Access Keys Will Kill You

Before You Kill The Password

Loïc Simon

Who Am I?

- Loïc Simon
- Principal Security Engineer @ NCC Group
- Author of Scout2
  - Security Auditing Tool for AWS environments
    - Static analysis of AWS resources
    - Security-oriented views of key resources
- Author of AWS-recipes
  - Repository of various tools and policies

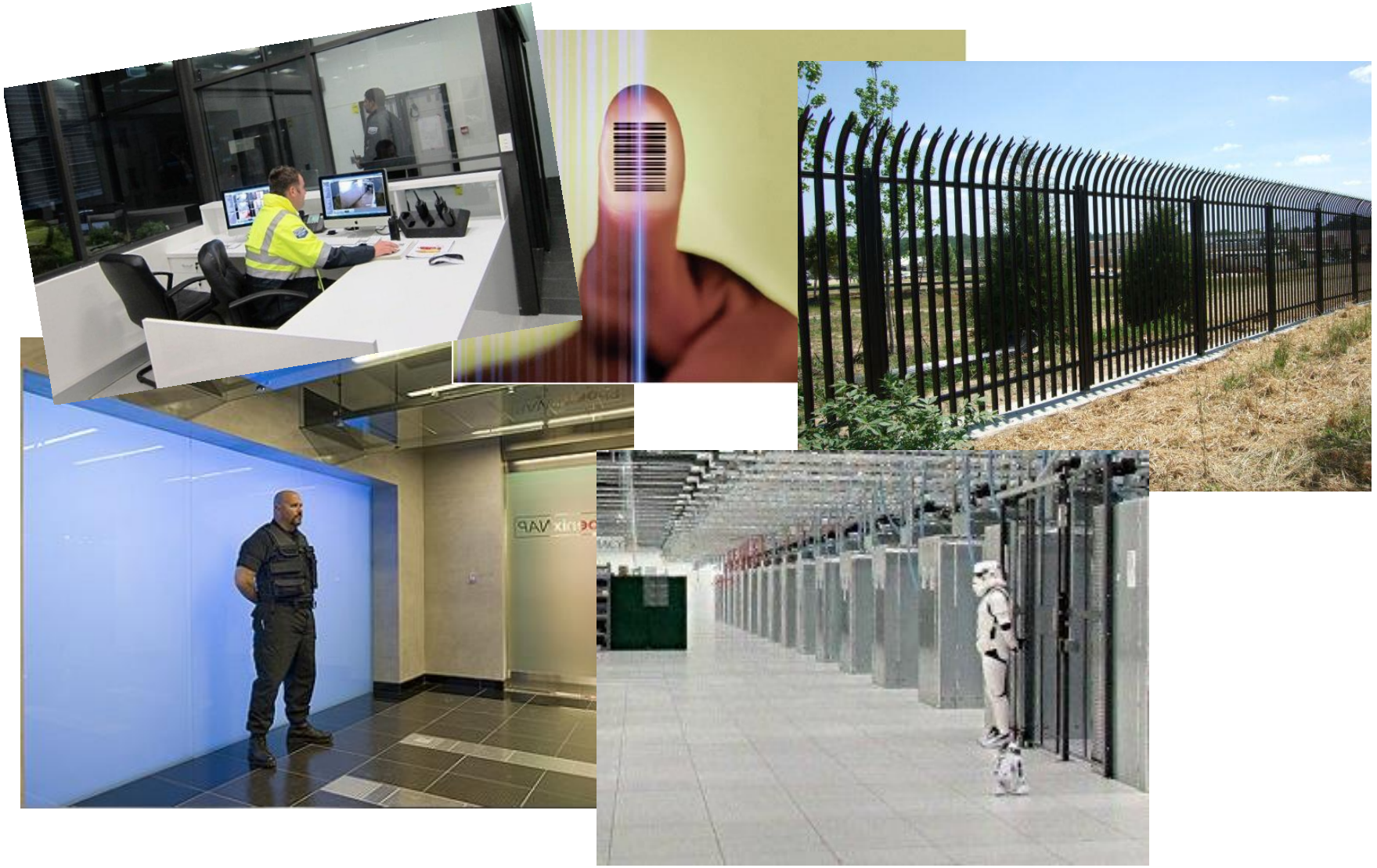
What is that all about?

