Guide Complet : Sécurisation IAM AWS pour Applications SaaS

Version: 1.0
Date: Novembre 2025
Destiné à: Équipes de sécurité et DevSecOps

Table des Matières

Vue d'Ensemble

Contexte de Sécurité 2025

Selon le rapport Verizon 2024 sur les violations de données, **plus de 80% des incidents de sécurité cloud** sont liés à des configurations incorrectes, souvent dues à des règles d'accès trop permissives. En 2025, IAM constitue l'épine dorsale de la sécurité cloud, avec l'utilisation croissante d'outils IAM pilotés par IA pour la détection de menaces, la classification des données et la gouvernance.

Statistiques Clés

- **65% des violations de données** proviennent de contrôles d'accès trop permissifs (CISA 2024)
- **57% des escalades de privilèges** résultent d'autorisations excessives (Flexera 2024)
- **50% des violations d'identité** exploitent l'absence de vérifications contextuelles (Gartner 2024)

Principe du Moindre Privilège (PoLP)

1. Définition et Importance

Le principe du moindre privilège consiste à accorder aux utilisateurs et rôles **uniquement les actions spécifiques requises**, en évitant les permissions larges comme `"Action": "*"`.

2. Stratégies de Mise en Œuvre

2.1 Utilisation d'IAM Access Analyzer

```
# Analyser les permissions non utilisées
aws accessanalyzer list-analyzers

# Générer une politique de moindre privilège basée sur l'activité
aws accessanalyzer generate-finding-recommendations \
    --analyzer-arn arn:aws:access-analyzer:region:account-id:analyzer/analyzer-name \
    --resource-arn arn:aws:iam::account-id:role/role-name
```

IAM Access Analyzer analyse les services et actions que vos rôles IAM utilisent réellement, puis génère une politique de moindre privilège que vous pouvez utiliser.

2.2 Éviter les Politiques Trop Larges

❌ **Mauvaise Pratique:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
```

```
      "Action": "*",
      "Resource": "*"
    }
  ]
}
```

✅ **Bonne Pratique:**

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Resource": "arn:aws:s3:::my-saas-bucket/${aws:userid}/*"
    }
  ]
}
```

2.3 Politiques de Pré-Production

Avant de lancer votre application en production, vous pouvez **générer une politique IAM basée sur l'activité d'accès** d'un rôle IAM pendant la phase de développement.

```
# Générer une politique basée sur CloudTrail
aws iam generate-service-last-accessed-details \
    --arn arn:aws:iam::123456789012:role/MyRole
```

3. Audits Réguliers

Les politiques IAM doivent être auditées :
- **À chaque changement majeur d'infrastructure**
- **Au moins une fois tous les 3 mois**
- **Après tout incident de sécurité**

AWS Security Hub et CloudTrail fournissent une visibilité sur 47% des violations résultant d'une visibilité insuffisante des changements de configuration (Gartner 2024).

Authentification Multi-Facteurs (MFA)

1. Importance Critique

L'authentification multi-facteurs (MFA) est **essentielle** pour protéger les comptes privilégiés. 50% des violations d'identité en 2024 ont exploité l'absence de vérifications d'accès contextuel (Gartner).

2. Types de MFA Acceptables

Type de MFA	Sécurité	Cas d'Usage
Hardware MFA (YubiKey, Gemalto)	⭐⭐⭐⭐⭐	Comptes root, administrateurs
Virtual MFA (Google Authenticator, Authy)	⭐⭐⭐⭐	Utilisateurs réguliers
U2F Security Keys	⭐⭐⭐⭐⭐	Développeurs, ops

3. Stratégies d'Application MFA

3.1 MFA pour tous les Utilisateurs Console

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyAllExceptListedIfNoMFA",
      "Effect": "Deny",
      "NotAction": [
        "iam:CreateVirtualMFADevice",
        "iam:EnableMFADevice",
        "iam:GetUser",
        "iam:ListMFADevices",
        "iam:ListVirtualMFADevices",
        "iam:ResyncMFADevice",
        "sts:GetSessionToken"
      ],
      "Resource": "*",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": "false"
        }
      }
    }
```

```
  ]
}
```

Cette politique **bloque toutes les actions** sauf la configuration MFA si l'utilisateur n'est pas authentifié avec MFA.

3.2 MFA via Service Control Policies (SCP)

Pour une application au niveau organisationnel :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "BoolIfExists": {
          "aws:MultiFactorAuthPresent": "false"
        }
      }
    }
  ]
}
```

3.3 MFA pour AWS CLI

Les utilisateurs IAM doivent d'abord récupérer leur token MFA avec l'opération AWS STS `GetSessionToken` :

```
# Obtenir un token de session avec MFA
aws sts get-session-token \
    --serial-number arn:aws:iam::123456789012:mfa/user \
    --token-code 123456 \
    --duration-seconds 129600

# Utiliser les credentials temporaires
export AWS_ACCESS_KEY_ID=ASIAIOSFODNN7EXAMPLE
export AWS_SECRET_ACCESS_KEY=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
export AWS_SESSION_TOKEN=AQoDYXdzEJr...
```

4. Auto-Enregistrement MFA

Stratégie permettant aux utilisateurs qui n'ont pas encore de dispositif MFA enregistré de s'auto-inscrire lors de la connexion, permettant de sécuriser les environnements AWS avec MFA sans distribuer individuellement les dispositifs.

Gestion des Rôles IAM

1. Rôles vs Utilisateurs

Pourquoi Privilégier les Rôles ?

Critère	Rôles IAM	Utilisateurs IAM
Credentials	Temporaires	Permanents
Rotation	Automatique	Manuelle
Sécurité	⭐⭐⭐⭐⭐	⭐⭐⭐
Cas d'usage	Applications, services	Humains (limité)

Recommandation AWS: Utilisez des rôles IAM pour les utilisateurs humains et les charges de travail accédant à vos ressources AWS afin qu'ils s'appuient sur des **credentials temporaires**.

2. Credentials Temporaires

```
# Assumer un rôle pour obtenir des credentials temporaires
aws sts assume-role \
    --role-arn arn:aws:iam::123456789012:role/MyRole \
    --role-session-name MySession \
    --duration-seconds 3600
```

Les credentials temporaires :
- ✅ Se désactivent automatiquement après expiration
- ✅ Réduisent le risque de compromission
- ✅ Sont conformes aux meilleures pratiques 2025

3. IAM Roles for Service Accounts (IRSA) - Kubernetes/EKS

Pour les charges de travail Kubernetes sur EKS :

```
apiVersion: v1
kind: ServiceAccount
metadata:
```

```
  name: my-service-account
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/my-role
```

Avantages:
- **Moindre privilège** : Permissions spécifiques à un service account
- **Auditabilité** : Logs disponibles via CloudTrail
- **Isolation** : Chaque pod peut avoir ses propres permissions

4. Trust Policies et External ID

Pour les rôles cross-account ou partagés avec des tiers :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::EXTERNAL-ACCOUNT-ID:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "unique-external-id-12345"
        },
        "IpAddress": {
          "aws:SourceIp": "203.0.113.0/24"
        }
      }
    }
  ]
}
```

External ID protège contre le "confused deputy problem" où un attaquant pourrait tromper votre service pour utiliser ses permissions.

Multi-Tenant et Isolation

1. Isolation des Tenants avec ABAC

Attribute-Based Access Control (ABAC) permet de gérer la complexité des politiques IAM dans les environnements multi-tenants.

Principe:

Les attributs de session et de ressource doivent correspondre (ex: `TenantID: yellow` peut uniquement accéder aux ressources taggées `TenantID: yellow`).

Exemple de Politique ABAC:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "*",
      "Condition": {
        "StringEquals": {
          "s3:ExistingObjectTag/TenantID": "${aws:PrincipalTag/TenantID}"
        }
      }
    }
  ]
}
```

Cette politique permet aux utilisateurs d'accéder uniquement aux objets S3 dont le tag `TenantID` correspond à leur propre `TenantID`.

