

## 개요

이번 AWS 관리기초 과제에서 선택한 가이드항목은 AWS를 내가 처음 사용하기도 하고 AWS를 사용할 때 가장 기본적으로 적용해야하는 것들을 선택했다.

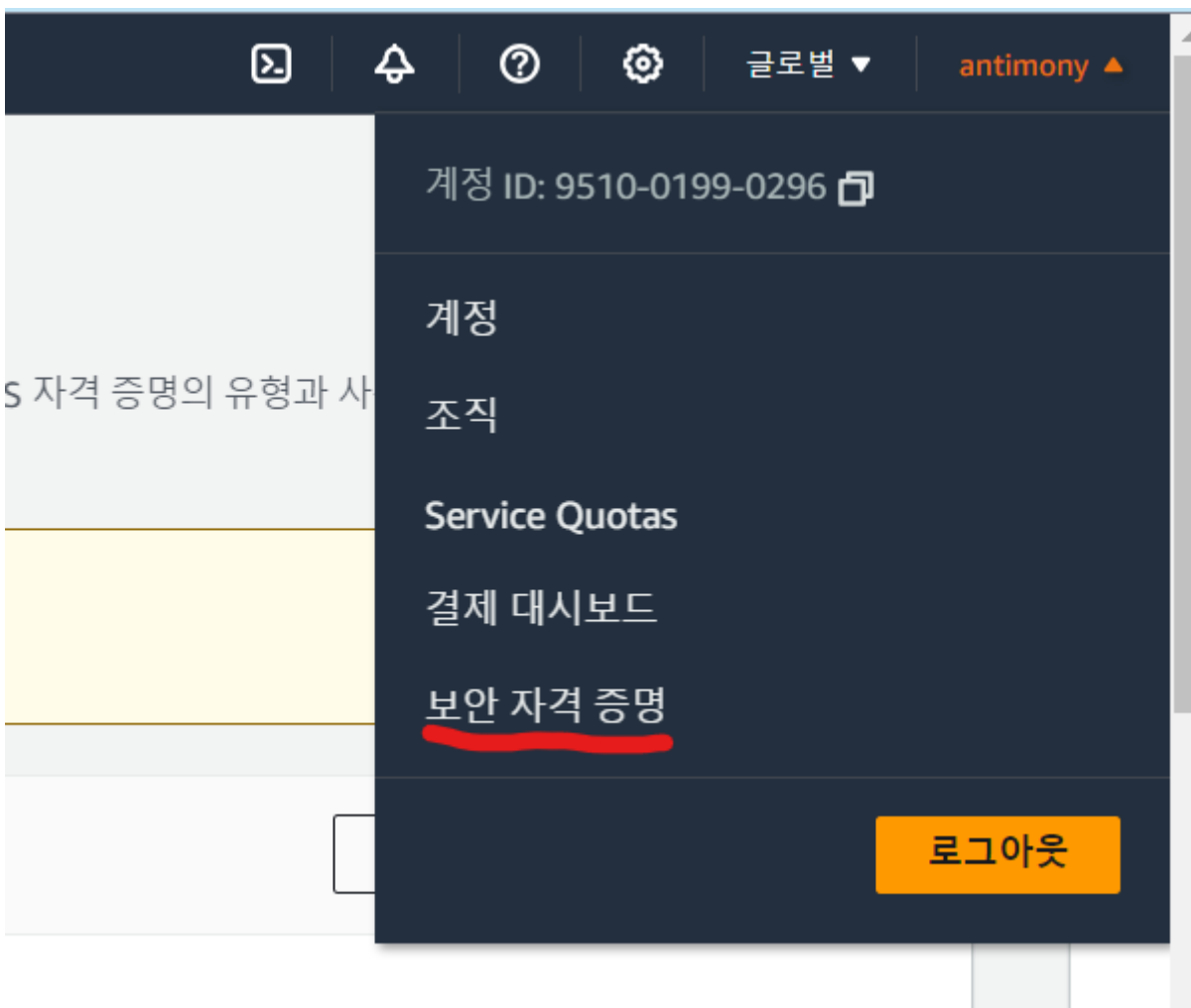
- 관리 콘솔 접근 시 MFA 적용
- 인스턴스 세부 정보에서 "EC2 인스턴스 연결" 기능 비활성화
- Access key 주기적 변경 -IAM 계정 비밀번호 복잡도 및 변경 주기 정책 설정

### 관리 콘솔 접근 시 MFA 적용

관리 콘솔을 접근하기 위해 로그인 시 2차인증인 MFA를 적용해 보안을 강화하는 것이다.

