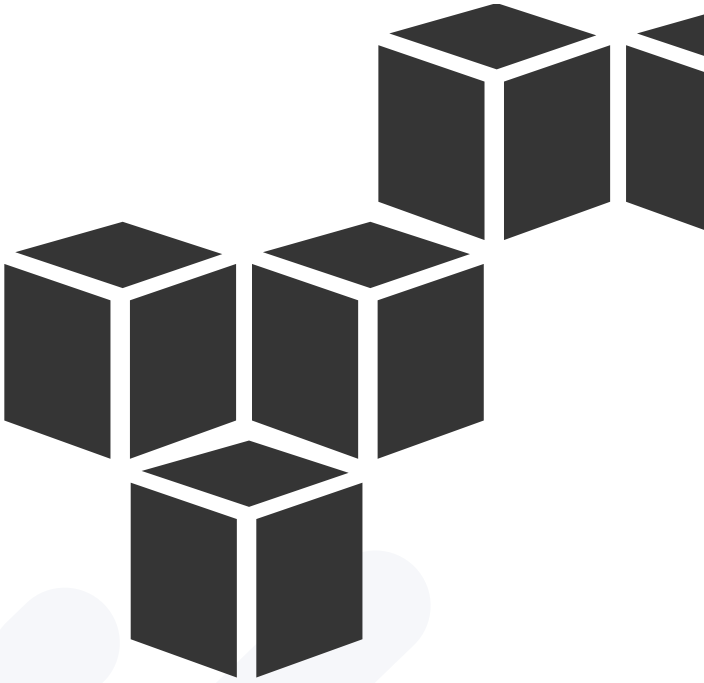
sqreen



AWS Security Best Practices

You use AWS. It's secure out of the box, but introducing security issues through misconfiguration is easy. This checklist will help guide you to potential security issues exposed by your AWS configuration, and will help you to tighten up the security of your AWS infrastructure.

The purpose of this article is to remind you of the most urgent security measures that should be taken on your AWS infrastructure. It is by no means exhaustive, and it should be adapted to your specific business use cases.

Enforce security settings for Identity and Access Management (IAM) accounts

The AWS IAM accounts are the most important part of your AWS setup, as they are where configuring the whole platform starts. Take these steps to secure them:

Use multi-factor authentication to protect your users against password theft.

Enable a strong password policy for good sources of information.

Here is our recommended account password policy:

```
$ aws iam get-account-password-policy

{
    "PasswordPolicy": {
        "RequireUppercaseCharacters":
true,
        "MinimumPasswordLength": 10,
        "RequireSymbols": true,
        "RequireNumbers": true,
        "HardExpiry": false,
        "RequireLowercaseCharacters":
true,
        "AllowUsersToChangePassword":
true,
        "ExpirePasswords": false
    }
}
```

The credential report can also provide you with additional access information related to each user. The report provides you with valuable information regarding your users, such as MFA status (also known as 2FA or

two factors authentication), last usage date of access keys. Two steps are necessary: first the report generation...

```
$ aws iam generate-credential-report

{ "State": "COMPLETE" }
```

...then the report retrieval.

```
$ aws iam get-credential-report | jq
'.Content' -r | base64 -D
user,arn,user_creation_time,password_e
nabled,password_last_used,password_las
t_changed,password_next_rotation,mfa_a
ctive,access_key_1_active,access_key_1
_last_rotated,access_key_1_last_used_d
ate,access_key_1_last_used_region,acce
ss_key_1_last_used_service,access_key_
2_active,access_key_2_last_rotated,acc
ess_key_2_last_used_date,access_key_2_
last_used_region,access_key_2_last_use
d_service,cert_1_active,cert_1_last_ro
tated,cert_2_active,cert_2_last_rotate
d
<root_account>,arn:aws:iam::
12345:root,
2015-08-03T12:09:08+00:00,not_supporte
d,
2016-06-23T11:56:59+00:00,not_supporte
d,not_supported,true,false,N/A,N/A,N/
A,N/A,false,N/A,N/A,N/A,N/A,false,N/
A,false,N/A
```

Parsing this CSV file is easy using e.g. Google Sheets, Numbers, or Excel. You can as well use the "Credential Report" tool from IAM in order to download the same CSV report. This exhaustive list will allow you to warn non-conforming users with a strict deadline.

Rotate your AWS keys

In a typical production environment, AWS keys get spread across various services, which have various privilege needs. Many employees will have access to multiple keys — DevOps will have access to most keys, DBAs will have access to the database keys, backend team to the log keys...