2. Politiques Générées Dynamiquement

Pour Lambda et EC2 dans des architectures multi-tenants :

```
import json
import boto3

def generate_tenant_policy(tenant_id):
    """Génère une politique IAM dynamique pour un tenant"""
    policy = {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Action": [
                    "dynamodb:GetItem",
                    "dynamodb:PutItem",
                    "dynamodb:Query"
                ],
                "Resource": f"arn:aws:dynamodb:region:account:table/TenantData",
```

```
                "Condition": {
                    "ForAllValues:StringEquals": {
                        "dynamodb:LeadingKeys": [tenant_id]
                    }
                }
            }
        ]
    }
    return json.dumps(policy)
```

3. Amazon Verified Permissions

Service permettant de gérer les politiques d'accès avec un **policy store par tenant** :

```
# Créer un policy store pour un tenant
aws verifiedpermissions create-policy-store \
    --validation-settings mode=STRICT \
    --description "Policy store for tenant-123"
```

Audit et Surveillance

1. Services de Surveillance IAM

Service	Fonction	Fréquence
AWS CloudTrail	Audit des appels API IAM	Temps réel
IAM Access Analyzer	Détection des accès externes	Continue
AWS Security Hub	Conformité et recommandations	Quotidienne
GuardDuty	Détection de menaces IAM	Temps réel

2. Alertes CloudWatch pour IAM

```
# Créer une alarme pour les changements de politique IAM
aws cloudwatch put-metric-alarm \
    --alarm-name IAM-Policy-Changes \
    --alarm-description "Alert on IAM policy changes" \
    --metric-name PolicyChanges \
    --namespace AWS/IAM \
    --statistic Sum \
    --period 300 \
```

```
    --threshold 1 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1
```

3. Filtres CloudWatch Logs pour Événements IAM

```
{
  "$.eventName": "AttachRolePolicy",
  "$.eventName": "DetachRolePolicy",
  "$.eventName": "PutRolePolicy",
  "$.eventName": "DeleteRolePolicy",
  "$.eventName": "CreateAccessKey",
  "$.eventName": "DeleteAccessKey"
}
```

4. Surveillance des Credentials Compromis

```
# Vérifier le rapport de credentials IAM
aws iam generate-credential-report
aws iam get-credential-report --output text --query 'Content' | base64 -d > credentials.csv

# Analyser les clés non utilisées depuis 90+ jours
awk -F, '$4 == "true" && $11 != "N/A" {
    cmd = "date -d "$11" +%s"
    cmd | getline last_used
    close(cmd)
    cmd = "date +%s"
    cmd | getline now
    close(cmd)
    if ((now - last_used) / 86400 > 90) print $1
}' credentials.csv
```

Automatisation et Conformité

1. Service Control Policies (SCP)

Les SCP permettent d'établir des **guardrails de permissions** au niveau organisationnel :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": [
```

```
        "iam:DeleteUser",
        "iam:DeleteRole"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalOrgID": "${aws:PrincipalOrgID}"
        }
      }
    }
  ]
}
```

2. AWS Organizations - Comptes Séparés

Recommandation: Utilisez des comptes séparés pour isoler les environnements de développement et de test des environnements de production, réduisant les risques de perturbations ou de conflits.

3. Identity Federation avec IAM Identity Center

Pour une gestion centralisée des accès :

```
# Configurer IAM Identity Center pour la fédération
aws sso-admin create-permission-set \
    --instance-arn arn:aws:sso:::instance/ssoins-1234567890abcdef \
    --name PowerUserAccess \
    --description "Power user access for developers"
```

4. Outils d'Automatisation

```
import boto3

def enforce_mfa_for_console_users():
    """Vérifie et applique MFA pour tous les utilisateurs console"""
    iam = boto3.client('iam')

    users = iam.list_users()['Users']

    for user in users:
        username = user['UserName']

        # Vérifier si l'utilisateur a un mot de passe console
        try:
            iam.get_login_profile(UserName=username)
            has_console = True
        except:
            has_console = False
```

```
        if has_console:
            # Vérifier MFA
            mfa_devices = iam.list_mfa_devices(UserName=username)['MFADevices']

            if not mfa_devices:
                print(f"WARNING: User {username} has console access without MFA!")
                # Ici, vous pourriez attacher une politique de refus
```

Checklist de Sécurité IAM

✅ Niveau Critique (Priorité 1)

- [ ] **Root Account MFA activé**
- [ ] **Pas d'access keys sur le compte root**
- [ ] **MFA appliqué pour tous les utilisateurs privilégiés**
- [ ] **Politiques basées sur le moindre privilège**
- [ ] **CloudTrail activé et logs conservés**
- [ ] **IAM Access Analyzer activé**
- [ ] **Pas de credentials codés en dur dans le code**
- [ ] **Rotation automatique des secrets (Secrets Manager)**

✅ Niveau Important (Priorité 2)

- [ ] **Service Control Policies (SCP) configurés**
- [ ] **Stratégie multi-comptes implémentée**
- [ ] **Identity Federation avec IAM Identity Center**
- [ ] **Politiques d'isolation multi-tenant (ABAC)**
- [ ] **Alarmes CloudWatch pour changements IAM**
- [ ] **Audit IAM trimestriel**
- [ ] **GuardDuty activé pour détection de menaces**
- [ ] **Credentials temporaires uniquement pour applications**

✅ Niveau Recommandé (Priorité 3)

- [ ] **Politiques générées par IAM Access Analyzer**
- [ ] **Tags de ressources appliqués systématiquement**

- [ ] **Conditions IP dans les trust policies**
- [ ] **External ID pour rôles tiers**
- [ ] **Session duration limitée (< 12h)**
- [ ] **Suppression automatique des clés inactives (> 90 jours)**
- [ ] **Security Hub pour conformité continue**
- [ ] **Documentation des rôles et permissions**

Références et Ressources

Documentation Officielle AWS

- IAM Best Practices
- IAM Access Analyzer
- SaaS Tenant Isolation with ABAC
- IAM Roles for Service Accounts

Rapports de Sécurité 2024-2025

- Verizon Data Breach Investigations Report 2024
- Gartner Identity Security Survey 2024
- CISA Cloud Security Guidelines 2024
- SailPoint State of Identity Security 2025

Outils et Services

- **AWS Security Hub** - Conformité et alertes
- **AWS CloudTrail** - Audit des API
- **AWS GuardDuty** - Détection de menaces
- **AWS Config** - Suivi des configurations
- **IAM Access Analyzer** - Analyse des politiques

Cas d'Usage Détaillés par Scénario

1. Architecture Serverless (Lambda + API Gateway)

Problématique

Applications serverless nécessitant des accès granulaires à DynamoDB, S3, et d'autres services AWS.

Solution IAM Optimale

Rôle d'exécution Lambda (un par fonction):

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DynamoDBReadWrite",
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:Query"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/Users",
        "arn:aws:dynamodb:us-east-1:123456789012:table/Users/index/*"
      ]
    },
    {
      "Sid": "S3UserUploads",
      "Effect": "Allow",
      "Action": [
        "s3:PutObject",
        "s3:PutObjectAcl"
      ],
      "Resource": "arn:aws:s3:::user-uploads-bucket/uploads/${aws:username}/*"
    },
    {
      "Sid": "CloudWatchLogs",
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:us-east-1:123456789012:log-group:/aws/lambda/my-function:*"
    },
    {
```

```
      "Sid": "SecretsManagerRead",
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/api/*"
    },
    {
      "Sid": "KMSDecrypt",
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt",
        "kms:DescribeKey"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/12345678-1234-1234-1234-12345678901
      "Condition": {
        "StringEquals": {
          "kms:ViaService": [
            "secretsmanager.us-east-1.amazonaws.com",
            "dynamodb.us-east-1.amazonaws.com"
          ]
        }
      }
    }
  ]
}
```

Trust Policy avec conditions restrictives:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Service": "lambda.amazonaws.com"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "aws:SourceAccount": "123456789012"
        },
        "ArnLike": {
          "aws:SourceArn": "arn:aws:lambda:us-east-1:123456789012:function:my-function"
        }
      }
    }
  ]
}
```

Terraform pour automation:

```
resource "aws_iam_role" "lambda_execution_role" {
  name = "lambda-execution-${var.function_name}"

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        Service = "lambda.amazonaws.com"
      }
      Action = "sts:AssumeRole"
      Condition = {
        StringEquals = {
          "aws:SourceAccount" = var.account_id
        }
        ArnLike = {
          "aws:SourceArn" = "arn:aws:lambda:${var.region}:${var.account_id}:function:${var.fun
        }
      }
    }]
  })

  tags = {
    Environment = var.environment
    ManagedBy   = "Terraform"
    Function    = var.function_name
  }
}

# Attach least privilege policy
resource "aws_iam_role_policy" "lambda_policy" {
  name = "lambda-policy-${var.function_name}"
  role = aws_iam_role.lambda_execution_role.id

  policy = data.aws_iam_policy_document.lambda_permissions.json
}
```

---

2. Architecture Container (ECS/EKS)

ECS Task Roles

Problématique: Containers nécessitant des accès à AWS sans credentials statiques.

Solution:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Sid": "S3BucketAccess",
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:ListBucket"
      ],
      "Resource": [
        "arn:aws:s3:::app-config-bucket",
        "arn:aws:s3:::app-config-bucket/*"
      ]
    },
    {
      "Sid": "ParameterStoreAccess",
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter",
        "ssm:GetParameters",
        "ssm:GetParametersByPath"
      ],
      "Resource": "arn:aws:ssm:us-east-1:123456789012:parameter/prod/app/*"
    },
    {
      "Sid": "RDSIAMAuth",
      "Effect": "Allow",
      "Action": [
        "rds-db:connect"
      ],
      "Resource": "arn:aws:rds-db:us-east-1:123456789012:dbuser:cluster-*/app_user"
    }
  ]
}
```

