How to secure AWS services?

Craig Godden-Payne
Oct 26, 2018 · 6 min read ★

AWS takes security very seriously, and many physical security measures are taken place to improve security, such as the location of the AWS data centres, are not public knowledge, therefore not easy to find. Each data centre has controlled physical access and has been rated the best in class in terms of data centre security. AWS has a lot of reputational damage to lose, so security is of utmost importance.

AWS has procedures and protocols in place and received accreditations from many security organisations such as:

HIPAA, Soc1, SSAE16, ISAE3402, Soc2, Soc3, PCI DSS, ISO 27000, RedRAMP, DIACAP, FISMA, ITAR, FIPS140–2, CSA, MPAA and more.

Shared Security Responsibility

AWS works based on shared security responsibility, which means that to remain secure, as a consumer, you are expected to take some of the security responsibility. This tends to be split by:

AWS responsibility

- Virtual Host Security

- Physical Storage Security
- Network Security
- Data Centre Security
- Database Server Security

Customer responsibility

- AWS Account Security (MFA + API as well as console)
- Operating Systems
- Database Systems
- Applications running on Compute
- Data Encryption
- Authentication
- Network integrity.

Security methods + Connectivity

There are many products available from AWS which aim to make the platform more secure and are relatively simple to use.

Virtual private cloud level

You can use security groups and network control lists. A security group works as a virtual firewall to control inbound and outbound traffic. They work at the instance level, rather than the subnet level and a network control list acts as a firewall for controlling traffic in and out of one or more subnets.

Here is a more in-depth comparison.

Security Groups

- Operates at the instance level (first layer of defence)
- Supports allow rules only
- Is stateful: Return traffic is automatically allowed, regardless of any rules
- They evaluate all rules before deciding whether to allow traffic
- Applies to an instance only if someone specifies the security group when launching the instance, or associates the security group with the instance later on

Network ACLs

- Operates at the subnet level (second layer of defence)
- Supports allow rules and deny rules
- Is stateless: Return traffic must be explicitly approved by rules
- They process rules in number order when deciding whether to grant traffic
- Automatically applies to all instances in the subnets it's associated with (backup layer of defence, so you don't have to rely on someone specifying the security group)

Other defences

AWS offers a more secure offering for network traffic between your office and the AWS data centre, called Direct Connect. It can reduce network costs, increase bandwidth throughput, and provide a more consistent network experience than Internet-based connections.

There is also a product that is available as a service called WAF (Web Application Firewall). A web application firewall can help to protect web applications from common web exploits that could affect application availability, compromise security, or consume excessive resources.

More of a compliance issue, but it is also possible to provision compute resources as “Dedicated Servers” which cost much more, but mean that the compute resources you use are not multi-tenant.

Creating a Simple, Low-Cost Twitter Bot, utilising Serverless Technologies.

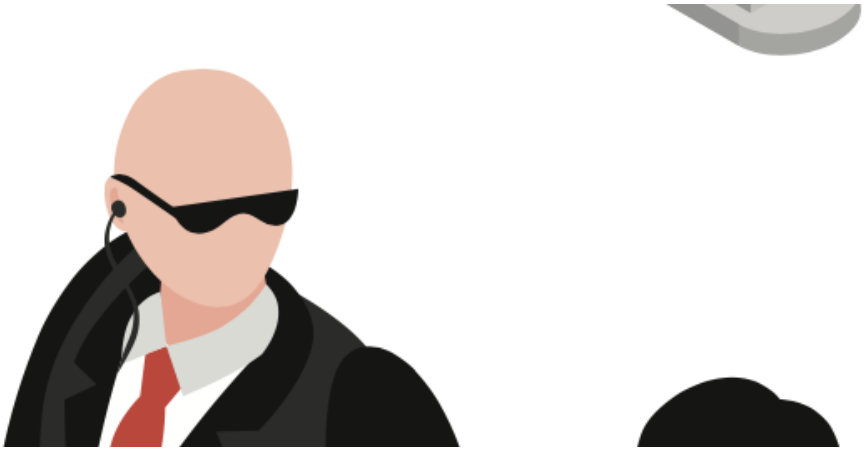
People have a love-hate relationship with twitter bots.

medium.com

Identity and Access management.

IAM allows you to manage access to AWS services and resources securely. You can use it to create and manage AWS users and groups, and use permissions to allow and deny their access to AWS resources. IAM is also free of charge!

There are four main types of identity and access management and use an RBAC type model.



Users

- Users can have permissions directly, or belong to a group, where the group stipulates the permissions

Groups

- Are used to group a bunch of permissions, for easier assigning to a user or multiple users.

Roles

- You apply a role to a User or an Instance. Roles are meant to be “assumed” rather than granted like you would in a group (i.e. its a temporary inflation of privileges).

Policies

- Policies are a collection of permissions that can be applied to users, groups or policies.

