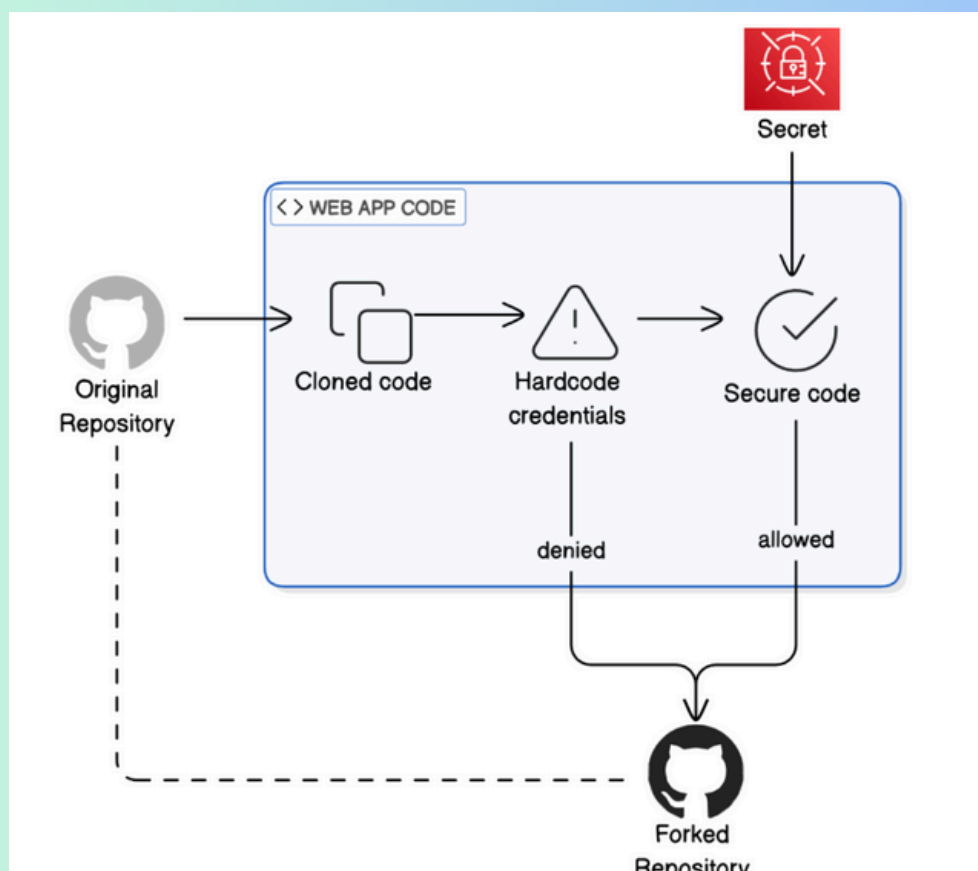


Secure Secrets with Secrets Manager



Kanika Mathur
github.com/KanikaGenesis





Kanika Mathur
github.com/KanikaGenesis

NextWork.org

Introducing Today's Project!

In this project, I will demonstrate how to use AWS Secrets Manager for secure credentials management. I did this project to learn how to protect my credentials when I connect live (in production) apps to my AWS environment and databases, etc.

Tools and concepts

Services I used were Secrets Manager, GitHub, S3 and IAM. Key concepts I learned include GitHub Secret scanning, the risk of hardcoding credentials and using git rebase to rewrite the commit history of a repository.

Welcome to My AWS Demo App

This application lists your S3 buckets behind the scenes.

Currently, the AWS credentials are **hard-coded** in `config.py`. In the next step, we'll replace them with **AWS Secrets Manager**.

[View My S3 Buckets](#)



Hardcoding credentials

In this project, a sample web app exposes the developer's AWS credentials, i.e. their access key ID and secret access key. It is unsafe to hardcode credentials because once the code is made public in a repository, anyone can get access to those credentials and access our AWS environment, e.g. delete resources, steal data, etc.