- Goal
  - Present hardening solutions for AWS environments that I have recommended and implemented
  - Demonstrate how accessible such solutions are when using the right policies and tools

- Agenda
  - Passwords, Access Keys, and Security
  - Fun with IAM Policies
  - Tools

Passwords, Access Keys, and Security

Outside of the cloud…



In the cloud…

In the cloud…

- Infrastructure management via web app
  - Credentials give you access to ***everything***
    - Stored data
    - Databases
    - Application servers
    - Firewall configuration
    - Logging and monitoring
    - …

In the cloud…

- Different security model than on premises
  - Strong access controls are available

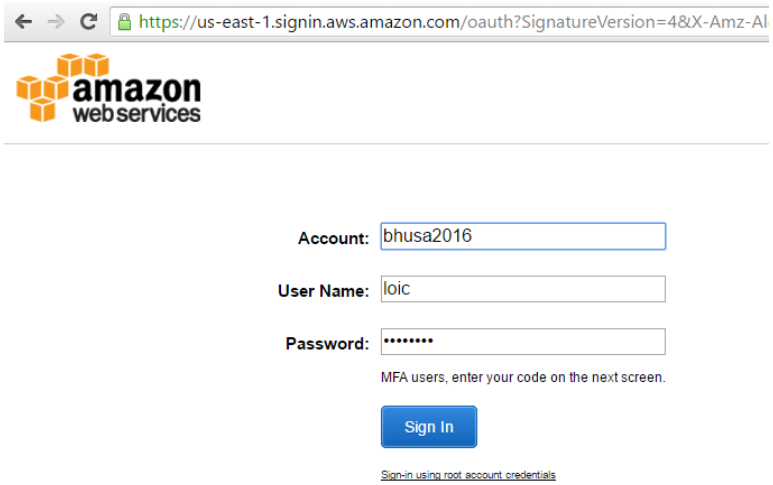
- Apply as many layers of defense as possible
  - **Require MFA**
  - Have short session timeout
  - IP-based restrictions
  - Require use of TLS

Authentication in AWS

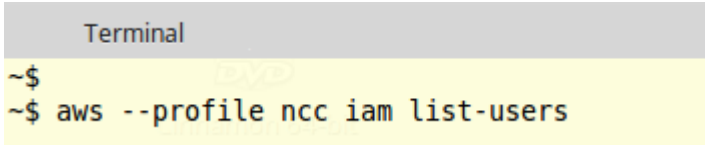
- Identity and Access Management (IAM)
  - AWS' "directory" (users and groups)
  - AWS' access controls (done via policies)
  - IAM credentials valid until user deletes/changes them
- Security Token Service (STS)
  - Issues temporary, limited-privilege credentials
  - STS credentials valid between 15 minutes and 36 hours

Authentication in AWS

- Web Console
  - Account ID if using IAM
  - Username
  - Password



- Tools via the API
  - Long Lived IAM Credentials (AKIA…)
    - AWS Access Key ID
    - AWS Secret Access Key



Passwords vs Access Keys *

	AWS Passwords	Access Keys
Unique, Random value	Maybe	Yes
Shared between users	Maybe	Maybe
Hardcoded in source	No	Yes
Stored on Post-It note	No	No
Stored in plaintext files	No	Yes
Rotated periodically	Maybe	No
Rotation enforceable	Yes	No
MFA available	Yes	Yes
MFA required if token exists	Yes	No
MFA enforced	No	No

* Based on past security assessments

Passwords vs Access Keys *

	AWS Passwords	Access Keys
Unique, Random value	Maybe	Yes
Shared between users	Maybe	Maybe
Hardcoded in source	No	Yes
Stored on Post-It note	No	No
Stored in plaintext files	No	Yes
Rotated periodically	Maybe	No
Rotation enforceable	Yes	No
MFA available	Yes	**Yes**
MFA required if token exists	Yes	**No**
MFA enforced	**No**	**No**

* Based on past security assessments

Passwords vs Access Keys *

- AWS admins have decent behavior password-wise
  - Use a password manager
  - MFA enabled as part of onboarding process