Setting up policies

The policy language you use when creating permissions using IAMs has two elements. Specification (defining access policies) and Enforcement (evaluating policies). Policies are written in JSON, and each statement should contain “PARC”

- Principle
- Action
- Resource
- Condition

Principal

A principal is an entity which is allowed or denied access to a resource. This can be indicated in different ways e.g.

```
– Anonymous Users
"Principal":"AWS":"*.*"

– Specific Accounts
"Principal":{"AWS":"arn:aws:iam::123456789012:root"}

– Individual IAM User
"Principal":{"AWS":"arn:aws:iam::123456789012:user/name"}

– Federated User
"Principal":{"Federated":"www.amazon.com"}

– Specific Role
"Principal":{"AWS":"arn:aws:iam::123456789012:role/rolename"}

– Specific Service
"Principal":{"Service":"ec2.amazonaws.com"}
```

Actions

An action describes the type of access that should be allowed or denied. It must include either an Action or NotAction. e.g.

```
– EC2 Action
"Action":"ec2:StartInstances"
```

```
- IAM Action
"Action":"iam:ChangePassword"

- S3 Action
"Action":"s3:GetObject"

- Multiple Action
"Action":["sqs:SendMessage", "sqs:RecieveMessage"]

- Wildcard Action
"Action":"sns:*"

- Wildcard Action (would deny all sns)
"NotAction":"sns:*"

- Using Multiples (this example will allow everything, except for
IAM)
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": "*",
    "Resource": "*"
  },
  {
    "Effect": "Deny",
    "Action": "iam:*",
    "Resource": "*"
  }]
}
```


Resource

A resource describes the object or objects that are being requested. It must include either a Resource or NotResource. e.g.

```
- S3 Bucket
"Resource": "arn:aws:s3:::/bucketname/*"

- SQS Queue
"Resource": "arn:aws:sqs:eu-west-2:123456789012:queuename"
```

```
- Multiple Dynamo Tables
"Resource": ["arn:aws:dynamodb:eu-west-
2:123456789012:table/tablename","arn:aws:dynamodb:eu-west-
2:123456789012:table/tablename2"]

- EC2 instances for an account in a region
"Resource": "arn:aws:ec2:eu-west-2:123456789012:instance/*"
```

Conditions

If you add a condition, it must evaluate to true, for the policy to evaluate as true. A condition can contain multiple conditions, and the keys can contain multiple values. e.g.

```
"Condition": {
  "DateGreaterThan": { "aws:CurrentTime": "2018-05-01T00:00:00Z" },
  "DateLessThan": { "aws:CurrentTime": "2018-05-22T00:00:00Z" }
  "IpAddress": { "aws:SourceIp": ["192.0.1.0/24", "192.0.2.0/24"] }
}
```

Policy Variables

You can use policy variables within your policies. The documentation is pretty good, over at https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_variables.html.

Some examples are:

```
aws:CurrentTime This can be used for conditions that check the date
and time.
aws:EpochTime This is the date in epoch or UNIX time, for use with
date/time conditions.
aws:SecureTransport This is a Boolean value that represents whether
the request was sent using SSL.
aws:SourceIp This is the requester's IP address, for use with IP
address conditions.
aws:UserAgent This value is a string that contains information about
the requester's client application.
aws:userid This value is the unique ID for the current user—see the
chart that follows.
aws:username This is a string containing the friendly name of the
current user—see the chart that follows.
```

Here is an example of using these policy variables:

```
{
  "Version": "2012-10-17",
  "Statement" : [{
    "Effect": "Allow",
    "Action": ["s3:ListBucket"],
    "Resource": ["arn:aws:s3:::bucketname/home/${aws:username}"]
  }]
}
```

Gotchas

There are some gotchas with policies.

- A deny will always win over an allow
- When using policy variables, you must always specify the version element
- Not all AWS services are supported. Check the documentation first!
- https://aws.amazon.com/documentation/iam/

Some Examples

Here is an example of allowing permission for a user to stop, start and terminate any EC2 instance within the account.

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Effect": "Allow",
    "Action": ["ec2:TerminateInstances"],
    "Resource": "arn:aws:ec2:eu-west-2:123456789012:instance/*"
  }]
}
```

More Examples

Here is an example of creating a “limited” IAM administrator.

The first statement shows allowing creating users, managing keys and setting passwords. The second statement shows this is limited to only attaching on the policy AmazonDynamoDBFullAccess

```
{
  "Version": "2012-10-17",
  "Statement": [{
    "Sid": "ManageUsersPermissions",
    "Effect": "Allow",
    "Action":
["iam:ChangePassword","iam:CreateAccessKey","iam:CreateLoginProfile"
,"iam:CreateUser",

"iam:DeleteAccessKey","iam:DeleteLoginProfile","iam:DeleteUser","iam
:UpdateAccessKey",
                "iam:ListAttachedUserPolicies","iam:ListPolicies"],
    "Resource": "*"
  },
  {
    "Sid": "LimitedAttachmentPermissions",
    "Effect": "Allow",
```

```
        "Action": ["iam:AttachUserPolicy", "iam:DetachUserPolicy"],
        "Resource": "*",
        "Condition": {
          "ArnEquals": {
            "iam:PolicyArn": [
              "arn:aws:iam::aws:policy/AmazonDynamoDBFullAccess"
            ]
          }
        }
      }]
    }
```

Graphics Attributions:
https://www.freepik.com/free-photos-vectors/technology
macrovector

AWS Iam Security