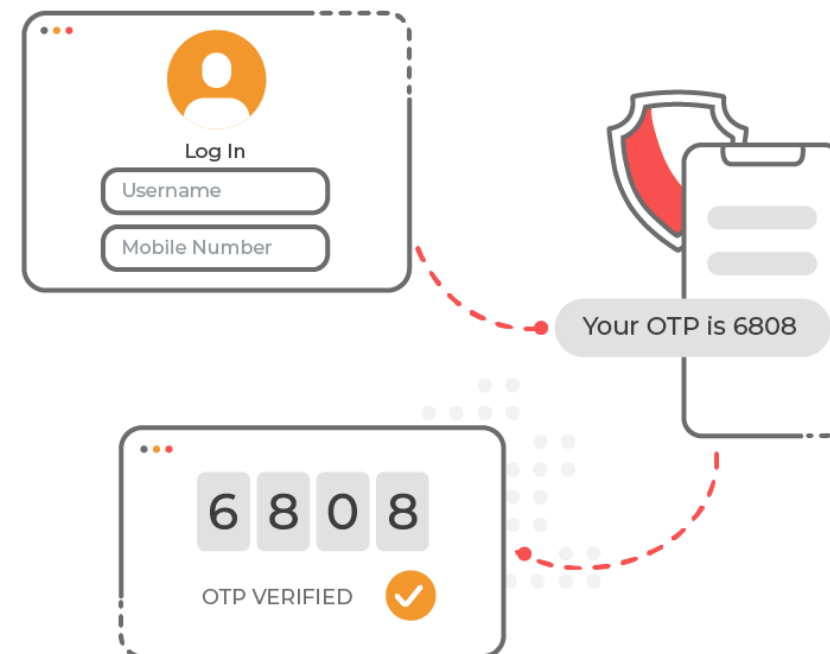


Current key Challenges faced with managing and storing secrets

- **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 robust 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-native environments



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



Agenda of Workshop

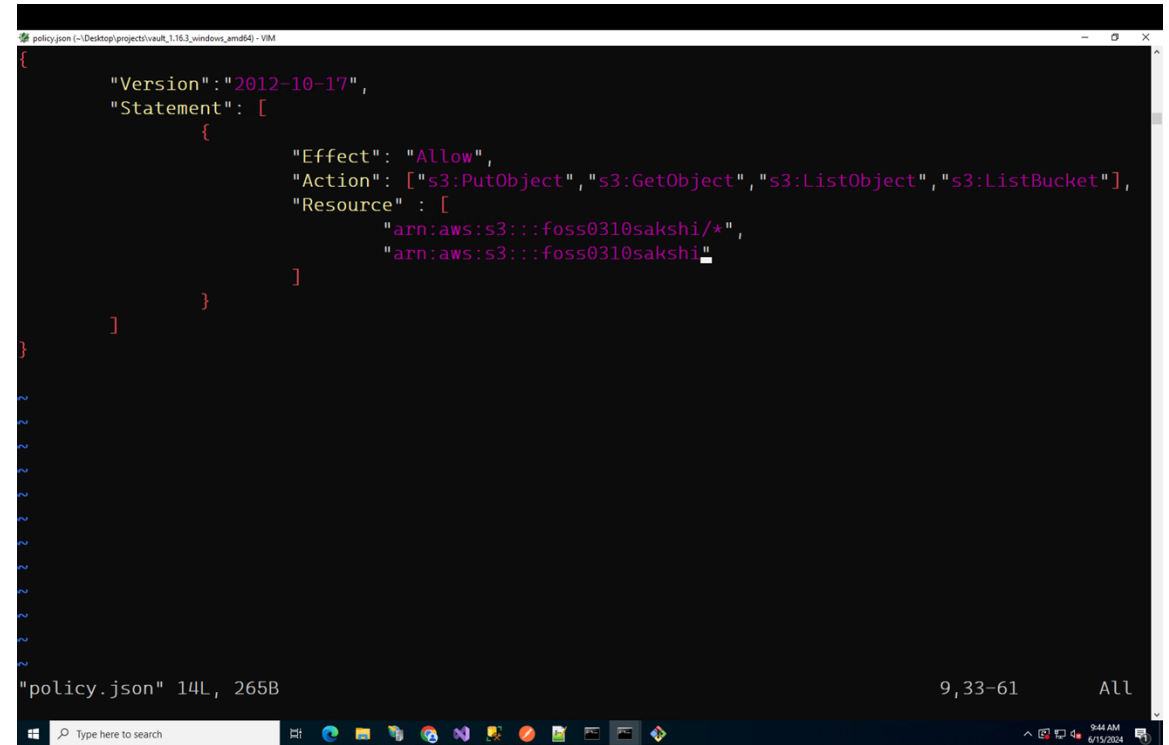
1. Introduction
2. Installation
3. Basic CRUD Operations
4. Using AWS secret Engine
5. Generating Dynamic Secrets : harnessing the POWER of FOSS tool
6. Operations by Using AWS Dynamic Secrets
7. Lease : renewal, revocation and inspection
8. AddingTTL (Expiry)
9. Disabling AWS Secret Engine
- 10.Auto 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