- Access keys are the weakest link
  - Found everywhere
    - Github
    - Internally accessible configuration files
    - Baked into public binaries
    - Stored on laptops under ~/.aws/credentials

* Based on past security assessments

MFA with Access Keys

- Require all human users to use MFA
- Regardless of how they access the API

  - Password-based authentication
    - Just create an MFA device
    - Problem: user may disable and delete MFA device if authorized

  - Access key-based authentication
    - Need to create and apply a policy
    - The policy will address the above problem

Authentication in AWS (with MFA)

- Web Console
  - Account ID if using IAM
  - Username
  - Password
  - MFA code

- Tools via the API
  - STS: long-lived credentials
    - AWS Access Key ID (AKIA…)
    - AWS Secret Access Key
    - MFA Code
  - All other services: short Lived Credentials
    - AWS Access Key ID (ASIA…)
    - AWS Secret Access Key
    - Session Token

Authentication in AWS (with MFA)

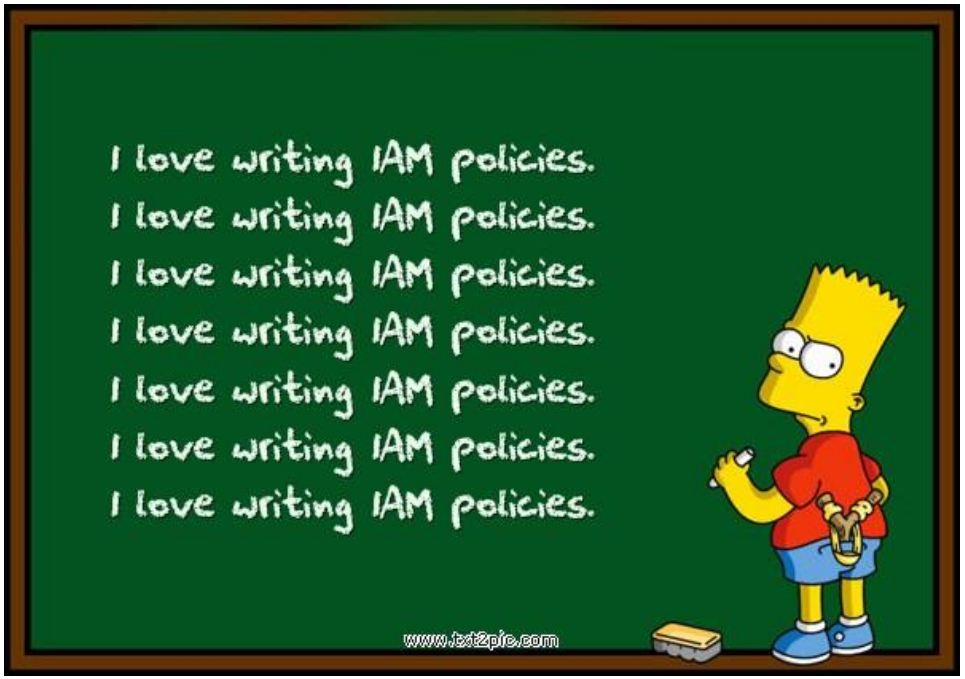
Long lived credentials + MFA code

=

Short lived credentials

- Long lived credentials
  - AWS Access Key ID (AKIA…) + AWS Secret Access Key
  - Username + Password
- Short lived credentials
  - AWS Access Key ID (ASIA…)
  - AWS Secret Access Key
  - Session Token