ECS Task Definition:

```
{
  "family": "my-app",
  "taskRoleArn": "arn:aws:iam::123456789012:role/my-ecs-task-role",
  "executionRoleArn": "arn:aws:iam::123456789012:role/ecsTaskExecutionRole",
  "containerDefinitions": [
    {
      "name": "app",
      "image": "123456789012.dkr.ecr.us-east-1.amazonaws.com/my-app:latest",
      "memory": 512,
      "cpu": 256,
      "essential": true
    }
  ]
}
```

EKS avec IRSA (IAM Roles for Service Accounts)

Service Account Kubernetes:

```
apiVersion: v1
kind: ServiceAccount
metadata:
  name: my-app-sa
  namespace: production
  annotations:
    eks.amazonaws.com/role-arn: arn:aws:iam::123456789012:role/my-app-eks-role
    eks.amazonaws.com/sts-regional-endpoints: "true"
```

Trust Policy pour IRSA:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/oidc.eks.us-east-1.amazonaws.com
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:sub": "system
          "oidc.eks.us-east-1.amazonaws.com/id/EXAMPLED539D4633E53DE1B71EXAMPLE:aud": "sts.ama
        }
      }
    }
  ]
}
```

Deployment avec Service Account:

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-app
  namespace: production
spec:
  replicas: 3
  selector:
    matchLabels:
      app: my-app
  template:
    metadata:
      labels:
        app: my-app
    spec:
      serviceAccountName: my-app-sa  # Utilise le SA avec IAM role
      containers:
      - name: app
        image: my-app:latest
```

```
        env:
        - name: AWS_REGION
          value: "us-east-1"
        - name: AWS_ROLE_ARN
          value: "arn:aws:iam::123456789012:role/my-app-eks-role"
```

---

3. Application Multi-Tenant SaaS

Isolation par Tenant avec DynamoDB Leading Keys

Problématique: Garantir que tenant A ne peut jamais accéder aux données de tenant B.

Politique IAM avec Leading Key Condition:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DynamoDBTenantIsolation",
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:Query"
      ],
      "Resource": [
        "arn:aws:dynamodb:us-east-1:123456789012:table/SaaSTenantData"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys": [
            "${aws:PrincipalTag/TenantID}"
          ]
        }
      }
    }
  ]
}
```

Schema DynamoDB:

```
Table: SaaSTenantData
Partition Key: TenantID (String)
Sort Key: RecordID (String)

Item example:
```

```
{
  "TenantID": "tenant-abc-123",
  "RecordID": "user-456",
  "Name": "John Doe",
  "Email": "john@company.com"
}
```

Code Lambda pour récupération avec TenantID:

```
import boto3
import json
import os

dynamodb = boto3.resource('dynamodb')
table = dynamodb.Table('SaaSTenantData')

def lambda_handler(event, context):
    # Récupérer TenantID depuis les claims JWT ou authorizer context
    tenant_id = event['requestContext']['authorizer']['claims']['custom:tenant_id']

    # Query avec partition key (TenantID)
    response = table.query(
        KeyConditionExpression='TenantID = :tenant_id',
        ExpressionAttributeValues={
            ':tenant_id': tenant_id
        }
    )

    # IAM policy garantit qu'on ne peut accéder qu'aux items de notre tenant
    return {
        'statusCode': 200,
        'body': json.dumps(response['Items'])
    }
```

S3 Access Points par Tenant

Problématique: Partager un bucket S3 entre tenants avec isolation stricte.

Architecture:

```
S3 Bucket: saas-tenant-data
│
├─> Access Point: tenant-a-ap
│   └─> Policy: Accès uniquement prefix tenant-a/*
│
├─> Access Point: tenant-b-ap
│   └─> Policy: Accès uniquement prefix tenant-b/*
│
└─> Access Point: tenant-c-ap
    └─> Policy: Accès uniquement prefix tenant-c/*
```

Création d'Access Point:

```
# Créer un access point pour tenant A
aws s3control create-access-point \
    --account-id 123456789012 \
    --name tenant-a-ap \
    --bucket saas-tenant-data \
    --vpc-configuration VpcId=vpc-12345678

# Policy pour l'access point
aws s3control put-access-point-policy \
    --account-id 123456789012 \
    --name tenant-a-ap \
    --policy '{
      "Version": "2012-10-17",
      "Statement": [{
        "Effect": "Allow",
        "Principal": {
          "AWS": "arn:aws:iam::123456789012:role/tenant-a-role"
        },
        "Action": ["s3:GetObject", "s3:PutObject"],
        "Resource": "arn:aws:s3:us-east-1:123456789012:accesspoint/tenant-a-ap/object/tenant-a
      }]
    }'
```

IAM Policy pour utiliser l'Access Point:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject"
      ],
      "Resource": "arn:aws:s3:us-east-1:123456789012:accesspoint/tenant-${aws:PrincipalTag/Ten
    }
  ]
}
```

Scénarios d'Attaque IAM et Mitigation

Attaque 1: IAM Role Privilege Escalation

Scénario:
Un attaquant avec un rôle IAM limité tente d'escalader ses privilèges en créant une nouvelle politique ou en s'attachant une politique administrative.

Vecteurs d'attaque courants:

```
# Attaque 1: Créer une nouvelle politique administrative
aws iam create-policy \
    --policy-name AdminPolicy \
    --policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Action":"*","Re

# Attaque 2: S'attacher une politique gérée AWS
aws iam attach-user-policy \
    --user-name attacker \
    --policy-arn arn:aws:iam::aws:policy/AdministratorAccess

# Attaque 3: Modifier une politique existante
aws iam put-user-policy \
    --user-name attacker \
    --policy-name MyPolicy \
    --policy-document '{"Version":"2012-10-17","Statement":[{"Effect":"Allow","Action":"*","Re

# Attaque 4: Créer des access keys pour un utilisateur privilégié
aws iam create-access-key --user-name admin

# Attaque 5: Assumer un rôle privilégié
aws sts assume-role \
    --role-arn arn:aws:iam::123456789012:role/AdminRole \
    --role-session-name attacker-session
```

Mitigation:

1. Service Control Policy (SCP) pour bloquer les escalations:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyPrivilegeEscalation",
      "Effect": "Deny",
      "Action": [
        "iam:CreatePolicy",
        "iam:CreatePolicyVersion",
        "iam:AttachUserPolicy",
        "iam:AttachGroupPolicy",
```

```
        "iam:AttachRolePolicy",
        "iam:PutUserPolicy",
        "iam:PutGroupPolicy",
        "iam:PutRolePolicy",
        "iam:CreateAccessKey",
        "iam:UpdateAssumeRolePolicy"
      ],
      "Resource": "*",
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalArn": [
            "arn:aws:iam::123456789012:role/SecurityAdminRole",
            "arn:aws:iam::123456789012:role/TerraformRole"
          ]
        }
      }
    },
    {
      "Sid": "DenyAssumeAdminRoles",
      "Effect": "Deny",
      "Action": "sts:AssumeRole",
      "Resource": [
        "arn:aws:iam::123456789012:role/*Admin*",
        "arn:aws:iam::123456789012:role/*Security*"
      ],
      "Condition": {
        "StringNotEquals": {
          "aws:PrincipalArn": "arn:aws:iam::123456789012:role/SecurityAdminRole"
        }
      }
    }
  ]
}
```

2. IAM Permission Boundaries:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PermissionsBoundary",
      "Effect": "Allow",
      "Action": [
        "s3:*",
        "dynamodb:*",
        "lambda:*"
      ],
      "Resource": "*"
    },
    {
      "Sid": "DenyIAMActions",
      "Effect": "Deny",
      "Action": [
```

```
        "iam:*",
        "organizations:*",
        "account:*"
      ],
      "Resource": "*"
    }
  ]
}
```

Application de la Permission Boundary:

```
# Créer un rôle avec permission boundary
aws iam create-role \
    --role-name DeveloperRole \
    --assume-role-policy-document file://trust-policy.json \
    --permissions-boundary arn:aws:iam::123456789012:policy/DeveloperBoundary
```

3. Détection avec GuardDuty:

```
# Activer GuardDuty
aws guardduty create-detector --enable

# GuardDuty détectera automatiquement:
# - Policy:IAMUser/RootCredentialUsage
# - UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration
# - PrivilegeEscalation:IAMUser/AnomalousBehavior
```

