AWS Security Maturity Roadmap

Written by Scott Piper <scott@summitroute.com>

SummitRoute.com

April 2019

Introduction

Who am I and who is this for

I've worked as an independent AWS security consultant for the past two years for my own company Summit Route[1] helping over a dozen companies, ranging from small start-ups with barely a footprint in AWS to Fortune 100s with hundreds of AWS accounts, from companies that did a lift and shift of large numbers of EC2s and have large footprints in datacenters to companies who exclusively use serverless, from companies that have been in AWS for a decade when it was first coming out to those just getting started with AWS.

I've performed security assessments, consulted on best practices, provided training, written custom code, performed research, and more. I developed the popular (and free) AWS security CTFs flaws.cloud[2] and flaws2.cloud[3], and developed the open-source tools CloudMapper[4] and CloudTracker[5] with Duo Security.

I have no affiliation with Amazon, but I only do work related to AWS security. This is my opinionated, actionable, guide to using AWS securely in 2019.

Keeping up-to-date

Some of what I write here may unfortunately become out-of-date. To stay up with what is happening in AWS security, I recommend the following:

- ☐ Subscribe to the weekly newsletter https://lastweekinaws.com/ by Corey Quinn.
- ☐ Subscribe to Corey's podcast as well https://www.screaminginthecloud.com/
- ☐ Read the weekly summary of general cloud news http://highscalability.com/, on twitter at @highscal.
- ☐ On Twitter, follow @jeffbarr (Chief evangelist for AWS), @AWSSecurityInfo (official AWS security twitter account), @StephenSchmidt (AWS CISO), @ben11kehoe (AWS hero), @esh (AWS hero), @jim_scharf (VP of AWS Identity), @QuinnyPig (aforementioned LastWeekInAWS author), @SummitRoute (my company), and @0xdabbad00 (my twitter account).
- ☐ Join the Slack og-aws[6] for general AWS discussions, and for me the most valuable part is the #announcements channel which automatically pulls in AWS announcements from RSS feeds.
- ☐ Join the Slack "awssecurityforum". There is no way to sign up, so ask myself or anyone else doing AWS security things for an invite.

Learning the foundations

This paper is mostly about current best practices, which requires you to know a lot of foundations of AWS and security in general.

The best book for learning about security concepts you'll likely use in your work around AWS is Securing DevOps: Security in the Cloud by Julien Vehent, who manages Firefox Operations Security.

If you want to get a cert in order to give yourself a goal to work toward, and proof that you have some knowledge in the area, the only cert I recommend getting is the AWS Security Specialty[7] certification. To prepare for the exam, you can take the acloud.guru[8] course and read the various white-papers from AWS along with the FAQs for security related AWS services. There are other AWS offered certs which you may also wish to get in order to learn the foundations of AWS.

[1] https://summitroute.com
[2] http://flaws.cloud
[3] http://flaws2.cloud
[4] https://github.com/duo-labs/cloudmapper
[5] https://github.com/duo-labs/cloudtracker
[6] https://og-aws-slack.lexikon.io/
[7] https://aws.amazon.com/certification/certified-security-specialty/
[8] https://acloud.guru/learn/aws-certified-security-specialty

Most common security incidents

In order to discuss the steps to secure AWS environments, you should be aware of what the common security incidents on AWS are so you can plan your defensive strategies accordingly. Generically, the most common AWS related incidents are:

1. Publicly accessible resources such as S3 buckets or ElasticSearch clusters.
2. Leaked access keys. For example, access keys posted to GitHub.
3. Compromised IAM Roles through SSRF or RCE against an EC2, resulting in access to the metadata service at 169.254.169.254.

There are many other issues possible, and many issues that are not AWS specific, but I believe extra effort should be spent in addressing these.

AWS Security Maturity Roadmap

In this section I've grouped security goals into stages that somewhat build on each other. These are the steps to take your company from having no cloud security program for AWS to having a solid program. You may find you do some of these steps in a different order, partially complete steps for some accounts before others, or have to go back and make slight adjustments, but this should help give you a roadmap for making progress.