I've set up the initial configuration of this web app with a Secret Access Key and Access Key ID. These credentials are just examples because using test credentials prevents exposing my actual AWS credentials/ access while doing this project.

File Edit View

```
# config.py - TEMPORARY for demonstration only
# WARNING: This is NOT safe for production! We'll fix it with Secrets Manager.

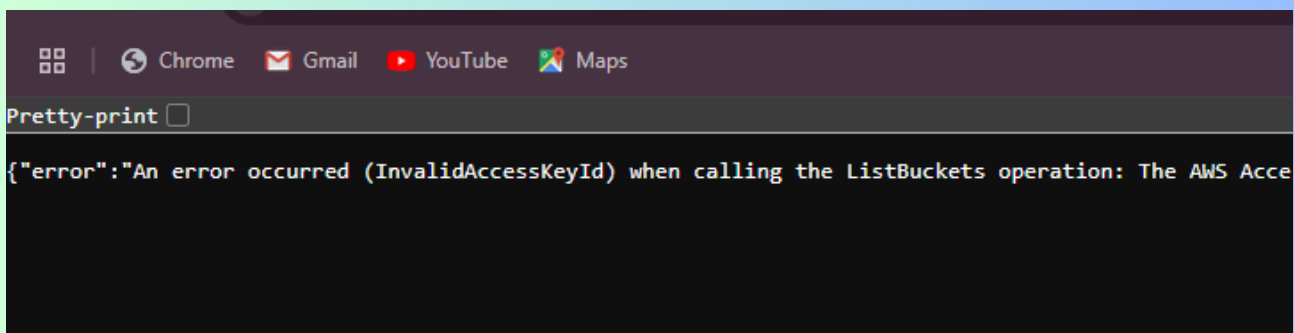
AWS_ACCESS_KEY_ID = "AKIAW3MEFRAFTQM5FHKE"
AWS_SECRET_ACCESS_KEY = "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"
AWS_REGION = "us-east-2"
```



Using my own AWS credentials

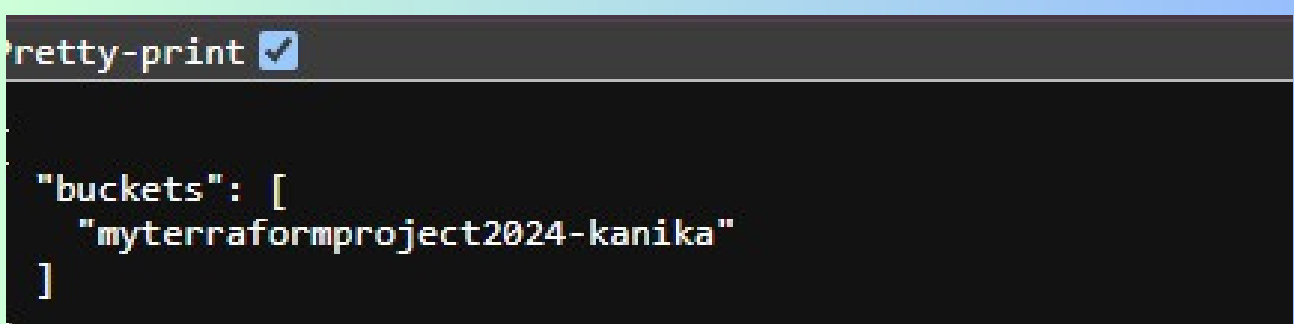
As an extension for this project, I also decided to run the web app locally using my own AWS credentials. To set up my virtual environment, I installed packages like: boto3, uvicorn and fast API, which are packages that my app's main file (app.py) uses to connect my web app to AWS.

When I first ran the app, I ran into an error (InvalidAccessKeyId) because I was using test credentials inside config.py, those credentials don't actually point to the real AWS account.