4. Alarmes CloudWatch pour actions IAM suspectes:

```
# Filtre CloudWatch Logs pour escalation de privilèges
aws logs put-metric-filter \
    --log-group-name /aws/cloudtrail/logs \
    --filter-name IAM-Privilege-Escalation \
    --filter-pattern '{
      ($.eventName = AttachUserPolicy) ||
      ($.eventName = AttachRolePolicy) ||
      ($.eventName = PutUserPolicy) ||
      ($.eventName = PutRolePolicy) ||
      ($.eventName = CreateAccessKey) ||
      ($.eventName = CreateLoginProfile)
    }' \
    --metric-transformations \
      metricName=IAMPrivilegeEscalationAttempts,\
      metricNamespace=Security,\
      metricValue=1

# Créer une alarme
aws cloudwatch put-metric-alarm \
    --alarm-name IAM-Privilege-Escalation-Alarm \
    --alarm-description "Alert on IAM privilege escalation attempts" \
```

```
    --metric-name IAMPrivilegeEscalationAttempts \
    --namespace Security \
    --statistic Sum \
    --period 300 \
    --threshold 1 \
    --comparison-operator GreaterThanOrEqualToThreshold \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

5. Réponse automatique avec EventBridge + Lambda:

```
import boto3
import json

iam = boto3.client('iam')
sns = boto3.client('sns')

def lambda_handler(event, context):
    """Réponse automatique à une tentative d'escalation de privilèges"""

    # Parse CloudTrail event
    detail = event['detail']
    event_name = detail['eventName']
    user_arn = detail['userIdentity']['arn']
    username = detail['userIdentity']['userName']

    # Actions suspectes
    suspicious_actions = [
        'AttachUserPolicy',
        'AttachRolePolicy',
        'PutUserPolicy',
        'CreateAccessKey'
    ]

    if event_name in suspicious_actions:
        print(f"SECURITY ALERT: {username} attempted {event_name}")

        # Option 1: Désactiver les access keys de l'utilisateur
        try:
            access_keys = iam.list_access_keys(UserName=username)
            for key in access_keys['AccessKeyMetadata']:
                iam.update_access_key(
                    UserName=username,
                    AccessKeyId=key['AccessKeyId'],
                    Status='Inactive'
                )
                print(f"Disabled access key: {key['AccessKeyId']}")
        except Exception as e:
            print(f"Error disabling keys: {e}")

        # Option 2: Attacher une politique de refus
        deny_policy = {
            "Version": "2012-10-17",
```

```
            "Statement": [{
                "Effect": "Deny",
                "Action": "*",
                "Resource": "*"
            }]
        }

        try:
            iam.put_user_policy(
                UserName=username,
                PolicyName='SECURITY_LOCKDOWN',
                PolicyDocument=json.dumps(deny_policy)
            )
            print(f"Applied lockdown policy to {username}")
        except Exception as e:
            print(f"Error applying lockdown: {e}")

        # Option 3: Notifier l'équipe de sécurité
        sns.publish(
            TopicArn='arn:aws:sns:us-east-1:123456789012:SecurityIncidents',
            Subject=f'CRITICAL: IAM Privilege Escalation Attempt by {username}',
            Message=f"""
            SECURITY INCIDENT DETECTED

            User: {username}
            ARN: {user_arn}
            Action: {event_name}
            Time: {detail['eventTime']}
            Source IP: {detail.get('sourceIPAddress', 'N/A')}

            AUTOMATED RESPONSE TAKEN:
            - Access keys disabled
            - Lockdown policy applied

            MANUAL ACTION REQUIRED:
            - Investigate CloudTrail logs
            - Interview user
            - Determine if account is compromised
            - Consider rotating all credentials
            """
        )

    return {'statusCode': 200, 'body': 'Processed'}
```

EventBridge Rule:

```
{
  "source": ["aws.iam"],
  "detail-type": ["AWS API Call via CloudTrail"],
  "detail": {
    "eventName": [
      "AttachUserPolicy",
      "AttachRolePolicy",
```

```
      "PutUserPolicy",
      "PutRolePolicy",
      "CreateAccessKey",
      "CreatePolicyVersion"
    ]
  }
}
```

Attaque 2: Credential Stuffing / Brute Force

Scénario:
Attaquant tente de deviner les mots de passe IAM ou utilise des credentials volés.

Détection:

```
# Filtre CloudWatch pour échecs de connexion console
aws logs put-metric-filter \
    --log-group-name /aws/cloudtrail/logs \
    --filter-name Console-Login-Failures \
    --filter-pattern '{
      ($.eventName = ConsoleLogin) &&
      ($.errorMessage = "Failed authentication")
    }' \
    --metric-transformations \
      metricName=ConsoleLoginFailures,\
      metricNamespace=Security,\
      metricValue=1

# Alarme pour 5+ échecs en 5 minutes
aws cloudwatch put-metric-alarm \
    --alarm-name Console-Brute-Force-Attempt \
    --metric-name ConsoleLoginFailures \
    --namespace Security \
    --statistic Sum \
    --period 300 \
    --threshold 5 \
    --comparison-operator GreaterThanThreshold \
    --evaluation-periods 1 \
    --alarm-actions arn:aws:sns:us-east-1:123456789012:SecurityAlerts
```

Mitigation:

1. Password Policy forte:

```
aws iam update-account-password-policy \
    --minimum-password-length 14 \
    --require-symbols \
    --require-numbers \
```

```
    --require-uppercase-characters \
    --require-lowercase-characters \
    --allow-users-to-change-password \
    --max-password-age 90 \
    --password-reuse-prevention 24 \
    --hard-expiry
```

2. MFA obligatoire (voir section MFA ci-dessus)

3. IP Whitelisting:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NotIpAddress": {
          "aws:SourceIp": [
            "203.0.113.0/24",
            "198.51.100.0/24"
          ]
        },
        "Bool": {
          "aws:ViaAWSService": "false"
        }
      }
    }
  ]
}
```

4. AWS WAF pour API Gateway (si API publique):

```
{
  "Name": "RateLimitRule",
  "Priority": 1,
  "Statement": {
    "RateBasedStatement": {
      "Limit": 100,
      "AggregateKeyType": "IP"
    }
  },
  "Action": {
    "Block": {}
  },
  "VisibilityConfig": {
    "SampledRequestsEnabled": true,
    "CloudWatchMetricsEnabled": true,
    "MetricName": "RateLimitRule"
```

```
  }
}
```

Attaque 3: Cross-Account Assume Role Abuse

Scénario:
Attaquant compromet un compte AWS et tente d'assumer des rôles dans d'autres comptes de l'organisation.

Vecteur d'attaque:

```
# Énumérer les rôles assumables
aws iam list-roles --query 'Roles[?AssumeRolePolicyDocument.Statement[?Principal.AWS]].RoleNam

# Tenter d'assumer chaque rôle
aws sts assume-role \
    --role-arn arn:aws:iam::TARGET-ACCOUNT:role/CrossAccountRole \
    --role-session-name attacker-session
```

Mitigation:

1. External ID obligatoire:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::SOURCE-ACCOUNT:root"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "unique-random-string-12345678"
        }
      }
    }
  ]
}
```

2. Conditions restrictives multiples:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
```

```
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789012:role/TrustedRole"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "${EXTERNAL_ID}",
          "aws:SourceAccount": "123456789012"
        },
        "IpAddress": {
          "aws:SourceIp": ["203.0.113.0/24"]
        },
        "StringLike": {
          "aws:userid": "AROAEXAMPLEID:*"
        }
      }
    }
  ]
}
```

3. Monitoring avec CloudTrail:

```
# Query CloudTrail pour AssumeRole from external accounts
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=AssumeRole \
    --max-results 50 \
    --query 'Events[?contains(CloudTrailEvent, `"sourceIPAddress"`)].CloudTrailEvent' \
    --output text | jq -r 'select(.userIdentity.accountId != "123456789012")'
```

4. GuardDuty Findings:

GuardDuty détectera automatiquement:
- `UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.InsideAWS`
- `UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration.OutsideAWS`

Intégration Identité Fédérée

1. SAML 2.0 avec Active Directory

Architecture:

```
Corporate AD/LDAP
      |
      ├─> ADFS (SAML Identity Provider)
      |     |
      |     └─> SAML Assertion
```

```
      │           │
      │           ▼
      ├─> AWS IAM Identity Center (AWS SSO)
      │     │
      │     └─> Permission Sets
      │           │
      │           ▼
      └─> AWS Accounts (Production, Staging, Dev)
```

Configuration IAM Identity Provider:

```
# Créer un Identity Provider SAML
aws iam create-saml-provider \
    --name CorporateADFS \
    --saml-metadata-document file://adfs-metadata.xml
```

Trust Policy pour rôle fédéré:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:saml-provider/CorporateADFS"
      },
      "Action": "sts:AssumeRoleWithSAML",
      "Condition": {
        "StringEquals": {
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        },
        "StringLike": {
          "SAML:sub": "*@company.com"
        }
      }
    }
  ]
}
```

Mapping des groupes AD aux rôles IAM:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:saml-provider/CorporateADFS"
      },
      "Action": "sts:AssumeRoleWithSAML",
```

```
      "Condition": {
        "StringEquals": {
          "SAML:aud": "https://signin.aws.amazon.com/saml"
        },
        "ForAnyValue:StringLike": {
          "SAML:groups": [
            "AWS-Admins",
            "AWS-Developers",
            "AWS-ReadOnly"
          ]
        }
      }
    }
  ]
}
```

---

2. OIDC avec GitHub Actions

Cas d'usage: CI/CD sans credentials statiques dans GitHub.