Stage 1: Inventory

- ☐ Identify all AWS accounts in the company and their points of contact.
- ☐ Integrate AWS accounts into AWS Organizations.
- ☐ Have an AWS account for Security.

In this phase, you'll want to create a spreadsheet of all the AWS accounts you have (name and 12-digit account ID), their POC (point of contact), associated payer, etc. You can also use this spreadsheet to keep track of which accounts have met certain goals that will be discussed in this paper by including additional columns. See the post How to inventory AWS accounts[9] for more information on how to inventory your AWS accounts. You'll want to ask the different groups at your company what accounts they have, the finance team what accounts are being paid for, talk to your TAM if you have one to figure out what accounts have been registered under your company email, search your company emails, and possibly search network logs.

If your accounts are not already in an AWS Organization, you'll want to do so. The root for the AWS Organization should not have any AWS resources in it (ie, no EC2's, S3 buckets, etc.). By putting the accounts in an Organization you'll get consolidated billing, an easier ability to create new accounts, and ability to use SCPs (discussed later).

You should have an AWS account used by the Security team that will be used for log collection. If you don't already have lots of AWS accounts, recognize that at this point you now have a minimum of 3 AWS accounts (an account that has your business stuff in it, the AWS Organization root account with nothing in it, and the Security account).

Stage 2: Have backups

- ☐ Create regular backups.

One of the most notorious AWS security incidents was the destruction of a business named Code Spaces. An attacker gained access to their account and destroyed all of their data and backups, resulting in the business shutting down within 12 hours of the incident.

As you learn what data you have in AWS, you should ensure that data is backed up to a separate AWS account in a separate region from where the data comes from (or possibly even somewhere off of AWS). You can use the AWS Backup[10] service to do this along with S3 Object Lock[11]. Planning for Disaster Recovery involves a trade-off between how quickly you need to recover and how much money you're willing to spend, so you should consider using Glacier Deep Archive[12] for some data which is $1/TB/mo to store data, but will take up to 12 hours to retrieve.

[9] https://summitroute.com/blog/2018/06/18/how_to_inventory_aws_accounts/
[10] https://aws.amazon.com/backup/
[11] https://aws.amazon.com/about-aws/whats-new/2018/11/s3-object-lock/
[12] https://aws.amazon.com/about-aws/whats-new/2019/03/S3-glacier-deep-archive/

Stage 3: Visibility and initial remediation

- ☐ Turn on CloudTrail logs for all accounts to send their logs to the Security account.
- ☐ Create an IAM role in every account that grants view access into the account from the Security account.
- ☐ Run a one-time scanning tool to identify tactical remediations.
- ☐ Turn on S3 Public Block Access.
- ☐ Develop an account initialization script and new account creation process.

CloudTrail logs should be turned on via the Organization root via an organization trail[13]. By using the organization trail you not only have a centrally configured CloudTrail that is enabled by default for all newly created accounts in the Organization, but also these CloudTrail logs cannot be turned off by the child accounts. You'll create an S3 bucket in the Security account, and then from the Master account, enable the CloudTrail logs. You'll need to setup permissions on this bucket for your organization to send logs to it, and you'll also want to setup read-only privileges for the child accounts to be able to view their own logs.

Create an IAM role in every account that grants access for the Security team. You can make a copy of the CloudFormation template I use when I perform assessments (be sure to change the account ID!) and you'll need to ask people what the account ID is for the accounts they ran this in. See here[14] for how the URL is formed and the template file that allows people to easily grant you access.

Once you have access, you'll want to run a one-time scanning tools such as CloudMapper[15], Prowler[16], or NCC's ScoutSuite[17]. This will give you a feel for how much work lies ahead. These will help point out tactical issues that you can work on fixing now, but later phases in this maturity model will help strategically stop these issues from happening in the first place. For now, the main things you should worry about are any public S3 buckets with sensitive contents, public AWS managed ElasticSearch servers, or any EC2's running publicly accessible services that should be private, such as ElasticSearch again, databases, or other "soft" targets, which can largely be identified by the port numbers that have been made public.