#### 취약점 확인

- root 및 IAM 계정으로 관리 콘솔 로그인 시 MFA를 적용하지 않은 경우 **취약**
- root 및 IAM 계정으로 관리 콘솔 로그인 시 MFA를 적용한 경우 **취약하지 않음**

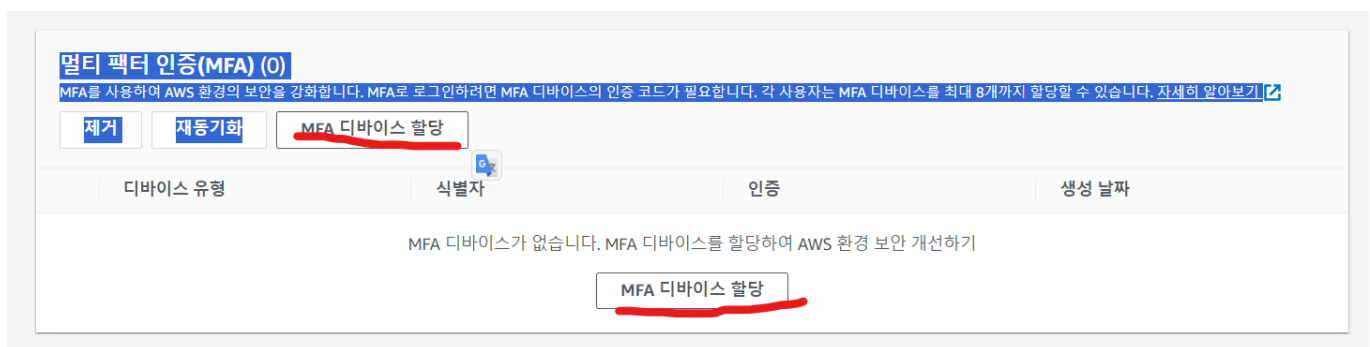


root 및 IAM 계정으로 관리 콘솔 로그인 후 오른쪽 상단의 계정명 클릭 후 보안 자격 증명 클릭



[멀티 팩터 인증(MFA)] 항목에서 MFA 할당 여부 확인

### 취약점 조치 방법



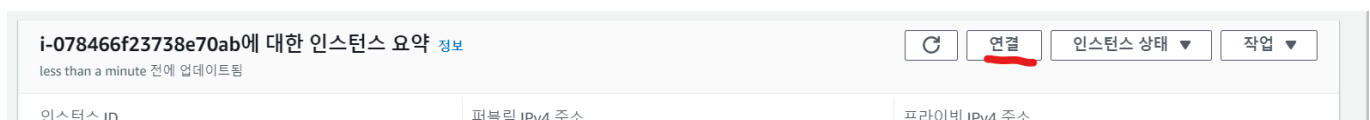
root 또는 IAM 계정으로 관리 콘솔 로그인 후 오른쪽 상단의 계정명 클릭, 보안 자격 증명 클릭, 멀티 팩터 인증 (MFA) 항목에서 **MFA 디바이스 할당** 클릭

인스턴스 세부 정보에서 "EC2 인스턴스 연결" 기능 비활성화

### 취약점 확인

- [EC2 인스턴스 연결] 기능이 활성화되어 있는 경우 취약
- [EC2 인스턴스 연결] 기능이 비활성화되어 있는 경우 취약하지 않음

EC2 서비스에서 인스턴스 -> 인스턴스 세부정보 -> 연결 클릭



EC2 인스턴스 연결 탭의 연결 버튼 클릭 시 인스턴스 접속 가능 여부 확인

```

aws | 서비스 | Q 검색 | [알트+S]

https://ubuntu.com/aws/pro

Expanded Security Maintenance for Applications is not enabled.

0 updates can be applied immediately.

Enable ESM Apps to receive additional future security updates.
See https://ubuntu.com/esm or run: sudo pro status

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-5-234:~$