It is often necessary to renew these keys (e.g. when someone leaves a team). This procedure should be straightforward and risk-free, so that you can do it frequently, and more importantly, in urgent situations.

You can list the active access keys for a given user in this way:

```
$ aws iam list-access-keys --user-name
Elliot --query 'AccessKeyMetadata[?
Status==`Active`]'
{
      "AccessKeyMetadata": [
          {
              "CreateDate":
"2015-10-12T16:52:19Z",
              "UserName": "Elliot",
              "Status": "Active",
              "AccessKeyId":
"AKXXXXXXXXXXXX"
          },
          {
              "CreateDate":
"2016-12-06T19:17:25Z",
              "UserName": "Elliot",
              "Status": "Active",
              "AccessKeyId":
"AKZZZZZZZZZZZZZZZZ"
      }
    ]
}
```

This will help you distinguish which keys are used, and which are not. Keeping your set of keys as reduced as possible will help you managing these critical secrets!

Do not commit AWS access keys or credentials

AWS access keys are meant to be used by your infrastructure and/or your code. Do not commit them into your source code. It would else make them available to a lot of 3rd parties, such as contractors or continuous integration tools. It will also make them very difficult to change. A good way to approach this is to use environment variables. It would also allow you to easily run your code in a non-production environment. These ideas are described in The Twelve-actor App.

You can use a script to list all of the access keys configured in your AWS account and look for them in your applications' code.

```
$ aws iam list-access-keys --user-name
Elliot --query
'AccessKeyMetadata[].AccessKeyId'
[ "AKXXXXXXXXXXX", "AKZZZZZZZZZZZZZZZZ
Z" ]
```

You can save these key IDs for searching in your source code.

```
$ aws iam list-access-keys --user-name
Elliot --query
'AccessKeyMetadata[].AccessKeyId' | jq
-r '.[]' > my-keys.txt $ grep -f my-
keys.txt -r source_code1 source_code2
…
```

All found keys should rather be read from the environment. This requires updating your deploy process and maybe your infrastructure. Once the new deploy mechanism is working, make sure your source code does not keep any hard-coded keys. Then, all the keys need to be rotated: generate new ones, and replace the old with the new ones.

AWS offers Parameter Store for this purpose. Else, systems to store secrets can vary from environment variables in your Jenkins, to dedicated servers such as Vault.

Keep your security groups minimal

By default, any AWS element has an empty security policy, meaning that nothing is allowed to access it. You should only give access to the IPs and ports that are really needed for the service, and block all the rest.

You can list the security groups that do not limit IP addresses connecting to them using this script:

```
sgs=$(aws ec2 describe-security-groups
--filters "Name=ip-
permission.cidr,Values=0.0.0.0/0" --
query "SecurityGroups[].[GroupId,
```

```
GroupName]" --output text)
while read -r line; do
 sgid=$(echo $line | awk '{print
$1;}')
 sgname=$(echo $line | awk '{print
$2;}')
 c=$(aws ec2 describe-network-
interfaces --filters "Name=group-
id,Values=$sgid" --query
"length(NetworkInterfaces)" --output
text)
 echo "$sgid,$c,$sgname"
done <<< "$sgs"
```

The result will look like this: group-id, number of assignations, group name:

```
sg-7xxxxx,11,https_everywhere
sg-7xxxxx,2,http_everywhere
sg-7xxxxx,0,demo
```

If the number of assignations is > 0, then this group is used. If its name does not explicitly state that it is meant to be public, you should check it: EC2 -> Network & Security -> Security Groups and make sure this group is legitimate.

In this example, the https_everywhere and http_everywhere are legitimate: they are operating load balancers.

Use a private VPC

AWS makes it very easy to configure the networks. To make the most of it, your VPC should be private, and all the instances in your VPC should have an internal IP address.

It means your machines will have private IP addresses, preventing by default connections to the internet and from being accessible from the Internet.

If external internet access is required on your machines, they should use an AWS NAT gateway as their only way to access the Internet

If you need to grant public access to these machines from the outside, use an Elastic Load Balancer, or an Application Load Balancer. They are the AWS dedicated elements that allow you to easily operate and scale public accesses.

For internal access (e.g. a microservice), it is better to create an internal Load Balancer (that will be restricted to your VPC) in order to decouple the network configuration of this specific machine from the configuration of its clients.