The feature S3 Public Block Access[18] should be turned on in accounts which disables buckets and their objects from being made public. For cases where you do still need some S3 buckets public for some reason, this can be configured to ensure no new buckets are made public.

As you granted access to the accounts for the Security account, you should use this as an initialization script for future accounts so that you have a common, secure baseline, that new accounts you create, or old accounts that are newly acquired, can conform to. You should document a process for creating new accounts. What team should do that and for what needs? Where should the root password and MFA (Multi-Factor Authentication) be kept? What email address pattern should be used for the account? Who should be on the distribution list of that email address? What phone number and billing information do these accounts tie back into it?

[13] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/creating-trail-organization.html
[14] https://summitroute.com/aws_security_assessments/
[15] https://github.com/duo-labs/cloudmapper
[16] https://github.com/toniblyx/prowler
[17] https://github.com/nccgroup/ScoutSuite
[18] https://aws.amazon.com/blogs/aws/amazon-s3-block-public-access-another-layer-of-protection-for-your-accounts-and-buckets/

Stage 4: Detection

- ☐ Turn on GuardDuty in all active regions.
- ☐ Detect issues from logs and events in near real-time.
- ☐ Perform regular scanning of the accounts for security issues.
- ☐ Document your security guidelines for your company.

GuardDuty is a threat detection service from AWS for detecting compromised access keys, EC2 instances, and more. You should have it enabled in all active regions, for the reasons I describe in Should you use GuardDuty?[19] Once you have GuardDuty enabled you need to be able to collect the alerts from all of your accounts, in all active regions, and forward those alerts to your Security account and then to a place where you'll see them (Slack, PagerDuty, etc.). I describe one way of accomplishing this in GuardDuty Event Collection via CloudWatch Events[20]. Another method is to use GuardDuty's Master/Member relationships and have that tied into AWS's Security Hub service.

The method I described for collecting GuardDuty alerts via CloudWatch Events also collects CloudTrail events in real-time, allowing you to quickly respond to violations of best practices. As a simpler alternative, or in addition to the CloudWatch Event collection, you can also monitor the CloudTrail logs you had sent to the S3 bucket earlier. These are delayed by 15 minutes, but include some log events that the CloudWatch Events do not (ex. List and Describe calls), include events from all regions, and are easier to ship into SIEMs or other log monitoring and analysis platforms. If you do not give people access to your SIEM to analyze their own CloudTrail events, you should set up Athena tables for people to use to more easily query and analyze their events.

Some best practices can't be detected based purely off of a log event, for example detecting an IAM role that has been inactive for over 90 days. Your event detection solution may also break down at some point, or you'll want to add new rules that accounts might already have violated at some point in the past. For these reasons, you'll want something to perform regular scanning of your environments.

Now that you're detecting various issues, you'll want to document what those issues are so you're not just telling your team they did something wrong without them knowing the rules you want them to play by. Many of the vendor and open-source solutions in this space detect things that you will not care about. For example, you might have accepted the risk of things like a bastion host that allows SSH access globally. There are also things that out of the box these tools don't detect that are specific to your environment that you want done. For example, you might want all public network resources to only be in a few specific accounts and subnets, or that all S3 buckets that contain certain types of data should be replicated to another specific account and region to ensure Disaster Recovery and fail-over. You should work toward building these additional checks into your detection solution where possible.

Stage 5: Secure IAM access

- ☐ Use SSO for access.
- ☐ Remove all IAM users.
- ☐ Reduce the privileges of service roles to necessary services.
- ☐ Implement pre-commit hooks for secret detection.

