



Long Term Access Keys

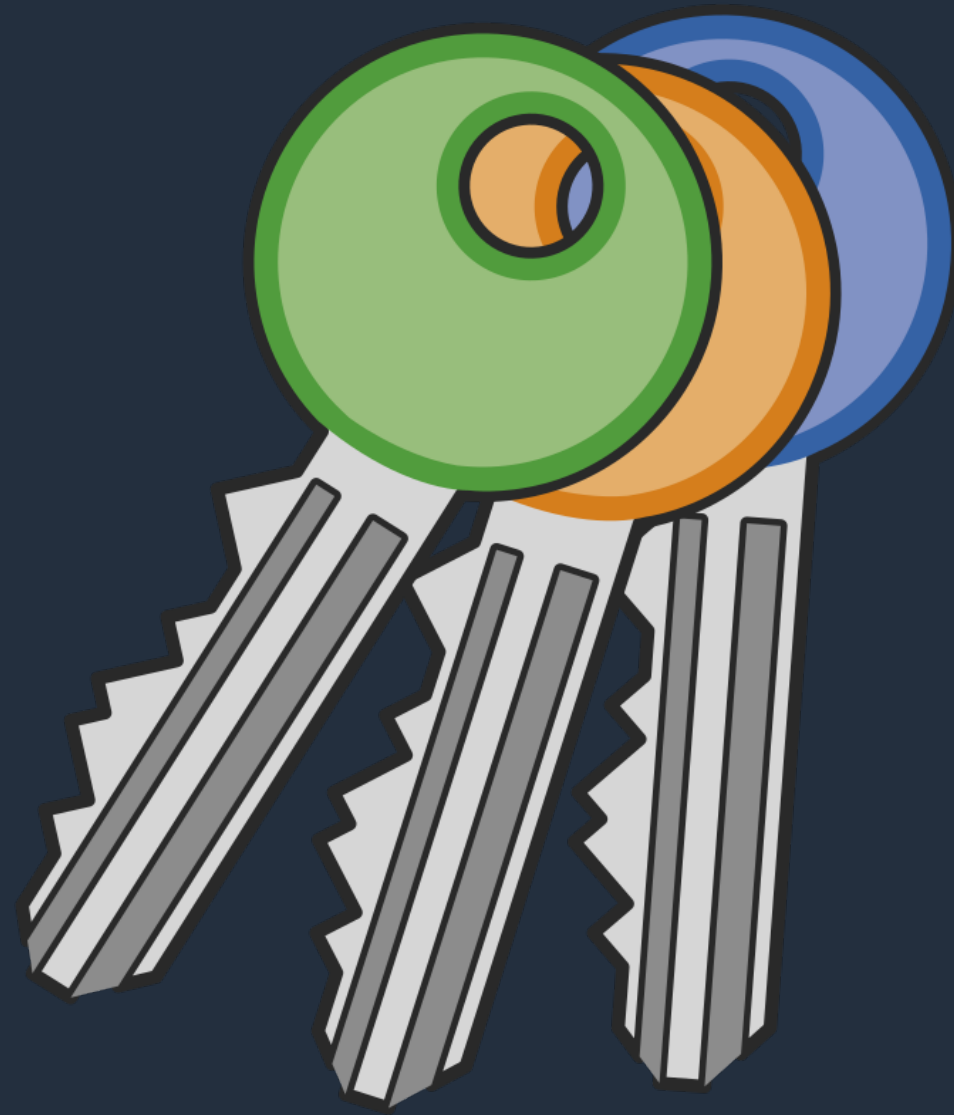
HealthCheck

Presenter



Agenda

- Why are we here?
- What are access keys?
- Should I use access keys?
- Best Practices



Why are we here?

If any of these conditions apply

- You have lost control of access keys.
- You have received an email from AWS about exposed access keys.
- You are worried about losing access keys?
- You don't want to stop using access keys!
- You need to use an access key, what should you do?
- You have an HRI flagged on SEC2 or SEC3 in your WAR review.

What are access keys?

