509.3

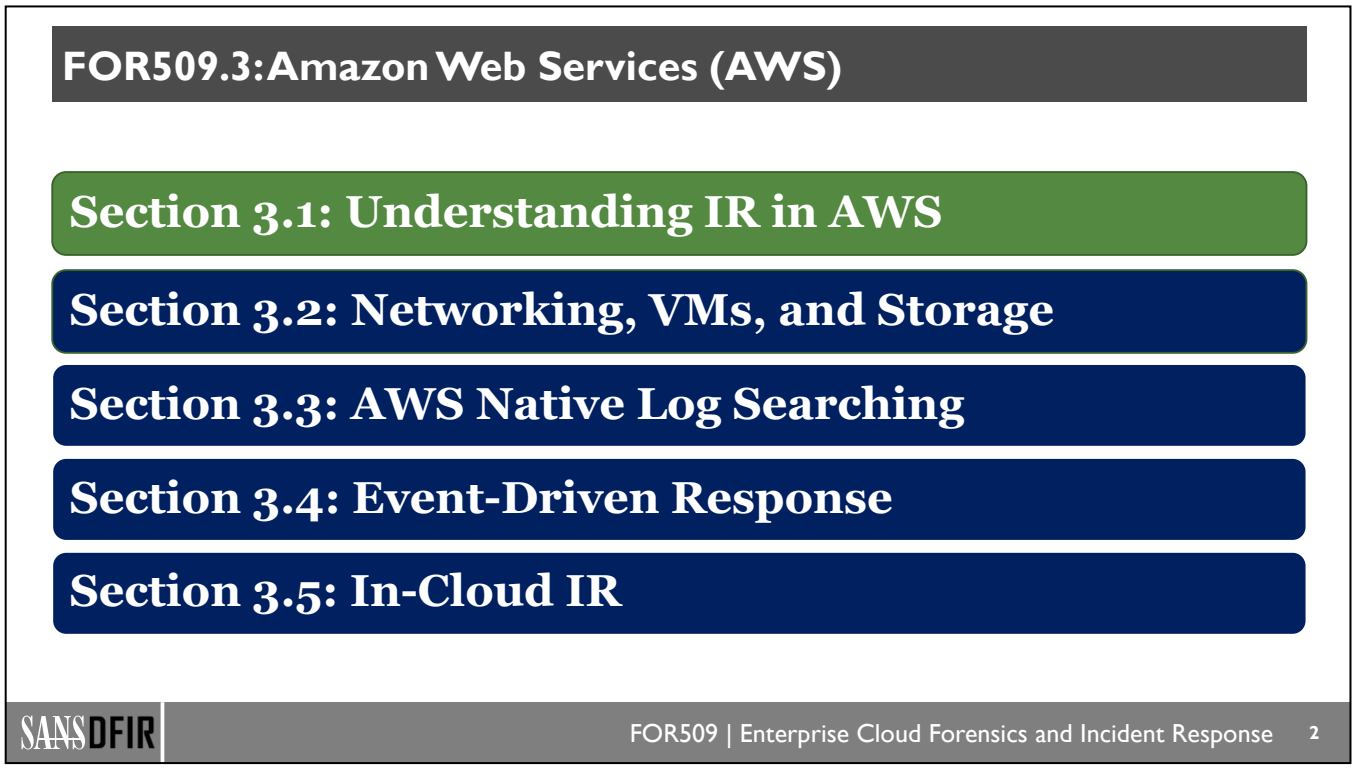
Amazon Web Services (AWS)

All reference links are operational in the browser-based delivery of the electronic workbook.

FOR509.3 | Enterprise Cloud Forensics and Incident Response

Amazon Web Services (AWS)

Author:
David Cowen: dlcowen@gmail.com
https://twitter.com/HECFBlog
https://www.hecfblog.com/



Here are the five sections for this portion of the course.

So, we're going to start with:
Section 3.1: A little bit of AWS so you can speak the vernacular for those who are not as fluent. Every cloud uses a different way to describe its services, as they all have different meanings. So, we're going to talk about that, and then we'll go into **CloudTrail.**

Section 3.2: Then we move into talking about **virtual machines**, as well as compute resources and **storage.** We're going to talk about **snapshots**, Lastly, we'll be talking about **flow logs.**

Section 3.3: Next, we will talk about additional places to get logs and work with logs, how to search for logs within AWS, and some of the tooling that AWS provides.

Section 3.4: We're going to talk about automation, something AWS calls event-driven response. It's the idea that we can have automated actions kick off based on events that AWS records. AWS has produced a white paper called "Event-Driven Response" that covers how you can use different triggers within AWS to fire off functions.

Section 3.5: This portion of the course ends with what you need to do to be able to perform some effective cloud IR. It concludes with a discussion about the good ways we would recommend doing it and things to avoid.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- AWS Organizations
- AWS Organizations for IR
- IAM
- IAM Methods of Access
- AWS Shared Responsibility Model
- CloudTrail
- CloudTrail Insights
- CloudTrail Hunting
- **Lab 3.1**: Reviewing CloudTrail Logs

This page intentionally left blank.

FOR509.3: Amazon Web Services (AWS)

English-to-AWS translation

A quick language primer so you can speak AWS natively

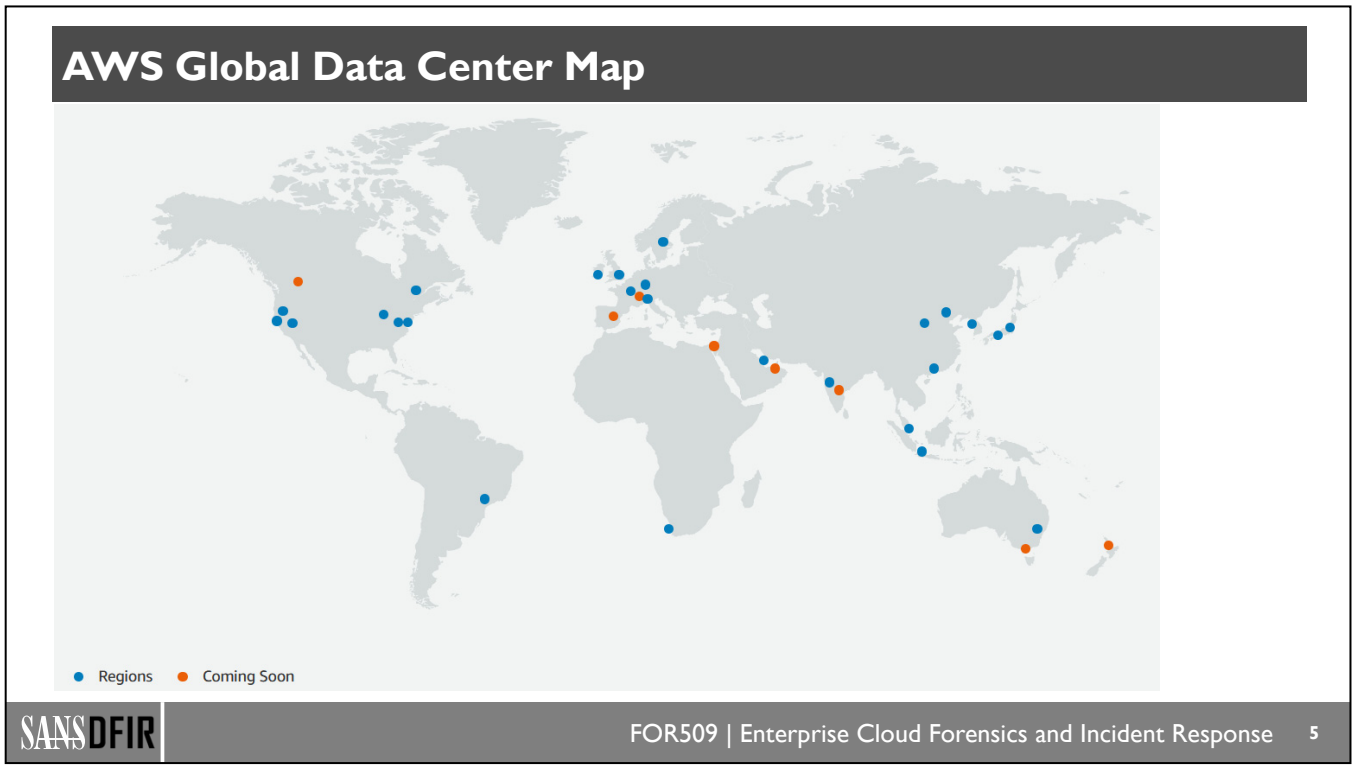
This section will introduce you to the language of AWS, the major services that are involved with most environments, as well as the forensic data we can expect to receive from them. This course is not meant to teach you how to configure, maintain, or deploy AWS services but how to investigate and respond to breaches within them.

When you're talking to AWS professionals, having the ability to speak their language will go a long way in their ability to assist you in your investigation.

For instance, here are several questions that result in good forensic data:

"Where do you store CloudTrail logs?"
"Do you have flow logs for this VPC?"
"Can this AKS cluster reach the metadata service?"

Don't understand what all these things mean? No problem. That's what this section will help you understand, while making sure those in the AWS world can understand you!



For each one of the cloud providers, we have these maps to show globally where all the data centers are.

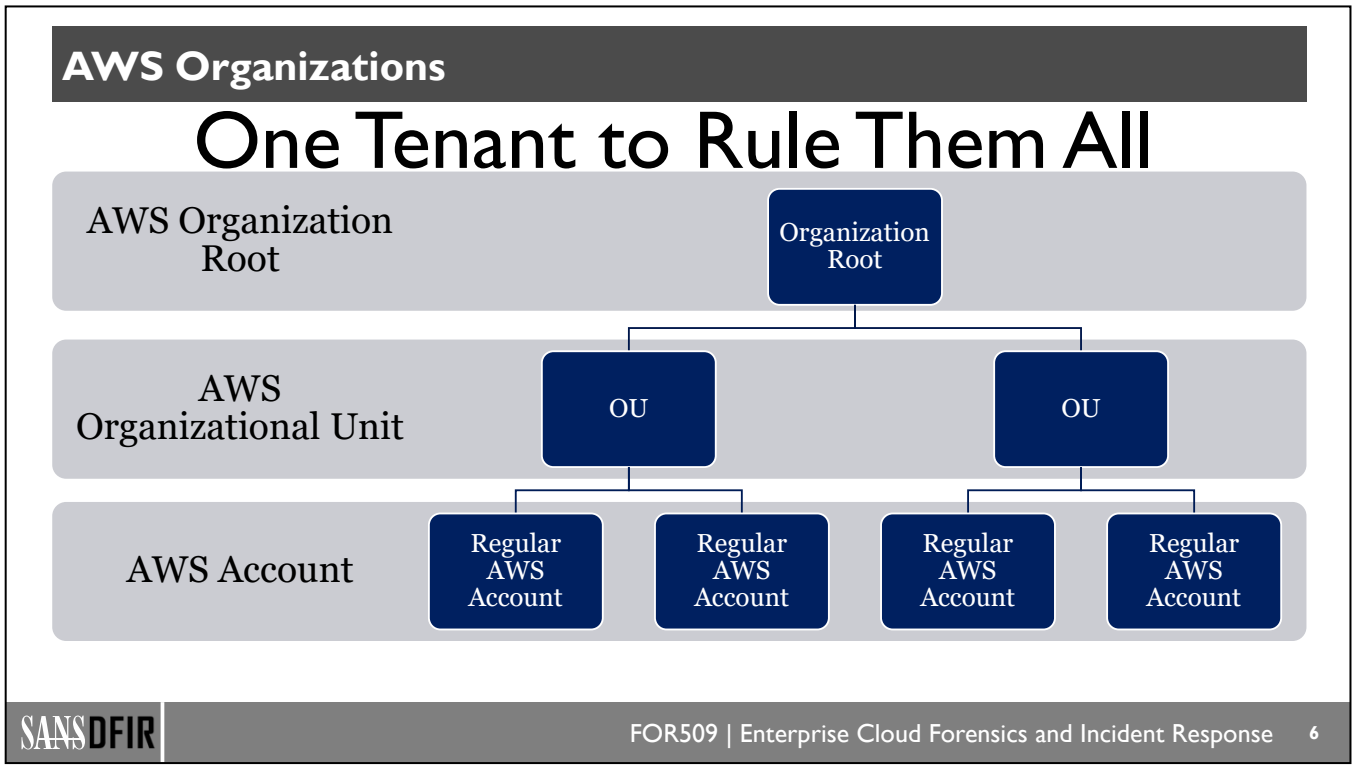
What's interesting here is you can stage your resources to be geocentric to just where the data or users are. In our case, we can locate our investigative resources to where the incidents are for faster access and cheaper data transfer.

There are large amounts of gray, where there are no data centers, but at least they're on the same physical content, which means they have the most optimal speeds to transfer data within that region.

Think of the regions in three ways:

1. If you are triaging an on-premises incident in the cloud, you want to pick a data center closest to where the data is being transferred from. This is easy to do with S3 Transfer Acceleration and something that really will increase the speed and reliability of these collections.
2. If you are doing in-cloud investigations, there are cost and speed considerations when pulling data between regions. Having your lab infrastructure as something you can stand up with terraform or AWS cloud formation means you can move your investigation closer to the data.
3. Many services are region specific, while few are global. If you are in the wrong region within your scoping, you may miss incidents or evidence entirely.

Credit: Amazon Web Services (AWS)
Picture as of 5/3/22
Source: https://aws.amazon.com/about-aws/global-infrastructure/



AWS organizations are a way to group all your companies AWS accounts into one tree so that you can:

- Manage access between AWS accounts
- Establish single sign-on across your AWS accounts
- Recognize cost savings by allowing your usage to be accumulated across all tenants for volume discounts

You can learn more about them here: https://aws.amazon.com/organizations/

So, the first thing in AWS that we need to be able to talk about is called organizations. Organizations in AWS are a way to group accounts together in kind of an Active Directory forest, if you're familiar with Active Directory. Basically, they're a way to chain together your accounts. And I think, in this way, AWS is a little different. For instance, we are talking about Azure in the next section.

Azure has a way to logically group accounts, but not in the same way that AWS does. For example, each account in Azure has an IAM provider and maybe even some cross-tenant trust. Google also has the ability to do this, but AWS seems to be the most mature of the offerings in this regard.

The very first thing you have to do if you want to be able to use something like an organization is to validate you know all the accounts your enterprise has. Once you've gotten all the account owners to join the org then AWS organizations allows you to have some common call controls and policies and compliance enforcement and cross-tenant roles. So, with AWS orgs you can have an IM role for IR that lets you have read-only access to every resource in all the accounts that have joined.

You're going to have to set up one account to basically be the organizational root. This is going to be a special account that very few people should have the ability to log in to. And maybe you have one or two accounts that can log in to it. Inside of those accounts, you'll be able to basically have a root identity and access master.

For every single AWS account, there is a root account that has full super-admin control over that whole AWS account that is not in any way, shape, or form managed within IAM, and when you log in, it tells you all sorts of things. You shouldn't log in with your root account except when necessary.

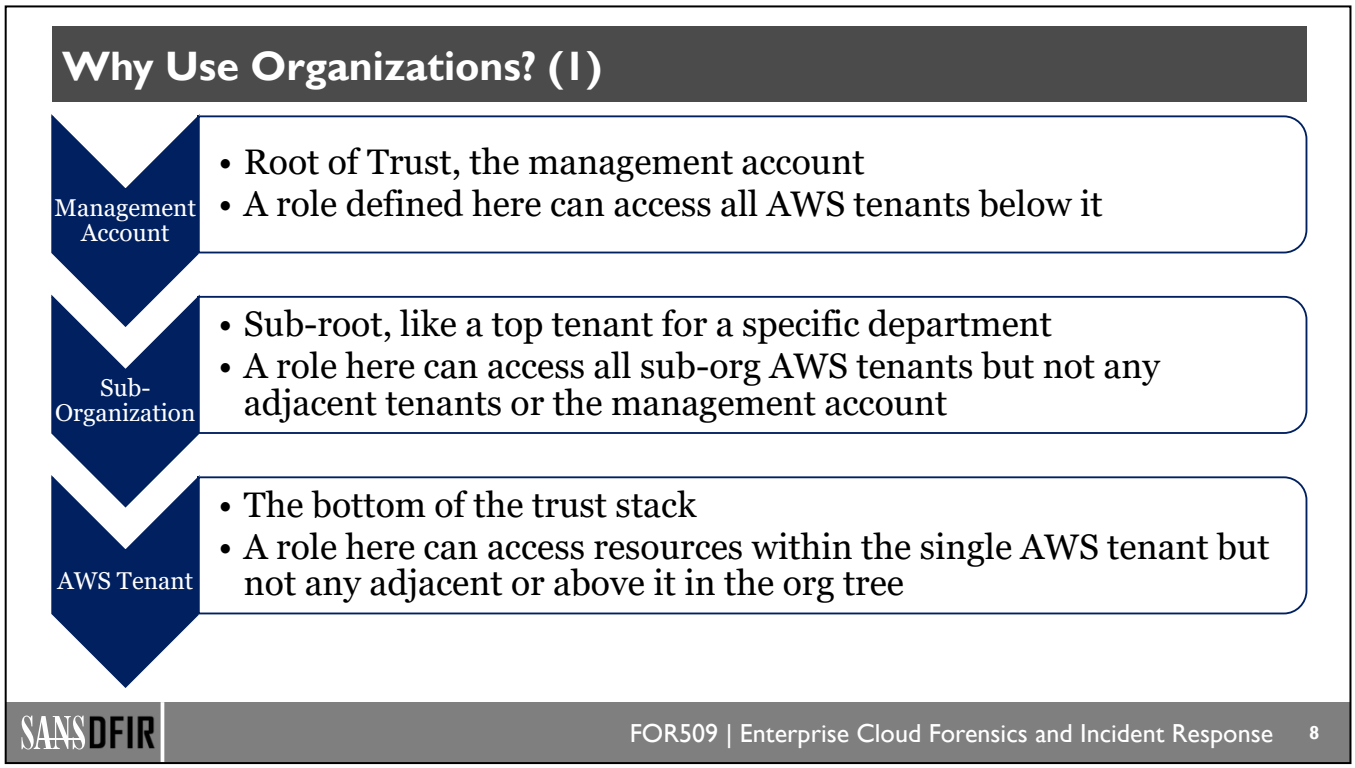
All of the AWS accounts belong to the same organization, and you can organize these into sub-organizations so that you can have different policies for compliance. The way we often see this happen is there will be one sub-organization for production, another sub-organization for development, and another sub-organization for QA. Three different sub-organizations, each with different policies, enforcement, and compliance. In this way, they can have cross-sub-org roles with each other—unless, of course, someone messes it up with over-provisioning, which is a real problem.

When you're talking about AWS organizations and cross-sub-org roles, an IAM role that is not properly scoped could be accidentally giving your development role access to your production environment.

As an IR responder, you want your company to make a role for you, or your resources. With a root org role, you can have full read-only access to every resource in your entire organization.

We would recommend against a generic security account and encourage you to have a security role attached to the individual accounts of the responders. You still want to have accountability, even on the IR team, of who's accessing what.

You can read more about organizations here: https://aws.amazon.com/organizations/.



In thinking about the scope of access of an account or role, think about where it exists within the organization.

Roles/accounts that are defined within the management account or root organizational account can be applied to all members of the org. These are the most powerful roles that can be assigned and should be tightly controlled. The idea here is that if you have something like an IR role within your organization, you can have all the resources available as a read-only evidence source whenever it's needed.

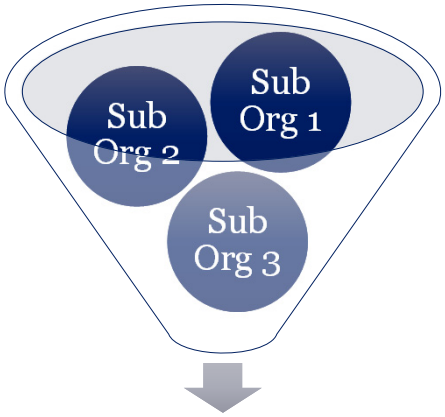
At the sub-organizational level, you are defining roles that apply to all of the accounts that are assigned beneath it. In the previous page's notes, we discussed typical sub-orgs like a development sub-org that gives access to developers to create, test, and stage resources without affecting production resources. This would allow you to set different levels of compliance rules that vary on the use of the resources. A service like AWS Audit Manager[1] can help to bring that level of compliance to various standards. AWS Config[2] would allow you to set custom compliance metrics and force compliance on an organizational, sub-org, or individual account basis. Remember that a role in a sub-org has the ability to access all resources within that sub-org, but not the org above it or other sub-orgs, without specific cross-role definitions.