IAM User Access Keys never expire. They regularly wind up in source code that finds it's way onto public GitHub repos and when configured to be used by the AWS CLI, they exist as plain-text in `~/.aws/credent`

[19] https://summitroute.com/blog/2019/03/05/should_you_use_guardduty/
[20] https://summitroute.com/blog/2019/03/06/guardduty_event_collection_via_cloudwatch_events/

`ials`, which poses risks. For these reasons, you should avoid IAM Users entirely and instead use IAM Roles. This isn't always possible for some older solutions or solutions that run on-prem outside of AWS.

For human users, you can use SSO for access. This gives you a central location for creating and removing users, or rolling their credentials if they are compromised. You can either have a single "Identity" AWS account that users log into via SSO and then assume into roles from there to the different accounts, or you can have them SSO directly into the different AWS accounts they have access to. When working from the command-line with SSO, aws-okta[21] can be used for Okta, and other solutions may exist for other providers.

Your SSO solution should be enforcing strong password policies and MFA access, but if you do have any human IAM users, you should enforce these restrictions in your AWS account of having a strong password policy and apply the policy at Self-Manage an MFA Device[22] to these users which enforces MFA even on access key usage. They should also be using aws-vault[23].

For services that don't run on AWS, you may be able to use AdRoll's hologram[24] which uses LDAP, or Hashicorp Vault where the on-prem server authenticates to the Vault to get AWS session keys, but these will likely require code changes.

You should audit your IAM Roles to identify ones that have not been used for a certain amount of time and use Access Advisor to reduce the privileges of the IAM roles to only those services they use. Ideally, you should reduce this further to only the necessary Actions and Resources, but that is a harder problem, we'll aim for later.

You should implement a solution to detect secrets being committed to source-code repos to avoid having IAM User access keys added to public, or even private, repos. One solution is Yelp's detect-secrets[25] project.

Stage 6: Network attack surface reduction

- ☐ Have no publicly facing EC2s or S3 buckets.
- ☐ Move all non-public network resources into private subnets and proxy outbound requests so you can filter and block them.
- ☐ Restrict Security Groups.

Resources such as EC2s and S3 buckets should not be publicly accessible, but instead should have an ELB or CloudFront in front them. This has a number of benefits including being able to make use of AWS Shield (DDoS protection), AWS WAF, and a more standardized means of logging access. This has operations improvements such as an ability to scale and reduced latency. If you do have ports open, such as SSH, this also ensures a level of abstraction between the public web service and the public SSH service to mitigate the likelihood of a targeted attack there. CloudMapper's `report` and `public` commands will show you what public resources you have. You should also look into using SSM Session Manager[26] instead of SSH as it removes the need to manage SSH keys and provides better auditing.

Once you have an ELB in front of your EC2s, you can then move those EC2s into private subnets, which helps reduce the possibility of other ports being exposed on the instance, and also makes it easier to restrict outbound network requests, to be done later.

As you focus on these network goals, you should work toward avoiding Security Groups with CIDR's and instead use named Security Groups to avoid having the hard exterior network surface and soft center by avoiding having something like `10.0.0.0/8` open on your network resources.

[21] https://github.com/segmentio/aws-okta
[22] https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_policies_examples_iam_mfa-selfmanage.html
[23] https://github.com/99designs/aws-vault
[24] https://github.com/AdRoll/hologram
[25] https://github.com/Yelp/detect-secrets
[26] https://aws.amazon.com/blogs/aws/new-session-manager/

Stage 7: Reproducibility and supply chain management

- ☐ Control AMI and package sourcing.
- ☐ Use Infrastructure as code.

Not all of the public AMIs (Amazon Machine Images) used to create EC2s are vetted by AWS, and there is not an easy way to identify the trust-worthiness of the source account ID that an AMI comes from. There have been issues[27] in the past of public AMIs with malware in them. You should identify what AMIs you want to be allowed to run in your environment. You should also document guidelines around the instance types to be used and the OS's, so you don't end up with a a dozen flavors of Linux running in your accounts without good reason.

One strategy you should consider is building all of your AMIs yourself, possibly with all of their code and packages pre-configured. This would mean that every code release would be a pre-built AMI, which is the strategy used by Netflix in their post How We Build Code at Netflix. Alternatively, you can deploy code and updates using something like salt, ansible, puppet, chef, or another solution.