Configuration:

```
# Créer OIDC Identity Provider pour GitHub
aws iam create-open-id-connect-provider \
    --url https://token.actions.githubusercontent.com \
    --client-id-list sts.amazonaws.com \
    --thumbprint-list 6938fd4d98bab03faadb97b34396831e3780aea1
```

Trust Policy pour GitHub Actions:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "arn:aws:iam::123456789012:oidc-provider/token.actions.githubusercontent.
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "token.actions.githubusercontent.com:aud": "sts.amazonaws.com"
        },
        "StringLike": {
          "token.actions.githubusercontent.com:sub": "repo:my-org/my-repo:*"
        }
      }
    }
```

```
  ]
}
```

GitHub Actions Workflow:

```
name: Deploy to AWS
on:
  push:
    branches: [main]

permissions:
  id-token: write
  contents: read

jobs:
  deploy:
    runs-on: ubuntu-latest
    steps:
      - uses: actions/checkout@v3

      - name: Configure AWS Credentials
        uses: aws-actions/configure-aws-credentials@v2
        with:
          role-to-assume: arn:aws:iam::123456789012:role/GitHubActionsRole
          role-session-name: github-actions-deploy
          aws-region: us-east-1

      - name: Deploy to S3
        run: |
          aws s3 sync ./build s3://my-website-bucket/

      - name: Invalidate CloudFront
        run: |
          aws cloudfront create-invalidation \
            --distribution-id EDFDVBD6EXAMPLE \
            --paths "/*"
```

Avantages:

- ✅ Pas de credentials statiques dans GitHub Secrets
- ✅ Credentials temporaires (1h par défaut)
- ✅ Audit complet via CloudTrail
- ✅ Révocation facile (désactiver le rôle)

Gestion Avancée des Secrets

1. AWS Secrets Manager avec Rotation Automatique

Créer un secret RDS avec rotation:

```
# Créer le secret
aws secretsmanager create-secret \
    --name prod/rds/master \
    --description "Master password for production RDS" \
    --secret-string '{
      "username": "admin",
      "password": "CHANGE-ME-12345",
      "engine": "postgres",
      "host": "mydb.cluster-abc123.us-east-1.rds.amazonaws.com",
      "port": 5432,
      "dbname": "production"
    }'

# Configurer la rotation automatique (30 jours)
aws secretsmanager rotate-secret \
    --secret-id prod/rds/master \
    --rotation-lambda-arn arn:aws:lambda:us-east-1:123456789012:function:SecretsManagerRDSPost
    --rotation-rules AutomaticallyAfterDays=30
```

Lambda Function pour utiliser le secret:

```
import boto3
import json
from botocore.exceptions import ClientError

def get_secret(secret_name):
    """Récupère un secret depuis Secrets Manager"""
    client = boto3.client('secretsmanager', region_name='us-east-1')

    try:
        response = client.get_secret_value(SecretId=secret_name)
        secret = json.loads(response['SecretString'])
        return secret
    except ClientError as e:
        raise Exception(f"Error retrieving secret: {e}")

def lambda_handler(event, context):
    # Récupérer les credentials RDS
    secret = get_secret('prod/rds/master')

    # Utiliser les credentials
    import psycopg2
    conn = psycopg2.connect(
        host=secret['host'],
```

```
        port=secret['port'],
        user=secret['username'],
        password=secret['password'],
        database=secret['dbname']
    )

    # Exécuter des requêtes...
    cursor = conn.cursor()
    cursor.execute("SELECT version();")
    version = cursor.fetchone()

    return {
        'statusCode': 200,
        'body': json.dumps({'version': version[0]})
    }
```

IAM Policy pour accès au secret:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "secretsmanager:GetSecretValue"
      ],
      "Resource": "arn:aws:secretsmanager:us-east-1:123456789012:secret:prod/rds/*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "secretsmanager.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

---

2. Parameter Store pour Configuration

Hiérarchie recommandée:

```
/prod/
  ├─> app/
  │   ├─> database/url (SecureString)
```

```
  │   ├─> database/user (SecureString)
  │   ├─> api/key (SecureString)
  │   └─> feature/flags (String)
  ├─> infrastructure/
  │   ├─> vpc/id (String)
  │   └─> subnet/ids (StringList)
  └─> third-party/
      ├─> stripe/key (SecureString)
      └─> sendgrid/key (SecureString)
```

Créer des paramètres:

```
# SecureString (encrypted with KMS)
aws ssm put-parameter \
    --name /prod/app/database/password \
    --value "MySecurePassword123!" \
    --type SecureString \
    --key-id alias/aws/ssm \
    --description "Production database password" \
    --tags Key=Environment,Value=Production Key=Compliance,Value=PCI

# Get parameter
aws ssm get-parameter \
    --name /prod/app/database/password \
    --with-decryption \
    --query 'Parameter.Value' \
    --output text
```

Récupération par path (Lambda):

```
import boto3

ssm = boto3.client('ssm')

def get_parameters_by_path(path):
    """Récupère tous les paramètres d'un path"""
    response = ssm.get_parameters_by_path(
        Path=path,
        Recursive=True,
        WithDecryption=True
    )

    params = {}
    for param in response['Parameters']:
        key = param['Name'].split('/')[-1]  # Extract last part of path
        params[key] = param['Value']

    return params

# Usage
```

```
db_config = get_parameters_by_path('/prod/app/database')
# Returns: {'url': '...', 'user': '...', 'password': '...'}
```

IAM Policy pour accès par path:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "ssm:GetParameter",
        "ssm:GetParameters",
        "ssm:GetParametersByPath"
      ],
      "Resource": [
        "arn:aws:ssm:us-east-1:123456789012:parameter/prod/app/*"
      ]
    },
    {
      "Effect": "Allow",
      "Action": [
        "kms:Decrypt"
      ],
      "Resource": "arn:aws:kms:us-east-1:123456789012:key/*",
      "Condition": {
        "StringEquals": {
          "kms:ViaService": "ssm.us-east-1.amazonaws.com"
        }
      }
    }
  ]
}
```

Troubleshooting et Debugging IAM

1. Policy Simulator

Tester une politique avant déploiement:

```
# Tester si un utilisateur peut effectuer une action
aws iam simulate-principal-policy \
    --policy-source-arn arn:aws:iam::123456789012:user/alice \
    --action-names s3:GetObject s3:PutObject \
    --resource-arns arn:aws:s3:::my-bucket/*

# Output:
{
```

```
  "EvaluationResults": [
    {
      "EvalActionName": "s3:GetObject",
      "EvalResourceName": "arn:aws:s3:::my-bucket/*",
      "EvalDecision": "allowed"
    },
    {
      "EvalActionName": "s3:PutObject",
      "EvalResourceName": "arn:aws:s3:::my-bucket/*",
      "EvalDecision": "explicitDeny",
      "MatchedStatements": [...]
    }
  ]
}
```

Tester une politique custom:

```
aws iam simulate-custom-policy \
    --policy-input-list file://policy.json \
    --action-names dynamodb:GetItem dynamodb:PutItem \
    --resource-arns arn:aws:dynamodb:us-east-1:123456789012:table/MyTable
```

---

2. Access Denied Troubleshooting

Erreur commune:

```
User: arn:aws:iam::123456789012:user/developer is not authorized to perform: s3:PutObject on r
```

Checklist de debugging:

```
# 1. Vérifier les politiques inline de l'utilisateur
aws iam list-user-policies --user-name developer

# 2. Vérifier les politiques attachées
aws iam list-attached-user-policies --user-name developer

# 3. Vérifier les groupes de l'utilisateur
aws iam list-groups-for-user --user-name developer

# 4. Vérifier les politiques des groupes
aws iam list-group-policies --group-name developers
aws iam list-attached-group-policies --group-name developers

# 5. Vérifier les SCPs (si Organizations)
aws organizations list-policies-for-target \
    --target-id 123456789012 \
    --filter SERVICE_CONTROL_POLICY
```

```
# 6. Vérifier les permission boundaries
aws iam get-user --user-name developer \
    --query 'User.PermissionsBoundary'

# 7. Vérifier la bucket policy S3
aws s3api get-bucket-policy --bucket my-bucket

# 8. Vérifier le Block Public Access S3
aws s3api get-public-access-block --bucket my-bucket
```