On the individual account or AWS tenant level, your roles only apply to resources created within the single account. These are the lowest-risk roles to work with but also become an easy place for things to go wrong if you don't have a good structure to assess and require compliance with security standards.

1. https://aws.amazon.com/audit-manager/
2. https://aws.amazon.com/config/

Why Use Organizations? (2)

- Consolidated Billing
  - Discounts for bulk usage
  - AWS provides discounts the more of it you use
  - Consolidated billing allows all of the tenants in your organization to add up all their usage for a larger discount



Consolidated Billing

What if you're working for a company right now that is not using organizations and there's pushback because people are looking for ways to avoid being forced into compliance?

You need a compelling reason to override that argument. In order to override arguments and business complaints, its often useful to point toward cost and ways to save money called consolidated billing.[1] Amazon gives you a discount the more of AWS you use. Without an organization, the different sub-accounts that are driving the costs will get their bills and usage siloed into their own accounts. With organizations, AWS will combine the overall company's AWS usage, allowing for a much more substantial discount.

1. https://docs.aws.amazon.com/awsaccountbilling/latest/aboutv2/consolidated-billing.html

IR Roles in Organizations

Automatic Write Blocking!

Read-Only Snapshots
Read-only access to all evidence sources

You can define read-only global access too:

S3 Buckets
EC2 Virtual Machines
EBS Data Stores
Databases
Containers
And More!

Centralized Logging

All of your AWS accounts log stored in one S3 Bucket!

Cross Org Account Access

Create an IR org that has access to your production org

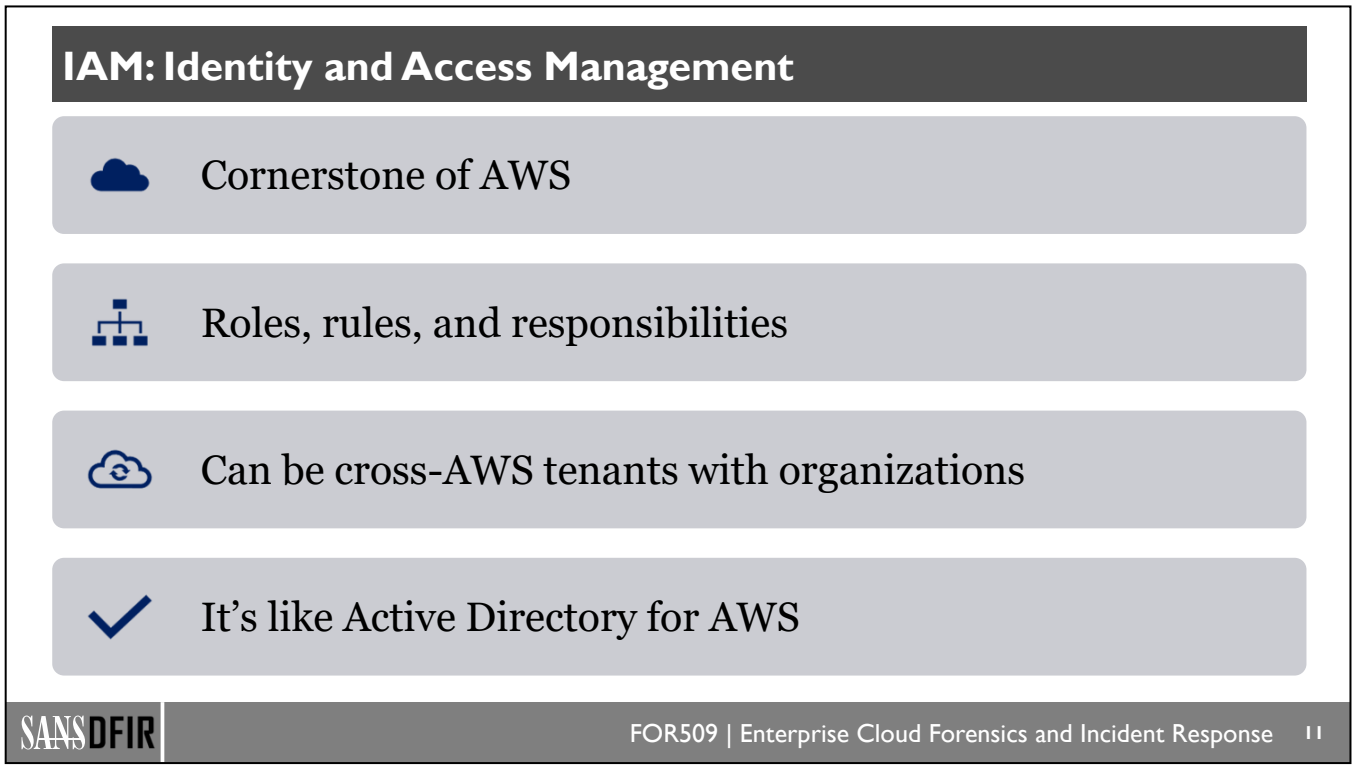
SANS DFIR | FOR509 | Enterprise Cloud Forensics and Incident Response 10

What does an IR team want from an AWS organization?

Mainly the ability to have an org-spanning IAM role with read-only access to the org resources. You can still have an IR sub-org VPC network you can pull back evidence to. You can also create org-wide automated alerts to automatically respond to all of your org's AWS services that are being hosted.

That includes S3 buckets, EC2, EBS data stores, databases, containers, and anything else that AWS creates in the future.

There is an argument to be made to put the IR group inside of a separate org to prevent a breach of an org leading to a breach of IR resources, in which case you can use cross-org roles and trusts. You can do this by adding a role that ties to the ARN (Amazon Resource Name) of the IR account in the separate AWS org for IR or by creating a SAML token, which we'll discuss on the next slide.



In every cloud provider, including Azure and GCP, IAM[1] is the cornerstone of everything.

Without it, you can't have the ability to define roles, policies, and access.

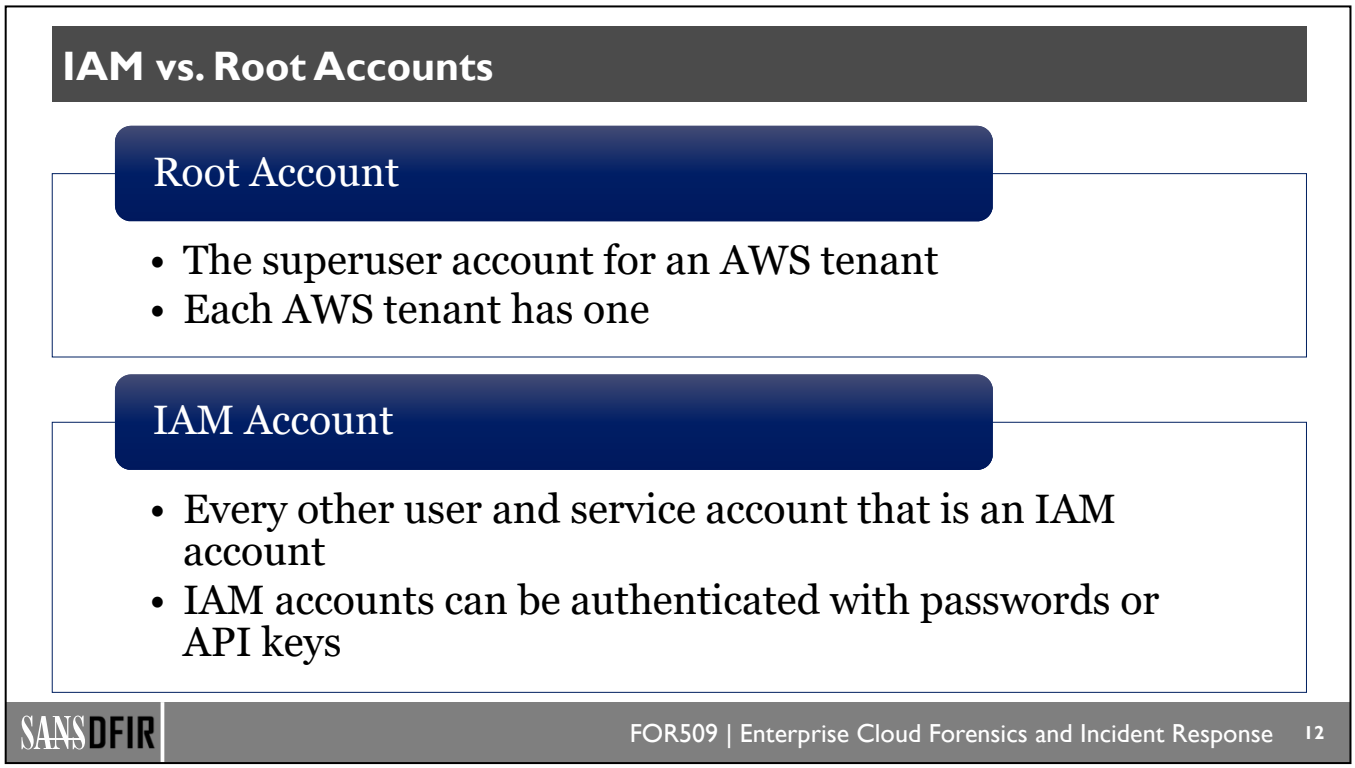
Roles: The individual groupings of policies that can be applied to an account
Rules: The compliance framework you establish that forces a certain level of security within your org resources. For instance, making VPC flow logs mandatory!
Responsibilities: This is the concept of placing the burden of compliance on the individual user but having the automated compliance framework to make sure you can ensure they follow it.

Cross-Tenant / Cross-Org Users: Every AWS user is given a unique ARN (Amazon Resource Name). As an example, if your account had the AWS ID of 12346789012 and you created an IAM user named JohnDoe, then their ARN would be:
arn:aws:iam::123456789012:user/JohnDoe

These ARNs are unique per account, as the number (123456789012) will change for every AWS account created. This means that from one account, sub-org, or org, you can make reference to an external ARN and provide access to your org's resources to it. In this way, you can establish a separate security or IR account that prevents a breach of your org from exposing your investigation.

IAM for UNIX users is NIS; for Windows users it's Active Directory. Whatever analogy works for you, it's the central place where credentials, roles, and policies are stored and applied for access.

1. https://aws.amazon.com/iam/



There is one root account per AWS account that is just called Root.

Root is the super account for every AWS tenant. AWS does not recommend using the Root account unless necessary.

Root accounts always have all roles and privileges for their AWS account.

An IAM account can be created for any user or service that needs one.

Each IAM account has a password, API key, or both!

IAM accounts have roles and privileges assigned to them.

IAM Roles and Policies

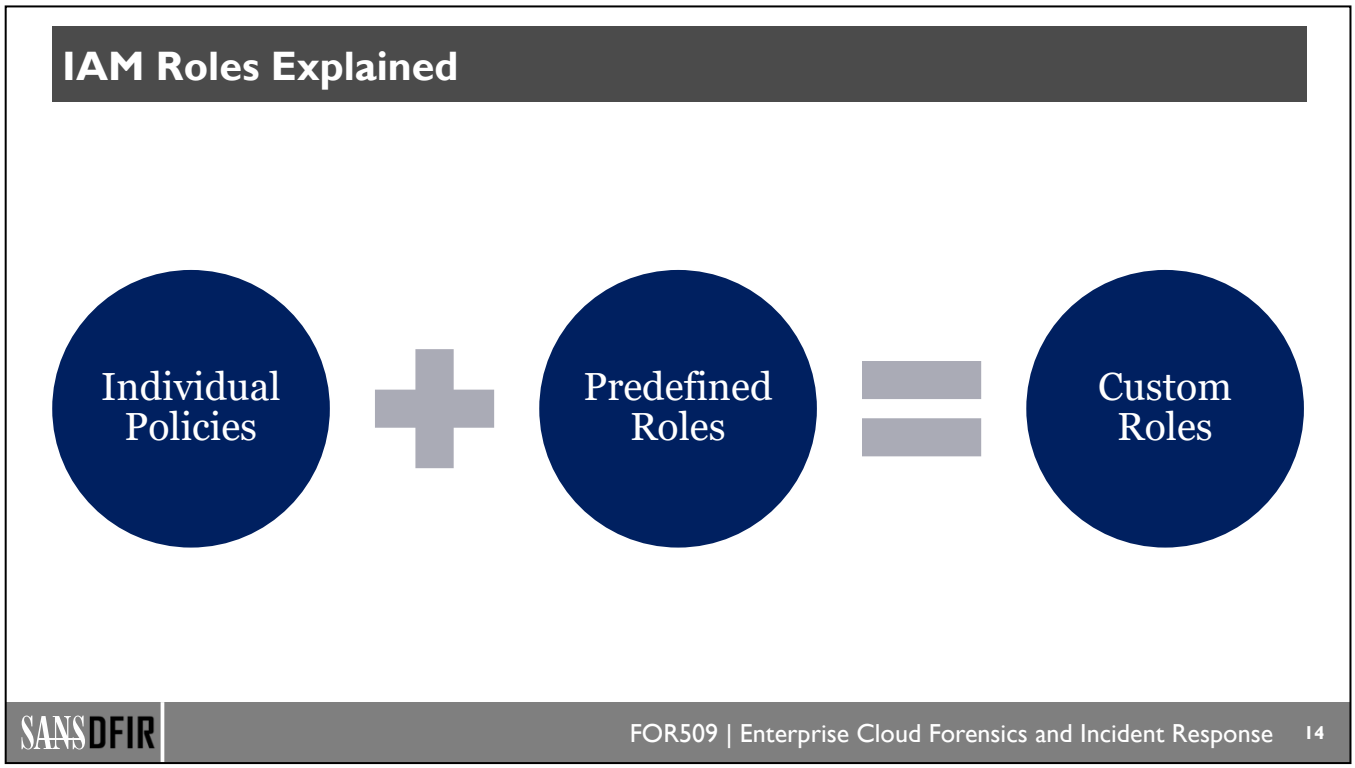
Privileges are given to IAM accounts with roles and policies

Policy	Predefined Roles	Custom Roles
• The most atomic level of defining privileges. Policies are a JSON document of API privileges that you specify	• Predefined roles are predefined sets of policies created by AWS to support access to AWS services • Roles usually come in Admin/Read-Only variants	• Predefined roles and your own policies can be combined to make custom roles

A policy is the smallest amount of privilege that can be granted. It works by permitting or denying whatever resource the policy is attaching to the ability to make API calls against specific resources.
A simple example of a policy would be the ability for a user to have read access to a S3 storage bucket. The policy would allow s3 operations like list and read against s3 buckets for that one user or group. Policies can be grouped together into roles.

Predefined roles are those that AWS has created to provide examples of policies needed to perform operations against AWS services. They are provided as examples in the hopes that you will then change those policies within them to reduce them to just the privileges needed. Ever AWS service has a set of predefined roles created for it providing examples for conditions like administrator access for that service or just read-only access as just some examples.

You can take predefined roles and individual policies you've created and turn those into custom roles that can be standardized and used throughout your organization.



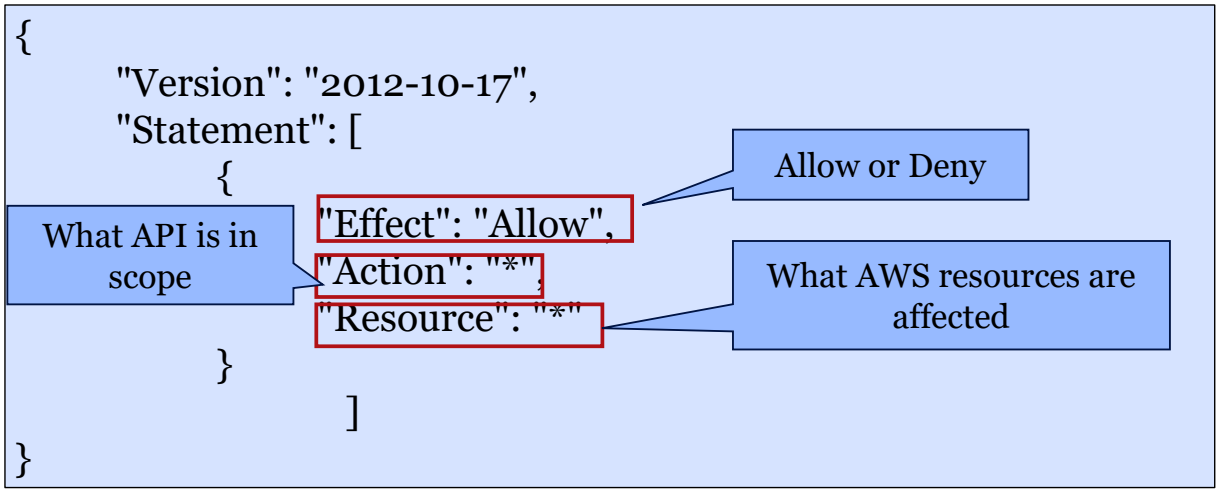
If the prior slide confused you, let's explain more.

The smallest unit of privileges that can be given to a user or IAM role is a policy. A policy defines in a JSON statement what API calls a user can or cannot make against which resources. In the next slide you will see an example of a single policy that is defined to allow all access to all resources.

A predefined role is a set of policies that AWS has created as an example of a set of privileges necessary to operate a service. For almost every AWS service there is at least a read-only and a full access predefined role. For instance, for Lambda (one of AWS's serverless services) there is a Lambda Full Access predefined role that provides the account it's attached to full access to all Lambda API calls for all resources. While a Lambda read-only role would only permit access to functions that can list or detail Lambda functions but no ability to change, execute, or delete them.

A custom role is any combination of predefined roles with the option of adding individual custom policies to create a curated set of API privileges on resources you specify that the account you are attaching them to needs to do its job. Custom roles allow you to finely set granular access controls to a user or IAM role so that it limits the damage that could be done if the account was compromised.

Analyzing IAM Policies

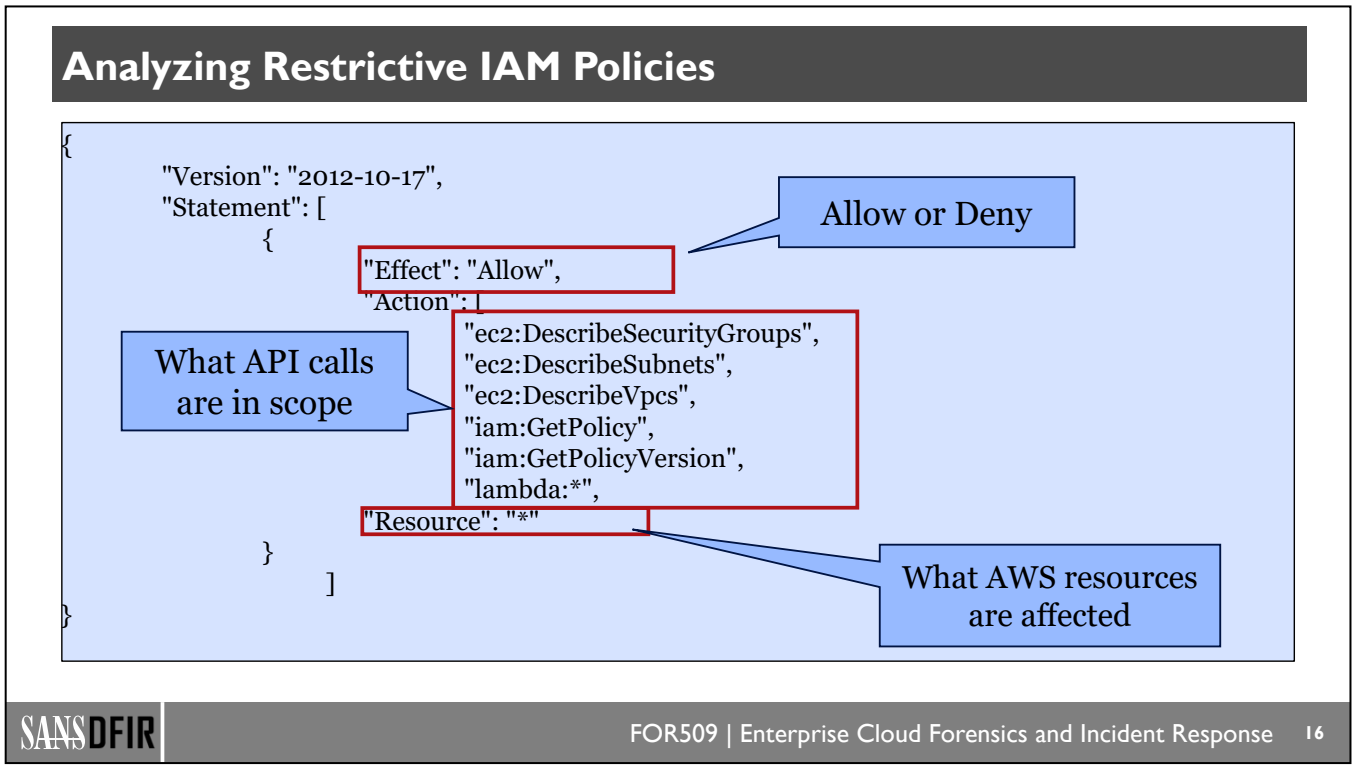


This is an example of an Administrator policy.