```

## 취약점 조치 방법

```

The list of available updates is more than a week old.
To check for new updates run: sudo apt update

The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-172-31-5-234:~$ ec2-instance-connect
ec2-instance-connect: command not found
ubuntu@ip-172-31-5-234:~$ sudo dpkg -l | grep ec2-instance-connect
ii  ec2-instance-connect  1.1.14-0ubuntu1.1      all
    Configures ssh daemon to accept EC2 Instance Connect ssh keys
ubuntu@ip-172-31-5-234:~$ sudo dpkg -r ec2-instance-connect
(Reading database ... 64725 files and directories currently installed.)
Removing ec2-instance-connect (1.1.14-0ubuntu1.1) ...
Deleted system user ec2-instance-connect
ubuntu@ip-172-31-5-234:~$

```

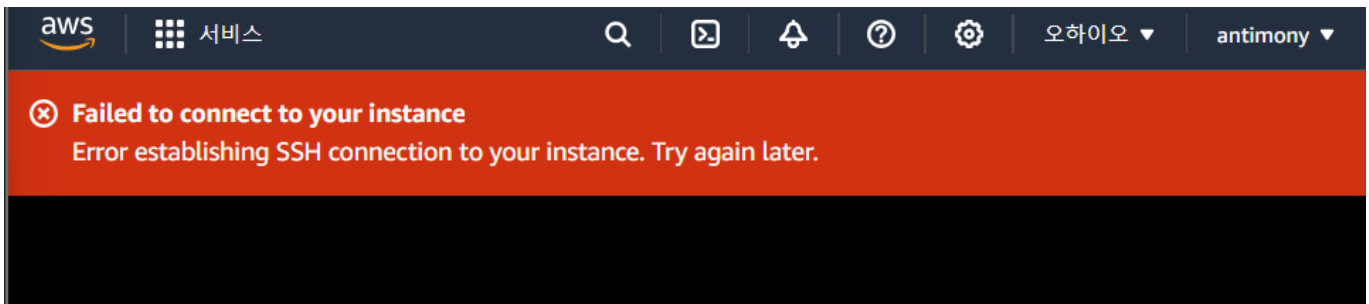
## 설치된 ec2-instance-connect 패키지 삭제

```

# 데비안 계열
sudo dpkg -r ec2-instance-connect

```

```
# 레드햇 계열
sudo yum remove ec2-instance-connect
```



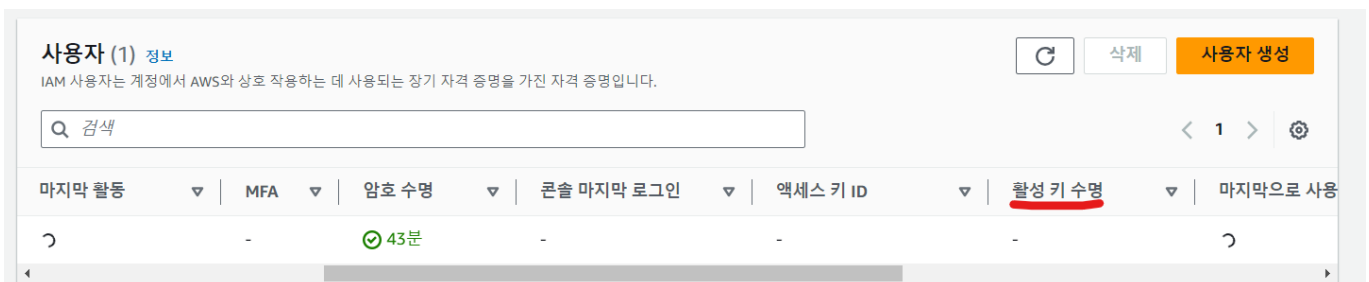
접속 불가 확인

## Access key 주기적 변경

AWS 리소스 핸들링을 위해 IAM 계정에 Access key(Access key ID/Seret access key)를 발급한 경우 주기적으로 변경 필요하다.

## 취약점 확인

- Access key 변경 주기를 지정하지 않고, 주기적으로 Access key를 변경하지 않은 경우 **취약**
- Access key 변경 주기를 지정하고, 주기적으로 Access key를 변경한 경우 **취약하지 않음**
- 관리 콘솔에서 [IAM] 검색 → [사용자] 메뉴 클릭
- \*[활성 키 수명(Active key age)]\*이 내부 정책에서 정의한 Access key 변경 주기를 초과하는지 여부 확인



## 취약점 조치 방법

보안, 자격 증명 및 규정 준수

# AWS Secrets Manager

## 보안 암호의 수명 주기 동안 보안 암호를 손쉽게 교체, 관리 및 검색

AWS Secrets Manager는 애플리케이션, 서비스 및 IT 리소스에 대한 액세스를 보호하는 데 도움이 됩니다. 데이터베이스 자격 증명, API 키 및 기타 보안 암호를 수명 주기 동안 손쉽게 교체하고 관리하고 검색할 수 있습니다.