- You use an access key (an access key ID and secret access key) to make programmatic requests to AWS.
- Access keys are long term credentials that do not change, unless you change them.
- Some things you shouldn't do with them
 - Do not embed access keys in code.
 - Do not share access keys between users in your AWS accounts.

What about root access keys?

- Do not create an access key for the root user.
- As a best practice, do not use your AWS account root user*.
- If you do have an access key for your AWS account root user, deactivate then delete it.
- Root accounts should always have Multi-Factor Authentication(MFA) enabled.
- Root access gives full access to your AWS account including billing information.
- SCPs can reduce root permissions in member accounts but not the payer account for AWS Organizations.

Should I use access keys?

It is the recommendation of AWS that you do not use or move away from using long term access keys!

You may think you need to but there are viable alternatives.



Should I use access keys?



Best Practices

Prevent access keys from being created

Prevent keys from being created

1. Use AWS Organizations to restrict key creation with Service Control Policy (SCP)

- Create an OU with an SCP that explicitly denies key creation

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "AllowsAllActions",
      "Effect": "Allow",
      "Action": "*",
      "Resource": "*"
    },
    {
      "Sid": "DenyCreateAccessKey",
      "Effect": "Deny",
      "Action": "iam:CreateAccessKey",
      "Resource": "*"
    }
  ]
}
```

- Create an OU with an SCP that allows key creation, use this as an exception OU

Prevent keys from being created

2. Consider using IAM policy to deny creation of IAM access keys

- Add deny to user or role permissions
 1. Create a policy that explicitly denies the creation of access keys

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCreateAccessKey",
      "Effect": "Deny",
      "Action": "iam:CreateAccessKey",
      "Resource": "*"
    }
  ]
}
```

Security break glass account

Only use for emergencies

1. Should exist within the Security OU in AWS Organizations
2. Allows for appropriate access to AWS accounts
 - Cross account
 - Read only
 - Can contain AWS IAM Roles that can assume roles in other accounts
3. Automatically rotate these keys every 90 days*

Vendor required access keys

1. Some vendors require access keys for use with their software
2. Before providing long term access keys, ask the vendor to support AWS IAM Roles instead
3. If no alternative is available, frequently rotate these keys
4. Apply a policy that grants the minimum permissions required for the service to function
5. Check with the vendor frequently about added support for AWS IAM Roles.

Audit your long term access keys

Audit your keys

Identify long term access keys

1. Generate a credential report

- `aws iam generate-credential-report`
- `aws iam get-credential-report`

2. Analyze your credential audit report

- Look for keys 90+ days and rotate
- Look for unused keys and remove
- Naming convention can identify outlier users
- Establish a baseline for users/keys and compare – more/less users than baseline?
- Check the last used date, is this expected or unexpected?

Audit your keys

Automate monitoring

```
{ "source": [ "aws.iam" ],  
  "detail-type": [ "AWS API Call via CloudTrail" ],  
  "detail": {  
    "eventSource": [ "iam.amazonaws.com" ],  
    "eventName": [ "CreateAccessKey" ]  
  }  
}
```

1. Monitor for creation of long term keys

- CloudWatch event rule based on IAM API event
- AWS Config check for Root access key availability
 - Enable AWS managed config rule - iam-root-access-key-check
 - Checks whether the root user access key is available.
 - The rule is compliant if the user access key does not exist.

2. Automate detection and remediation of exposed access keys


- Deploy automated workflows using AWS Health API
- https://github.com/aws/aws-health-tools/tree/master/automated-actions/AWS_RISK_CREDENTIALS_EXPOSED

Audit your keys

Automate monitoring

3. Monitor for unrotated keys

- AWS Security Hub Foundational Security Best Practices
 - Enable Security Hub FSBP checks
 - This adds a check for unrotated keys older than 90 days
- AWS Config check for unrotated keys
 - Enable AWS managed config rule - access-keys-rotated
 - Parameters set maxAccessKeyAge to 90

| | | | |
|---|---------------------|---------------|--|
|  | access-keys-rotated | IAM, Periodic | Checks whether the active access keys are rotated within the number of days specified in maxAccessKeyAge. The rule is non-compliant if the access keys have not been rotated for more than maxAccessKeyAge number of days. |
|---|---------------------|---------------|--|

