import boto3

from botocore.exceptions import ClientError

import bcrypt

from datetime import datetime, timedelta

# Initialize DynamoDB client

dynamodb = boto3.client('dynamodb')

dynamodb\_resource = boto3.resource("dynamodb")

# Maximum number of login attempts before locking the account

MAX\_LOGIN\_ATTEMPTS = 5

# Lockout period (e.g., 15 minutes)

LOCKOUT\_DURATION\_MINUTES = 15

def generate\_client\_table\_name(company\_id):

"""Generate the Table name string based on the id"""

return f"{company\_id}Employee"

def generate\_auth\_table\_name(company\_id):

"""Generate the Table name string based on the id"""

return f"{company\_id}Employee\_Auth"

def get\_id\_from\_CompanyName(company\_name):

"""Return the company ID based on company name"""

try:

company\_profile\_table = dynamodb\_resource.Table("CompaniesProfiles")

response = company\_profile\_table.get\_item(

Key={"organizationName": company\_name}

)

if "Item" in response:

return response["Item"].get("id", None)

except ClientError as e:

print(f"Error fetching company ID for {company\_name}: {e}")

# Lambda function handler

def lambda\_handler(event, context):

# Extract email and password from the event

email = event.get('email')

password = event.get('password')

organizationName = event.get('organizationName')

company\_id = get\_id\_from\_CompanyName(organizationName)

auth\_table\_name = generate\_auth\_table\_name(company\_id)

client\_table\_name = generate\_client\_table\_name(company\_id)

# Validate parameters

if not email or not password or not auth\_table\_name or not client\_table\_name:

return {

'statusCode': 400,

'body': 'Email, password, auth\_table\_name, and client\_table\_name are required.'

}

# Step 1: Query the {auth\_table\_name} to get the user details by email

try:

response = dynamodb.get\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}}

)

if 'Item' not in response:

return {

'statusCode': 404,

'body': 'User not found.'

}

auth\_data = response['Item']

guid = auth\_data['GUID']['S'] # Retrieve the GUID for the user

login\_attempts = int(auth\_data.get('LoginAttempts', {}).get('N', '0'))

blocked\_until = auth\_data.get('BlockedUntil', {}).get('S')

except Exception as e:

return {

'statusCode': 500,

'body': f'Error querying auth table: {str(e)}'

}

# Check if user is blocked due to too many failed login attempts

if blocked\_until:

blocked\_until\_dt = datetime.strptime(blocked\_until, '%Y-%m-%dT%H:%M:%S')

if blocked\_until\_dt > datetime.now():

return {

'statusCode': 403,

'body': f'Account locked. Please try again after {blocked\_until\_dt}.'

}

# Step 2: Verify the provided password against either FirstTimePassword or UserEnteredPassword (using bcrypt)

first\_time\_password\_db = auth\_data.get('FirstTimePassword', {}).get('S')

user\_entered\_password\_db = auth\_data.get('UserEnteredPassword', {}).get('S')

password\_correct = False # Flag to check if password matches

# Check first-time password (if exists)

if first\_time\_password\_db:

# Check if FirstTimePassword is hashed

if first\_time\_password\_db.startswith('$2b$'): # It's a bcrypt hashed password

if bcrypt.checkpw(password.encode('utf-8'), first\_time\_password\_db.encode('utf-8')):

password\_correct = True

print("First-time password (hashed) used for login.")

else:

# FirstTimePassword is in plain text (old system), so check it directly

if password == first\_time\_password\_db:

password\_correct = True

print("First-time password (plain text) used for login.")

# Hash the plain-text password and store it as UserEnteredPassword

hashed\_password = bcrypt.hashpw(password.encode('utf-8'), bcrypt.gensalt()).decode('utf-8')

try:

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET UserEnteredPassword = :new\_password, FirstTimePassword = :hashed\_password, LoginAttempts = :zero\_attempts",

ExpressionAttributeValues={

':new\_password': {'S': hashed\_password},

':hashed\_password': {'S': hashed\_password}, # Update FirstTimePassword to hashed version

':zero\_attempts': {'N': '0'}

},

ReturnValues="UPDATED\_NEW"

)

print(f"Updated UserEnteredPassword for {email}.")

except Exception as e:

return {

'statusCode': 500,

'body': f"Error updating UserEnteredPassword: {str(e)}"

}

# Check user-entered password (for subsequent logins)

if not password\_correct and user\_entered\_password\_db:

if bcrypt.checkpw(password.encode('utf-8'), user\_entered\_password\_db.encode('utf-8')):

password\_correct = True

print("User-entered password used for login.")