Script Python de debugging:

```
import boto3
import json

def debug_iam_permissions(user_name, action, resource):
    """Debug IAM permissions for a user"""
    iam = boto3.client('iam')

    print(f"Debugging permissions for user: {user_name}")
    print(f"Action: {action}")
    print(f"Resource: {resource}\n")

    # 1. Get user policies
    print("=== User Inline Policies ===")
    user_policies = iam.list_user_policies(UserName=user_name)
    for policy_name in user_policies['PolicyNames']:
        policy = iam.get_user_policy(UserName=user_name, PolicyName=policy_name)
        print(f"Policy: {policy_name}")
        print(json.dumps(policy['PolicyDocument'], indent=2))

    # 2. Get attached policies
    print("\n=== User Attached Policies ===")
    attached = iam.list_attached_user_policies(UserName=user_name)
    for policy in attached['AttachedPolicies']:
        print(f"Policy: {policy['PolicyName']}")
        policy_version = iam.get_policy(PolicyArn=policy['PolicyArn'])
        default_version = policy_version['Policy']['DefaultVersionId']
        policy_doc = iam.get_policy_version(
            PolicyArn=policy['PolicyArn'],
            VersionId=default_version
        )
        print(json.dumps(policy_doc['PolicyVersion']['Document'], indent=2))

    # 3. Get groups
    print("\n=== User Groups ===")
    groups = iam.list_groups_for_user(UserName=user_name)
    for group in groups['Groups']:
        print(f"Group: {group['GroupName']}")

    # 4. Simulate policy
    print("\n=== Policy Simulation ===")
    simulation = iam.simulate_principal_policy(
```

```
        PolicySourceArn=f"arn:aws:iam::123456789012:user/{user_name}",
        ActionNames=[action],
        ResourceArns=[resource]
    )

    for result in simulation['EvaluationResults']:
        print(f"Action: {result['EvalActionName']}")
        print(f"Decision: {result['EvalDecision']}")
        if 'MatchedStatements' in result:
            print("Matched Statements:")
            for stmt in result['MatchedStatements']:
                print(f"  - Source: {stmt['SourcePolicyId']}")

# Usage
debug_iam_permissions('developer', 's3:PutObject', 'arn:aws:s3:::my-bucket/*')
```

---

3. CloudTrail pour Audit

Requête CloudWatch Logs Insights:

```
# Tous les appels API IAM dans les dernières 24h
fields @timestamp, userIdentity.userName, eventName, errorCode, errorMessage
| filter eventSource = "iam.amazonaws.com"
| sort @timestamp desc
| limit 100

# Appels API échoués (Access Denied)
fields @timestamp, userIdentity.arn, eventName, errorCode, errorMessage, requestParameters
| filter errorCode = "AccessDenied"
| sort @timestamp desc
| limit 50

# Actions IAM sensibles
fields @timestamp, userIdentity.userName, eventName, requestParameters
| filter eventName in ["AttachUserPolicy", "AttachRolePolicy", "PutUserPolicy", "CreateAccessK
| sort @timestamp desc

# AssumeRole depuis comptes externes
fields @timestamp, userIdentity.accountId, userIdentity.arn, eventName, requestParameters.role
| filter eventName = "AssumeRole" and userIdentity.accountId != "123456789012"
| sort @timestamp desc
```

Compliance et Audit IAM

Mapping ISO 27001:2022

Control	Description	Implémentation IAM
A.5.15	Access control	IAM policies, least privilege
A.5.16	Identity management	IAM users, roles, Identity Center
A.5.17	Authentication information	MFA, password policy
A.5.18	Access rights	IAM Access Analyzer, regular audits
A.8.2	Privileged access rights	MFA for admin, Session Manager
A.8.5	Secure authentication	MFA enforcement policy

Mapping SOC 2 Trust Service Criteria

Critère	Contrôle IAM	Evidence
CC6.1	Logical and physical access controls	IAM policies, MFA
CC6.2	Access termination	Automated deprovisioning
CC6.3	Access review	Quarterly IAM audits
CC7.2	System monitoring	CloudWatch alarms for IAM events

IAM Permission Boundaries

Concept et Cas d'Usage

Les **Permission Boundaries** limitent les permissions maximales qu'une entité IAM peut avoir, même si des politiques plus permissives sont attachées.

Use Case Principal: Déléguer la création de rôles IAM sans risque d'escalade de privilèges.

```
Sans Permission Boundary:
Developer crée un rôle → Attache AdministratorAccess → Escalade de privilèges ⚠️

Avec Permission Boundary:
Developer crée un rôle → Tente d'attacher AdministratorAccess → Boundary limite à DeveloperAcc
```

Implémentation Terraform

```
# permission_boundaries.tf

# 1. Créer la politique de boundary
resource "aws_iam_policy" "developer_boundary" {
  name        = "DeveloperPermissionBoundary"
  description = "Maximum permissions for developer-created roles"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "AllowedServices"
        Effect = "Allow"
        Action = [
          "ec2:*",
          "s3:*",
          "dynamodb:*",
          "lambda:*",
          "logs:*",
          "cloudwatch:*",
          "apigateway:*"
        ]
        Resource = "*"
      },
      {
        Sid    = "DenyAdminActions"
        Effect = "Deny"
        Action = [
          "iam:*",
          "organizations:*",
          "account:*",
          "kms:ScheduleKeyDeletion",
          "kms:Delete*"
        ]
        Resource = "*"
      },
      {
        Sid    = "AllowIAMReadOnly"
        Effect = "Allow"
        Action = [
          "iam:Get*",
          "iam:List*"
        ]
        Resource = "*"
```

```
      }
    ]
  })
}

# 2. Politique permettant aux développeurs de créer des rôles
resource "aws_iam_policy" "allow_role_creation_with_boundary" {
  name = "AllowRoleCreationWithBoundary"

  policy = jsonencode({
    Version = "2012-10-17"
    Statement = [
      {
        Sid    = "AllowRoleCreation"
        Effect = "Allow"
        Action = [
          "iam:CreateRole",
          "iam:AttachRolePolicy",
          "iam:DetachRolePolicy",
          "iam:PutRolePolicy",
          "iam:DeleteRolePolicy"
        ]
        Resource = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:role/developer
        Condition = {
          StringEquals = {
            "iam:PermissionsBoundary" = aws_iam_policy.developer_boundary.arn
          }
        }
      },
      {
        Sid    = "RequirePermissionsBoundary"
        Effect = "Deny"
        Action = [
          "iam:CreateRole",
          "iam:PutRolePermissionsBoundary"
        ]
        Resource = "*"
        Condition = {
          StringNotEquals = {
            "iam:PermissionsBoundary" = aws_iam_policy.developer_boundary.arn
          }
        }
      },
      {
        Sid    = "PreventBoundaryRemoval"
        Effect = "Deny"
        Action = "iam:DeleteRolePermissionsBoundary"
        Resource = "*"
      }
    ]
  })
}

# 3. Rôle développeur avec boundary
resource "aws_iam_role" "developer" {
```

```
  name                 = "DeveloperRole"
  permissions_boundary = aws_iam_policy.developer_boundary.arn

  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        AWS = "arn:aws:iam::${data.aws_caller_identity.current.account_id}:root"
      }
      Action = "sts:AssumeRole"
      Condition = {
        StringEquals = {
          "sts:ExternalId" = var.developer_external_id
        }
      }
    }]
  })
}

resource "aws_iam_role_policy_attachment" "developer_create_roles" {
  role       = aws_iam_role.developer.name
  policy_arn = aws_iam_policy.allow_role_creation_with_boundary.arn
}
```

Validation et Testing

```
# Test script pour valider la permission boundary
import boto3

iam = boto3.client('iam')

def test_permission_boundary():
    """Tester que la boundary empêche l'escalade de privilèges"""

    # Test 1: Créer un rôle sans boundary (devrait échouer)
    try:
        iam.create_role(
            RoleName='test-role-no-boundary',
            AssumeRolePolicyDocument='...'
        )
        print("❌ FAIL: Role created without boundary")
    except Exception as e:
        print("✅ PASS: Boundary enforcement working")

    # Test 2: Créer un rôle avec boundary (devrait réussir)
    try:
        iam.create_role(
            RoleName='test-role-with-boundary',
            AssumeRolePolicyDocument='...',
            PermissionsBoundary='arn:aws:iam::123456789012:policy/DeveloperPermissionBoundary'
        )
        print("✅ PASS: Role created with boundary")
```

```
    except Exception as e:
        print(f"❌ FAIL: {e}")

    # Test 3: Tenter d'attacher AdministratorAccess (devrait être limité par boundary)
    try:
        iam.attach_role_policy(
            RoleName='test-role-with-boundary',
            PolicyArn='arn:aws:iam::aws:policy/AdministratorAccess'
        )
        # Même si attaché, les permissions effectives sont limitées par la boundary
        print("⚠️ Policy attached but limited by boundary")
    except Exception as e:
        print(f"Info: {e}")
```

Break Glass Procedures (Accès d'Urgence)