- AWS Trusted Advisor
 - Available for enterprise support customers
 - Information comes from IAM credential report

Use roles and SSO

Alternatives to access keys

Single sign on (SSO)

1. Centralized identity management
2. Centralized MFA enforcement
3. Connect with existing identity provider (e.g., Active Directory, Okta)
4. Using SSO is recommended for AWS Management Console access
5. Enable AWS CLI to use SSO
 - <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-sso.html>

Alternatives to access keys

Using AWS IAM roles

1. Most AWS services support the use of temporary security credentials
2. Services that support it automatically retrieve temporary security credentials
3. EC2 instances use Instance Profiles; can replace access keys deployed to instances
4. AWS IAM Roles can enable cross-account access

Find and avoid using hard coded credentials

Find hard coded credentials

Using tools to find embedded access keys

1. Amazon Macie

- Built-in managed identifier to find AWS credentials in files

| Data type | Keyword required | Additional information | Countries and regions |
|-----------------|--|------------------------|-----------------------|
| AWS secret keys | Yes, including: aws_secret_access_key, credentials, secret access key, secret key, set-awscredential | – | Any |

2. awslabs/git-secrets

- Prevents you from committing passwords and other sensitive information to a git repository
- <https://github.com/awslabs/git-secrets>

3. AWS Trusted Advisor checks for exposed access keys

- Checks popular code repositories for access keys that have been exposed
- Checks for irregular Amazon EC2 usage that could be the result of a compromised access key.

Avoid using hard coded credentials

Store keys without embedding them in code

1. AWS Secrets Manager

- Easily rotate, manage, and retrieve database or other credentials
- Users and applications retrieve secrets with a call to Secrets Manager APIs
- Eliminates the need to hardcode sensitive information in plain text

2. AWS Systems Manager Parameter Store

- Provides secure, hierarchical storage for configuration data management and secrets management
- Store data such as passwords, database strings, Amazon Machine Image (AMI) IDs, and license codes
- Reference stored values by unique name in scripts, commands and automation workflows

Implement least privilege

Implement least privilege

Polices on IAM users and roles

1. Scope down permissions in policy

- Give every user or process the minimal amount of permissions

2. Write policy based on conditions

- Write policy that is specific to your organization
- VPC as a boundary condition
- OrgID as a requirement

Restrict access from SourceIp and VPC

```
"Condition": {
  "IpAddressIfExists": {"aws:SourceIp" : ["xxx"] },
  "StringEqualsIfExists" : {"aws:SourceVpc" : ["yyy"]}
}
```

Deny if MFA doesn't exist

```
"Effect" : "Deny",
"Condition" : { "BoolIfExists" : {
  "aws:MultiFactorAuthPresent" : "false" } }
```

IAM policy that allows users with the tagManager=true tag to manage IAM users, groups, or roles

```
"Effect": "Allow",
  "Action": "iam:*",
  "Resource": "*",
  "Condition": {"StringEquals":
{"aws:PrincipalTag/tagManager": "true"}}
```

https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_condition-keys.html

Implement least privilege

Utilize AWS managed policies

1. Designed to provide permissions for many common use cases
2. Easier for you to assign appropriate permissions to users, groups, and roles
3. Don't have to write the policies yourself

Example Use Case:

Attach the *AmazonS3ReadOnlyAccess* policy to a role used by an Amazon EC2 instance that allows read-only access to all Amazon S3 buckets in your account.