You should control the packages (code and software) that go into your code bases. Your production servers should not be reaching out to various GitHub repos for updates. You should have a CI/CD pipeline that builds things for you in a reproducible way and consider having mirrors of package repos or libraries.

In addition to better controlling how your software is built, you should also control how changes to your AWS environment happen. Changes should only be made through Infrastructure as Code (IaC), such as Terraform or CloudFormation. This has a number of benefits such as ensuring you have a reproducible environment and can better audit changes. This is often done gradually, beginning with the VPC, Security Group, and some IAM configurations, and eventually encompassing all resources. Once you control infrastructure changes through code, you can also use two-man rule deployments, where one person proposes the changes and another signs off in order for the change to be implemented.

Stage 8: Enforce protections

- ☐ Apply SCP restrictions.
- ☐ Automated remediation.
- ☐ Refine IAM policies.

SCPs (Service Control Policies) now have the ability to use Resources and Conditions, which opens up the possibly of restricting what regions can be used, denying the ability to remove security related IAM roles or CloudWatch Event collection configurations, requiring encryption on S3 buckets, and more. For examples see the Example Service Control Policies[28]. These ensure that admins and even root users are restricted.

In cases where actions cannot be restricted by SCPs, you can setup automated remediations to correct issues. This can mean automatically removing inactive IAM Roles after some time period, removing their unused privileges via something like RepoKid[29], making network resources private that have been made public that shouldn't be, and more.

Review and refine your IAM policies to better restrict them. Your policies should have specific Actions, Resources, and include Conditions. You should avoid using AWS Managed IAM policies because they are not restricted enough.

[27] https://summitroute.com/blog/2018/09/24/investigating_malicious_amis/
[28] https://docs.aws.amazon.com/organizations/latest/userguide/orgs_manage_policies_example-scps.html
[29] https://github.com/Netflix/repokid

Stage 9: Advanced defense

- ☐ Restrict metadata service access.
- ☐ Restrict egress network traffic.
- ☐ Deploy honey tokens.

If an attacker manages to get SSRF against an EC2, they can use this to access the metadata service at 169.254.169.254 where they can take over the IAM Role for the service. Will Bengston of Netflix describes solutions to this in his Preventing Credential Compromise in AWS[30] article and associated project AWS Metadata Proxy[31]. He shows how to restrict the IP range that credentials can be used from and how to proxy the metadata service to use a user-agent restriction capability to limit SSRF from accessing the service in the first place.

As an additional network precaution, you can use AWS PrivateLink to restrict access to resources to only those coming from certain VPCs. You may need to have a proxy to access third-party vendors, but you should be able to restrict egress network traffic so nothing calls out to the Internet, or does so only through a proxy. This is not only applicable to EC2s, but also Lambdas and containers.

Use "fake" access keys as honey tokens for an attacker to find and use so you can more easily detect them. I describe how to do this in Guidance on deploying honey tokens[32].

Stage 10: Incident preparation

- ☐ Limit the blast radius of incidents.
- ☐ Practice responding to incidents.

Consider if it makes sense to break up accounts further. Ensure there is a separation of duties so the security team can monitor the production accounts, and those with production access, cannot impact the security team. Consider breaking up services into pieces so that if one is compromised the blast radius is more limited.

If you're lucky enough to not have any security incidents, you should practice what happens when systems or IAM roles are compromised. How long does it take you to identify, understand, and react? Use the lessons learned to improve.

Conclusion

I work as an independent AWS security consultant. If you're looking for help implementing the above concepts or anything else with your AWS security, reach out to me at scott@summitroute.com or read more at summitroute.com

[30] https://medium.com/netflix-techblog/netflix-information-security-preventing-credential-compromise-in-aws-41b112c15179
[31] https://github.com/Netflix-Skunkworks/aws-metadata-proxy
[32] https://summitroute.com/blog/2018/06/22/guidance_on_deploying_honey_tokens/