시작하기

데이터베이스 자격 증명 또는 다른 유형의 보안 암호 저장

새 보안 암호 저장

관리콘솔에서 Secret Manager검색 후 새 보안 암호 저장 클릭

키/값 페어 정보

키/값

일반 텍스트

key

value

+ 행 추가

암호화 키 정보

Secrets Manager가 생성하는 KMS 키 또는 사용자가 생성한 고객 관리형 KMS 키를 사용하여 암호화할 수 있습니다.

aws/secretsmanager

▼

↺

새 키 추가

취소

다음

Access Key를 저장할 Secret Manager생성(키/값에는 임시 값 입력)

## 교체 구성 - 선택 사항

### 자동 교체 구성 정보

이 보안 암호를 자동으로 교체하도록 AWS Secrets Manager를 구성합니다.

☐ 자동 교체

### 교체 일정 정보

☐ 예약 표현식 빌더

☐ 예약 표현식

시간 단위

시간

시간 ▼

23

기간 - 선택 사항

4h

시간(h)을 입력합니다.

자동 교체 구성 비활성화

정책 (1132) 정보

정책은 권한을 정의하는 AWS의 객체입니다.

🔄

작업 ▼

삭제

정책 생성

🔍 검색

필터링 기준 유형

모든 유형 ▼

< 1 2 3 4 5 6 7 ... 57 > ⚙️

정책 이름 ▲	유형 ▼	다음... ▼	설명
<input type="radio"/> AccessAnalyzerSer...	AWS 관리형	없음	Allow Access Analyzer to analyze resource metadata
<input type="radio"/> AdministratorAccess	AWS 관리형	없음	Provides full access to AWS services and resources

관리콘솔에서 IAM검색, 정책 메뉴, 정책 생성 클릭

IAM > 역할

**역할 (2) 정보** 🔄 삭제 역할 만들기

IAM 역할은 단기간 동안 유효한 자격 증명을 가진 특정 권한이 있는 자격 증명입니다. 신뢰할 수 있는 개체가 역할을 맡을 수 있습니다.

🔍 검색

<input type="checkbox"/>	역할 이름	▲ 신뢰할 수 있는 개체	마지막 활동 ▼
<input type="checkbox"/>	<a href="#">AWSServiceRoleForSupport</a>	AWS 서비스: support (서비스 연결 역할)	-
<input type="checkbox"/>	<a href="#">AWSServiceRoleForTrustedAdvisor</a>	AWS 서비스: trustedadvisor (서비스 연결 역할)	-