Prerequisites

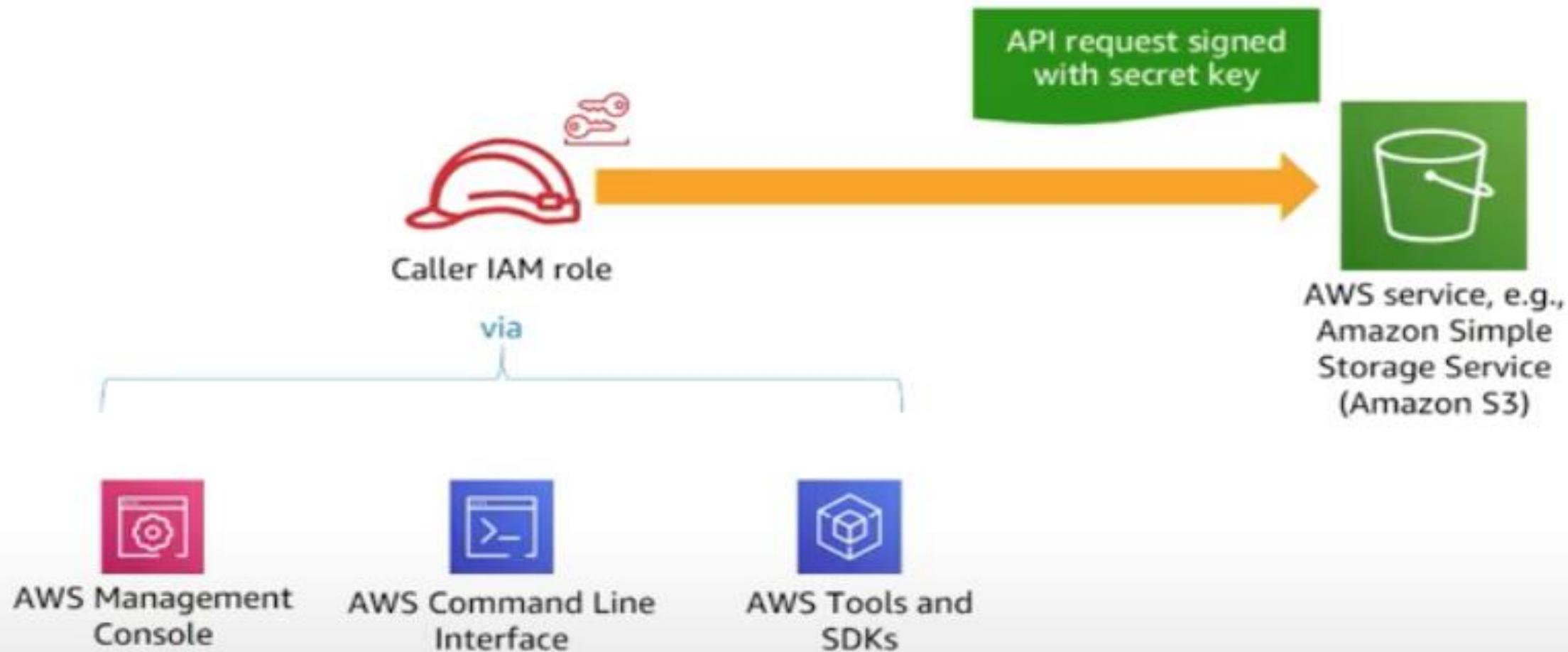
- Amazon Simple Storage Service (Amazon S3) is an object storage service. [To know more](#)
- AWS IAM (securely control access to AWS resources)
 - [IAM USER](#): entity that you create in AWS (human user or workload)
 - [IAM Policy](#): You manage access/permissions in AWS by creating policies and attaching 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



```
{
  "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"
      ]
    }
  ]
}
```