This is like the Any/Any firewall rule. It allows the IAM role or account to which it is attached to execute any API command against any AWS service.

Effect: Should be Allow or Deny. Since you can have multiple permissions and restrictions in one policy, you would stack these to meet your needs.
Action: This is the API calls you are allowing. Here we are indicating *, which means all functions. In the next slide we will be specifying a specific set of API calls.
Resource: This states what AWS resources (such as S3 buckets or EBS volumes) are in scope for this policy. Again, * means all resources.

More than once we've seen developers get frustrated in testing and put this kind of policy in place on a standard IAM role for a system. Everything works, and then they walk away and forget, leaving an EC2 instance that, if compromised, could act as an administrator. You need to list out and review all of the attached roles and policies to a compromised account or role to truly scope an incident.



This is an excerpt from the Lambda Full Access role.

Notice that every Lambda function will be given the ability to list out your EC2 instances, IAM roles, and access Lambda functions.

“Describe” here means to list, so this is allowing whatever has this role to list out your virtual networks and firewall roles.

IAM GetPolicyVersion allows whatever has this role to list out all of the policies that exist, which is useful to hunt for over-provisioned policies.

This means that if someone were to get a Lambda to run hostile code, it could easily scan your environment and return data, while also exposing the IAM credentials of the Lambda function itself.

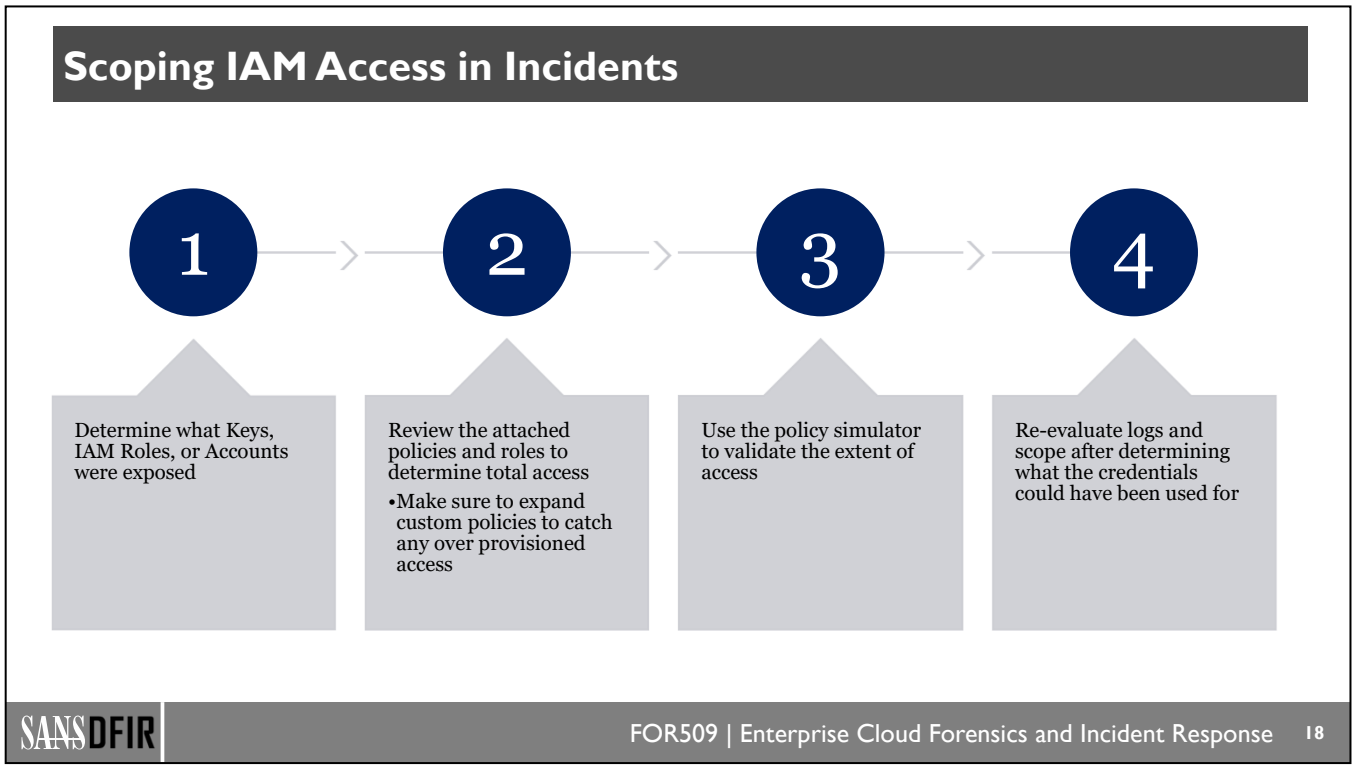
IAM Policy Simulator

- Want to quickly see what a given user or policy provides access to?
- AWS made IAM Policy Simulator for exactly this.
- You can have it test specific services or all services to determine if the given policy/user would have access.

Sometimes all of the roles and policies (and the roles that policies inherit) can get a little overwhelming.

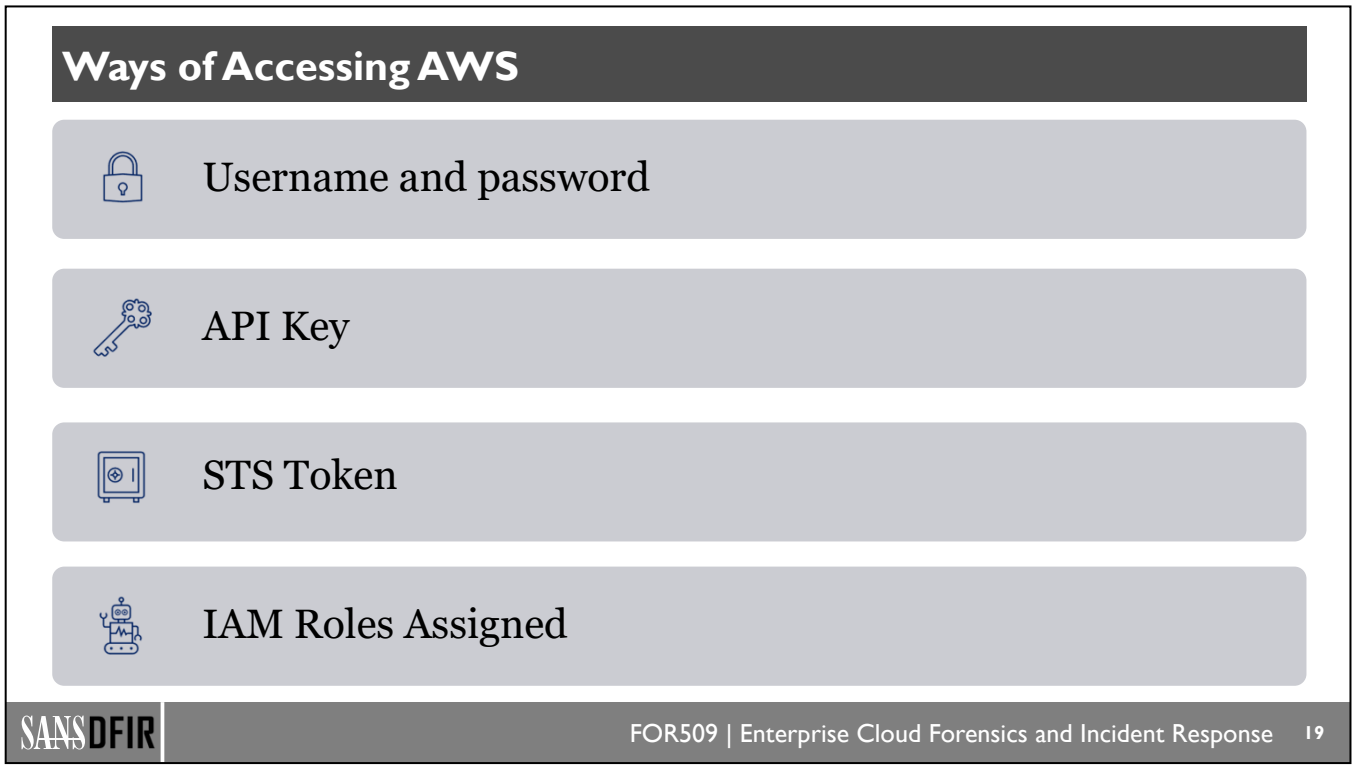
AWS provides the IAM Policy Simulator,[1] which will test the policies against an AWS service you specify, showing not only if the API call would be allowed, but what policy would allow it.

1. https://policysim.aws.amazon.com/home/index.jsp?#groups



Once you know a compromise has occurred, it's important to properly scope the incident. Failure to scope the incident to understand its impact and how an attacker could have moved laterally through your environment could result in never closing the breach or knowing what truly happened.

Following this process will allow you to pivot from a single account, IAM role, or API key to all of the access that stolen credential could have been used for. This will shape what queries you run within your log analysis, systems you review in your endpoint analysis, and potential for movement and exfil.



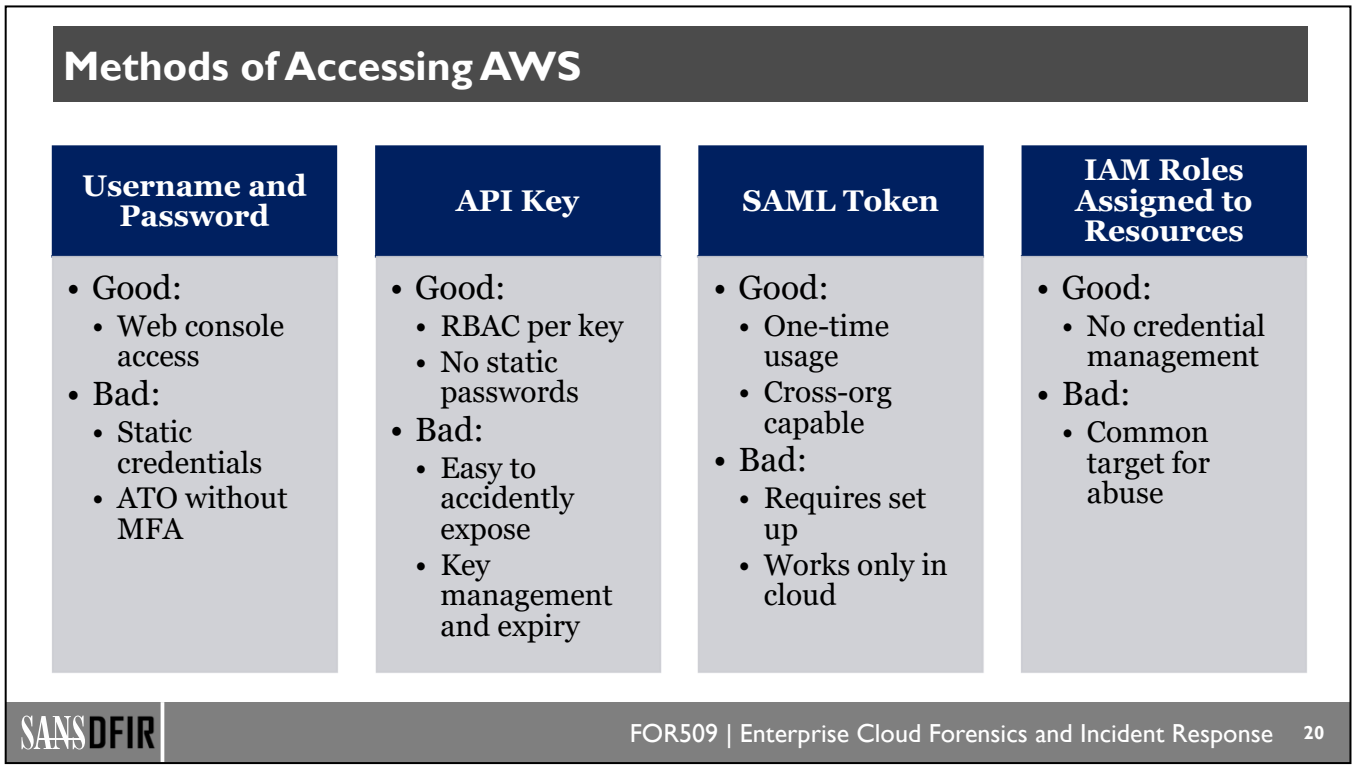
There are multiple ways to access AWS:

Username and password is the standard method of access for most people who first access AWS. However, these are the credentials most at risk in most circumstances. In order to use a username and password, you must access the AWS web console (aws.amazon.com) and specify your organization's ID or provide the root account email address. We call this the "tired" method since it limits your ability to automate or work with multiple resources at once. You'll notice in the AWS documentation that they typically try to direct you to perform tasks with either the command line interface (CLI) or programmatically with the software development kits (SDKs), both of which don't support usernames and passwords.

API keys are defined as the "new normal" because that's what the AWS documentation expects. API keys can be created on a per-user account, service, role, or resource basis. With an API key, you can define the scope of the access needed, much like a service account, and then provide it to the CLI or SDK to programmatically access your AWS resources. API keys can be easily audited and set to automatically expire, which helps to prevent stale credentials.

SAML (Security Assertion Markup Language) generates tokens that expire within a specified time window. These tokens can last for seconds, minutes, hours, or days and provide access to a set of resources you specify, where a token is provided as a file or embedded within a URL. The benefit of SAML is that it lets you dynamically generate tokens that access AWS resources that are meant to expire, without the key management overhead required for API keys.

IAM roles mean a set of policies that have been added to specific roles within the AWS IAM system. From here, you can attach a role directly to an AWS resource, allowing it to make AWS queries with those rights inherited, rather than worrying about hard-coding credentials into a service or script.



Username and Password
Good:
- Good for using the web console, but shouldn't be hard-coded anywhere else
- Make sure to turn on MFA

Bad:
- Static credentials that allow full access
- Keeping track of multiple tenants for multiple roles
- Without MFA, your account could be taken over

API Key
Good:
- Defining role-based access with specific permissions
- Allows access without exposing passwords

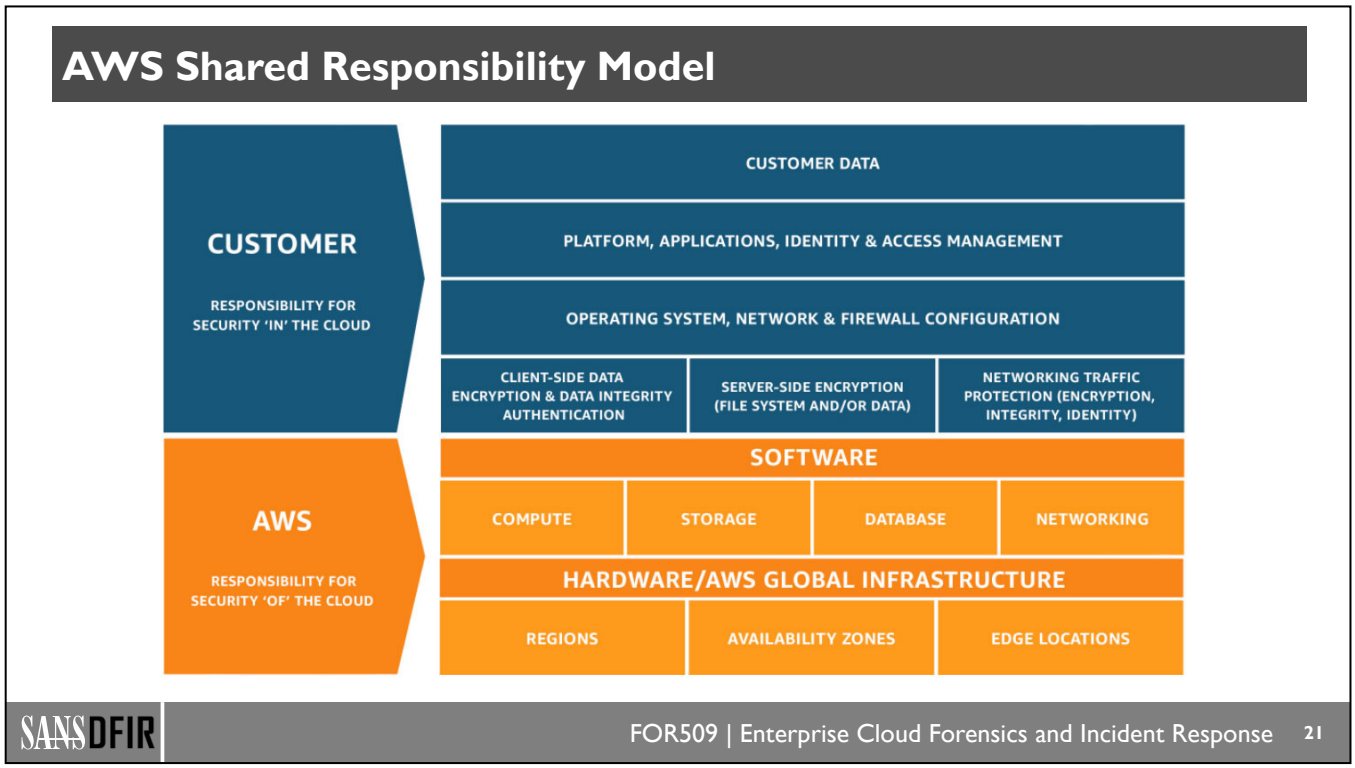
Bad:
- Bad guys and bug bounty hunters are already searching the world for API keys
- Typically, don't expire and can easily continue to exist when someone leaves

SAML Token
Good:
- Allows for one-time usage of an AWS resource
- Allows access to resources outside of your organization

Bad:
- Requires someone with Admin permissions to set up the STS service to provide the token

IAM Roles Assigned to Resources
Good:
- Allows access without credentials

Bad:
- Once compromised, provides those roles without credentials



Just like the Microsoft shared responsibility model slide you saw in Section 1, AWS has its own shared responsibility model. The important thing to think about here is that using AWS infrastructure services is caveated as "secure at your own risk." Things like EC2 instances, containers, and S3 buckets are your responsibility to secure with good practices and IAM roles.

For those infrastructure services, AWS makes sure that the hypervisor is secured, but everything else is your problem. If you're trying to explain this to people, how do you define the line? Your org is responsible for everything you choose to set up and run.

Reference:
https://aws.amazon.com/compliance/shared-responsibility-model/

What Do You Maintain the Security Of?

You are responsible to secure, maintain, and monitor

- Operating system you choose to run and manage
- Firewall you configure
- S3 bucket you provision
- Web applications you decide to run

AWS will help with monitoring, for a fee

AWS will help you if you choose to opt in and pay for a service like AWS Detective or AWS GuardDuty to be able to look for threats and problems. But they don't have any kind of managed security service provided by AWS for you to monitor or secure your org.