IAM 서비스에서 역할 메뉴, 역할 만들기 클릭

json을 활용해 json정책 생성

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "SecretsManagerReadWritePermission",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue",
        "secretsmanager:DescribeSecret",
        "secretsmanager:PutSecretValue",
        "secretsmanager:UpdateSecretVersionStage",
        "secretsmanager:ListSecretVersionIds"
      ],
      "Resource": [
        "SECRETS_MANAGER의 ARN"
      ]
    },
    {
      "Sid": "KMSWritePermission",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:Encrypt"
      ],
      "Resource": [
        "aws/secretsmanager의 ARN"
      ]
    },
    {
      "Sid": "SecretsManagerKMSTListingPermission",
      "Effect": "Allow",
      "Action": [
        "kms:ListKeys",
        "secretsmanager:ListSecrets"
      ],
      "Resource": "*"
    }
  ]
}
```

## 정보

## 신뢰할 수 있는 엔터티 유형

- |  |   |   |
|--|---|---|
| <p><b>○ AWS 서비스</b><br/>EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.</p>              | <p><b>○ AWS 계정</b><br/>사용자 또는 서드 파티에 속한 다른 AWS 계정의 엔티티가 이 계정에서 작업을 수행하도록 허용합니다.</p>     | <p><b>○ 웹 자격 증명</b><br/>지정된 외부 웹 자격 증명 공급자와 연동된 사용자가 이 역할을 맡아 이 계정에서 작업을 수행하도록 허용합니다.</p> |
| <p><b>○ SAML 2.0 연동</b><br/>기업 디렉터리에서 SAML 2.0과 연동된 사용자가 이 계정에서 작업을 수행할 수 있도록 허용합니다.</p> | <p><b>○ 사용자 지정 신뢰 정책</b><br/>다른 사용자가 이 계정에서 작업을 수행할 수 있도록 사용자 지정 신뢰 정책을 생성할 수 있습니다.</p> |   |

## 사용 사례

EC2, Lambda 등의 AWS 서비스가 이 계정에서 작업을 수행하도록 허용합니다.

## 서비스 또는 사용 사례

Lambda

지정된 서비스에 대한 사용 사례를 선택합니다.

## 사용 사례

- **Lambda**  
Allows Lambda functions to call AWS services on your behalf.

## Json을 활용해 만든 정책과 AWSLambdaBasicExecutionRole 정책 할당

## Lambda > 함수

## 함수 (0)

마지막으로 가져온 항목 2분 전

작업 ▼

## 함수 생성

🔍 태그 및 속성별 필터 또는 키워드별 검색

< 1 > 

함수 이름	설명	패키지 유형	런타임	마지막 수정
-------	----	--------	-----	--------

표시할 데이터가 없습니다.



**런타임 정보**  
함수를 작성하는 데 사용할 언어를 선택합니다. 콘솔 코드 편집기는 Node.js, Python 및 Ruby만 지원합니다.

Python 3.11

**아키텍처 정보**  
함수 코드에 대해 원하는 명령 세트 아키텍처를 선택합니다.

☒ x86\_64  
☐ arm64

**권한 정보**  
기본적으로 Lambda는 Amazon CloudWatch Logs에 로그를 업로드하는 권한을 가진 실행 역할을 생성합니다. 이 기본 역할은 나중에 트리거를 추가할 때 사용자 지정할 수 있습니다.

▼ 기본 실행 역할 변경

**실행 역할**  
함수에 대한 권한을 정의하는 역할을 선택합니다. 사용자 지정 역할을 생성하려면 [IAM 콘솔](#)로 이동하십시오.

☐ 기본 Lambda 권한을 가진 새 역할 생성  
☒ 기존 역할 사용  
☐ AWS 정책 템플릿에서 새 역할 생성

**기존 역할**  
생성한 기존 역할 중에 이 Lambda 함수와 함께 사용할 역할을 선택합니다. 이 역할에는 Amazon CloudWatch Logs에 로그를 업로드할 수 있는 권한이 있어야 합니다.

test

IAM 콘솔에서 [test 역할을 확인](#)합니다.

관리 콘솔에서 Lambda검색, 함수 생성 클릭

**권한 정책 (1/2)**  
사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.

필터링 기준 유형: 모든 유형

검색

정액 이름	유형	연결 방식
<input type="checkbox"/> IAMUserChangePassword	AWS 관리형	직접
<input checked="" type="checkbox"/> test	고객 인라인	인라인

▶ 권한 경계 (설정되지 않음)

함수 생성 설정, Lambda을 활용해 함수 작성, Deploy클릭

```
import boto3
import json
import logging
import os
import time

logger = logging.getLogger()
logger.setLevel(logging.INFO)

def lambda_handler(event, context):
```

```

arn = event['SecretId']
token = event['ClientRequestToken']
step = event['Step']
# Setup the client
secretsmanager_client = boto3.client('secretsmanager')
# Make sure the version is staged correctly
metadata = secretsmanager_client.describe_secret(SecretId=arn)
logging.info(repr(metadata))
versions = metadata['VersionIdsToStages']
if token not in versions:
    logger.error("Secret version %s has no stage for rotation of secret %s." %
(token, arn))
    raise ValueError("Secret version %s has no stage for rotation of secret
%s." % (token, arn))
    if "AWSCURRENT" in versions[token]:
        logger.info("Secret version %s already set as AWSCURRENT for secret %s." %
(token, arn))
        return
    elif "AWSPENDING" not in versions[token]:
        logger.error("Secret version %s not set as AWSPENDING for rotation of
secret %s." % (token, arn))
        raise ValueError("Secret version %s not set as AWSPENDING for rotation of
secret %s." % (token, arn))
    if step == "createSecret":
        logging.debug("createSecret %s" % arn)
        logging.info("for IAM user access keys secret creation is handled by IAM
")
    elif step == "setSecret":
        logging.debug("setSecret %s" % arn)
        current_dict = get_secret_dict(secretsmanager_client, arn, "AWSCURRENT",
required_fields=['username'])
        username = current_dict['username']
        master_dict = get_secret_dict(secretsmanager_client,
current_dict['masterarn'], "AWSCURRENT")
        master_iam_client = boto3.client('iam',
aws_access_key_id=master_dict['accesskey'],
aws_secret_access_key=master_dict['secretkey'])
        # load any pre-existing access keys. sorted by created descending. if the
count is 2+ remove the oldest key
        existing_access_keys =
sorted(master_iam_client.list_access_keys(UserName=username)['AccessKeyMetadata'],
key=lambda x: x['CreateDate'])
        if len(existing_access_keys) >= 2:
            logger.info("at least 2 access keys already exist. deleting the oldest
version: %s" % existing_access_keys[0]['AccessKeyId'])
            master_iam_client.delete_access_key(UserName=username,
AccessKeyId=existing_access_keys[0]['AccessKeyId'])
            # request new access key and gather the response
            new_access_key = master_iam_client.create_access_key(UserName=username)
            current_dict['accesskey'] = new_access_key['AccessKey']['AccessKeyId']
            current_dict['secretkey'] = new_access_key['AccessKey']['SecretAccessKey']
            logging.info('applying new secret value to AWSPENDING')
            # save the new access key to the pending secret
            secretsmanager_client.put_secret_value(SecretId=arn,

```