Architecture Break Glass

Les procédures "Break Glass" permettent un accès d'urgence en cas d'incident majeur tout en maintenant l'audit et la sécurité.

```
Situation Normale:
┌──────────────────┐
│ Users → Roles    │
│ with MFA         │
└──────────────────┘

Situation d'Urgence (Break Glass):
┌──────────────────────────────────┐
│ 1. Récupérer credentials depuis  │
│    sealed envelope physique      │
│ 2. Utiliser token MFA hardware   │
│ 3. AssumeRole vers BreakGlass    │
│ 4. Toutes actions loggées        │
│ 5. Alertes immédiates envoyées   │
└──────────────────────────────────┘
```

Implémentation Complète

```
# break_glass.tf

# 1. Break Glass Role
resource "aws_iam_role" "break_glass" {
  name        = "BreakGlassEmergencyAccess"
  description = "Emergency access role - Use only in critical incidents"

  max_session_duration = 3600  # 1 heure maximum
```

```
  assume_role_policy = jsonencode({
    Version = "2012-10-17"
    Statement = [{
      Effect = "Allow"
      Principal = {
        AWS = [
          "arn:aws:iam::${data.aws_caller_identity.current.account_id}:user/break-glass-user-1
          "arn:aws:iam::${data.aws_caller_identity.current.account_id}:user/break-glass-user-2
        ]
      }
      Action = "sts:AssumeRole"
      Condition = {
        Bool = {
          "aws:MultiFactorAuthPresent" = "true"
        }
        IpAddress = {
          "aws:SourceIp" = var.emergency_ip_ranges  # IPs autorisées
        }
      }
    }]
  })

  tags = {
    Name     = "Break Glass Emergency Access"
    Critical = "true"
  }
}

resource "aws_iam_role_policy_attachment" "break_glass_admin" {
  role       = aws_iam_role.break_glass.name
  policy_arn = "arn:aws:iam::aws:policy/AdministratorAccess"
}

# 2. CloudWatch Alarm pour utilisation Break Glass
resource "aws_cloudwatch_log_metric_filter" "break_glass_usage" {
  name           = "break-glass-role-usage"
  log_group_name = "/aws/cloudtrail/organization"

  pattern = "{($.userIdentity.sessionContext.sessionIssuer.userName = \"BreakGlassEmergencyAcc

  metric_transformation {
    name      = "BreakGlassRoleUsage"
    namespace = "Security/BreakGlass"
    value     = "1"
  }
}

resource "aws_cloudwatch_metric_alarm" "break_glass_alert" {
  alarm_name          = "CRITICAL-BreakGlass-Role-Used"
  alarm_description   = "Break Glass emergency role has been used!"
  comparison_operator = "GreaterThanOrEqualToThreshold"
  evaluation_periods  = 1
  metric_name         = "BreakGlassRoleUsage"
  namespace           = "Security/BreakGlass"
```

```
  period              = 60
  statistic           = "Sum"
  threshold           = 1
  treat_missing_data  = "notBreaching"

  alarm_actions = [
    aws_sns_topic.critical_security_alerts.arn,
    aws_sns_topic.exec_team_sms.arn  # SMS au CISO et CEO
  ]
}

# 3. Lambda pour notification enrichie
resource "aws_lambda_function" "break_glass_notifier" {
  filename         = "break_glass_notifier.zip"
  function_name    = "break-glass-emergency-notifier"
  role             = aws_iam_role.break_glass_notifier.arn
  handler          = "index.handler"
  runtime          = "python3.11"
  source_code_hash = filebase64sha256("break_glass_notifier.zip")

  environment {
    variables = {
      SLACK_WEBHOOK_URL = var.slack_security_webhook
      PAGERDUTY_KEY     = var.pagerduty_api_key
    }
  }
}
```

Lambda de Notification Break Glass

```
# break_glass_notifier/index.py
import boto3
import json
import requests
import os
from datetime import datetime

def handler(event, context):
    """Notifier l'équipe en cas d'utilisation du Break Glass"""

    # Parser l'événement CloudTrail
    message = json.loads(event['Records'][0]['Sns']['Message'])

    cloudtrail = boto3.client('cloudtrail')

    # Récupérer les détails de l'événement
    events = cloudtrail.lookup_events(
        LookupAttributes=[{
            'AttributeKey': 'Username',
            'AttributeValue': 'BreakGlassEmergencyAccess'
        }],
        MaxResults=10
    )
```

```
    for evt in events['Events']:
        event_detail = json.loads(evt['CloudTrailEvent'])

        notification = {
            'timestamp': datetime.now().isoformat(),
            'severity': 'CRITICAL',
            'event': 'BREAK_GLASS_ACTIVATED',
            'user': event_detail.get('userIdentity', {}).get('principalId'),
            'source_ip': event_detail.get('sourceIPAddress'),
            'user_agent': event_detail.get('userAgent'),
            'event_name': event_detail.get('eventName'),
            'aws_region': event_detail.get('awsRegion'),
            'actions_taken': []
        }

        # Notifier via Slack
        send_slack_alert(notification)

        # Créer un incident PagerDuty
        create_pagerduty_incident(notification)

        # Logger dans S3 pour forensics
        log_to_forensics_bucket(notification)

        # Envoyer SMS aux executives
        send_executive_sms(notification)

    return {'statusCode': 200}

def send_slack_alert(notification):
    """Envoyer une alerte Slack"""
    webhook_url = os.environ['SLACK_WEBHOOK_URL']

    message = {
        'text': '🚨 CRITICAL: Break Glass Access Used',
        'attachments': [{
            'color': 'danger',
            'fields': [
                {'title': 'User', 'value': notification['user'], 'short': True},
                {'title': 'Source IP', 'value': notification['source_ip'], 'short': True},
                {'title': 'Region', 'value': notification['aws_region'], 'short': True},
                {'title': 'Time', 'value': notification['timestamp'], 'short': True}
            ]
        }]
    }

    requests.post(webhook_url, json=message)

def create_pagerduty_incident(notification):
    """Créer un incident PagerDuty de haute priorité"""
    api_key = os.environ['PAGERDUTY_KEY']

    incident = {
        'incident': {
```

```
            'type': 'incident',
            'title': '🚨 Break Glass Emergency Access Used',
            'service': {'id': 'SECURITY_SERVICE_ID', 'type': 'service_reference'},
            'urgency': 'high',
            'body': {
                'type': 'incident_body',
                'details': json.dumps(notification, indent=2)
            }
        }
    }

    headers = {
        'Authorization': f'Token token={api_key}',
        'Content-Type': 'application/json',
        'Accept': 'application/vnd.pagerduty+json;version=2'
    }

    requests.post(
        'https://api.pagerduty.com/incidents',
        headers=headers,
        json=incident
    )
```

Procédure Opérationnelle

```
# Break Glass Emergency Access Procedure

## ⚠️ À N'UTILISER QU'EN CAS D'URGENCE CRITIQUE

### Situations Autorisées:
- Compte root compromis
- Tous les administrateurs IAM perdus/compromis
- Incident de sécurité majeur nécessitant accès immédiat
- Panne critique empêchant accès normal

### Procédure d'Activation:

1. **Obtenir Approbation**
   - [ ] Contacter CISO (téléphone)
   - [ ] Documenter la raison (ticket incident)
   - [ ] Obtenir approbation verbale + écrite

2. **Récupérer Credentials**
   - [ ] Ouvrir l'enveloppe scellée (coffre-fort physique)
   - [ ] Vérifier numéro de série de l'enveloppe
   - [ ] Credentials: break-glass-user-1 ou break-glass-user-2

3. **Activer MFA Hardware**
   - [ ] Récupérer token MFA physique (coffre-fort)
   - [ ] Vérifier que le token est fonctionnel

4. **Connexion**
   ```bash
```
```

```
   aws configure --profile break-glass
   # Enter Access Key ID: [FROM ENVELOPE]
   # Enter Secret Access Key: [FROM ENVELOPE]

   # AssumeRole avec MFA
   aws sts assume-role \
       --role-arn arn:aws:iam::123456789012:role/BreakGlassEmergencyAccess \
       --role-session-name "emergency-$(date +%Y%m%d-%H%M%S)" \
       --serial-number arn:aws:iam::123456789012:mfa/break-glass-mfa \
       --token-code [6-DIGIT-CODE] \
       --profile break-glass
   ```

5. **Actions d'Urgence**
   - [ ] Documenter CHAQUE action effectuée
   - [ ] Résoudre l'incident
   - [ ] Temps limite: 1 heure

6. **Post-Incident (OBLIGATOIRE)**
   - [ ] Rotation des credentials break-glass
   - [ ] Nouvelle enveloppe scellée
   - [ ] Rapport d'incident complet
   - [ ] Revue avec CISO

Alertes Déclenchées:
✅ SMS au CISO et CEO
✅ Incident PagerDuty haute priorité
✅ Message Slack #security-critical
✅ Email à l'équipe executive
✅ Logs forensics dans S3

Post-Mortem Requis:
- Analyse de la nécessité du break glass
- Actions effectuées pendant l'accès
- Recommandations pour éviter future utilisation
```

---

# Just-in-Time (JIT) Access

## Architecture JIT

Le JIT Access permet d'élever temporairement les privilèges uniquement quand nécessaire, réduisant la surface d'attaque.

