AWS CLOUDFORMATION – AWS IAM: HOW TO DELEGATE DEPLOYMENT SAFELY

AWS CloudFormation | AWS Identity and Access Management (IAM) | DevSecOps | Infrastructure as Code (IaC)

beSharp | 30 April 2020

Day by day the number of companies that look with interest at the cloud computing world increases and many of them wonder how the cloud can help their business and which security flaws it can introduce. One of the most complicated tasks of the security team in fact is to understand how to exploit novel cloud technologies, with their unprecedented flexibility, without introducing security vulnerabilities.

One of the most useful tools to enforce compliance with security best practices is certainly Infrastructure as Code (IaC), which allows us to handle our resources inside the cloud environment through templating systems. This makes it possible to deploy the infrastructure in a repeatable and reproducible way.

What is AWS CloudFormation?

In this article we will focus on the AWS service called AWS CloudFormation.

CloudFormation lets you create, update and handle resources in your AWS Cloud Environment through the use of JSON or YAML templates in which you can describe resource by resource your own infrastructure.

This leads to a huge set of questions: how can I have control on who deploy what? If I can handle each resource, can I also handle resources permissions? If I can handle permissions, how can I delegate the use of CloudFormation to other parties or team members without the risk of privileges escalation attacks?

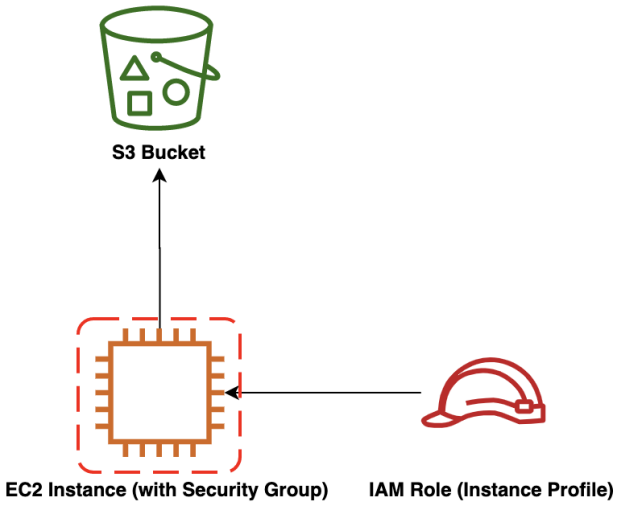
To answer these questions we need to introduce another service: AWS Identity Access Management (AWS IAM).

What is AWS IAM?

AWS Identity and Access Management (IAM) is the AWS service that allows one to handle all permissions inside your AWS Cloud Environment. If you want to execute any action (using the Console, the CLI or the SDK) the permission to do so has to be written inside a policy attached to your “user”.

Use Case

Now that we know which are the instruments we can use, let’s describe a very common situation: we created a CloudFormation template that provisions an infrastructure composed by a pre-configured EC2 instance (through Amazon Machine Image) which needs to access an S3 bucket. Being a simple infrastructure, we would like to allow the developers to deploy this template without asking the Security team.



More farsighted people know that to deploy this infrastructure the CloudFormation template needs to create an IAM Role for the EC2. Thus we need to make sure that the developer who will run the template will not be able to modify the policies of this role in order to give the EC2 (and thus himself) wider permission (e.g. admin access).