A screenshot of a web browser window. The address bar shows icons for Chrome, Gmail, YouTube, and Maps. Below the address bar, there is a 'Pretty-print' button with a checkbox. The main content area displays a JSON error message: {"error": "An error occurred (InvalidAccessKeyId) when calling the ListBuckets operation: The AWS Acce".

To resolve the 'InvalidAccessKeyId' error, I updated the configuration file (config.py) to use my AWS account's access key ID and secret access key- no more test credentials.



A screenshot of a web browser window. The address bar shows icons for Chrome, Gmail, YouTube, and Maps. Below the address bar, there is a 'Pretty-print' button with a checked checkbox. The main content area displays a JSON response: {"buckets": ["myterraformproject2024-kanika"]}



Pushing Insecure Code to GitHub

Once I updated the web app code with credentials, I forked the repository to simulate what it is like to push insecure code into your public repo. A fork is different from a clone because it also creates a new repository in your GitHub account (where clone just copies your code into your local computer).

To connect my local repository to the forked repository, I ran the command "git remote set-url origin". Then I used git add and git commit to save the changes made to config.py. Finally, git push uploaded those changes to the remote origin (the forked repository).

GitHub blocked my push because its secret scanning feature detected that I was hardcoding AWS credentials. This is a good security feature because it stops me from revealing the sensitive information in a public repository.

[illegible]



Kanika Mathur
github.com/KanikaGenesis

NextWork.org

Secrets Manager

Secrets Manager is a service that helps you securely store and manage secrets, such as database credentials, API keys, and other sensitive information. I'm using it to store my AWS Access Key ID and Secret Key credentials. Other common use cases include Database credentials, usernames and passwords for databases like MySQL, PostgreSQL, SQL Server, API keys (API keys for accessing third-party services), OAuth tokens (tokens for authentication and authorization), any other sensitive information (you can store any text-based secret in Secrets Manager).

Another feature in Secrets Manager is secrets rotation, which means the automatic rotation of the value of keys on a consistent schedule. Secrets Manager will generate new value for those secrets, and it is useful in high-risk situations when you'd like to minimize how long an attacker has access to your databases / API keys if it ever infiltrates your credentials/services.



Sample Code from Secrets Manager

Secrets Manager provides sample code in various languages, like Java, JavaScript, Python3. This is helpful because you can use these code samples directly in my config.py to update the way I retrieve credentials. Now I can use functions to connect to Secret Manager.

Sample code

Use these code samples to retrieve the secret in your application.

Java | JavaScript | C# | **Python3** | Ruby | Go | Rust

```
6 import boto3
7 from botocore.exceptions import ClientError
8
9
10 def get_secret():
11
12     secret_name = "aws-access-key-secretsmanager"
13     region_name = "us-east-1"
14
15     # Create a Secrets Manager client
16     session = boto3.session.Session()
17     client = session.client(
18         service_name='secretsmanager',
19         region_name=region_name
20     )
```

Python | Line 1, column 1 | ✖ Errors: 0 | ⚠ Warnings: 0



Updating the web app code

I updated the config.py file to retrieve the secret stored in my Secrets Manager programmatically. The get_secret() function will set a connection to Secrets Manager using boto3 and then it will try to retrieve the value of the secret with the secret name that I used when I set it up and store that secret value in a variable called "secret".

I also added code to config.py to extract the values of the secret that I stored in Secrets Manager. This is important because the configuration file now no longer has hardcoded credentials inside. It uses the function suggested by the code sample and splits up the secret's value to store the access key ID and secret access key in the same variables that my app.py will continue to use.

```
config.py M X
config.py > ...
You, 4 seconds ago | 2 authors (You and one other)
1 import boto3
2 from botocore.exceptions import ClientError
3
4
5 def get_secret():
6
7     secret_name = "aws-access-key-secretsmanager"
8     region_name = "us-east-1"
9
10    # Create a Secrets Manager client
11    session = boto3.session.Session()
12    client = session.client(
13        service_name='secretsmanager',
14        region_name=region_name
15    )
16
17    try:
18        get_secret_value_response = client.get_secret_value(
19            SecretId=secret_name
20        )
21    except ClientError as e:
22        # For a list of exceptions thrown, see
23        # https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretVa
24        raise e
25
26    secret = get_secret_value_response['SecretString']
27
```




Rebasing the repository

Git rebasing is a process of changing the command history of my repository. I used it to drop a commit where I hardcoded the AWS credentials. This was necessary because my AWS credentials are still living somewhere in my repository's commit history otherwise. And if it is still living in the commit history, it is still discoverable by attackers.