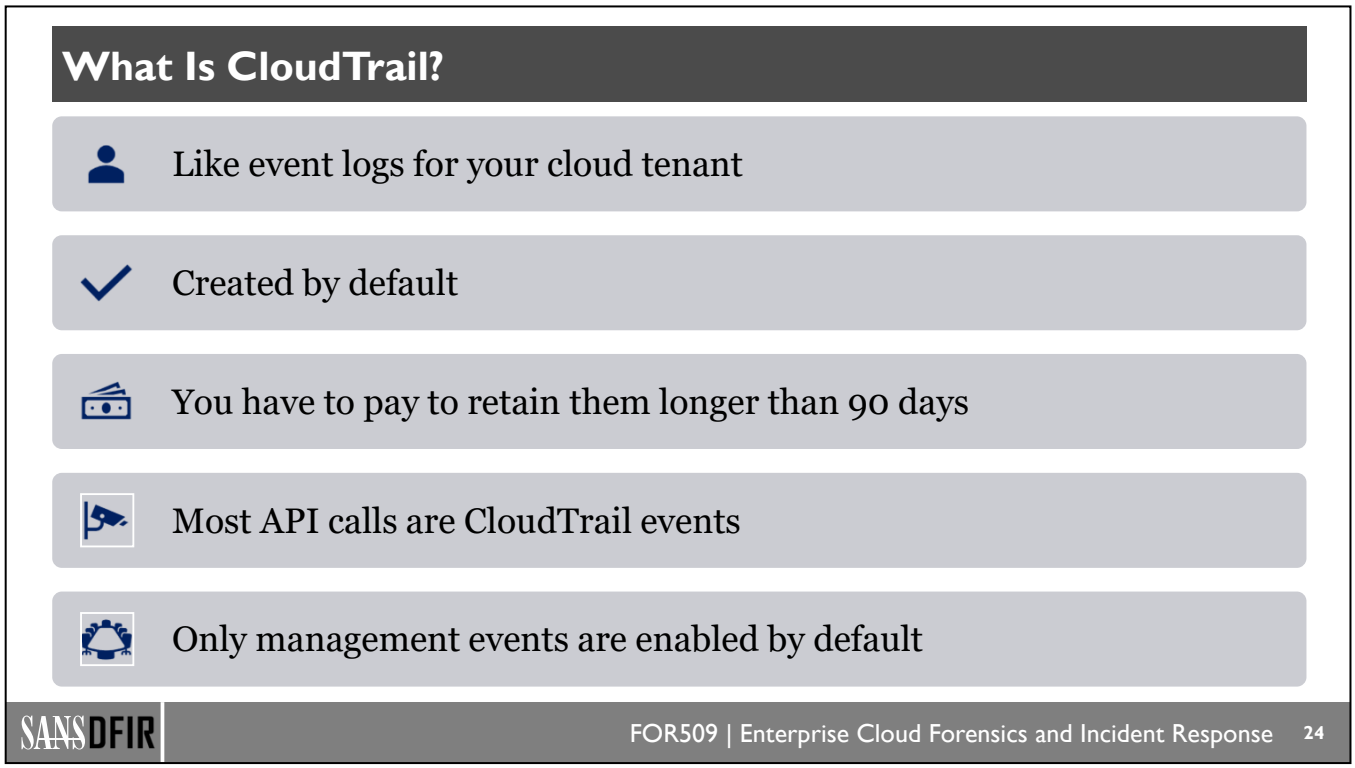
For services like S3, AWS provides the underlying security by making sure no one can access everyone's stored data, but you still set the policies of who can access your own data. In contrast, for a managed Platform as a Service (PaaS) like Elastic Beanstalk or any other type of managed platform, AWS takes care of the security. Another example would be AWS's managed databases. AWS will secure the database, but you define the roles and accounts that can access the underlying data.

What Does AWS Maintain?

- AWS-provided hypervisor (EC2)
- AWS-provided share storage space (S3)
- AWS-maintained web application
- AWS-maintained database
- If AWS maintains it, they secure it

This is a short list of examples in the shared security responsibility model of where AWS provides the security for a resource. This is not inclusive of the many services AWS provides but rather gives a good guideline of where the line is drawn. For whatever backend service AWS is providing, they will provide security for it.

So, for infrastructure services, they will secure the hypervisor, but not what you choose to run on it.
For platform services, they will secure the operating system in addition, but not the code you choose to run.
For databases, they will secure the underlying database and operating system, but not the data you store within it and who you provide access to.



How can you determine whether roles are being accessed? How can you determine how accounts are being used?

For AWS, CloudTrail is the audit log that stores this and much more!

CloudTrail[1] is like event logs for your cloud tenant.
CloudTrail is:

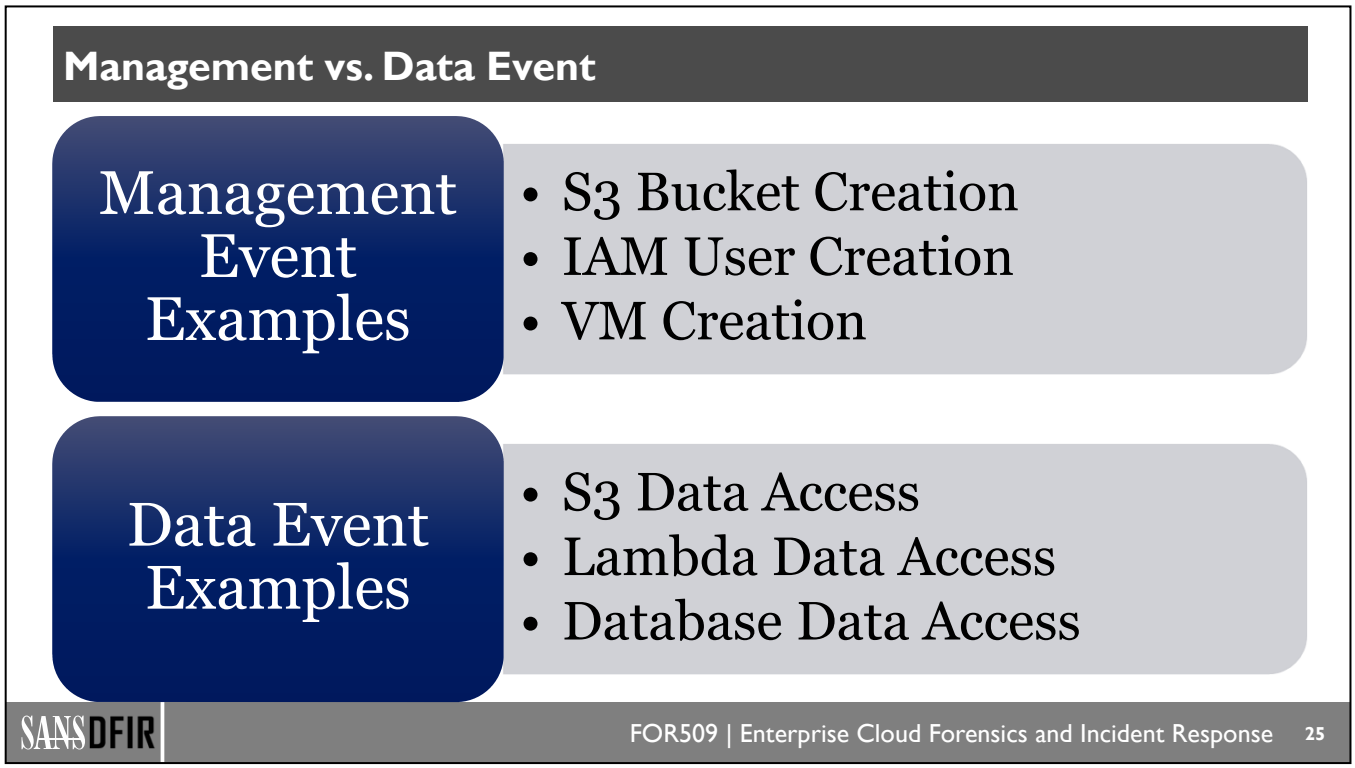
- Created by default.
- Free for storage of audit logs for 90 days.
- If you want to keep data for longer than 90 days, you have to set up a trail to get stored in an S3 bucket somewhere or ingested into some other platform.

In its most basic definition, CloudTrail records calls to AWS APIs, including what was being requested and what the result was. CloudTrail does not contain logs from the applications that are running in your AWS org (though you can use CloudWatch Logs to import those into AWS as well). Each log entry is an event, meaning an API request.

As an example, creating and starting an EC2 instance would be logged in CloudTrail. However, the events that are occurring within the EC2 instance would not be recorded in CloudTrail, unless those events involved making a request to the AWS API. All of the actions you can perform in the AWS web console are really API calls in the backend and are recorded as well.

It's also important to know that all CloudTrail events are recorded in UTC.

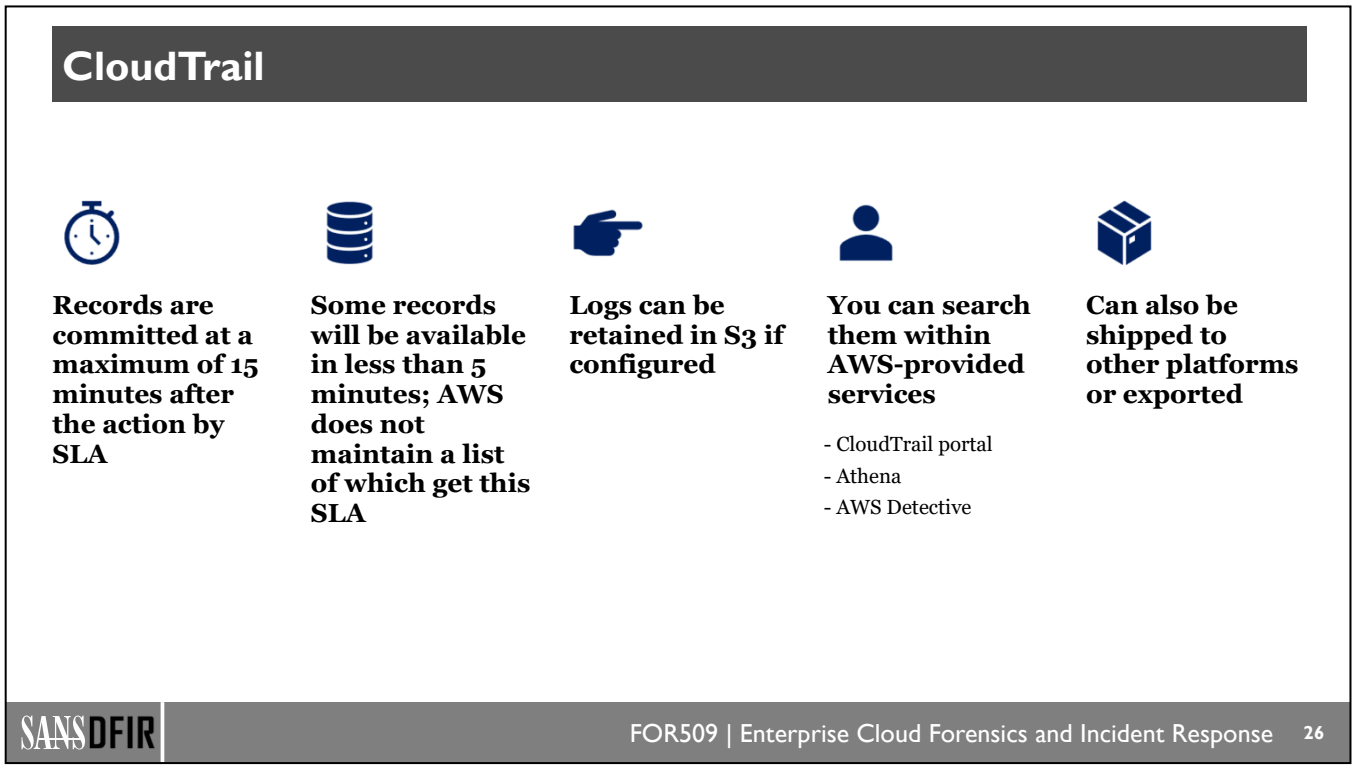
1. https://aws.amazon.com/cloudtrail/



In CloudTrail we have two kinds of events, management and data.

Management events represent those API requests being made to the control plane to interact with a service endpoint. These API calls are requesting information, creating resources, modifying resources or deleting resources. These management events will be recorded by default within CloudTrail logs. One thing to note is that as new API functions get developed and released by AWS there can be a lag time between their public release and when they have events about its usage recorded in CloudTrail.

Data events represent those API requests being made to interact with data being stored within a service. The three Data event sources currently available in CloudTrail for configuration are S3, Lambda, and Database (RDS). Data events are not enabled by default in CloudTrail. To enable the recording of Data events you must create a "trail" as we discuss later in the section and choose each of the data sources you want to include in your CloudTrail logs.



CloudTrail records are committed at a maximum SLA of 15 minutes. This means that a record could be created in CloudTrail in less than 15 minutes, but 15 minutes is the maximum time AWS guarantees that a record could be written in. In comparison, Azure and Microsoft 365 have an SLA of 30 minutes to 24 hours.

This means the ability to build automated responses to known malicious CloudTrail events within your account will always have a maximum 15-minute response time between when the event occurred and when you can detect and triage it. There are some AWS services that have a 5-minute commit time, on average. However, AWS does not provide a stated SLA beyond 15 minutes on a per-service basis.

Depending on the speed of your attacker by the time the log entry enters your detected environment, it may be too late, which means you still need endpoint visibility. CloudTrail is not a replacement for EDR/XDR/MDR; it is an audit trail of resources for hypervisor API actions.

If you want to retain CloudTrail logs longer than 90 days, you must create a "trail." This allows you to specify an S3 bucket within your organization that will store CloudTrail logs as long as you specify. By default, those logs will be kept forever in that S3 bucket.

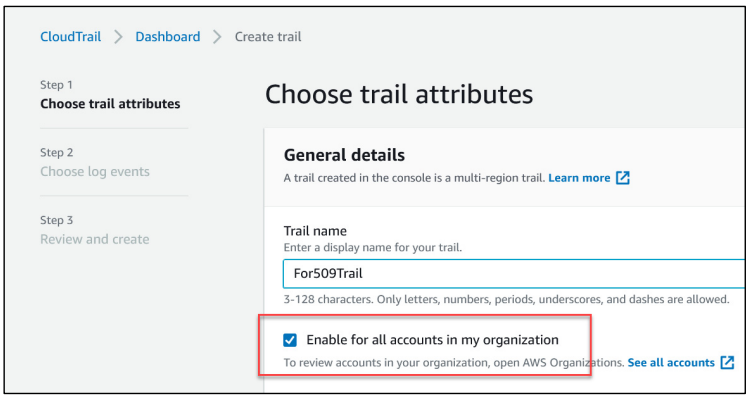
(Read more about accessing the base 90 days of CloudTrail logs here: https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events-console.html.)

You can search CloudTrail logs in the CloudTrail portal, or natively in AWS you could search them in **Athena,** which is their SQL-like search interface for just about anything, but it has predefined templates for CloudTrail. You could also search and review automated findings of your CloudTrail logs in something like AWS Detective.

Lastly, if you already have an existing SIEM or log management platform, you can just ship CloudTrail logs off to external services. AWS will still keep 90 days' worth of logs, regardless, in their S3 buckets.

CloudTrail Creation

- Creating trails allows you to retain access logs beyond 90 days
- You can force CloudTrail for your org
- You can centralize your org trail into one bucket or split your trails into multiple buckets
- Creating a trail allows you to capture data events.



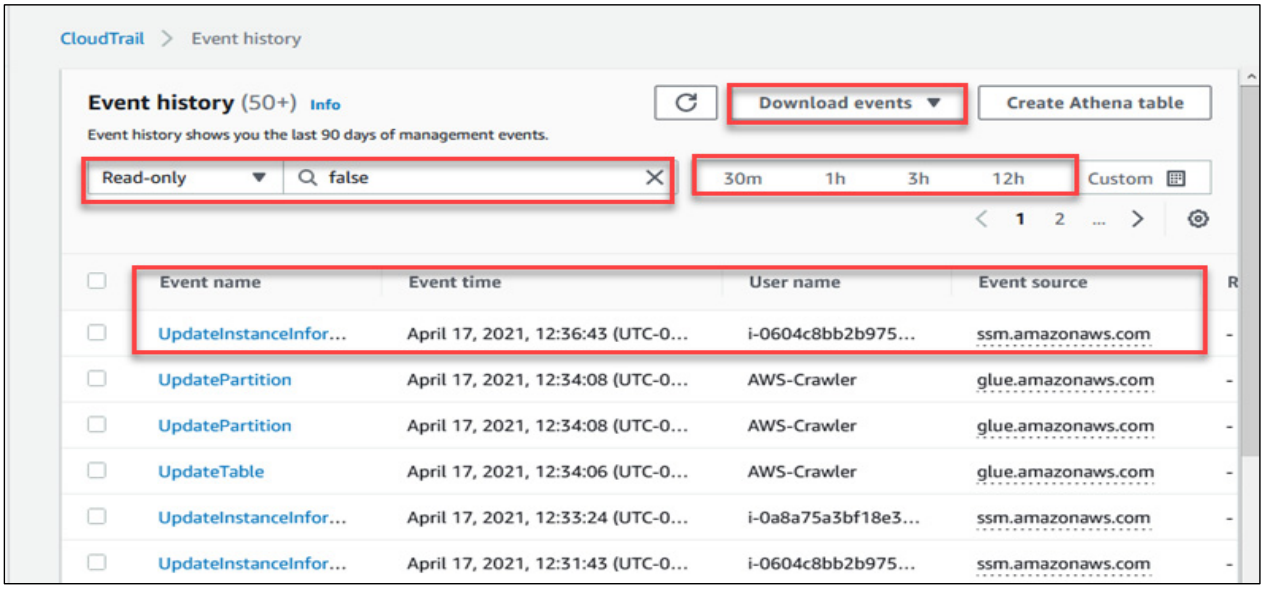
You can turn on CloudTrail at the organization level.[1]

This means all the accounts added to your AWS organization will store CloudTrail logs to a bucket you specify.

If you do not create a trail, you'll lose all CloudTrail logs older than 90 days.

1. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-create-and-update-a-trail.html

CloudTrail Portal



This is the default view of CloudTrail logs in the CloudTrail web portal.[1]

You can do basic filtering, set timeframes, and review API management events. When you find the events that you want to look at outside of the CloudTrail portal, you can download them in CSV or JSON format.

However, there is an issue with using this portal to review your CloudTrail logs, which is that API data events will not show up inside of this dashboard, meaning that if you're looking for a service being created, deleted, or modified, it should be searchable and viewable inside of this dashboard. Events such as S3 sync events or any other API calls that are affecting data and are not management functions will not be viewable in this view. Even though the CloudTrail events exist, they will not be viewable or searchable here; in addition, this view will always be limited to 90 days, even if you create a trail that stores the logs for longer.

1. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/view-cloudtrail-events-console.html

Downloading Default CloudTrail Logs

- If you don't have CloudTrail logs saved into an S3 bucket you can still retrieve the data visible in the CloudTrail Portal through the API.
- The AWS API allows you to download the default 90 day CloudTrail logs 50 events at a time.
- A python script called awsCloudTrailDownload.py is available in the class GitHub (for509.com/github) and the direct link is in the notes below.
- These logs will only contain management events meaning no:
  - S3 Data Events
  - Lambda Data Events
  - Database Data Events

If you find yourself in the unfortunate situation of only having the default 90 days of management events in the CloudTrail Portal, you don't have to just review them through the web console.

Using the script provided at:
https://github.com/dlcowen/sansfor509/blob/main/AWS/awsCloudTrailDownload.py

You can download the last 90 days of logs that are being retained in AWS default log storage that is not accessible directly. You should know that AWS limits these default log retrievals to 500 records per request and will throttle attempts to request them faster. The script has the following syntax:

awsCloudTrailDownload.py –access-key-id <api key id> --secret-key <API Secret key>

When executed, it will download all available logs until there are no records available to the directory it's executed from. CloudTrail is regional and this script in its current version only downloads logs for the region your CLI is currently set for. If you want to download from other regions either a) change your region in the CLI or b) wait for the next script release which will download from all regions.

Be aware that when you are working with the default CloudTrail logs that only Management API events are logged. What are Management events? Management events are those API calls that create/delete/start or stop a service or make a call into an existing service that is running. What the default CloudTrail logs do not log are Data events. These events include API calls to retrieve/store/query S3 data, calls to Lambda functions, as well as Database API calls. The only way to have Data API events logged is to enable a trail and turn on the optional events.

Downloading CloudTrail Logs from S3

- If the CloudTrail logs are stored in an S3 bucket you can download them much faster than the default logs.
- You can use utilities like:
  - S3 Browser (https://s3browser.com/)
  - Cyber Duck (https://cyberduck.io)
  - CloudBerry (https://www.msp360.com/explorer/windows/amazon-s3.aspx)
- Or just the AWS CLI

```
aws s3 cp s3://<name of log bucket>/AWSLogs . --recursive
```

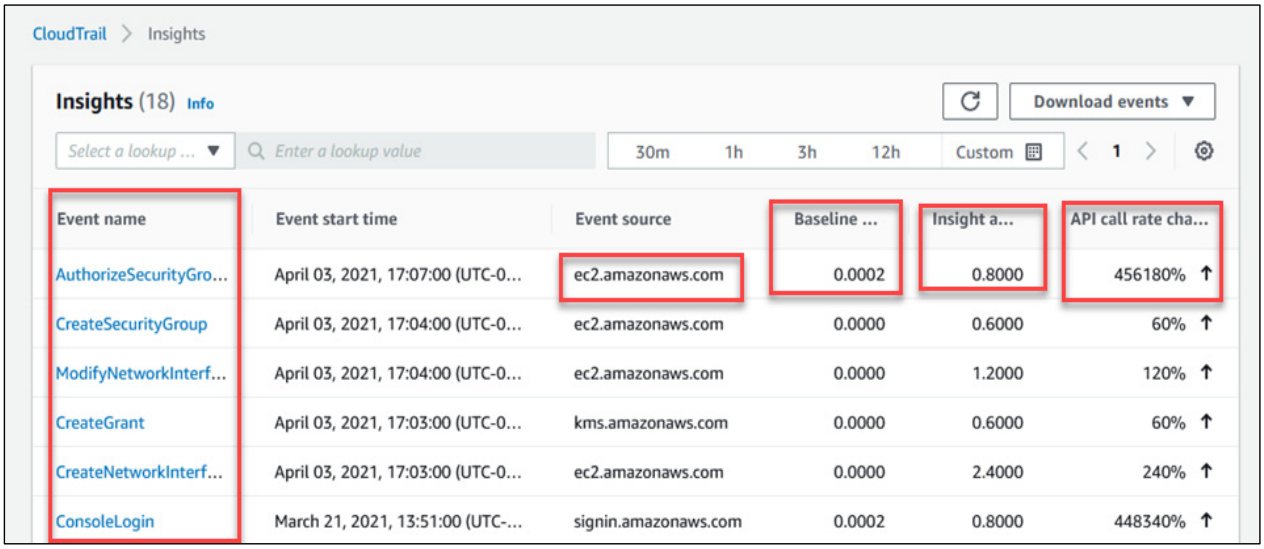
Will download all the logs in a named bucket to your local system

If you are lucky and someone did set up a trail in CloudTrail you will be able to retrieve the data directly from S3.