Without Permission Boundaries

The template that we will deploy is the following one:

```
---
AWSTemplateFormatVersion: '2010-09-09'
Description: |
 CloudFormation and IAM Permission Boundaries Demo


################################################################################
```

```
#                                    Metadata                                  #
##############################################################################
Metadata:

 AWS::CloudFormation::Interface:
   ParameterGroups:
     - Label: {default: 'Required parameters'}
       Parameters:
         - VpcId
         - AmiId
         - KeyName
         - SubnetId
     - Label: {default: 'Optional parameters'}
       Parameters:
         - NameSpace
         - ProjectName
         - Environment


##############################################################################
#                                   Parameters                                 #
##############################################################################
Parameters:

 NameSpace:
   Type: String
   Default: 'besharp'

 ProjectName:
   Type: String
   Default: 'permission-boundaries-demo'

 Environment:
   Type: String
   Default: 'dev'
  VpcId:
   Type: AWS::EC2::VPC::Id

 SubnetId:
   Type: AWS::EC2::Subnet::Id

 AmiId:
   Type: AWS::EC2::Image::Id

 KeyName:
   Type: AWS::EC2::KeyPair::KeyName


##############################################################################
#                                   Conditions                                 #
##############################################################################
Conditions: {}


##############################################################################
#                                    Mappings                                  #
##############################################################################
Mappings: {}
```

```yaml
##################################################################################
#                                  Resources                                     #
##################################################################################
Resources:

 ################################## S3 ##################################
 S3Bucket:
   Type: AWS::S3::Bucket
   Properties:
     BucketName: !Sub 'com.${NameSpace}.${ProjectName}'
     Tags:
       - Key: Name
         Value: !Sub '${NameSpace}-${ProjectName}'
       - Key: Owner
         Value: 'name.surname@besharp.it'

 ################################## EC2 ##################################
 EC2Instance:
   Type: AWS::EC2::Instance
   Properties:
     IamInstanceProfile: !Ref IAMInstanceProfile
     ImageId: !Ref AmiId
     InstanceType: t3a.micro
     KeyName: !Ref KeyName
     NetworkInterfaces:
       - AssociatePublicIpAddress: 'true'
         DeviceIndex: '0'
         GroupSet:
           - !Ref EC2SecurityGroup
         SubnetId: !Ref SubnetId
     Tags:
       - Key: Name
         Value: !Sub '${NameSpace}-${ProjectName}'
       - Key: Owner
         Value: 'name.surname@besharp.it'

 EC2SecurityGroup:
   Type: AWS::EC2::SecurityGroup
   Properties:
     GroupName: !Sub '${NameSpace}-${ProjectName}-ec2'
     GroupDescription: !Sub 'Security Group for ${NameSpace}-${ProjectName}-ec2'
     VpcId: !Ref VpcId
     Tags:
       - Key: Name
         Value: !Sub '${NameSpace}-${ProjectName}-ec2'
       - Key: Owner
         Value: 'name.surname@besharp.it'

 ################################## IAM ##################################
 IAMInstanceProfile:
   Type: AWS::IAM::InstanceProfile
   Properties:
     Roles:
       - !Ref IAMRole
     InstanceProfileName: !Sub '${NameSpace}-${ProjectName}'

 IAMRole:
   Type: AWS::IAM::Role
   Properties:
     RoleName: !Sub '${NameSpace}-${ProjectName}'
```

```
      AssumeRolePolicyDocument:
        Version: '2012-10-17'
        Statement:
          - Effect: Allow
            Principal:
              Service: ec2.amazonaws.com
            Action: sts:AssumeRole
      Path: '/'
      Policies:
        - PolicyName: 'EC2Access'
          PolicyDocument:
            Version: '2012-10-17'
            Statement:
              - Effect: 'Allow'
                Action:
                  - 's3:GetObject'
                Resource: !Sub '${S3Bucket.Arn}/*'
      Tags:
        - Key: Name
          Value: !Sub '${NameSpace}-${ProjectName}'
        - Key: Owner
          Value: 'name.surname@besharp.it'


###############################################################################
#                                    Outputs                                  #
###############################################################################
Outputs:

 StackName:
   Description: 'Stack name.'
   Value: !Sub '${AWS::StackName}'
```

Here we can find the information to create:

- S3 Bucket
- EC2 Instance + Security Group
- IAM Role + Instance Profile

We attached to the developer IAM user the following policy so that he can deploy that template:

```
{
   "Version": "2012-10-17",
   "Statement": [
       {
           "Sid": "CloudFormationReadAccess",
           "Effect": "Allow",
           "Action": [
               "cloudformation:DescribeStacks",
               "cloudformation:ListChangeSets",
               "cloudformation:ListExports",
               "cloudformation:ListImports",
               "cloudformation:ListStacks"
```

```
            ],
            "Resource": "*"
        },
        {
            "Sid": "CloudFormationWriteAccess",
            "Effect": "Allow",
            "Action": [
                "cloudformation:CreateStack",
                "cloudformation:DeleteStack",
                "cloudformation:TagResource",
                "cloudformation:UpdateStack",
                "cloudformation:ValidateTemplate"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2ReadAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:DescribeImages",
                "ec2:DescribeInstances",
                "ec2:DescribeKeyPairs",
                "ec2:DescribeSecurityGroupReferences",
                "ec2:DescribeSecurityGroups",
                "ec2:DescribeSubnets",
                "ec2:DescribeVpcs"
            ],
            "Resource": "*"
        },
        {
            "Sid": "EC2WriteAccess",
            "Effect": "Allow",
            "Action": [
                "ec2:AssociateIamInstanceProfile",
                "ec2:AuthorizeSecurityGroupEgress",
                "ec2:AuthorizeSecurityGroupIngress",
                "ec2:CreateSecurityGroup",
                "ec2:CreateTags",
                "ec2:DeleteSecurityGroup",
                "ec2:DeleteTags",
                "ec2:RevokeSecurityGroupEgress",
                "ec2:RevokeSecurityGroupIngress",
                "ec2:RunInstances",
                "ec2:StartInstances",
                "ec2:StopInstances",
                "ec2:TerminateInstances"
            ],
            "Resource": "*"
        },
        {
            "Sid": "IAMReadAccess",
            "Effect": "Allow",
            "Action": [
                "iam:GetInstanceProfile",
                "iam:GetRole",
                "iam:GetRolePolicy"
            ],
            "Resource": "*"
        },
        {
```

```
            "Sid": "IAMWriteAccess",
            "Effect": "Allow",
            "Action": [
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:CreateRole",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:PassRole",
                "iam:PutRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "iam:UntagRole"
            ],
            "Resource": "*"
        },
        {
            "Sid": "S3WriteAccess",
            "Effect": "Allow",
            "Action": [
                "s3:CreateBucket",
                "s3:DeleteBucket",
                "s3:PutBucketTagging"
            ],
            "Resource": "*"
        }
    ]
}
```

If we connect to the newly created EC2 instance, we can verify that it can download objects from the S3 Bucket:

```
ubuntu@ec2-demo:~$ aws s3 cp s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine .

download: s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine
 to ./if-you-download-me-you-are-fine
ubuntu@ec2-demo:~$ cat if-you-download-me-you-are-fine
test-ok
```

And that it can't, for example, create other buckets:

```
ubuntu@ec2-demo:~$ aws s3api create-bucket --bucket can-i-create-it

An error occurred (AccessDenied) when calling the CreateBucket operation: Access Denied
```

Now, the developer (regardless of the good or bad intention) might want to modify the set of permissions attached to the role to execute actions that normally he is not authorized to do. The IAM Role resource can be modified like this:

```
IAMRole:
  Type: AWS::IAM::Role
  Properties:
    RoleName: !Sub '${NameSpace}-${ProjectName}'
    AssumeRolePolicyDocument:
      Version: '2012-10-17'
      Statement:
        - Effect: Allow
          Principal:
            Service: ec2.amazonaws.com
          Action: sts:AssumeRole
    Path: '/'
    Policies:
      - PolicyName: 'AdministratorAccess'
        PolicyDocument:
          Version: '2012-10-17'
          Statement:
            - Effect: 'Allow'
              Action:
                - '*'
              Resource: '*'
    Tags:
      - Key: Name
        Value: !Sub '${NameSpace}-${ProjectName}'
```

In this way, for example, the instance can download objects from the bucket, but also create new buckets (not to mention that, modifying the policy roughly like that, in fact it has administrator permissions inside the account):

```
ubuntu@ec2-demo:~$ aws s3 cp s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine .

download: s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine
 to ./if-you-download-me-you-are-fine
ubuntu@ec2-demo:~$ cat if-you-download-me-you-are-fine
test-ok
ubuntu@ec2-demo:~$ aws s3api create-bucket --bucket can-i-create-it
{
    "Location": "/can-i-create-it"
}
```

Clearly the principal concern is not only the fact that a developer can gain unintended permissions, but also the increased attack surface exploitable from malicious entities: the old style linux privileges escalation ported to the cloud!