A merge conflict occurred during rebasing because the commit I removed had the hardcoded credentials, and for the next commit also I changed the config.py file. So Git's got confused. Git has paused the rebase and asked me to resolve the conflicts first. To resolve these conflicts, I needed to manually edit the conflicting file. I deleted the entire section between <<<<<<< HEAD and ===== (this removes the hardcoded credentials) Then, at the end of the file, I deleted the ===== line, I deleted the >>>>>>> feature-branch line, and I saved the file.

```
Command Prompt - git rebase X + v
pick 6c3a9ca first commit
pick 7e6497c Delete .DS_Store
pick 8c63f73 Update config.py
pick 201e292 Update config.py
pick 9186d38 Update config.py
pick 440e40c updated config.py
pick 60279e1 Updated config.py with Secrets Manager credentials

# Rebase 60279e1 onto fd81fbb (7 commands)
#
# Commands:
# p, pick <commit> = use commit
# r, reword <commit> = use commit, but edit the commit message
# e, edit <commit> = use commit, but stop for amending
# s, squash <commit> = use commit, but meld into previous commit
# f, fixup [-C | -c] <commit> = like "squash" but keep only the previous
#                               commit's log message, unless -C is used, in which case
#                               keep only this commit's message; -c is same as -C but
#                               opens the editor
# x, exec <command> = run command (the rest of the line) using shell
# b, break = stop here (continue rebase later with 'git rebase --continue')
# d, drop <commit> = remove commit
# l, label <label> = label current HEAD with a name
# t, reset <label> = reset HEAD to a label
# m, merge [-C <commit> | -c <commit>] <label> [# <oneline>]
#     create a merge commit using the original merge commit's
#     message (or the oneline, if no original merge commit was
git/rebase-merge/git-rebase-todo [unix] (21:32 17/03/2025)
~/Documents/nextwork-security-secretsmanager/.git/rebase-merge/git-rebase-todo [unix] 38L, 1713B
```



Kanika Mathur
github.com/KanikaGenesis

NextWork.org

Final Code in Repo

Once the merge conflict was resolved (I have updated the config.py, I verified that my GitHub repository is not exposing any of my credentials. I visited the config.py in the forked repository and can see that it uses code to extract my credentials instead of any hardcoding. I also successfully pushed my code without any blocking from GitHub Secret Scanning.

```
nextwork-security-secretsmanager / config.py
KanikaGenesis config file with credentials coming from Secrets Manager

Code Blame 36 lines (28 loc) · 1.09 KB

1  import boto3
2  from botocore.exceptions import ClientError
3  import json
4
5
6  def get_secret():
7
8      secret_name = "aws-access-key-secretsmanager"
9      region_name = "us-east-1"
10
11      # Create a Secrets Manager client
12      session = boto3.session.Session()
13      client = session.client(
14          service_name='secretsmanager',
15          region_name=region_name
16      )
17
18      try:
19          get_secret_value_response = client.get_secret_value(
20              SecretId=secret_name
21          )
22      except ClientError as e:
23          # For a list of exceptions thrown, see
24          # https://docs.aws.amazon.com/secretsmanager/latest/apireference/API_GetSecretValue.html
25          raise e
26
27      secret = get_secret_value_response['SecretString']
28      return json.loads(secret)
29
30 # Retrieve credentials from Secrets Manager
31 credentials = get_secret()
```



NextWork.org

Everyone should be in a job they love.

Check out nextwork.org for more projects