There are multiple methods of downloading CloudTrail logs (and any other logs) stored in S3.

1. Web Console: You could attempt to download these from the Web Console, but its file lists are paginated and attempting to download a large recursive directory structure will time out.
2. API-based GUI Tools: You can make use of the GUI tools listed above. You can provide either your username and password or API credentials and it will graphically display the folders/files, and most will permit multi-threaded downloads for accelerated transfer.
3. AWS CLI: The AWS CLI cp or sync commands will also allow you to download all of the logs present in a S3 bucket. In the slide above we are providing the syntax to use the s3 cp command to download all of the logs found under the AWSLogs folder in the S3 bucket and store them in the directory where we are running the command from. The –recursive tag is telling the CLI that we want to download not just the contents of this folder but all subfolders and files as well.

CloudTrail Insights



CloudTrail Insights[1] is the basic version of GuardDuty. It will provide observations of anomalous events, such as a rapid increase in certain API calls, that are found within your CloudTrail logs. There isn't a lot of "intelligence" to these events like with GuardDuty; instead, Insights is taking a base metric, which can be seen in the slide here, as "baseline" and creating an entry if the measured value is higher than the baseline. However, if you cannot afford GuardDuty (which we talk about later in this section), Insights may provide valuable clues into possible attacker activity or misconfigured services.

1. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/logging-insights-events-with-cloudtrail.html

CloudTrail Pricing

Trails

- There is one default trail created and maintained by AWS
- If you want to store logs longer or create another subset of logs that just contain certain events, you can create a "trail". You can have multiple trails.

Free Tier

- The built-in CloudTrail portal and the 90 days it stores is searchable at no cost
- You can create one “trail” in S3 for free and retain data longer than 90 days
- Does not include CloudTrail Insights

Paid Tier

- After the first “trail,” second copies of management events are $2 per 100k events
- Data events are $.10 per 100k events sent to S3
- CloudTrail Insights is $.35 per 100k events

AWS has two tiers of service for CloudTrail.

If you are using the free tier of AWS, meaning you are not paying for the service (yet), then your CloudTrail logs will still be stored for 90 days for free.

You are allowed to create one free trail that will store data longer than 90 days, but you will pay for the S3 storage costs associated with it.

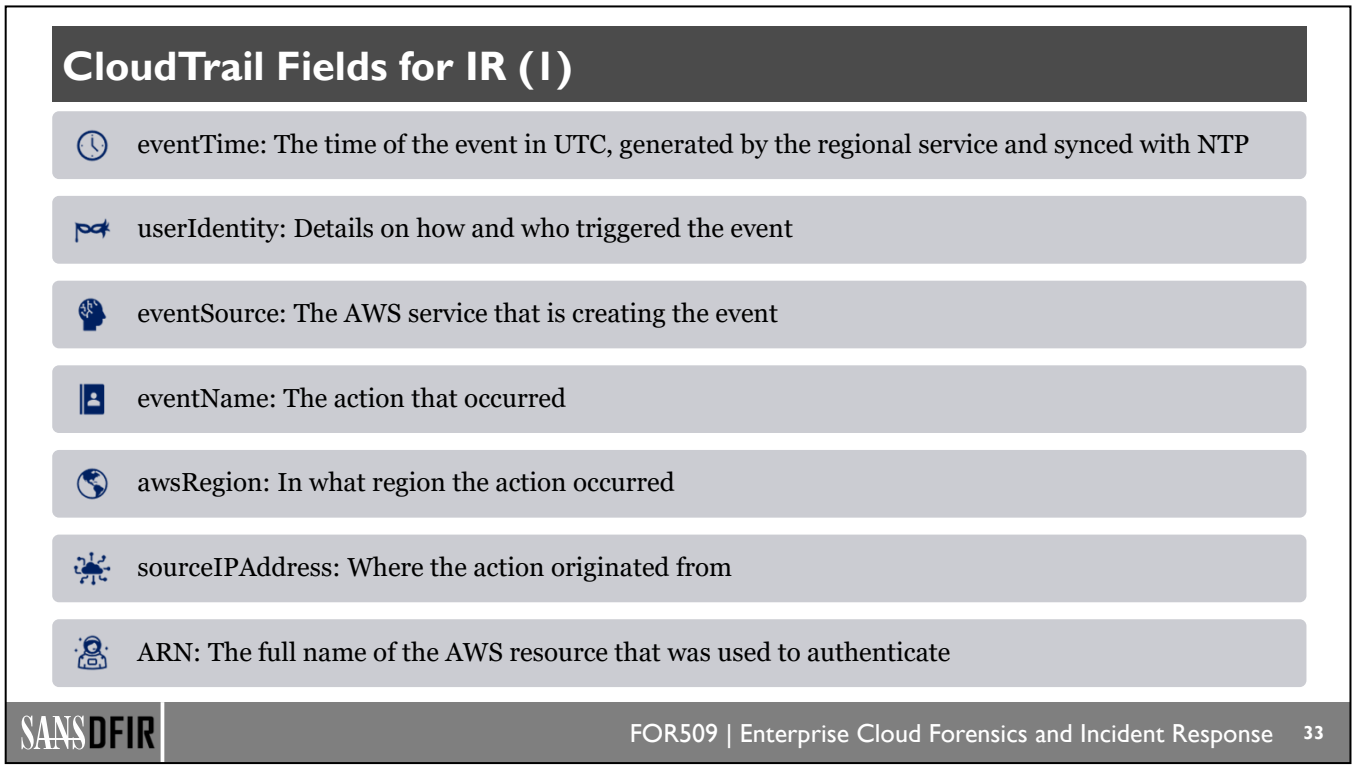
Also, free tier accounts do not get access to CloudTrail Insights, which points out possibly malicious or anomalous activities in your CloudTrail logs.

In the paid tier,[1] the following costs are involved:
Your second trail and every trail thereafter is charged at $2 per 100k management API-related CloudTrail events recorded. This is in addition to the storage costs being incurred in the S3 buckets where the logs are stored.

In addition, any data-related API CloudTrail events recorded will be priced at $.10 per 100k. Why? Because there are many more data events that management events.

Lastly, CloudTrail Insights is charged at $.35 for 100k events per trail.

1. https://aws.amazon.com/cloudtrail/pricing/



eventTime: When the record was created in UTC. Congratulations, you don't have to worry about time zones.[1]

userIdentity: The userIdentity I think this is very important. This is either an assumed role, an STS token, the user account, or whatever authentication mechanism was used. This can include federated accounts with something like Azure AD. Sometimes that will be the actual account that caused the event to be created; other times it may be an internal role that was used, in which case you'd have to work backward to figure out what triggered the function to see the actual user or stolen role that was responsible.

eventSource: In Section 1 we talked about Microsoft 365 workloads; AWS calls this an event source.

eventName: The actual action that was called, usually the name of the API function called.

awsRegion: Which AWS named data center processed this API call.

sourceIPAddress: The IP address that the request came from.

ARN: Amazon Resource Name. This is the full name of the identity that was used to authenticate the request.

1. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference.html

CloudTrail userIdentity Types

Root: The event was triggered by someone with the root identity account

IAMUser: The event was triggered by someone using IAM user credentials

AssumedRole: The event was triggered by an account that got credentials by IAM role or cross-account access

FederatedUser: The event was triggered via a temporary security token issued through federated authentication

AWSAccount: The event was triggered by an AWS account outside of your organization

AWSService: An AWS service triggered the event

Let's break down the kinds of userIdentity types you might see in the CloudTrail logs.

To our knowledge, this is not the complete list. Amazon can add new identity types at any time. However, at the time that this slide was written, this appears to be the complete list of identity types you will find.

Root: If you see root as the user identity, that means someone is actually using that root super-user account for that account ID, which is embedded in the ARN. Any time you see the root account being used, it could be interesting. The root account should be used sparingly and not something you should see very often in a corporate production environment. So, if you see a lot of root accesses, you might want to reach out; this could be a bad practice or maybe a compromised account.

IAMUser: This will record the name of the IAM user who authenticated. That authentication can be any of the prior methods (password, API key, SAML token, etc.).

AssumedRole: An existing IAM role was used to authenticate, and it was impersonated based on the privileges and policy. The full log entry should show you what role they assumed as well as who was assuming it.

FederatedUser: Federation means that your organization is allowing an external authentication source to allow access into your environment. If it's a federated account (let's say you're attaching Azure AD for authentication for single sign-on into your AWS environment), then the authentication to AWS of that Azure AD account will come in as a federated user and will log the Azure AD account name.

AWSAccount: Many people misunderstand what AWSAccount means when it comes to proper access scoping. An AWS account, as far as security permissions go, means an account on AWS not within your tenant, just anywhere on AWS. So, if you see AWSAccount in your log, that's probably not good, unless it's a public resource.

AWSService: The existence of this entry means that it's an internal AWS service making the API call.

Read more about the userIdentity types:
https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-user-identity.html

CloudTrail Fields for IR (2)

- userAgent: How the request was made
- requestParameters/responseElements: What was requested/what was returned?
- eventID: Unique event number in CloudTrail
- eventType: What type of event occurred?
- resources: What was accessed during the event?
- sessionCredentialFromConsole: Did this event occur from the AWS console?

userAgent: In the UAL, the user agent was either blank, web, or the name of the web browser. In AWS terms, it could be a console access, it could be an API access internally, or it could be an actual user agent straight from a browser. In addition, it could be from the API, and it can actually identify the type of request of the version and language of the API being used.

requestParameters: A JSON string of what was passed into the API call when the request was made.

responseElements: A JSON string of what was returned from the API call when the request was made. When we are dealing with requestParameters and responseElements, it's a lot like the audit data we were talking about in the Microsoft 365 section. We've used our Logstash rules to flatten these nested JSON structures out. We're actually pulling sub-elements up because it's a very nested structure. As you go through the lab, if you feel that anything is missing, you'll see in Lab 3.4 how to use jq to pull the raw data out.

eventID: The eventID is not an event ID in the Windows Event Logs sense. In other words, it's a unique number per event, not something that defines the kind of event being recorded. It's an incremental, unique number of that audit event, which can be helpful when you need to dig deeper.

eventType: The type of event, usually an API call.

Resources: This is a JSON string that will contain which resources were accessed/affected by this API call.

sessionCredentialFromConsole: Did this API call come from the web console?

ARN

ARN stands for Amazon Resource Name

It is a globally unique way to describe the resource requesting an action

Format

- arn:partition:service:region:account-id:resource-id
- arn:partition:service:region:account-id:resource-type/resource-id
- arn:partition:service:region:account-id:resource-type:resource-id

ARN[1] stands for Amazon Resource Name, it's like a windows SID as it includes the full description of the AWS resource that is making the authorization for a request. Each ARN is unique globally and identifies the exact service, in the exact AWS account, with the exact user, service, or role name that is making the request. ARN's are necessary to understand if you have a large number of AWS accounts within your organization and you need to know which account to focus on to get the evidence you need.

1. https://docs.aws.amazon.com/general/latest/gr/aws-arns-and-namespaces.html

ARN Field Examples

arn:partition:service:region:account-id:resource-type/resource-id

Partition
- aws: AWS Regions
- aws-cn: China Regions
- aws-us-gov: AWS GovCloud (US) Regions

Service
- S3
- STS
- Quicksight

Region
- us-east-1
- us-west-1
- eu-west-1

Account ID
- Unique Amazon ID of your tenant
- 197674874429

Resource Type
- user
- assumed-role
- federated-user

Resource Id
- IAM username
- Bucket name
- Assumed Role name

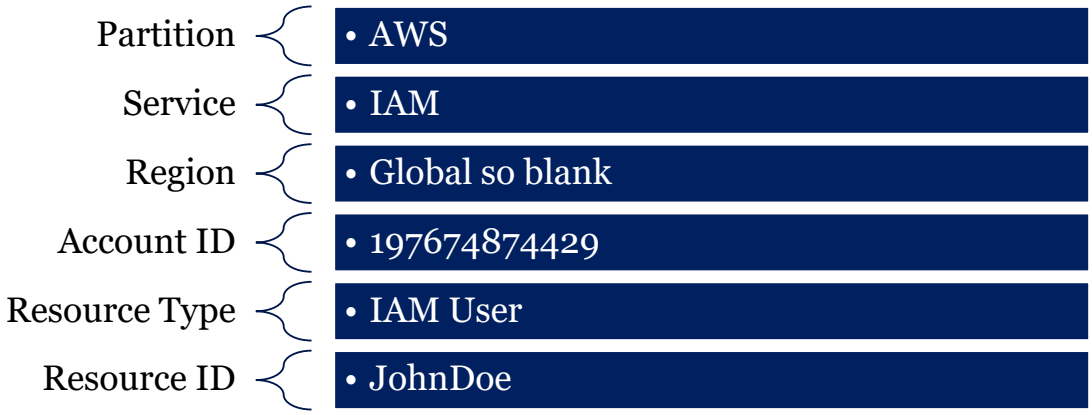
In the previous slide we talked about what an ARN is, here we talk about what can be in each of the fields.[1]

Partition: There are currently three possible values here. Either AWS the commercially available cloud, aws-cn for Chinese residency requirements and aws-us-gov for fed cloud requirements.
Service: This is the AWS service involved in the request such as STS for tokens generated for service requested accesses or IAM for user accounts requesting an operation to be performed.
Region: This is the region the resource is located in; if it's a global resource this field will be blank.
Account ID: This is the AWS account id of the account where the request originated from.
Resource Type: This is the type of resource such as an IAM user account, an assumed role, or a federated user from another cloud.
Resource Id: This is the name of the user, role, or resource that is making the request.

1. https://docs.aws.amazon.com/quicksight/latest/APIReference/qs-arn-format.html

ARN IAM User Example Breakdown

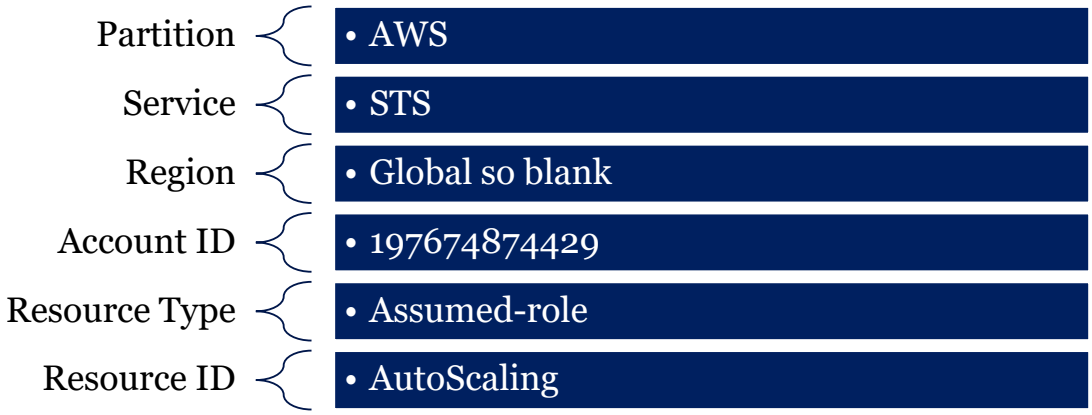
arn:aws:iam:: 197674874429 :user/JohnDoe



Here an example of an ARN belonging to an IAM user named JohnDoe.
We can see this is coming from the AWS publicly available cloud as the partition, it's being generated from the IAM service, IAM is global not regional so the region is blank.
Under the account ID we see 197674874429, which is the account ID of the AWS account that this IAM user exists in. Under Resource Type we see user, which means this is a IAM user not the root account or an IAM role. Lastly under Resource ID we see the username JohnDoe.

ARN Assumed Role Example Breakdown

arn:***aws***:***sts***::***197674874429***:***assumed-role***
/AWSServiceRoleForAutoScaling/AutoScaling



Here is an example of a service creating a temporary token to act under an assumed role. That is a long way of saying this is an AutoScaling service that is running in this AWS account performing an action so it is acting as its Service Account 'AWSServiceRoleForAutoScaling' to perform an operation in its work. If we break down this ARN we can see that the partition is again AWS. In this case the service has switched from IAM to STS. That's because STS[1] (Security Token Service) generates temporary API keys to allow a service, role, or other resource to perform actions as that role for a specified amount of time. The Region is again global so is not shown in the ARN, and the account ID is the same as before since this is occurring in the same account. Lastly the resource type is telling us this is an assumed role and the resource ID is letting us know the role being assumed is the AutoScaling role.

1. https://docs.aws.amazon.com/STS/latest/APIReference/welcome.html

CloudTrail IAM Investigation Examples

Tracking access to the console

Finding API key creations

Finding exposed API keys

Threat hunting in CloudTrail

So, what are common things we're looking for in CloudTrail logs as investigators?

1. People accessing the console, making queries to the console, and trying to expand what kind of access they have. That's usually kind of a prelude to a larger attack if someone has stolen roles, accesses, or keys.
2. Looking for people creating new API keys, a lot of the times when you have credentials compromised and the ability to create the key.
3. Looking for keys that could have been misused after they've been exposed. We see this a lot when people are accidently pushing API keys into GitHub projects.
4. We will also walk through some threat-hunting scenarios.

CloudTrail: Tracking Access to the Console

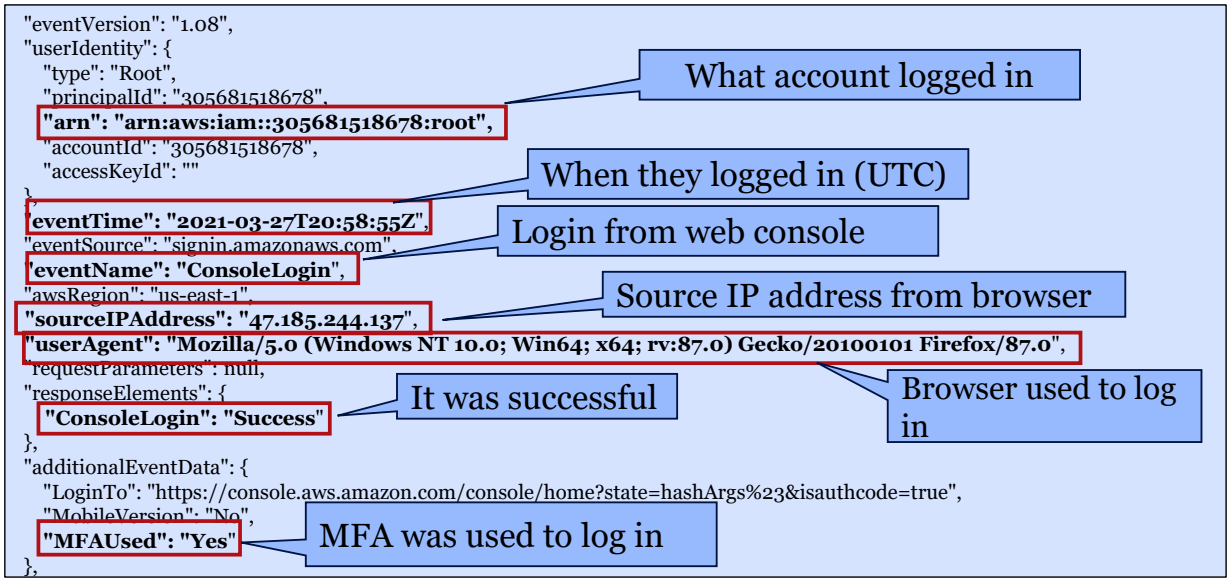
Important fields	
ARN	What account was logging in
eventTime	When they logged in
userAgent	What browser they used to log in
sourceIPAddress	Where did they come from?
responseElements	Was it successful?
MFAUsed	Did they use MFA to log in?

When we're talking specifically about tracking access to the console, these are the fields we should focus on:

- Things we should be looking for are the ARN; what accounts are being logged in to?
- What time did they logged in?
- What if the user agent isn't matched to the user agent that user is normally using?
- Checking the source IP address to see where they're coming from and comparing that to other logins.
- Lastly, if your organization has followed AWS's best practices, we should check if the authentication used MFA.