```
Flux JIT Access:
┌──┐
│ 1. Developer demande accès production (Slack)   │
└──────────────────────┬───────────────────────────┘
                       │
                       ▼
┌──┐
```
```

```
│ 2. Manager approuve (dans les 15 minutes)       │
└──────────────────────┬──────────────────────────┘
                       │
┌──────────────────────▼──────────────────────────┐
│ 3. Lambda crée temporary credentials (1 heure)  │
└──────────────────────┬──────────────────────────┘
                       │
┌──────────────────────▼──────────────────────────┐
│ 4. Developer utilise credentials temporaires    │
└──────────────────────┬──────────────────────────┘
                       │
┌──────────────────────▼──────────────────────────┐
│ 5. Auto-révocation après expiration             │
│    + Audit complet dans CloudTrail              │
└─────────────────────────────────────────────────┘
```

Implémentation avec Step Functions

```
# jit_access.tf

# DynamoDB pour tracking des accès JIT
resource "aws_dynamodb_table" "jit_access_requests" {
  name           = "JITAccessRequests"
  billing_mode   = "PAY_PER_REQUEST"
  hash_key       = "RequestId"
  range_key      = "Timestamp"

  attribute {
    name = "RequestId"
    type = "S"
  }

  attribute {
    name = "Timestamp"
    type = "N"
  }

  attribute {
    name = "UserId"
    type = "S"
  }

  attribute {
    name = "Status"
    type = "S"
  }

  global_secondary_index {
    name            = "UserIdIndex"
    hash_key        = "UserId"
    range_key       = "Timestamp"
    projection_type = "ALL"
  }
```

```
  global_secondary_index {
    name            = "StatusIndex"
    hash_key        = "Status"
    range_key       = "Timestamp"
    projection_type = "ALL"
  }

  ttl {
    attribute_name = "ExpirationTime"
    enabled        = true
  }

  tags = {
    Name = "JIT Access Tracking"
  }
}

# Step Function pour orchestration JIT
resource "aws_sfn_state_machine" "jit_access_workflow" {
  name     = "JITAccessWorkflow"
  role_arn = aws_iam_role.jit_step_function.arn

  definition = jsonencode({
    Comment = "JIT Access Request Workflow"
    StartAt = "ValidateRequest"
    States = {
      ValidateRequest = {
        Type     = "Task"
        Resource = aws_lambda_function.jit_validator.arn
        Next     = "RequestApproval"
        Catch = [{
          ErrorEquals = ["ValidationError"]
          Next        = "DenyRequest"
        }]
      }

      RequestApproval = {
        Type = "Task"
        Resource = "arn:aws:states:::lambda:invoke.waitForTaskToken"
        Parameters = {
          FunctionName = aws_lambda_function.jit_approval_requester.arn
          Payload = {
            "TaskToken.$" = "$$.Task.Token"
            "Request.$"   = "$"
          }
        }
        TimeoutSeconds = 900  # 15 minutes pour approuver
        Next           = "GrantAccess"
        Catch = [{
          ErrorEquals = ["States.Timeout", "ApprovalDenied"]
          Next        = "DenyRequest"
        }]
      }
```

```
      GrantAccess = {
        Type     = "Task"
        Resource = aws_lambda_function.jit_grant_access.arn
        Next     = "WaitForExpiration"
      }

      WaitForExpiration = {
        Type    = "Wait"
        Seconds = 3600  # 1 heure
        Next    = "RevokeAccess"
      }

      RevokeAccess = {
        Type     = "Task"
        Resource = aws_lambda_function.jit_revoke_access.arn
        End      = true
      }

      DenyRequest = {
        Type     = "Task"
        Resource = aws_lambda_function.jit_deny_notifier.arn
        End      = true
      }
    }
  })
}
```

Lambda JIT Grant Access

```
# jit_grant_access/index.py
import boto3
import json
from datetime import datetime, timedelta
import secrets

sts = boto3.client('sts')
iam = boto3.client('iam')
dynamodb = boto3.resource('dynamodb')

def handler(event, context):
    """Accorder l'accès JIT temporaire"""

    request_id = event['request_id']
    user_id = event['user_id']
    role_arn = event['requested_role_arn']
    justification = event['justification']
    approved_by = event['approved_by']

    # Générer des credentials temporaires
    session_name = f"jit-{user_id}-{datetime.now().strftime('%Y%m%d%H%M%S')}"

    temp_creds = sts.assume_role(
        RoleArn=role_arn,
```

```
        RoleSessionName=session_name,
        DurationSeconds=3600,  # 1 heure
        Tags=[
            {'Key': 'JITAccess', 'Value': 'true'},
            {'Key': 'RequestId', 'Value': request_id'},
            {'Key': 'UserId', 'Value': user_id}
        ]
    )

    # Stocker dans DynamoDB
    table = dynamodb.Table('JITAccessRequests')
    table.update_item(
        Key={
            'RequestId': request_id,
            'Timestamp': event['timestamp']
        },
        UpdateExpression='SET #status = :status, AccessKeyId = :key, SessionToken = :token, Ex
        ExpressionAttributeNames={
            '#status': 'Status'
        },
        ExpressionAttributeValues={
            ':status': 'GRANTED',
            ':key': temp_creds['Credentials']['AccessKeyId'],
            ':token': temp_creds['Credentials']['SessionToken'],
            ':expires': (datetime.now() + timedelta(hours=1)).isoformat(),
            ':approver': approved_by
        }
    )

    # Notifier l'utilisateur
    send_credentials_to_user(user_id, temp_creds, session_name)

    # Logger pour audit
    log_jit_grant(request_id, user_id, role_arn, approved_by, justification)

    return {
        'status': 'granted',
        'request_id': request_id,
        'expires_at': temp_creds['Credentials']['Expiration'].isoformat()
    }

def send_credentials_to_user(user_id, credentials, session_name):
    """Envoyer les credentials de manière sécurisée"""
    # Option 1: Via Secrets Manager avec accès temporaire
    secrets = boto3.client('secretsmanager')

    secret_name = f"jit-access/{user_id}/{session_name}"

    secrets.create_secret(
        Name=secret_name,
        SecretString=json.dumps({
            'AccessKeyId': credentials['Credentials']['AccessKeyId'],
            'SecretAccessKey': credentials['Credentials']['SecretAccessKey'],
            'SessionToken': credentials['Credentials']['SessionToken'],
            'Expiration': credentials['Credentials']['Expiration'].isoformat()
```

```
        }),
        Tags=[
            {'Key': 'Type', 'Value': 'JITAccess'},
            {'Key': 'ExpiresAt', 'Value': credentials['Credentials']['Expiration'].isoformat()}
        ]
    )

    # Notifier l'utilisateur du nom du secret
    send_slack_notification(user_id, f"Your JIT access is ready. Retrieve credentials from Secrets Manager: {secret_name}")

def log_jit_grant(request_id, user_id, role_arn, approved_by, justification):
    """Logger l'octroi d'accès JIT pour audit"""
    cloudwatch = boto3.client('logs')

    log_entry = {
        'timestamp': datetime.now().isoformat(),
        'event': 'JIT_ACCESS_GRANTED',
        'request_id': request_id,
        'user_id': user_id,
        'role_arn': role_arn,
        'approved_by': approved_by,
        'justification': justification,
        'duration': '1 hour'
    }

    cloudwatch.put_log_events(
        logGroupName='/aws/security/jit-access',
        logStreamName=datetime.now().strftime('%Y/%m/%d'),
        logEvents=[{
            'timestamp': int(datetime.now().timestamp() * 1000),
            'message': json.dumps(log_entry)
        }]
    )
```

Conclusion

La sécurisation IAM est un **processus continu** qui nécessite :
- Une application stricte du principe du moindre privilège
- Une authentification forte avec MFA
- Des audits réguliers et automatisés
- Une surveillance proactive des menaces
- **Permission Boundaries** pour délégation sécurisée
- **Break Glass Procedures** pour urgences
- **JIT Access** pour minimiser la surface d'attaque

En 2025, avec l'augmentation des attaques ciblant les identités cloud, IAM représente la première ligne de défense de votre infrastructure AWS.

L'implémentation de ce guide permettra de réduire significativement la surface d'attaque et de garantir la conformité avec les standards de sécurité actuels.

Document préparé pour: [Nom du Client]
Contact support: [Email de l'équipe sécurité]
Dernière mise à jour: Novembre 2025