The following command will display the list of public IP addresses that are used amongst your EC2 instances. If your instances are using internal IP addresses, only your NAT gateway should appear here.

```
$ aws ec2 describe-instances --query
"Reservations[*].Instances[*].
{ip:PublicIpAddress,id:InstanceId}"
{
    [
        {
            "ip": "34.xx.xx.xx.xx",
            "id": "i-yyyyyyyyyyy"
        }
```

```
    ]
}
```

You can now restrict this list to only display public IP addresses, along with the associated EC2 instance ID:

```
$ aws ec2 describe-instances --query
"Reservations[*].Instances[*].
{ip:PublicIpAddress,id:InstanceId}"
jq -c '.[]|.[]|select(.ip)'
{"ip": "1.2.3.4","id": "i-
xxxxx"} {"ip": "2.3.4.5","id": "i-
yyyyy"}
```

All of the displayed machines will have a public IP address. If you do not publicly expose these machines (but rather rely e.g. on a load balancer), there is no reason to have this. If one machine is only publicly accessed by a load balancer, then this machine should be on a private VPC, and the the load balancer should access it through this VPC.

Use Trusted advisor

AWS Trusted Advisor is a great way to retrieve many details about the security of your AWS setup. It also allows you to monitor billing or performance.

The free version of Trusted advisor will only tell you about the Security Groups with unrestricted ports, though the paying version has much more information available. The paid version will tell you about logging, your SSL certificates, exposed IAM keys and key rotation... The price is high, up to 10% of your infrastructure price. Only 4 checks are available by default, then you need to purchase Business support (100$ / month) to access all of them.

Enable CloudTrail

AWS CloudTrail is a logger that will record all the calls performed to AWS APIs with credentials that you own. All this information can be stored to S3 for further analysis (allowing low-cost retention). It is not a prevention against security incidents, though it is a way to be able to analyze what happened on your infrastructure in case of an incident, and examine which services were accessed.

In order to check the trails configured in your infrastructure, list the trails available:

```
$ aws cloudtrail describe-trails
{
    "trailList": [
        {
            "Name": "my-trail",
"LogFileValidationEnabled": true,
            "IsMultiRegionTrail":
true,
            "IncludeGlobalServiceEvent
s": true,
            "TrailARN":
"arn:aws:cloudtrail:eu-
west-1:0000000000:trail/my-trail",
            "HasCustomEventSelectors":
```

```
false,
            "S3BucketName": "mytrail",
            "HomeRegion": "eu-west-1"
        }
    ]
}
```

And check their status:

```
$ aws cloudtrail get-trail-status --
name my-trail
{
"LatestNotificationAttemptSucceeded":
"",
    "TimeLoggingStarted":
"2015-03-07T22:58:34Z",
    "LatestDigestDeliveryTime":
1497546412.316,
    "TimeLoggingStopped": "",
    "LatestDeliveryAttemptTime":
"2017-06-15T17:04:32Z",
    "LatestNotificationAttemptTime":
"",
    "IsLogging": true,
    "StartLoggingTime": 1283019477,
    "LatestDeliveryAttemptSucceeded":
"2017-06-15T17:04:32Z",
    "LatestDeliveryTime":
1497546272.808
}
```

You can see last time an event was logged with this trail. The same is available from the AWS console:

Access AWS CloudTrail now.

Update your Amazon Machine Images (AMI)

Just like any other piece of software, the OS needs to be upgraded to prevent security issues. Some of them should be corrected as soon as possible — even though some previous steps, such as reducing network exposure, can mitigate some of the issues created by an out-of-date OS.

In AWS, the OS is managed with the AMIs. You should ensure your AMIs are kept up to date.

When an AMI starts, it will by default download and apply the latest security patches. AWS keeps a very up to date list of the security issues corrected in the AWS instances.

AMIs can be displayed in the AWS Web console Pay attention when choosing the region were your EC2 instances are placed.

Choose your rights carefully

Just like your network, the other AWS services start with a zero-rights policy (nothing is allowed by default). So allowing certain entities to use this service is part of the service configuration. This configuration need to be tight, and must not contain any unnecessary privileges.

Some tools that are part of AWS IAM can help perform simulations of the rights you are building. The “Access Advisor” in IAM will help you fine tune the rights associated to the roles you create.

Monitor billing

Billing is not directly security related, though it can be an excellent indicator that something went wrong, or that your credentials have been used by a third party. It is not rare to see companies with dozens of new machines started to relay traffic or even mine cryptocurrencies (such as Bitcoin).

Billing information can be accessed from the AWS dashboard. Billing alarms can also be created in order to monitor this.

Going further

These AWS-authored articles will help you review what you have done in your organization and how you should improve your infrastructure:

AWS Auditing Security Checklist

AWS Security Best Practices

Don't forget, your infrastructure is only one piece of your company's security!

Check out Sqreen to learn how we can help you protect your apps deployed on AWS. Integrate continuous security in your infra. Monitor and protect your apps.