# Reset the failed login attempts

try:

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET LoginAttempts = :zero\_attempts",

ExpressionAttributeValues={

':zero\_attempts': {'N': '0'}

},

ReturnValues="UPDATED\_NEW"

)

except Exception as e:

return {

'statusCode': 500,

'body': f"Error resetting login attempts: {str(e)}"

}

# If password is incorrect

if not password\_correct:

login\_attempts += 1

if login\_attempts >= MAX\_LOGIN\_ATTEMPTS:

# Lock the account by setting a blocked timestamp

blocked\_until\_dt = datetime.now() + timedelta(minutes=LOCKOUT\_DURATION\_MINUTES)

blocked\_until\_str = blocked\_until\_dt.strftime('%Y-%m-%dT%H:%M:%S')

try:

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET LoginAttempts = :attempts, BlockedUntil = :blocked\_time",

ExpressionAttributeValues={

':attempts': {'N': str(login\_attempts)},

':blocked\_time': {'S': blocked\_until\_str}

},

ReturnValues="UPDATED\_NEW"

)

return {

'statusCode': 403,

'body': f'Too many failed attempts. Account locked until {blocked\_until\_str}.'

}

except Exception as e:

return {

'statusCode': 500,

'body': f"Error updating login attempts: {str(e)}"

}

else:

# Update the login attempts without blocking

try:

dynamodb.update\_item(

TableName=auth\_table\_name,

Key={'Email': {'S': email}},

UpdateExpression="SET LoginAttempts = :attempts",

ExpressionAttributeValues={

':attempts': {'N': str(login\_attempts)}

},

ReturnValues="UPDATED\_NEW"

)

except Exception as e:

return {

'statusCode': 500,

'body': f"Error updating login attempts: {str(e)}"

}

return {

'statusCode': 403,

'body': 'Invalid password.'

}

# Step 3: Fetch the user's data from {client\_table\_name} using the GUID

try:

user\_data\_response = dynamodb.get\_item(

TableName=client\_table\_name,

Key={'GUID': {'S': guid}}

)

if 'Item' not in user\_data\_response:

return {

'statusCode': 404,

'body': 'User data not found.'

}

except Exception as e:

return {

'statusCode': 500,

'body': f'Error fetching user data: {str(e)}'

}

# Prepare user data for return (excluding sensitive fields)

user\_data = user\_data\_response['Item']

user\_data\_cleaned = {

'GUID': user\_data['GUID']['S'],

'FirstName': user\_data.get('FirstName', {}).get('S', 'N/A'),

'LastName': user\_data.get('LastName', {}).get('S', 'N/A'),

'Email': user\_data.get('Email', {}).get('S', 'N/A'),

'Sex': user\_data.get('Sex', {}).get('S', 'N/A'),

'AddressLine1': user\_data.get('AddressLine1', {}).get('S', 'N/A'),

'City': user\_data.get('City', {}).get('S', 'N/A'),

'State': user\_data.get('State', {}).get('S', 'N/A'),

'ZipCode': user\_data.get('ZipCode', {}).get('S', 'N/A'),

'Country': user\_data.get('Country', {}).get('S', 'N/A'),

'Manager': user\_data.get('Manager', {}).get('S', 'N/A'),

'Position': user\_data.get('Position', {}).get('S', 'N/A'),

'Division': user\_data.get('Division', {}).get('S', 'N/A'),

'Telephone': user\_data.get('Telephone', {}).get('S', 'N/A')

}

return {

'statusCode': 200,

'body': {

'message': 'Login successful.',

'user\_data': user\_data\_cleaned

}

}