AWS will display a suggestion when you log in as root without MFA. It will suggest that you turn on MFA. But unlike Azure, Amazon doesn't complain too loudly if your IAM users don't have MFA—even if those users have full admin roles assigned.

CloudTrail Console Login



So, here's an example of a raw log from CloudTrail showing a console login.

In these slides, we start with a raw log to be able to walk through the full structure. When you go into the lab, you're going to see that Logstash has flattened the JSON structure. It's easier to work with, but I want you to see the raw data.

Focus on the ARN. Start with ARN followed by AWS so that this is an AWS-internal identity, meaning it is not federated. It's from the IAM service.

Next is the account number for the AWS account, and since this is the root account for that account authenticating, we see the word "root".

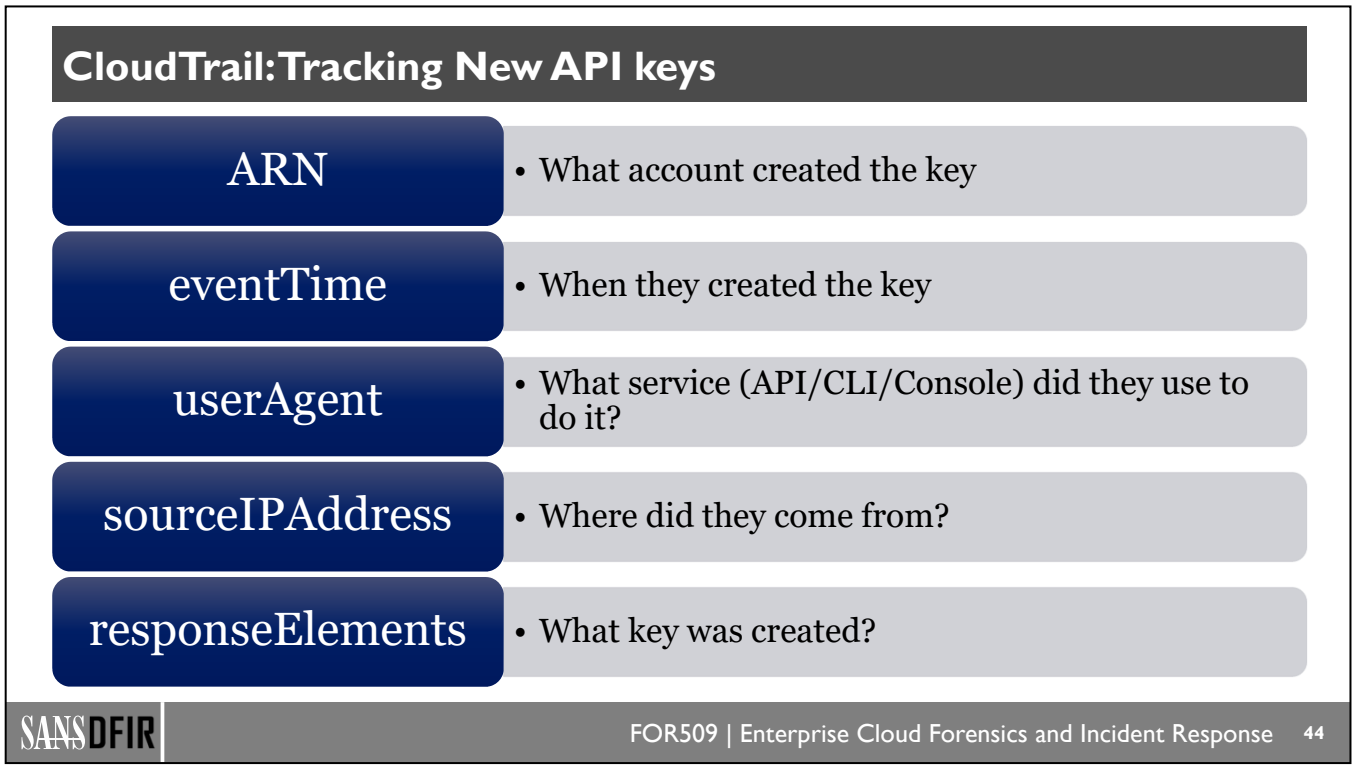
Again, you shouldn't see too many of these direct root logins unless you're in a test demo environment.

The event time here would be when they logged in. The event name states that this is a console login, meaning someone has gone into the web console and authenticated.

Next, we can see the IP address that I came from, followed by the user agent, which in this case is the browser I was using.

We can see that the responseElements are stating that the login was successful.

Lastly, we highlight the field that shows that MFA was used.



Let's change scenario to API key creation.

Important field to look at here are as follows:

ARN: What fully described identity was used to create the request to make the API key.

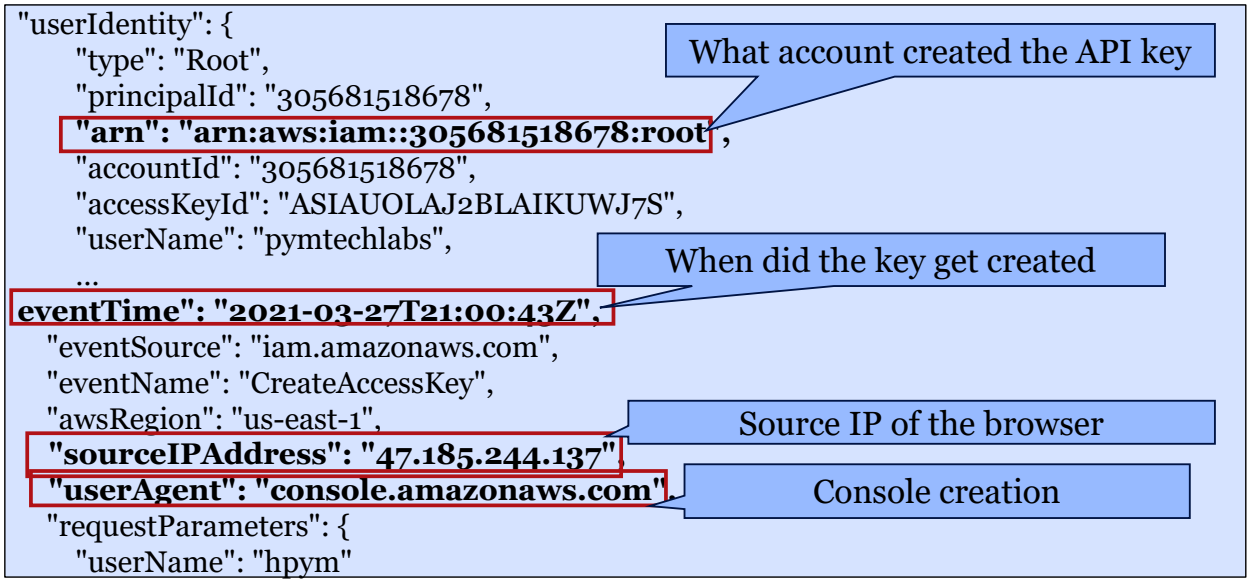
eventTime: When in UTC time the request was received to create the API key.

userAgent: This normally is going to record the service (web console, API, CLI) that was used to make the request.

sourceIPAddress: The IP address used to make the request.

responseElements: This will contain the results of the request, including if it was successful. Remember that the name of the key being requested will be stored in requestParameters.

CloudTrail: New API Key Created from Console (1)



Here we see a raw CloudTrail log being created from the console.

The ARN shows us this is being created using the root user's account.

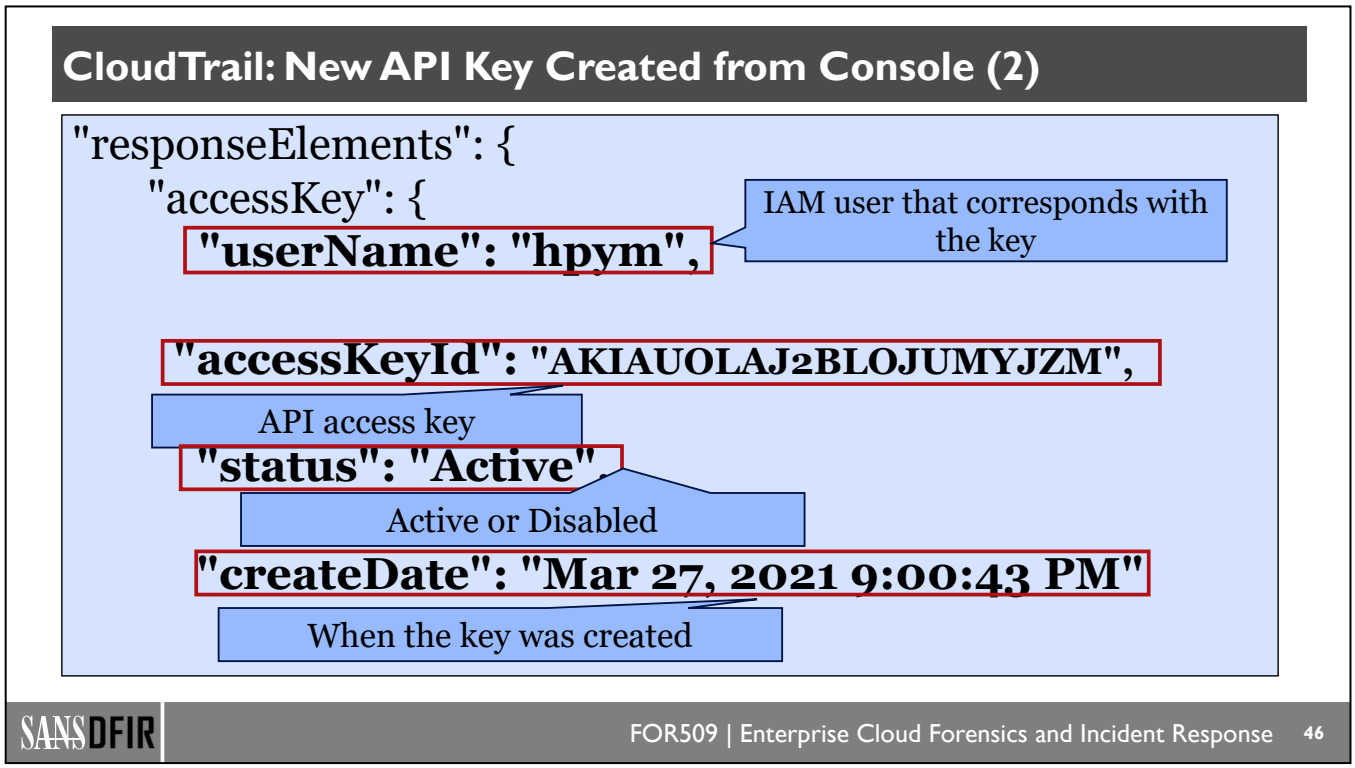
The eventTime is showing us when the API key was requested.

The sourceIPAddress is still the IP address of the user logged in. This is interesting, as the user agent was replaced but the source IP address was not changed to an Amazon IP.

The eventName is "CreateAccessKey," which is how we know what action is taking place here.

The API call being recorded as coming from the console is interesting because, rather than the user agent being the browser used to make the request, it's logging the intermediary step used to request it. The console generated the API function on the user's behalf to create the key.

Lastly, in the requestParameters, you can see which username is being included in the request. This means that an API key is being created that will allow the user to have hpym's privileges.



This is the responseElements block that would be returned from the successful createAPIKey function.

Here again we see that the key was created for the hpym account, but more importantly we get the accessKeyID. This is the identifier for the API key made. If we now wanted to investigate what accesses, changes, and deletions this key has made, we could filter down the rest of the logs for this specific API key.

The next two fields, status and createDate, are useful because they prove the key was created and give us the earliest possible time and date it could have been used.

Here are the most common examples you will see shown as user agents in CloudTrail logs. We are not saying this is an exclusive list (AWS can add more at any time), but these are the most common types we see.

Signin: If this is a console login, you will see the signin.amazonaws.com user agent. Note that API keys and other alternative methods of access don't "sign in". Instead, they provide the secret and authenticate on every request.

Console: API calls generated from interactions with the web console will generate console.amazonaws.com user agents in the logs.

Lambda: API calls made from running Lambda functions (the AWS version of serverless compute) will show lambda.amazonaws.com as the user agent in the logs.

AWS-CLI: If someone is using the AWS-CLI to make an API call, you will see AWS-CLI as the user agent, as well as the version and command they executed. If someone is running a script that runs AWS-CLI commands, you will still see AWS-CLI commands.

If someone is running AWS commands through a programmatic API like the Python Boto3 library, then you will see the name of the API library.

If someone is accessing the web console, then the user's browser user agent identifier will be recorded in the logs.

Access Keys Explained

- Access keys have meaning beyond identifiers
- Example: **"AKIAUOLAJ2BLOJUMYJZM“**
- AKIA means Access Key

When you are looking at an access key within CloudTrail logs the prefix of they key itself provides context into how the operation was requested and what is requesting the operation. The first four letters in the access key, in this example AKIA, defines what kind of access key we are dealing with. Is this an access key generated for a specific user (AKIA which it is in this case), is it being generated as an STS token for a service (ASIA as we will see in the next slides), or is this just a IAM account accessing and an access key is being stored in the log but wasn't provided (AIDA as we will see in the next slides)? The letters and numbers that follow the prefix are the unique identifier for that key.

Reference:
https://docs.aws.amazon.com/IAM/latest/UserGuide/reference_identifiers.html#identifiers-unique-ids

Decoding Access Key ID Prefixes (1)

ABIA	AWS STS service bearer token
ACCA	Context-specific credential
AGPA	User group
AIDA	IAM user
AIPA	Amazon EC2 instance profile
AKIA	Access key
ANPA	Managed policy

There are multiple prefixes, and in these next two slides we will highlight the ones that are most relevant to investigations.

If you see AIDA, this is someone who authenticated with an IAM user account and is making requests. It does not mean that an API key for that user was created and is being used via the CLI/SDK.
If you see the AKIA this does mean that someone generated an API key for a user or role and is using that key via the CLI/SDK to make requests.
AWS STS service bearer token:
https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_bearer.html.

Decoding Access Key ID Prefixes (2)

ANVA	Version in a managed policy
APKA	Public key
AROA	Role
ASCA	Certificate
ASIA	Temporary (AWS STS) access key IDs

ASIA is one of the prefixes we look for the most in the case of incidents. This is because attackers will attempt to use the STS service through either the metadata service or other AWS services (as we will see on the API Calls That Return Credentials slide) to generate these temporary API keys that allow them to make requests as that service.

CloudTrail: Finding Exposed API Keys

Role Enumeration

eventtime	eventsource	eventname	sourceipaddress	useragent	requestparameters	eventtype
2021-04-03T18:36:36Z	iam.amazonaws.com	ListRoles	47.185.244.137	aws-cli/2.1.32 Python/3.8.8 Windows/10 exe/AMD64 prompt/off c	null	AwsApiCall
2021-04-03T18:43:36Z	iam.amazonaws.com	ListGroupsForUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"userName":"hpym"}	AwsApiCall
2021-04-03T18:43:36Z	iam.amazonaws.com	GetUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	null	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicy	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"policyArn":"arn:aws:iam::aws:policy/AdministratorAccess"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicyVersion	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"policyArn":"arn:aws:iam::aws:policy/IAMUserChangePassword","versionId":"v2"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	ListUserPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"userName":"hpym"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	ListAttachedGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"groupName":"admin-access"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	ListAttachedUserPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"userName":"hpym"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicyVersion	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"policyArn":"arn:aws:iam::aws:policy/AdministratorAccess","versionId":"v1"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	ListGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"groupName":"admin-access"}	AwsApiCall
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicy	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"policyArn":"arn:aws:iam::aws:policy/IAMUserChangePassword"}	AwsApiCall
2021-04-03T18:44:25Z	iam.amazonaws.com	ListAttachedGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"groupName":"admin-access"}	AwsApiCall
2021-04-03T18:44:25Z	iam.amazonaws.com	ListGroupsForUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44	{"userName":"hpym"}	AwsApiCall

What you are seeing here is an export from the CloudTrail console of events that were recorded from running a privilege escalation script called aws_escalate.[1] You will get a chance to dig through these events in the lab that is coming next.

What is being shown is one specific source IP making a large number of API calls using the Python boto library. These API calls are enumerating roles, listing users, and getting policies, all from one IP address in a matter of seconds. This is not typical activity, unless you are doing an IAM audit looking for all your users and their assigned privileges. This is a very common threat actor technique as they seek to find policies and roles that are over-provisioned with access. It's not uncommon to find prior development fixes that are forgotten about that provide administrator-like privileges to roles and accounts that do not need them.

When attackers find these kind of "shadow admin" roles, they will work to target and steal those credentials in order to take over the cloud tenant.

1. https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/

API Calls That Return Credentials

- chime:CreateApiKey
- codepipeline:PollForJobs
- cognito-identity:GetOpenIdToken
- cognito-identity: GetOpenIdTokenForDeveloperIdentity
- cognito-identity: GetCredentialsForIdentity
- connect:GetFederationToken
- connect:GetFederationTokens
- ecr:GetAuthorizationToken
- gamelift:RequestUploadCredentials
- iam:CreateAccessKey
- iam:CreateLoginProfile
- iam:CreateServiceSpecificCredential
- iam:ResetServiceSpecificCredential
- iam:UpdateAccessKey
- lightsail:GetInstanceAccessDetails
- lightsail: GetRelationalDatabaseMasterUserPassword
- rds-db:connect
- redshift:GetClusterCredentials
- sso:GetRoleCredentials
- mediapackage:RotateChannelCredentials
- mediapackage:RotateIngestEndpointCredentials
- sts:AssumeRole
- sts:AssumeRoleWithSaml
- sts:AssumeRoleWithWebIdentity
- sts:GetFederationToken
- sts:GetSessionToken

When you're doing an investigation of a known compromised AWS resource and you are trying to determine if the compromised system was used to steal API keys that could be abused, then refer to this list.

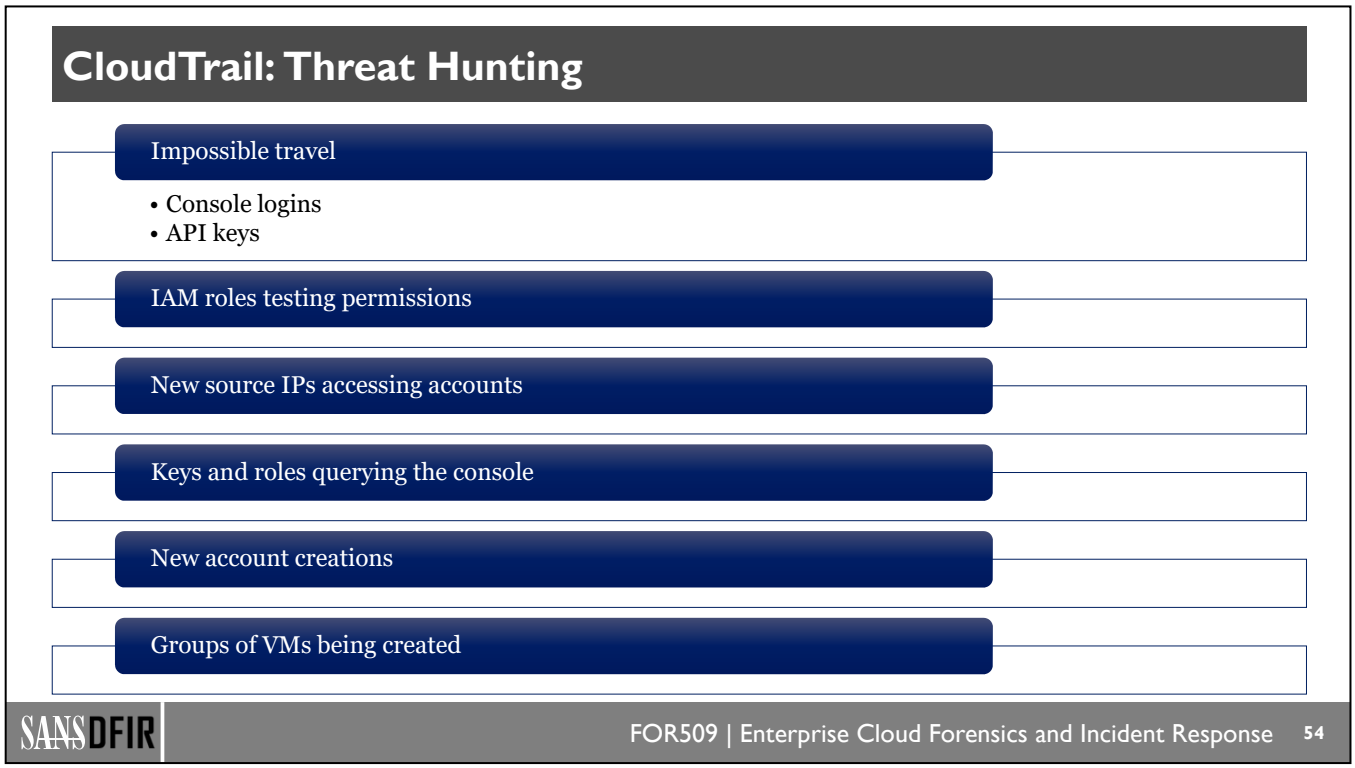
Any of these API calls, if the underlying service is configured and allowed for the assumed role being used, will return an STS token that would allow the attacker to steal/impersonate that service and use its privileges to make additional requests.