Fun with IAM policies

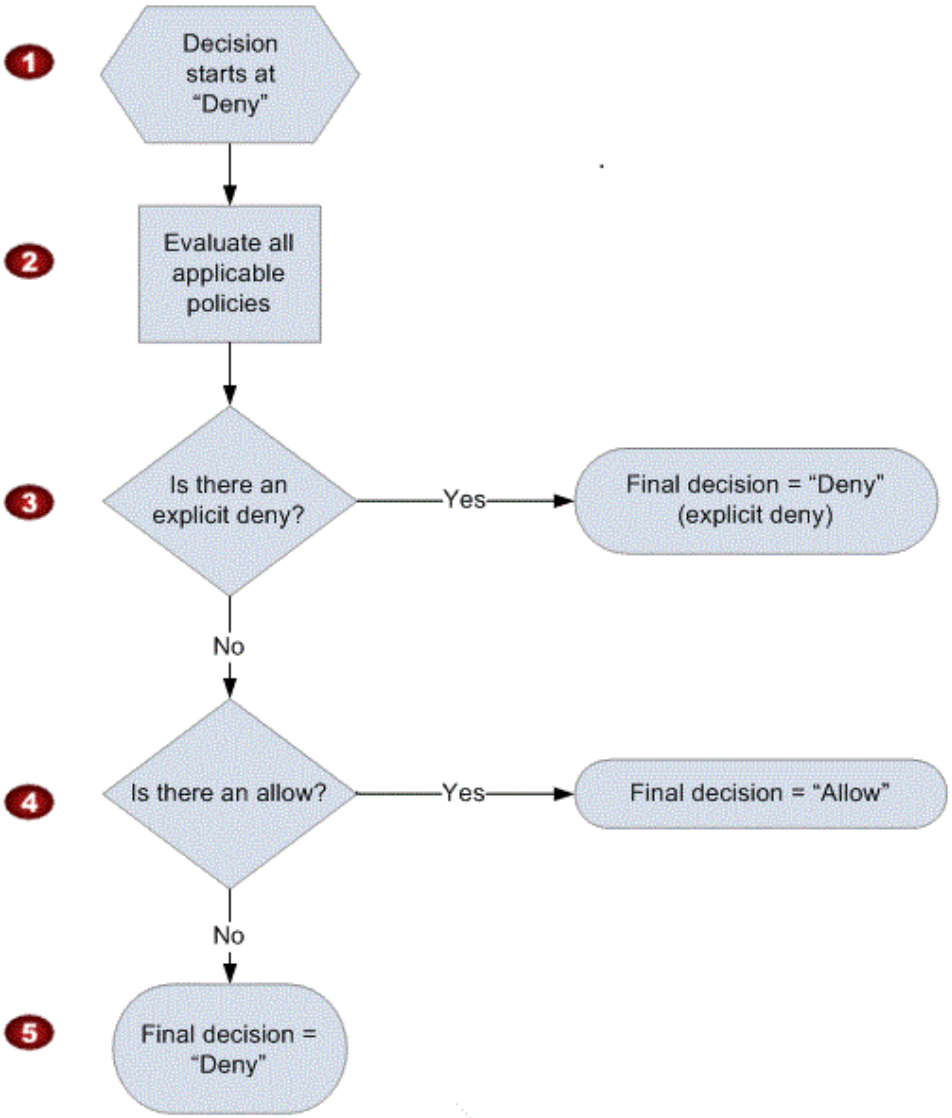


Reminder about IAM policies

- Policy
  - Set of permissions defined as a list of statements
  - JSON

- Statement
  - Rule defined by
    - Effect: Allow or **Deny**
    - Action
    - Resource: object the action applies to
    - **Condition**

Reminder about IAM policies



Policy#1: Strict MFA Enforcement

- Use the Deny effect
- Deny all actions
- Use conditions
  - aws:MultiFactorAuthPresent (Existence)
  - aws:MultiFactorAuthAge (Duration)

Policy#1: Strict MFA Enforcement


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:MultiFactorAuthAge": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NumericGreaterThan": {
          "aws:MultiFactorAuthAge": "28800"
        }
      }
    }
  ]
}
```

Policy#1: Strict MFA Enforcement

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:MultiFactorAuthAge": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NumericGreaterThan": {
          "aws:MultiFactorAuthAge": "28800"
        }
      }
    }
  ]
}
```