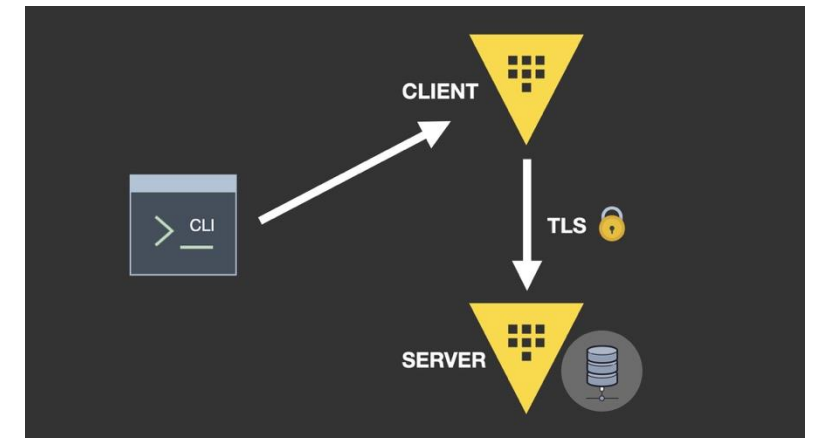
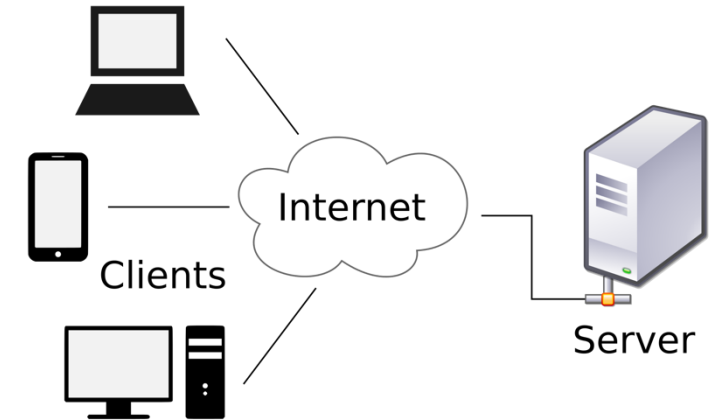
The screenshot shows a code editor window titled "policy.json" with a dark theme. It contains a JSON document defining an AWS IAM policy. The policy has a version of "2012-10-17" and a single statement that allows actions "s3:PutObject", "s3:GetObject", "s3:ListObject", and "s3:ListBucket" on the resources "arn:aws:s3:::foss0310sakshi/*" and "arn:aws:s3:::foss0310sakshi". The status bar at the bottom indicates the file is "policy.json", 14 lines long, 265 bytes, and the cursor is at line 9, column 33-61. The Windows taskbar is visible at the bottom of the screen.

How an authentication works in AWS



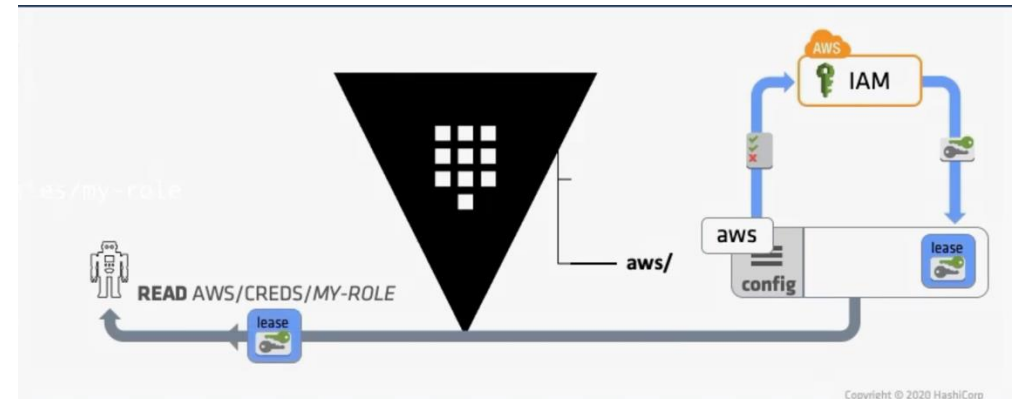
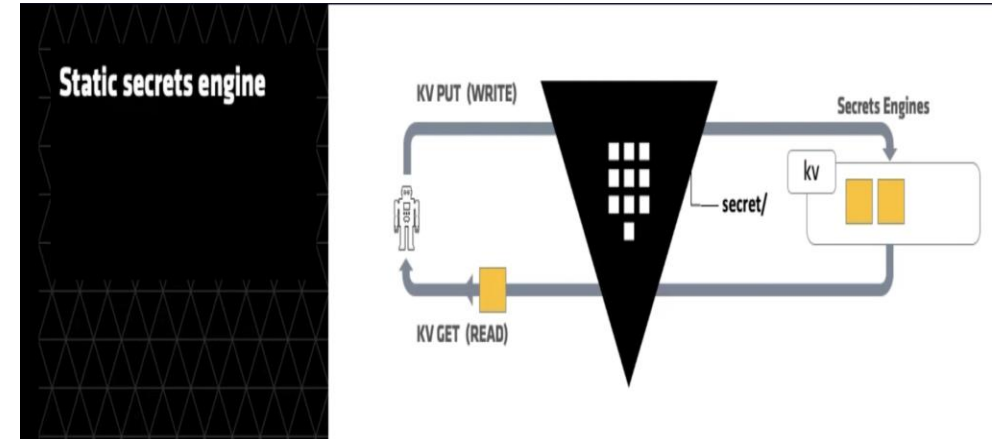
Prerequisites

