From Static to Dynamic: Leveraging Short-Lived SAS tokens for Better Security in Microsoft Azure

Sakshi Nasha

Software Engineer II - Cohesity

How much security is too much security for you ?


NYPD
Authorization
Authentication

One step at a time

AUTHENTICATION

Who are you?

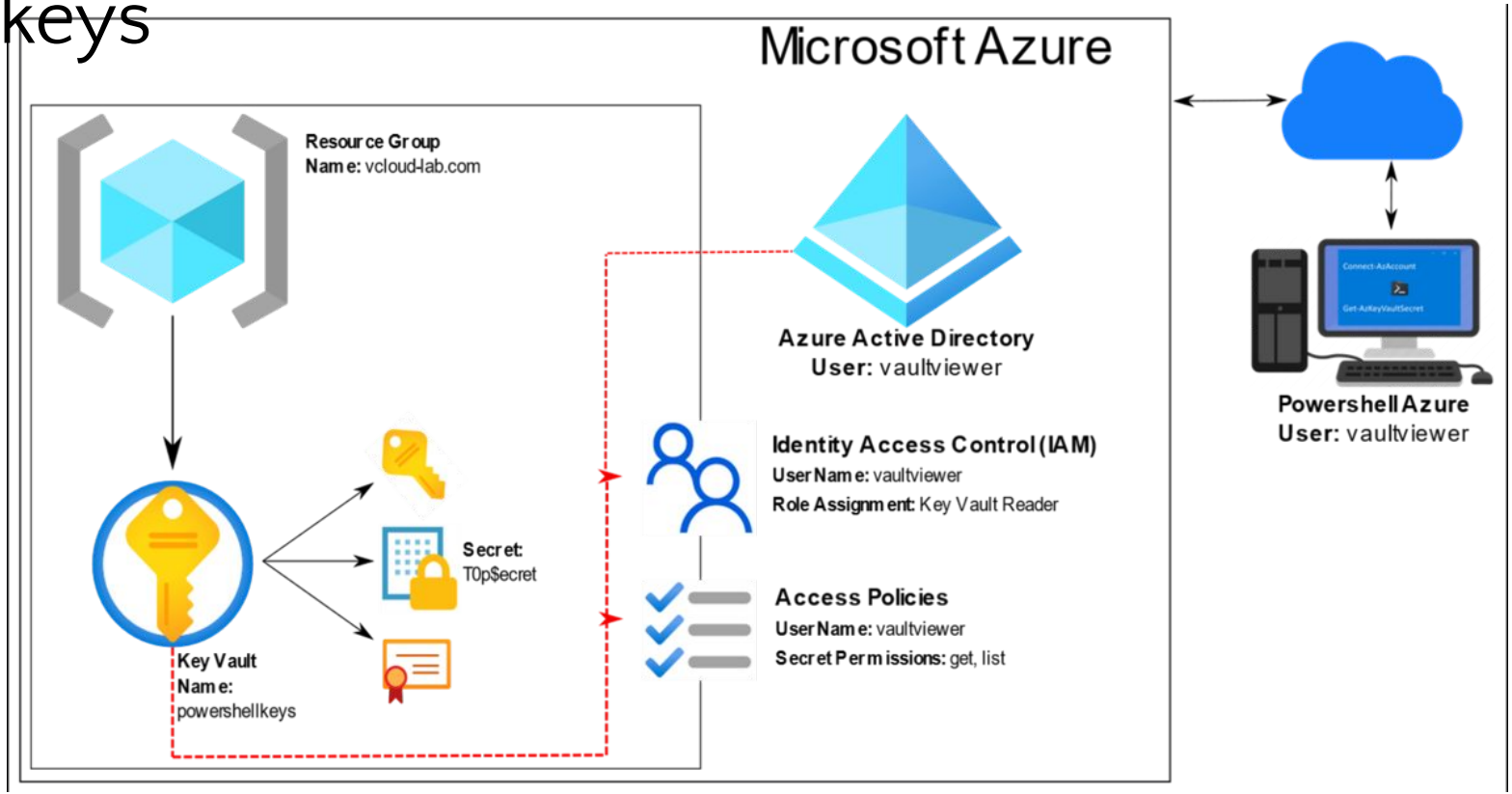
Verify the user's identity

AUTHORIZATION

What are you allowed to do?