If the key “MultiFactorAuthAage” does not exist

Policy#1: Strict MFA Enforcement

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:MultiFactorAuthAge": "true"
        }
      }
    },
    {
      "Effect": "Deny",
      "Action": "*",
      "Resource": "*",
      "Condition": {
        "NumericGreaterThan": {
          "aws:MultiFactorAuthAge": "28800"
        }
      }
    }
  ]
}
```

If the key "MultiFactorAuthAage" does not exist

If the value of "MultiFactorAuthAage" is more than 8 hours (28800 seconds)

How to use Policy #1?

- Use “Category” groups
  - AllUsers
    - Every single IAM user
  - AllHumanUsers
    - Every IAM user associated with a human
  - AllServiceUsers *
    - Every IAM user used by a service

* Hopefully empty

How to use Policy #1?

- Use "Category" groups
  - AllUsers
    - Every single IAM user
  - AllHumanUsers
    - Every IAM user associated with a human

Enforce MFA

  - AllServiceUsers *
    - Every IAM user used by a service

* Hopefully empty

How to use Policy #1?

- Create the AllHumanUsers group
- Place all human users in the AllHumanUsers group
- Attach Policy#1 to this group

How to use Policy #1?

- Create the AllHumanUsers group
- Place all human users in the AllHumanUsers group
- Attach Policy#1 to this group

- Nothing works anymore, you're secure !
  - Need to deploy MFA-protected API access slowly…

Policy#1: Enforce MFA

- Works
- May be too restrictive for some AWS users
  - All IAM management must be done by IAM admins
- Credentials generated on a limited number of machines
  - IAM Admin's computers

Better workflow?

- Suggestion
  - Admin creates new IAM users
  - Admin generates a temporary password for that user
  - User connects and changes their password
  - User enrolls in MFA on their own
    - User cannot access other services until they authenticate with MFA
  - User logs out, logs in, and can access other services

- Advantages
  - Admin never knows user chosen/generated credentials
  - Users can manage their own credentials

Better workflow?

- Requirements
  - Need two new IAM policies
    - Policy#2: management of credentials
      - Only for the authenticated user
    - Policy#3: new MFA enforcement policy
      - Looser to allow MFA enrolment

Policy#2: credentials management


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*AccessKey*",
        "iam:*Password",
        "iam:*MFADevice*",
        "iam:UpdateLoginProfile"
      ],
      "Resource": "arn:aws:iam::AWS_ACCOUNT_ID:user/${aws:username}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateVirtualMFADevice",
        "iam:DeleteVirtualMFADevice"
      ],
      "Resource": "arn:aws:iam::AWS_ACCOUNT_ID:mfa/${aws:username}"
    }
  ]
}
```

Policy#2: credentials management


```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "iam:*AccessKey*",
        "iam:*Password",
        "iam:*MFADevice*",
        "iam:UpdateLoginProfile"
      ],
      "Resource": "arn:aws:iam::AWS_ACCOUNT_ID:user/${aws:username}"
    },
    {
      "Effect": "Allow",
      "Action": [
        "iam:CreateVirtualMFADevice",
        "iam:DeleteVirtualMFADevice"
      ],
      "Resource": "arn:aws:iam::AWS_ACCOUNT_ID:mfa/${aws:username}"
    }
  ]
}
```

Policy#2: credentials management

- Authorizes users to
  - Manage their passwords
  - Manage their access keys
  - Manage their MFA devices

- For readability, this policy uses wildcard
  - Expand the list of actions when creating the policy

Policy#3: MFA enforce

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Deny",
      "NotAction": [
        "iam:ChangePassword",
        "iam:CreateVirtualMFADevice",
        "iam:EnableMFADevice",
        "iam:GetUser",
        "iam:ListMFADevices",
        "iam:ListUsers",
        "iam:ListVirtualMFADevices"
      ],
      "Resource": "*",
      "Condition": {
        "Null": {
          "aws:MultiFactorAuthAge": "true"
        }
      }
    },
```

```
{
    "Effect": "Deny",
    "NotAction": [
      "iam:ChangePassword",
      "iam:CreateVirtualMFADevice",
      "iam:EnableMFADevice",
      "iam:GetUser",
      "iam:ListMFADevices",
      "iam:ListUsers",
      "iam:ListVirtualMFADevices"
    ],
    "Resource": "*",
    "Condition": {
      "NumericGreaterThan": {
        "aws:MultiFactorAuthAge": "28800"
      }
    }
  }
 ]
}
```

Policy#3: MFA Enforce

- Deny NotAction [list] instead of Deny Action *

Action	Usage
iam:ChangePassword	Change temporary password upon 1st login
iam:CreateVirtualMFADevice	MFA enrollment
iam:EnableMFADevice	MFA enrollment
iam:GetUser	MFA enrollment via CLI
iam:ListUsers	MFA enrollment via AWS web console
iam:ListMFADevices	MFA enrollment via AWS web console
iam:ListVirtualMFADevices	MFA enrollment via AWS web console

- Same MFA conditons as policy#1

Policy#3: MFA Enforce

- Trust Of First Use
  - Gaps compared to strict policy#1
    - 1st login
    - When MFA is disabled
  - To prevent gap #2, forbid deleting and disabling MFA
    - Infrequent request
    - Require an IAM admin to do that on behalf of user

Tools

Tool#1: Enable MFA

- Requirements
  - Already configured long-lived credentials for CLI

- Usage