```

ClientRequestToken=token, SecretString=json.dumps(current_dict), VersionStages=
['AWSPENDING'])
    elif step == "testSecret":
        logging.debug("testSecret %s" % arn)
        # load the pending secret for testing
        pending_dict = get_secret_dict(secretsmanager_client, arn, "AWSPENDING",
required_fields=['username'], token = token)
        # attempt to call an iam service using the credentials
        test_client = boto3.client('iam',
aws_access_key_id=pending_dict['accesskey'],
aws_secret_access_key=pending_dict['secretkey'])
        try:
            test_client.get_account_authorization_details()
        except test_client.exceptions.ClientError as e:
            # the test fails if and only if Authentication fails. Authorization
failures are acceptable.
            if e.response['Error']['Code'] == 'AuthFailure':
                logging.error("Pending IAM secret %s in rotation %s failed the
test to authenticate. exception: %s" % (arn, pending_dict['username'], repr(e)))
                raise ValueError("Pending IAM secret %s in rotation %s failed the
test to authenticate. exception: %s" % (arn, pending_dict['username'], repr(e)))
            elif step == "finishSecret":
                logging.debug("finishSecret %s" % arn)
                # finalize the rotation process by marking the secret version passed in as
the AWSCURRENT secret.
                metadata = secretsmanager_client.describe_secret(SecretId=arn)
                current_version = None
                for version in metadata["VersionIdsToStages"]:
                    if "AWSCURRENT" in metadata["VersionIdsToStages"][version]:
                        if version == token:
                            # The correct version is already marked as current, return
                            logger.info("finishSecret: Version %s already marked as
AWSCURRENT for %s" % (version, arn))
                            return
                        current_version = version
                        break
                # finalize by staging the secret version current
                secretsmanager_client.update_secret_version_stage(SecretId=arn,
VersionStage="AWSCURRENT", MoveToVersionId=token,
RemoveFromVersionId=current_version)
                logger.info("finishSecret: Successfully set AWSCURRENT stage to version %s
for secret %s." % (token, arn))
            else:
                raise ValueError("Invalid step parameter")

def get_secret_dict(secretsmanager_client, arn, stage, required_fields=[],
token=None):
    # Only do VersionId validation against the stage if a token is passed in
    if token:
        secret = secretsmanager_client.get_secret_value(SecretId=arn,
VersionId=token, VersionStage=stage)
    else:
        secret = secretsmanager_client.get_secret_value(SecretId=arn,

```

```

VersionStage=stage)
plaintext = secret['SecretString']
secret_dict = json.loads(plaintext)
# Run validations against the secret
for field in required_fields:
    if field not in secret_dict:
        raise KeyError("%s key is missing from secret JSON" % field)
# Parse and return the secret JSON string
return secret_dict

```

JSON(2)를 활용해 IAM 계정이 자신의 패스워드와 Access Key를 변경하도록 하는 정책 생성