With Permission Boundaries

Permission boundaries are nothing more than additional IAM policies attached to an IAM entity to limit its permissions. Indeed, the resulting permissions will be the intersection between the ones granted by the IAM policy and the ones allowed by the permission boundary. By themselves, they do not resolve any problem but we can force the developer to attach the permission boundary if he wants to create a role.

The IAM policy part becomes like this:

```
{
            "Sid": "IAMPermissionBoundaryWriteAccess",
            "Effect": "Allow",
            "Action": [
                "iam:CreateRole",
                "iam:PutRolePolicy",
                "iam:UpdateRole",
                "iam:UpdateRoleDescription"
            ],
            "Resource": "arn:aws:iam::111122223333:role/dev-namespace/*",
            "Condition": {
                "StringEquals": {
                    "iam:PermissionsBoundary": "arn:aws:iam::111122223333:policy/beshar
p-permission-boundary-demo"
                }
            }
        },
        {
            "Sid": "IAMPassRoleAccess",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": "arn:aws:iam::111122223333:role/dev-namespace/*"
        },
        {
            "Sid": "IAMWriteAccess",
            "Effect": "Allow",
            "Action": [
                "iam:AddRoleToInstanceProfile",
                "iam:CreateInstanceProfile",
                "iam:DeleteInstanceProfile",
                "iam:DeleteRole",
                "iam:DeleteRolePolicy",
                "iam:RemoveRoleFromInstanceProfile",
                "iam:TagRole",
                "iam:UntagRole"
            ],
            "Resource": "*"
        },
```

If during the creation phase of the IAM role through CloudFormation is not present the permission boundary attachment, the creation will fail.

To attach the permission boundary to the instance IAM role, the resource will be modified as following:

```
IAMRole:
   Type: AWS::IAM::Role
   Properties:
     RoleName: !Sub '${NameSpace}-${ProjectName}'
     PermissionsBoundary: !Ref PermissionBoundaryArn
     AssumeRolePolicyDocument:
       Version: '2012-10-17'
       Statement:
         - Effect: Allow
           Principal:
             Service: ec2.amazonaws.com
           Action: sts:AssumeRole
     Path: '/dev-namespace/'
     Policies:
       - PolicyName: 'AdministratorAccess'
         PolicyDocument:
           Version: '2012-10-17'
           Statement:
             Effect: 'Allow'
             Action:
                - '*'
             Resource: '*'
     Tags:
       - Key: Name
         Value: !Sub '${NameSpace}-${ProjectName}'
       - Key: Owner
         Value: 'name.surname@besharp.it'
```

Note: to make the solution safer and to limit the dangerous action iam:PassRole, we added the Path attribute, which in fact creates a namespace for developers inside IAM.

Attaching the permission boundary, the creation phase will succeed and the instance permissions (and the ones of any IAM role that the developer wants to create) will be limited.

```
ubuntu@ec2-demo:~$ aws s3 cp s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine .

download: s3://com.besharp.permission-boundaries-demo/if-you-download-me-you-are-fine
 to ./if-you-download-me-you-are-fine
ubuntu@ec2-demo:~$ cat if-you-download-me-you-are-fine
test-ok
ubuntu@ec2-demo:~$ aws s3api create-bucket --bucket can-i-create-it

An error occurred (AccessDenied) when calling the CreateBucket operation: Access Denied
```

In this configuration permission boundaries are the IAM policies under Security team control. The team can agree on which is the maximum set of permissions that a resource can have and AWS IAM will assure that all resources with the restricted IAM role associated with them will not be able to perform unintended actions or to create another resource which can do so.

Conclusion

We have seen how, thanks to a simple modification of the IAM entities configurations (Permission Boundaries), you could enforce the limitation of permissions. This approach usually identifies the developer as a delegated admin. In this way we can increase the independence of developers, reducing the number of daily tasks that the few people with wide permission inside the AWS Cloud Environment have to fulfill without introducing security exploits.

Satisfied? 🙂 feel free to leave comments or to contact us to discuss this or other possible solutions!

beSharp

Dal 2011 beSharp guida le aziende italiane sul Cloud. Dalla piccola impresa alla grande multinazionale, dal manifatturiero al terziario avanzato, aiutiamo le realtà più all'avanguardia a realizzare progetti innovativi in campo IT.

Get in touch

beSharp.it
proud2becloud@besharp.it