This is one of the ways we normally see attackers move laterally within AWS environments, as they hunt for additional keys that will be over-privileged and get them more access.

This list is maintained by **Kinnaird McQuade**.

Reference:
https://gist.github.com/kmcquade/33860a617e651104d243c324ddf7992a

eventtime	eventsource	eventname	sourceipaddress	useragent
2021-04-03T18:36:36Z	iam.amazonaws.com	ListRoles	47.185.244.137	aws-cli/2.1.32 Python/3.8.8 Windows/10 exe/AMD64 prompt/off c
2021-04-03T18:43:36Z	iam.amazonaws.com	ListGroupsForUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:36Z	iam.amazonaws.com	GetUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicy	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicyVersion	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	ListUserPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	ListAttachedGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	ListAttachedUserPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicyVersion	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	ListGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:43:37Z	iam.amazonaws.com	GetPolicy	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:44:25Z	iam.amazonaws.com	ListAttachedGroupPolicies	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44
2021-04-03T18:44:25Z	iam.amazonaws.com	ListGroupsForUser	47.185.244.137	Boto3/1.17.44 Python/3.9.1 Windows/10 Botocore/1.20.44



The following are common threat hunt scenarios you could perform inside of an AWS cloud tenant.

Some of these scenarios are covered in the labs in this class, such as IAM roles testing permissions, keys and roles accessing the console, and VMs being created.

Let's discuss the ones that are not yet covered in labs.

Impossible travel scenarios: This is something you will do in the Azure section. The general idea is that you are comparing logins to the same account from all sources (console, API authentications, CLI executions) and looking at the source IP addresses for scenarios where people's locations changed faster than they could travel between those two geographic locations. In those situations, you could have found:

a) An automation where that user's credentials are being used
b) A shared account
c) An active threat actor who gains access to an account's credentials

Some AWS services like GuardDuty and AWS Detective will highlight these for you. Otherwise, you can write queries in the log platform of your choice (Splunk, Elk, etc.) to make these queries once you have the locations geolocated with something like the MaxMind database.

Another common threat hunt scenario is looking at IPs accessing accounts over time and looking for new and unusual locations appearing. This could indicate a possible account takeover, even without an impossible travel scenario (for instance, multiple sessions coming at the same time from two local internet providers).

One very common scenario we look for is API keys/assumed roles being used for console access. Normally, only IAM users and root users will log in to the console. Seeing threat actors pivot from API calls to console access using API keys or assumed roles should be something you can look for.

Watching for new account creations may be valid if you have a tightly controlled AWS environment. If you are federated or have a large user base, this may not be as useful to you.

Lastly, creating triggers when a large number of VMs are created at the same time, especially if they all have GPUs, is a great threat hunt.

GuardDuty (1)

Now that you've learned how to manually review CloudTrail logs, don't you wish there was an easier way to find that likely threat?

Enter GuardDuty!

GuardDuty only looks forward from the time you enable the service. It does not retroactively go through your stored CloudTrail logs.

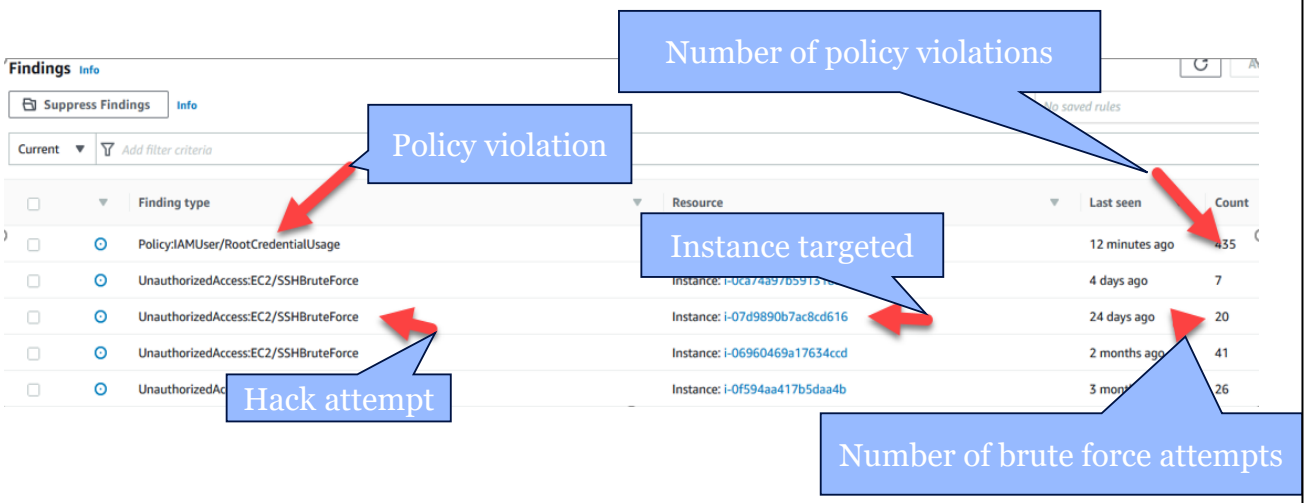
GuardDuty[1] is AWS's pay-per-usage CloudTrail analyzer. They have built within AWS machine learning scenarios of the most common incidents they work through. GuardDuty then takes all of those rules, detections, and alerting logic and scrapes through your CloudTrail logs, looking for possible matches. Unlike Insights, which just does statistical sampling, GuardDuty is looking for patterns of access, or just accesses that are against best practices.

GuardDuty does a very good job and can bring out some major events. However, not every GuardDuty finding is a valid attack. You shouldn't cancel everyone's weekend every time a GuardDuty alert appears, but it could be providing some very useful information, and it can do so at very large scales of logs. I see a lot of clients use GuardDuty as a first-pass triage of their logs, with a subset of CloudTrail logs being shipped off to their SIEM for additional alerting.

Just remember that GuardDuty isn't free. Even our class's small AWS tenant costs $6.68 per day. If I had less S3 activity and thus fewer CloudTrail logs, my cost would decrease dramatically.

1. https://aws.amazon.com/guardduty/

GuardDuty (2)

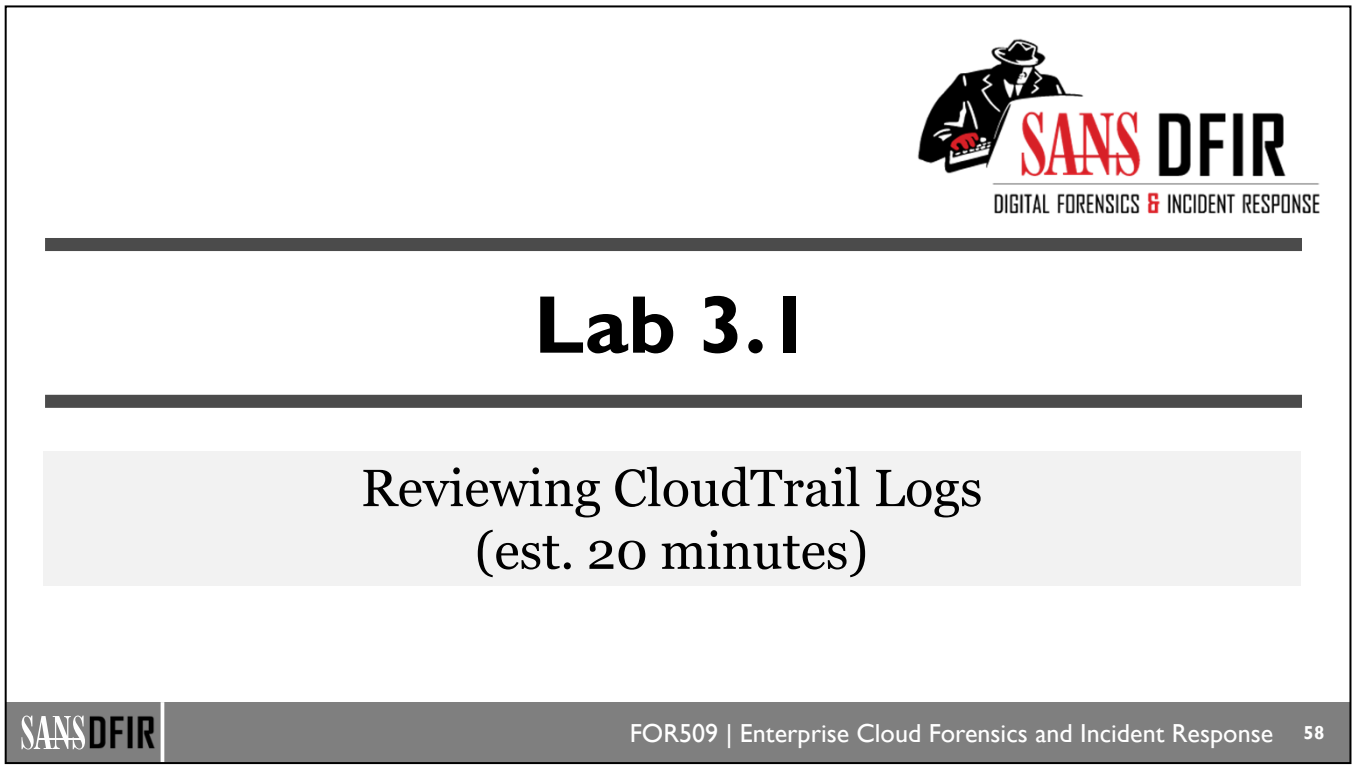


Here are some examples of GuardDuty findings in our class's cloud tenant.

You can see the highest count is Root Credential Usage. This is GuardDuty alerting that the root account has been used to access the cloud tenant 435 times. As we stated earlier, this is not a good practice, but it's something we did to generate data for a class environment. AWS does not recommend using the root account often and will generate alerts like these if your company is following bad practices.

Does this mean the tenant has been compromised? No, not necessarily, but as you can see, the finding starts with "Policy," meaning it's a best policy rule that was matched.

Looking at the rest of the examples, we can see SSH brute force attacks were attempted across four different instances (the AWS word for virtual machines). GuardDuty can't tell if the attacks were successful, as the underlying host logs don't get imported into CloudTrail. Instead, GuardDuty is recording a large number of SSH connections being made to these instances over a short period of time. This is a very common GuardDuty finding that you will find on any publicly exposed SSH or RDP service running within AWS.



This page intentionally left blank.

FOR509.3: Amazon Web Services (AWS)

Section 3.1: Understanding IR in AWS

Section 3.2: Networking, VMs, and Storage

Section 3.3: AWS Native Log Searching

Section 3.4: Event-Driven Response

Section 3.5: In-Cloud IR

This page intentionally left blank.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

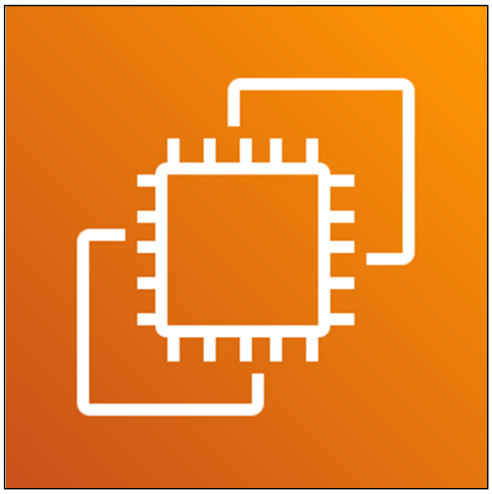
- EC2 Types
- CloudTrail: EC2
- EBS Types
- CloudTrail: EBS
- Snapshots
- CloudTrail: Snapshots
- EFS
- CloudTrail: EFS
- **Lab 3.2**: Finding Rogue VMs

This page intentionally left blank.

EC2: Elastic Compute Cloud

This is the cloud version of a virtual machine:

- AWS provides:
  - Initial tenant creation
  - IP address assignment
- You choose:
  - When you want it to run
  - CPUs, memory, storage
  - How you will maintain it
  - How you will access it
  - How you will secure it



EC2,[1] or Elastic Compute Cloud, is Amazon's way of providing virtual machines.

When you are considering the separation between what AWS maintains and what you control, here is a good way to think about it: AWS provides the IP address assignments; your choice of these IPs is public or private. Based on your configuration (or template selected), AWS will find the resources in their data center and launch that instance for you. If this is an auto-scaling cluster, AWS will even continue to spin up new instances if your demand triggers are set.

When it comes to the configuration of these virtual machine instances, you choose and secure everything else.

You choose how often the instance will run and how many resources (CPUs, RAM, gigabytes of storage) to provide it.

In addition, as you go through the network configuration, you will be choosing how to secure, access, and maintain the instances.

You could place them on a private subnet that isn't reachable publicly.
You could place them behind proxies or load balancers to provide some web application firewalls for additional security.
You could turn them into auto-scaling web solutions with multiple load-balanced routes.

In the end, how you configure it is not AWS's concern; they will bill you based on usage. What is important as an investigator is that you understand how the system was configured so you can scope the impact of a possible intrusion that originates from a compromised instance. Normally, I would start by looking at attached IAM roles and hard-coded keys within scripts on the instances to understand how a threat actor could leverage them into further access within your tenant.

1. https://aws.amazon.com/ec2/

EC2 Types

General-Purpose Prefixes	Compute Optimized Prefix	Memory Optimized	Accelerated Computing	Storage Optimized
• MAC: Bare-metal Macs • T: Burstable • M: Most Scenarios • A: ARM Based	• C: One CPU for every 2GB of RAM	• R: RAM • X: Extra-Large Memory • Z: High Frequency	• P: GPU • G: GPU • F: FPGA	• I: IO Focused • D: Dense Storage • H: Large Local Storage

These are the current EC2 instance types you can choose from.[1]

"General purpose" means that it has a balanced mix of RAM and CPUs.

Within "general purpose" you can find Mac instances; these will have an MAC prefix. Since the terms of the OS X license says you're not allowed to virtualize a Mac outside of a Mac, Amazon had to provide access to actual bare-metal systems. When we say, "bare metal," we mean that the physical Mac computers are in a data center that you are provisioned access to.

Instances prefixed with a T (like the standard t2.micro) are burstable. This means they can be configured to add on additional resources at trigger points you specify.

Instances prefixed with A indicate you have an ARM-based processor in them.
Instances prefixed with C indicate that there are more CPUs than memory.
Instances prefixed with R or X mean they have much more RAM than CPUs.
Instances prefixed with Z are high frequency, meaning super-fast RAM access.
Instances prefixed with P, G, or F indicate that a GPU or FPGA is attached to them. This is the instance type we see most abused by crypto miners.
Instances prefixed with I, D, or H are going to be configured to have super-fast storage or large data storage volumes.

1. https://aws.amazon.com/ec2/instance-types/

Regions Matter!

In the IAM section, we didn't have to talk much about regions, as accounts are valid across all regions.

However, many resources like EC2 VMs and EBS storage are region based, meaning you have to know the region to access them.

Additionally, if you are going transfer data, it is faster to do it within the same region.

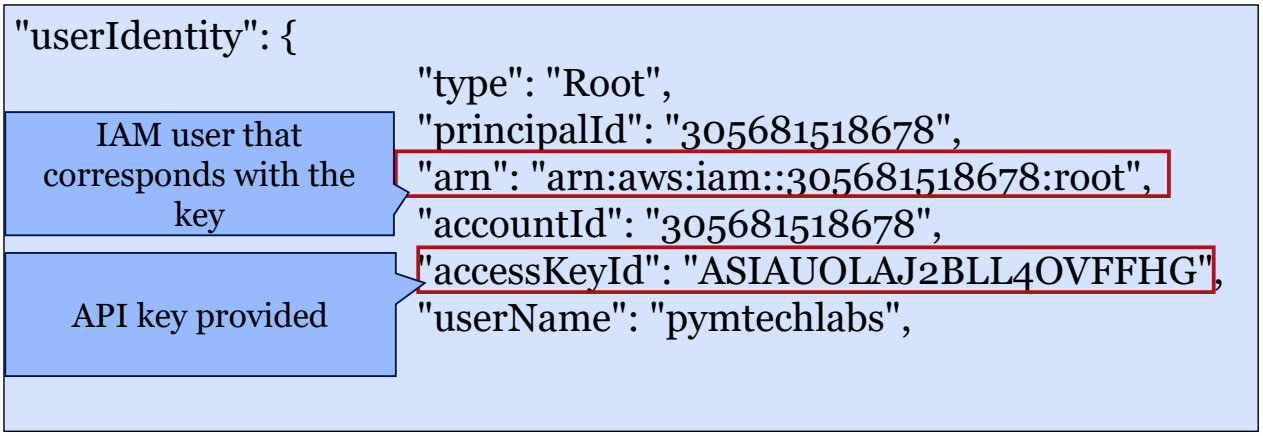
We didn't have to talk about regions in the IAM section because IAM is a global service, meaning that no matter what region your console was set to, you would still see the same data. Another example of a global service is S3, but it has regional specifications.

EC2 is regional, meaning that the instances are running in a specific data center somewhere in the world. As an investigator, this can make your work difficult because when you list out instances, you will only be shown those instances that exist in the region you have selected. This is why it's important to pull the region of out the CloudTrail logs to know in which region you should be looking for a possibly affected resource. It would be very easy to query one region and miss thousands of instances that are running in other regions around the world.

Other critical points are data transfer pricing and speed of transfer. If you're responding to an incident, and that incident is in US East 1, we would recommend standing up your forensic instances in the same region. Not only will the data transfer much faster within the same data center (even between different AWS orgs), but there may be no cost to transfer the data if it does not leave the region.

Another consideration is data residency. You might want to make sure the data stays in the same geographic area to prevent running into issues with regional data privacy laws around the world.

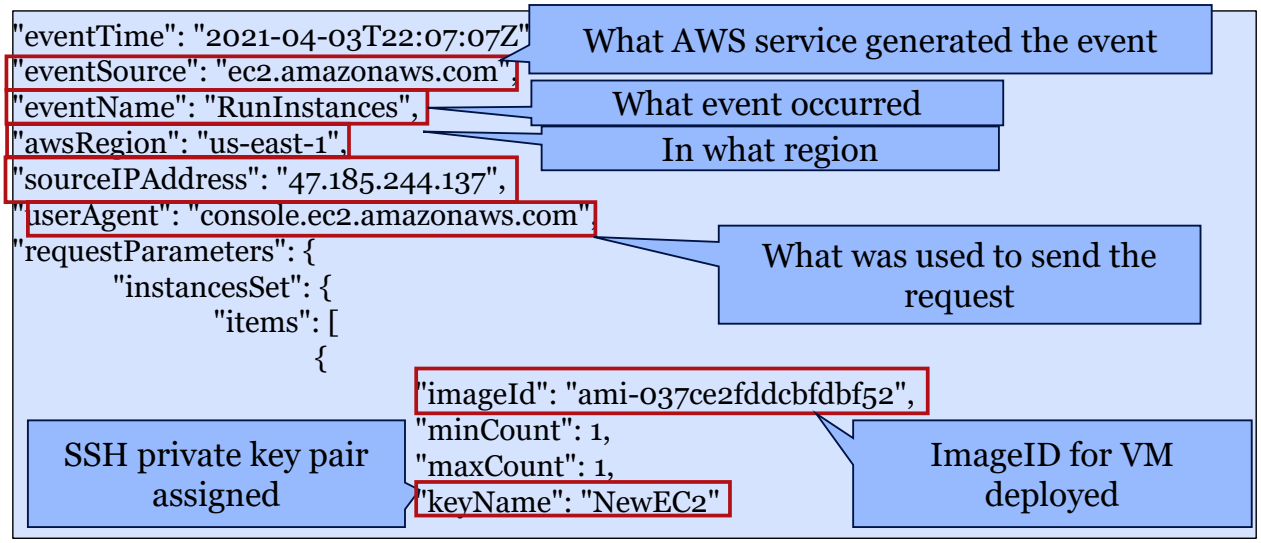
CloudTrail: Creating an EC2 Instance (1)



In this next set of slides, we are going to look at the raw CloudTrail log entry that gets created when you launch an EC2 instance. Launching an EC2 instance will create the virtual machine and create a CloudTrail entry that has a lot of configuration data within it. Within SOF-ELK, we will bring out the most important sub-elements, but to see the full structure, you can query the eventID for the specific log entry with a tool like jq, which we use in Lab 3.4.