```

{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:ListUsers",
        "iam:GetAccountPasswordPolicy"
      ],
      "Resource": "*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:*AccessKey*",
        "iam:ChangePassword",
        "iam:GetUser",
        "iam:*ServiceSpecificCredential*",
        "iam:*SigningCertificate*"
      ],
      "Resource": [
        "arn:aws:iam::*:user/${aws:username}"
      ]
    }
  ]
}

```

Access Key를 자동으로 교체할 IAM 계정에 JSON(2)를 활용해 만든 정책 할당

AWSLambda\_FullAccess 정책을 할당한 IAM계정의 AWS CLI접속후 명령어 실행

```

aws lambda add-permission --function-name [lambda의 ARN] --principal
secretsmanager.amazonaws.com --action lambda:InvokeFunction --statement-id
SecretsManagerAccess

```

13 / 17

## 교체 일정

☒ 예약 표현식 빌더

☐ 예약 표현식

시간 단위

일

일 ▼

90

기간 - 선택 사항

4h

시간(h)을 입력합니다.


☒ 보안 암호가 저장되면 즉시 교체합니다. 다음 교체가 일정에 따라 시작됩니다.

## 교체 함수

### Lambda 교체 함수

이 보안 암호를 교체할 수 있는 Lambda 함수를 선택합니다.

test ▼


[함수 생성](#)


Cancel


저장

관리콘솔에서 Secret Manager 검색, 생성한 보안 암호 클릭, 보안 암호값 검색 클릭, 편집 클릭

보안 암호값 편집

- accesskey: Access key를 변경하고자 하는 IAM계정의 Access Key값
- secretkey: Access key를 변경하고자 하는 IAM 계정의 Secret Access Key값
- username: Access key를 변경하고자 하는 IAM계정명
- masterarn: 해당 Secret Manager의 ARN

교체편집 클릭, 교체구성 설정

 test

15 / 17

암호 정책 정보

IAM 사용자에게 대한 암호 요구 사항을 구성합니다.

이 AWS 계정은 다음과 같은 기본 암호 정책을 사용합니다.

암호 최소 길이

8자

암호 강도

다음 문자 유형 조합 중 최소 3개 포함:

• 대문자

• 소문자

• 숫자

• 영숫자를 제외한 문자

기타 요구 사항

• 비밀번호가 만료되지 않음

• AWS 계정 이름 또는 이메일 주소와 동일할 수 없음

취약점 조치 방법

암호 정책 정보

IAM 사용자에게 대한 암호 요구 사항을 구성합니다.

편집

이 AWS 계정은 다음과 같은 기본 암호 정책을 사용합니다.

암호 최소 길이

8자

암호 강도

다음 문자 유형 조합 중 최소 3개 포함:

• 대문자

• 소문자

• 숫자

• 영숫자를 제외한 문자

기타 요구 사항

• 비밀번호가 만료되지 않음

• AWS 계정 이름 또는 이메일 주소와 동일할 수 없음

암호 정책에서 편집클

릭

16 / 17



## 암호 정책

☐ IAM 기본값  
기본 암호 요구 사항을 적용합니다.
 ☒ 사용자 지정  
사용자 지정 암호 요구 사항을 적용합니다.

암호 최소 길이,  
최소 문자 길이를 적용합니다.

자
 

6~128 사이여야 합니다.

암호 강도

- ☒ 1개 이상의 라틴 알파벳 대문자(A-Z) 필수
- ☒ 1개 이상의 라틴 알파벳 소문자(a-z) 필수
- ☒ 1개 이상의 숫자 필수
- ☒ 영숫자를 제외한 문자(! @ # \$ % ^ & \* ( ) \_ + - = [ ] { } | ' ) 1개 이상 필수

기타 요구 사항

- ☐ 암호 만료 활성화
- ☐ 암호 만료 시 관리자 재설정 필요
- ☒ 사용자 자신의 암호 변경 허용
- ☐ 암호 재사용 제한

취소
변경 사항 저장

사용자지정 선택후 위처럼 요구사항 선택

## 선택과제

내가 좋아하는 분야는 offensive security로 이에 해당하는 직문인 모의해킹, 레드팀 등의 직무를 갖고싶다. 화이트햇스쿨 이전 국정원에서 주관한 윤리적해커양성이라는 교육을 들은 후 국정원에 대한 목표가생겼다. 해당 목표를 달성하기 위해 2가지의 길을 선택중이다. 하나는 대학졸업후 바로 공채로 들어가기, 다른 회사에서 경력을 채운 뒤 경력직으로 들어가기.

## Reference

<https://rogue-gouda-f87.notion.site/MFA-e9289520fb5640d3aa885d43b353beaa>