- **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.



let the
adventure
BEGIN

Demo

- Link :

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](#)

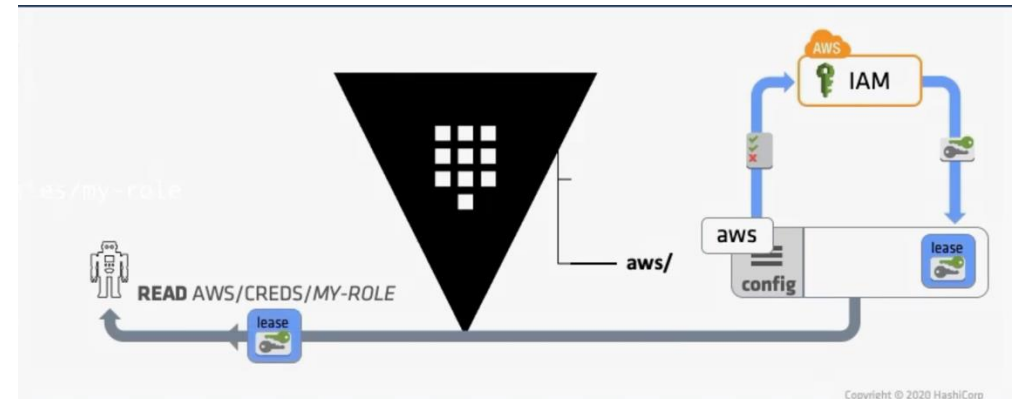
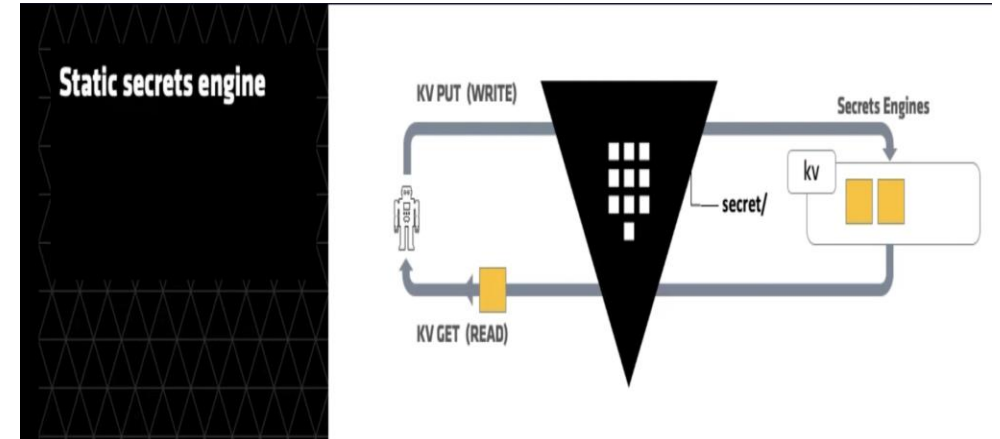
Extension to Databases

- [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



Thank
you

until
next
time