Determine user permissions

Traditional Way of generating Azure Secret keys



Microsoft

Disadvantages of using Standard Credentials in your application

- These standard credentials are **long lived** in nature.
- If compromised, they give attackers **ample time** to exploit the application.
- If they are stolen it would be a nightmare to discern which operations are legitimate.
- Thus, the only fail-safe choice is to cumbersomely **rotate the keys** and **redistribute to customers**. This is often overlooked action and adds **extra pain** for the DevOps.

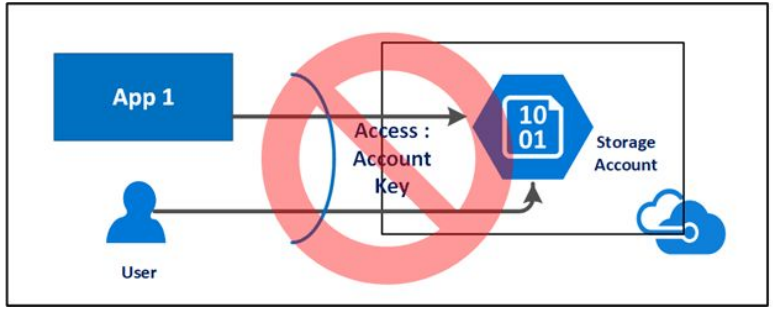
Hardcoding secrets in the code

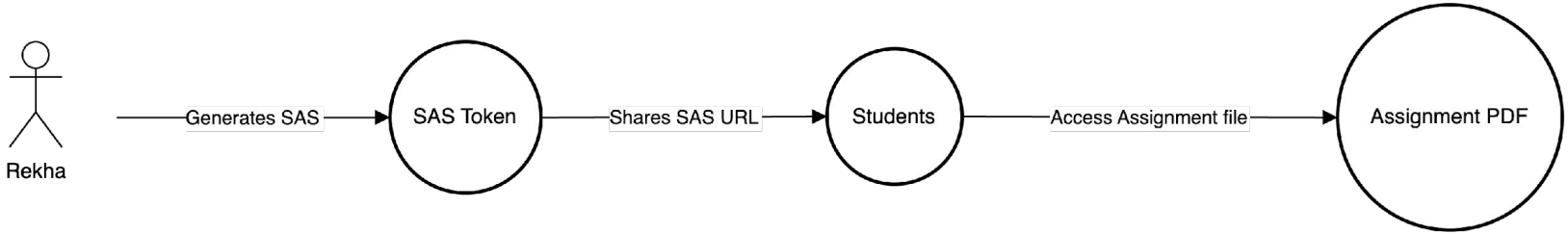
main.go

```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")
}
```

Introduction to Short Lived SAS tokens

- Scenario

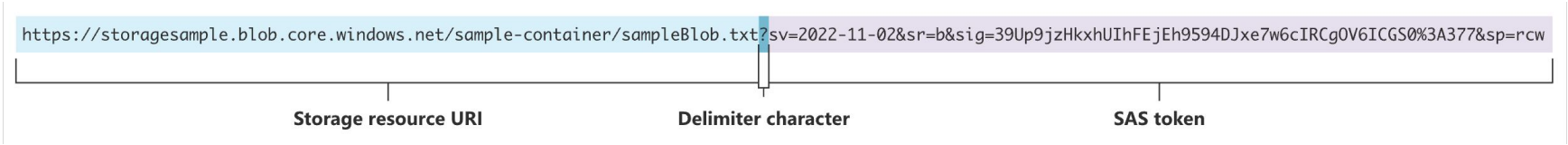




What are SAS ?

1. A shared access signature (SAS) is a **URI** that grants restricted access rights to Azure Storage resources.
2. 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.

Working of SAS ?



SAS is token that is appended to the URI
Token contains a special set of query parameters that indicate how the resources may be accessed by the client.

The SAS token: Everything after the ? is the temporary key.
sv : Version of the SAS. sr : Signed Resource (Blob, File, table, Queue)
se : The expiration time for the token.
sp=r: What they can do (in this case, just read).
sig=abcdefg12345: The security signature to make sure it’s valid.

Types of SAS Tokens

- Account SAS
- Service SAS
- User Delegation SAS

Decision Tree for selecting SAS token Type

Let the adventure began

Wish me luck !!

1. Azure CLI

Access file :
https://github.com/Sakshi-10/Az-SAS-token/blob/main/Program.cs

2. Azure SDK

```
from azure.storage.blob import generate_container_sas
```

Access the Code

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.

Thank You!

Connect with me 💁🏻‍♀️

Guidance

Microsoft

Q&A, Bring it on !

Microsoft