```
$ git clone https://github.com/nccgroup/AWS-recipes.git
$ cd AWS-recipes/Python
$ pip install -r requirements.txt
$ python aws_iam_enable_mfa.py --profile ncc
```

- Flow
  - Creates a new MFA virtual device
  - Displays the QR code
  - Prompts for two consecutive codes to enable the device
  - Saves the MFA serial

Tool#1: Enable MFA

```
[ncc]
aws_access_key_id = AKIA...
aws_secret_access_key = Hqas...
```

```
[ncc]
aws_access_key_id = AKIA...
aws_secret_access_key = Hqas...
aws_mfa_serial = arn:aws:iam::...:mfa/loic...
```

Tool#2: Init STS session

- Requirements
  - Already configured long-lived credentials and MFA serial

- Usage

```
$ git clone https://github.com/nccgroup/AWS-recipes.git
$ cd AWS-recipes/Python
$ pip install -r requirements.txt
$ python aws_recipes_init_sts_session.py --profile ncc
```

- Flow
  - Prompts for an MFA code
  - Saves STS credentials

Tool#2: Init STS session

```
[ncc]
aws_access_key_id = AKIA...
aws_secret_access_key = Hqas...
aws_mfa_serial = arn:aws:iam::...:mfa/loic...
```

```
[ncc]
aws_access_key_id = ASIAI...
aws_secret_access_key = xoEpg2t2aS...
aws_mfa_serial = arn:aws:iam::...
aws_session_token = AQoDYXdzEMv//...
[ncc-nomfa]
aws_access_key_id = AKIAJ...
aws_secret_access_key = Hqas...
aws_mfa_serial = arn:aws:iam::...
```

Tool#2: Init STS session

- Two profiles
  - ncc-nomfa
    - IAM Long lived credentials
  - ncc
    - STS short-lived credentials

- The tool knows to use the -nomfa profile to initiate new STS sessions
- If necessary, long-lived credentials are accessible using the -nomfa profile

Tool#3: Rotate Key

- Requirements
  - Already configured long-lived credentials

- Usage

```
$ git clone https://github.com/nccgroup/AWS-recipes.git
$ cd AWS-recipes/Python
$ pip install -r requirements.txt
$ python aws_iam_rotate_my_key.py --profile ncc
```

- Flow
  - Creates a new access key
  - If MFA is configured, prompts for an MFA code
  - Validates that new STS sessions can be established
  - Saves new IAM credentials

Tool#3: Rotate Key

```
[ncc]
aws_access_key_id = ASIAI...
aws_secret_access_key = x0Epg2t2aS...
aws_mfa_serial = arn:aws:iam::...
aws_session_token = AQoDYXdzEMv//...
[ncc-nomfa]
aws_access_key_id = AKIAJ...
aws_secret_access_key = Hqas...
aws_mfa_serial = arn:aws:iam::...
```

```
[ncc]
aws_access_key_id = ASIAI7RKWJGSI....
aws_secret_access_key = Fi8NbjwtoHrgNji
aws_mfa_serial = arn:aws:iam::...
aws_session_token = AQoDYXdzEMv////...
[ncc-nomfa]
aws_access_key_id = AKIAJFIF...
aws_secret_access_key = Iz5zcVUzIPz....
aws_mfa_serial = arn:aws:iam::...
```

Takeaways

- Access Keys are the root cause of many incidents in AWS
- MFA can be enforced consistently
  - Deny statements are powerful
- Tools exist to allow seamless work with enforced MFA

Thank You, Questions?

- Loïc Simon
  - Loic.Simon@nccgroup.trust

- Tools on GitHub
  - https://github.com/nccgroup/AWS-recipes
  - https://github.com/nccgroup/Scout2