Full access AWS managed policies

- AmazonDynamoDBFullAccess
- IAMFullAccess

Power-user AWS managed policies

- AWSCodeCommitPowerUser
- AWSKeyManagementServicePowerUser

Partial-access AWS managed policies

- AmazonMobileAnalyticsWriteOnlyAccess
- AmazonEC2ReadOnlyAccess

Implement least privilege

Avoid the use of "*"

1. Using the wildcard "*" can lead to overly permissive policies
2. But "star" in "Deny" statements is ok
3. Anti Pattern: Devs will want full access to ease production, which can lead to security inconsistencies
4. Solve by managing access centrally with Service Control Policies in AWS Organizations

| Not Recommended | Recommended |
|--|--|
| "Resource": "arn:aws:iam::123456789012:user/*" | "Resource": "arn:aws:iam::123456789012:user/Maria" |
| "Resource": "*" | "Resource": "arn:aws:s3:::awsexamplebucket1/*" |
| "Action": "ec2:*" | "Action": "ec2:RunInstances" |

There are minor caveats: API calls using AWSLambdaBasicExecutionRole uses "star" because Lambda no function to predefine what log group it is going to write to

Implement least privilege

Avoid the use of "*"

| Not Recommended | Recommended |
|---|---|
| <pre>{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["S3:*"], "Resource": "*" }] }</pre> | <pre>{ "Version": "2012-10-17", "Statement": [{ "Effect": "Allow", "Action": ["s3:PutObject", "s3:PutObjectAcl", "s3:GetObject", "s3:GetObjectAcl", "s3:DeleteObject"], "Resource": ["arn:aws:s3:::Production/*", "arn:aws:s3:::Production"] }] }</pre> |

Implement least privilege

IAM Access Analyzer

AWS IAM Access Analyzer is a free service that helps you identify the resources in your organization and accounts, such as Amazon S3 buckets or IAM roles, that are shared with an external entity.

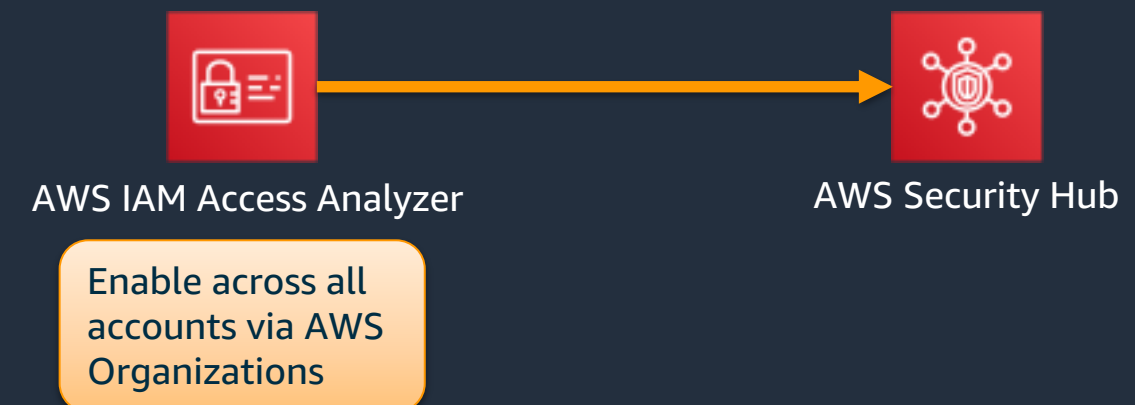
This lets you identify unintended access to your resources and data, which is a security risk.

The IAM Access Analyzer integration with Security Hub enables you to send findings from Access Analyzer to Security Hub. Security Hub can then include those findings in its analysis of your security posture.

IAM Access Advisor helps you audit access and remove unnecessary permissions. Use it frequently to scope appropriate permissions.

Supported resource types:

- Amazon S3 buckets
- AWS IAM roles
- AWS KMS keys
- AWS Lambda functions and layers
- Amazon SQS queues
- AWS Secrets Manager secrets



Implement least privilege

Resources

- Techniques for writing least privilege IAM policies
<https://aws.amazon.com/blogs/security/techniques-for-writing-least-privilege-iam-policies/>
- Security best practices and use cases in AWS Identity and Access Management
<https://docs.aws.amazon.com/IAM/latest/UserGuide/IAMBestPracticesAndUseCases.html>
- Security best practices in IAM
<https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html>

Resources to help you

- Your account team
 - Solutions Architect
 - Technical Account Managers
- Well Architected Review
- Enterprise Support

Thank you