The two important fields we are calling out here are the ARN (which account is authenticating to create the EC2 instance) and the accessKeyId (which shows which API key is being passed in). We can pivot off either of these to find recorded actions that occurred before or after this event, but I normally use accessKeyId, as the ARN would show more than just API entries.

CloudTrail: Creating an EC2 Instance (2)



eventSource: All EC2-related events will have an event source of ec2.amazonaws.com.

eventName: This is the actual event that was recorded (in this case, RunInstances). RunInstances shows that an EC2 instance was created.

awsRegion: This shows which region our instance is running in (in this case, us-east-1). If we wanted more data about this instance, or to make a snapshot of it, we need to know the region to specify.

sourceIPAddress: As before, this is the IP that was logged making the API call.

userAgent: Here we see console.ec2.amazonaws.com instead of just console.amazonaws.com. This means we are specifically logged in to the web console and using the ec2 web console to make this request.

imageId: This is the unique identifier for this instance. If you had an auto-scaling group and the same configuration was launched hundreds of times, then each would get its own unique imageId. The imageId is prefixed with AMI, which stands for Amazon Machine Image. AMIs are like VMX files in VMware. They contain all the configuration-related data needed to launch a virtual machine. You can use AWS template AMIs or upload your own.

keyName: This is a Linux instance, and there is an SSH key created for accessing it when you create the instance. In this case, this is letting us know that the SSH key we would need to access the instance is called NewEC2.

CloudTrail: Creating an EC2 Instance (3)


```
"attributes": {
                "mfaAuthenticated": "true",
                "creationDate": "2021-04-01T12:03:50Z"
                        "groupSet": {
                                "items": [
                                        {
                                        "groupId": "sg-00b98f2f5d3339900"
                                        },
                                        {
                                        "groupId": "sg-044c410b2327cc21f"
                                        }
                                        ]
                },
                "userData": "<sensitiveDataRemoved>",
                "instanceType": “t2.micro",
```


The type of EC2 VM that was created

The important part of the attributes shown here is the instanceType. This relates back to the list of instances types we discussed earlier in this section.

In this case, our launched instance is a T type (or burstable type) instance and is a second-generation micro-instance. This means it is following a configuration that has very few CPUs and a small amount of RAM. You will see these a lot because they are eligible for the AWS free tier, meaning they have no cost to run unless they exceed a threshold of resources.

CloudTrail: Creating an EC2 Instance with a GPU


```
"attributes": {
                                        "mfaAuthenticated": "true",
                                        "creationDate": "2021-04-03T18:03:51Z"
                                                "groupSet": {
                                "items": [
                                        {
                                                "groupId": "sg-00b98f2f5d3339900"
                                        },
                                        {
                                                "groupId": "sg-044c410b2327cc21f"
                                        }
                                ]
                        },
                        "userData": "<sensitiveDataRemoved>",
                        "instanceType": "g4ad.4xlarge",
```


A GPU-enabled VM was created

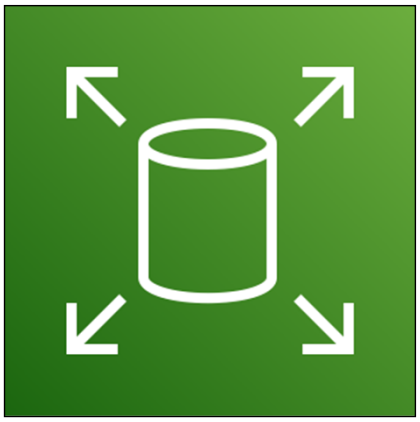
In this example, we see an instance being created with a G prefix, meaning it has a GPU attached.

In addition, you can see it is a "4xlarge," meaning it has a large number of GPUs, CPU, and RAM attached to it.

These are the kinds of instances you can expect to see when a crypto miner incident is occurring.

EBS: Elastic Block Store

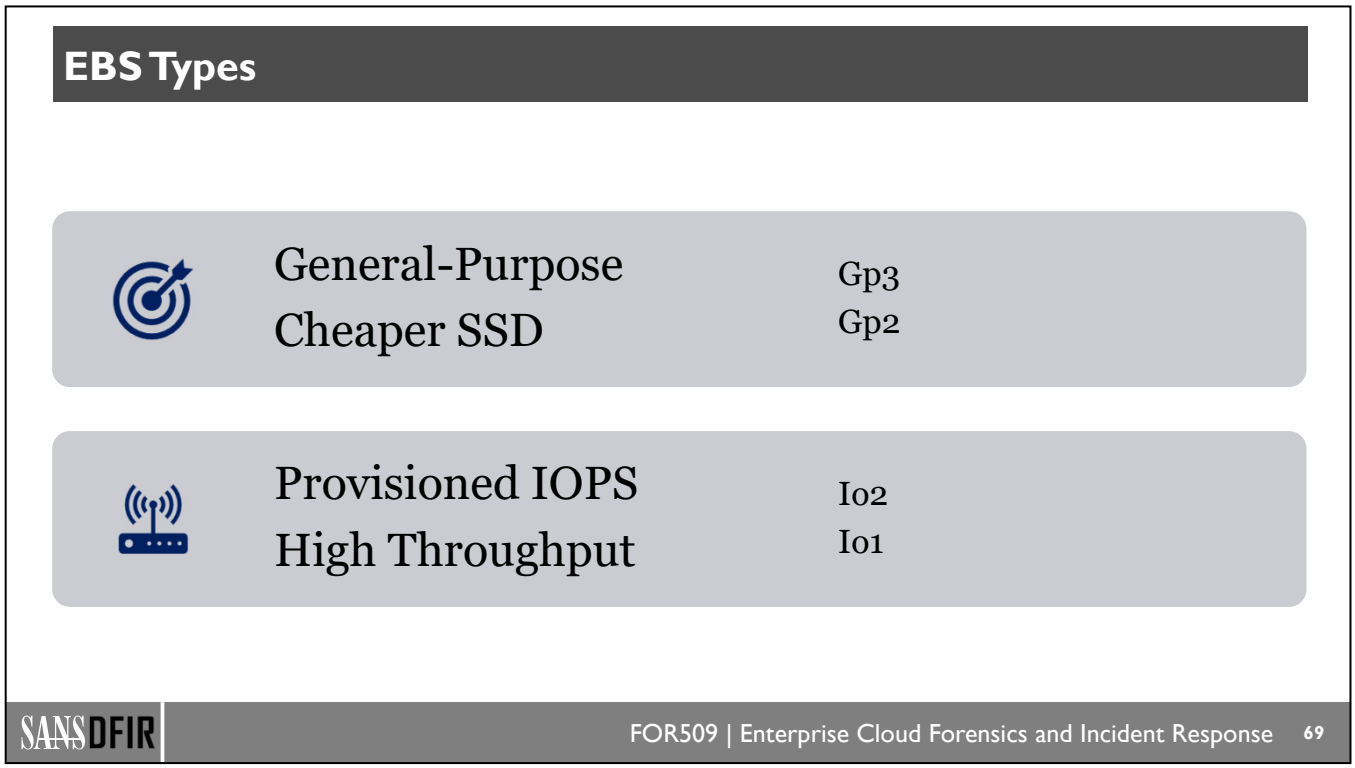
- Virtual storage for EC2
  - In virtual machine terms: VMDKs, VDIs, VHDs
- You can have as many virtual storage devices as you need



AWS refers to storage volumes for EC2 instances as EBSs,[1] or Elastic Block Stores.

If you are using another virtualization platform such as VMware, it would be a VMDK; VirtualBox would be a VDI, and Hyper-V would be a VHD. In any case, these are the storage volumes accessible to your instances as disks. All EC2 instances will have at least one EBS volume for the operating system, but they can have as many as needed.

1. https://aws.amazon.com/ebs/



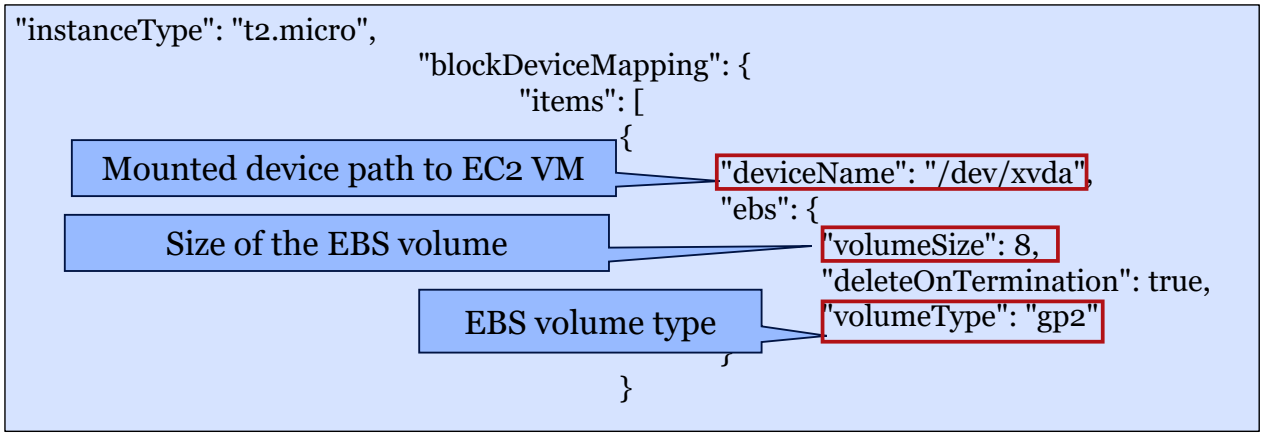
EBS volumes currently come in four types:[1]

General-purpose SSD: This comes in EBS volumes with types GP3 and GP2. Which GP is selected for the EC2 instance depends on the disk speed needed. These are the most common EBS volumes you will see deployed.

Specialized IOPS disks: This comes in EBS volumes with types IO2 and IO1. These disks provide guaranteed faster transfer rates and higher throughputs.

1. https://aws.amazon.com/ebs/volume-types/

CloudTrail: Creation of EBS



We are fairly deep within the RunInstance CloudTrail entry now to get to the EBS volume attached. We are within parameters, below the instance type, and in a sub-block called blockDeviceMapping. Within this is an array of EBS volumes that contain the following:

deviceName: The path to the operating system where the volume can be accessed

volumeSize: How large the volume is (in this case, 8GB)

volumeType: The kind of EBS volume (in this case, a general-purpose SSD)

Snapshots

- Snapshots create a disk image of the current state of the disk through the hypervisor
- First snapshot contains all the data
- Subsequent snapshots only contain differences on a per-block basis
- Copying snapshots between regions will get hit with a data transfer charge ($.01 - $.02 per GB)
- The DirectBlockAccess API allows you to access individual blocks of a snapshot without having to restore the entire snapshot

Most virtual hypervisors provide some kind of snapshot[1] functionality. AWS is no different and provides a mechanism to create differential snapshots of EBS volumes over time. The first time you create a snapshot, it will create a base image of the disk. Each subsequent snapshot created will only contain the delta of what changed on a per-EBS-block basis. This is much like how Windows shadow copies work.

When you are working with snapshots, you can work with just differential data between snapshots, or you can get the consolidated view of the whole disk at that time.

Copying snapshots between regions will cost a data transfer fee per copy. This again goes back to our earlier point of trying to keep your forensic environments in the same regions as your resource to prevent unnecessary costs as you perform your analysis. While the fee is small (.02 per GB), it can add up quickly.

DirectBlockAccess is an API that provides the ability to read individual blocks from any individual snapshot. With this API you can do a live triage of a snapshotted disk that allows you to read the individual block rather than mounting the snapshot on another instance. The idea here is that you could script out rapid triage across snapshots without going through the time or expense of restoring, mounting, and processing them in bulk.

1. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html

Want to Download a Snapshot?

- Using the DirectBlockAccess API you can download a snapshot without restoring it.
- AWS has made a tool available called 'Coldsnap' to perform this function.
  - https://github.com/awslabs/coldsnap
- Be aware that snapshots are differential as stated before. The snapshot you download only contains the blocks that changed since the last snapshot taken.

```
coldsnap --region <region> download <snapshot id> image.dd
```


One of the most interesting uses of the DirectBlockAccess API is its ability to allow you download data from snapshots directly without having to restore them first. While I first though this would be most useful for triaging data across large amounts of snapshots, others have realized this also allows you to download entire snapshots as disk images.

If the snapshot you download is first or only snapshot of a system then it will be a full snapshot and the image you download will be like any other full forensic image.

If the snapshot you download had multiple snapshots before it then it will only contain those blocks within the snapshot that have changed since the last snapshot. This means that it will not be a full forensic image and cannot just be loaded into standard forensic tools for analysis.

If you would like to be able to access a snapshot as a fully restored disk you will still have to restore the snapshot to a new volume. When you do the restore AWS's backend will layer the data from the previous backups to create a picture in time view of that disk when the snapshot was created. Once restored you can create a forensic image of the restored volume in order to have a full forensic image of the snapshotted disk.

Snapshot Pricing

- Snapshot Storage: Charged on total space allocated; free space is not included
  - Free Tier
    - 1GB of snapshot storage
  - Paid Tier
    - $.05GB/month of data stored
- Fast Snapshot Restoration
  - $.75 per 1 DSU (data service unit)
- Direct Block Access
  - List Blocks: $.0006 per thousand requests
  - Get Blocks: $.003 per thousand units
  - Put Blocks: $.006 per thousand units

There can be a cost associated with snapshot creation.

A free tier is provided that allows for one gigabyte (1GB) of storage, and that's for each individual snapshot, meaning that as long as your differential snapshots are small, they may not be charged to you.

Once your snapshots begin to grow beyond 1GB, they are charged at five cents a gigabyte. Again, this is for the differential; in other words, how much data has changed between snapshots.

As an example, if your VM operated for a month, and you had 500 megabytes of new data on the disk and made a snapshot, then the snapshot is still in the free tier. There would have to be more than a gigabyte of new data loaded onto the volume and recorded into the second snapshot to exceed the free tier, in which case you would be charged five cents a gigabyte for every gigabyte of new data in that snapshot. This fee, though, is charged monthly. So, if you had a 2GB snapshot, you would be charged 10 cents a month as long as the snapshot existed.

If you made a snapshot and it was 100GB in size, it would cost $5 a month to store the snapshot.

If you don't want to wait for standard restoration times, you can pay for fast restoration. Fast restoration is charged at 75 cents per data service hour.

The **Direct Block Access API** is charged at .0006 cents per 1,000 requests to list out blocks stored within the snapshot, as well as $.003 per 1,000 blocks (think disk sectors) retrieved, and $.006 per 1,000 blocks stored in the snapshot. All of these prices are in USD.

Reference:
https://aws.amazon.com/ebs/pricing/

Making a Snapshot with the CLI

```
C:\aws ec2 create-snapshot --volume-id <volumeid>
      --description "Making a snapshot"
{
    "Description": "Making a snapshot",
    "Encrypted": false,
    "OwnerId": "305681518678",
    "Progress": "",
    "SnapshotId": "snap-096a35e714244ea98",
    "StartTime": "2021-04-17T21:22:48+00:00",
    "State": "pending",
    "VolumeId": "<volumeid>",
    "VolumeSize": 8,
    "Tags": []
}
```


Here, for the first time, we are showing how to use the AWS CLI. We've discussed CloudTrail accesses being recorded from CLI accesses, but not how to use the CLI itself.

You can download the CLI from AWS and then, after configuring it with your API key, you can make requests to your AWS tenant from the command line.

In this example, the bolded text shows the CLI command we executed.

We executed "aws" (the name of the CLI command). We specified that we are accessing the "ec2" service and calling the "create-snapshot" command. The rest are parameters to the create-snapshot command, such as the volume-id of the individual EBS volume we are looking to snapshot and a description we can store with the snapshot so that in the future we can remember why we made it.

What follows within the {} text is the output from running the command. This lets us know that the snapshot is in process and is providing us the SnapshotID we need to access this specific snapshot in the future.

Accessing Snapshots from DFIR AMIs

- First, you need to create a volume out of the snapshot:

```
aws ec2 create-volume --availability-zone <zone where your DFIR
AMI is running>  --snapshot-id <snapshot-id>
```

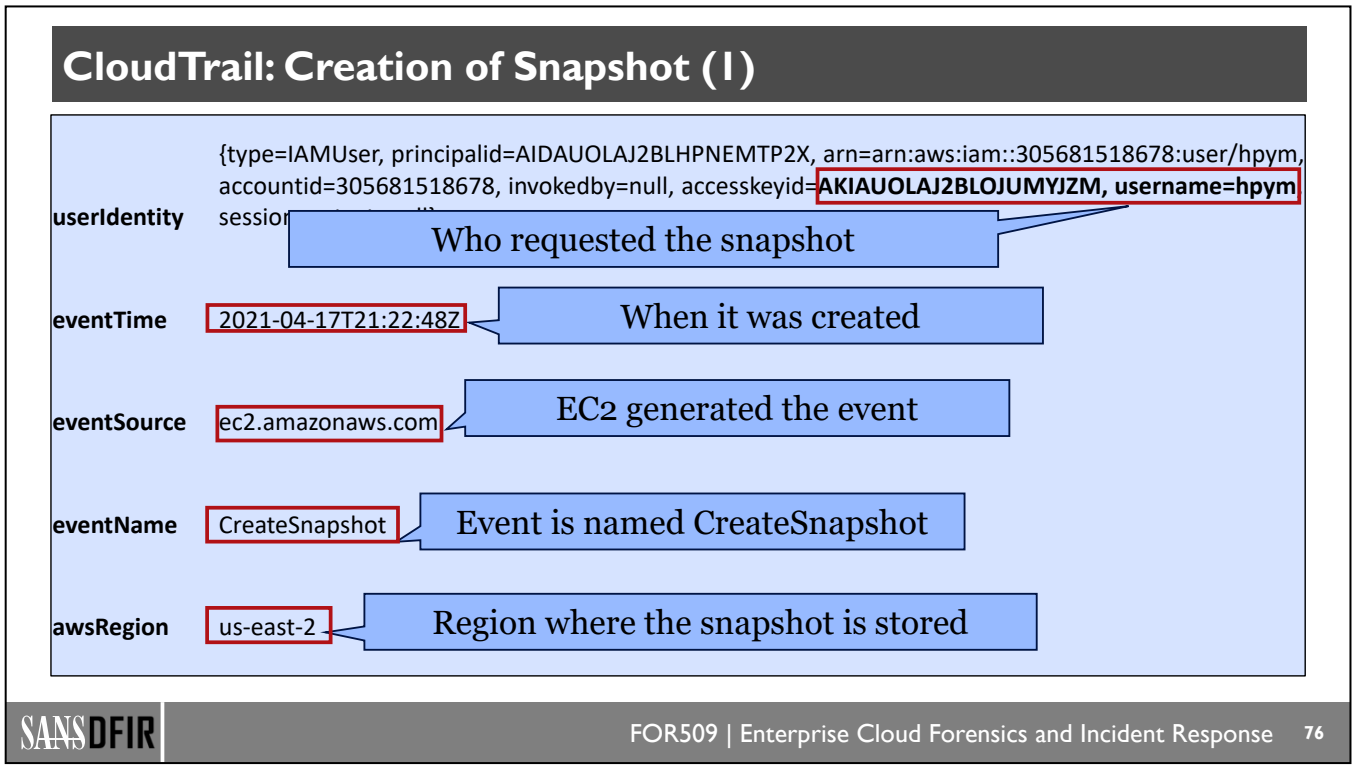
- Second, you have to attach it to your running EC2 instance:

```
aws ec2 attach-volume --volume-id <volume id returned from prior
command> --instance-id <your DFIR EC2 instance> --device
</dev/sdX>
```


From the API or CLI, you can create a volume from a snapshot. In the prior slide we showed how to make a snapshot; now we are showing how to take that snapshot and bring it back into a volume you can do analysis on within the DFIR EC2 workstation.

The first command, create-volume, is going to ask for two options. The first is the region where your EC2 instance is located, and the second is the snapshot-id of the snapshot you want to analyze. Once the command runs, it will wrap that snapshot into an EBS volume.

Now you need to attach the new EBS volume that contains the snapshot data to your DFIR EC2 workstation. To do that, we use the attach-volume command. You need three options here. The first is the volume-id, which is provided in the results of the prior command. The second is the instance-id, where you would provide the instance-id of the running DFIR EC2 workstation you want to use. Lastly, you need to give it a device path you want the volume to appear as. Once the command is executed, the data is available for analysis.



This is what it would look like in CloudTrail if you created a snapshot:

userIdentity: Who requested the snapshot. In this example, it's an IAM user named HYPM. In addition to the username is the acceskeyid, which represents the API key that was used, along with the full ARN of the account used to authenticate.

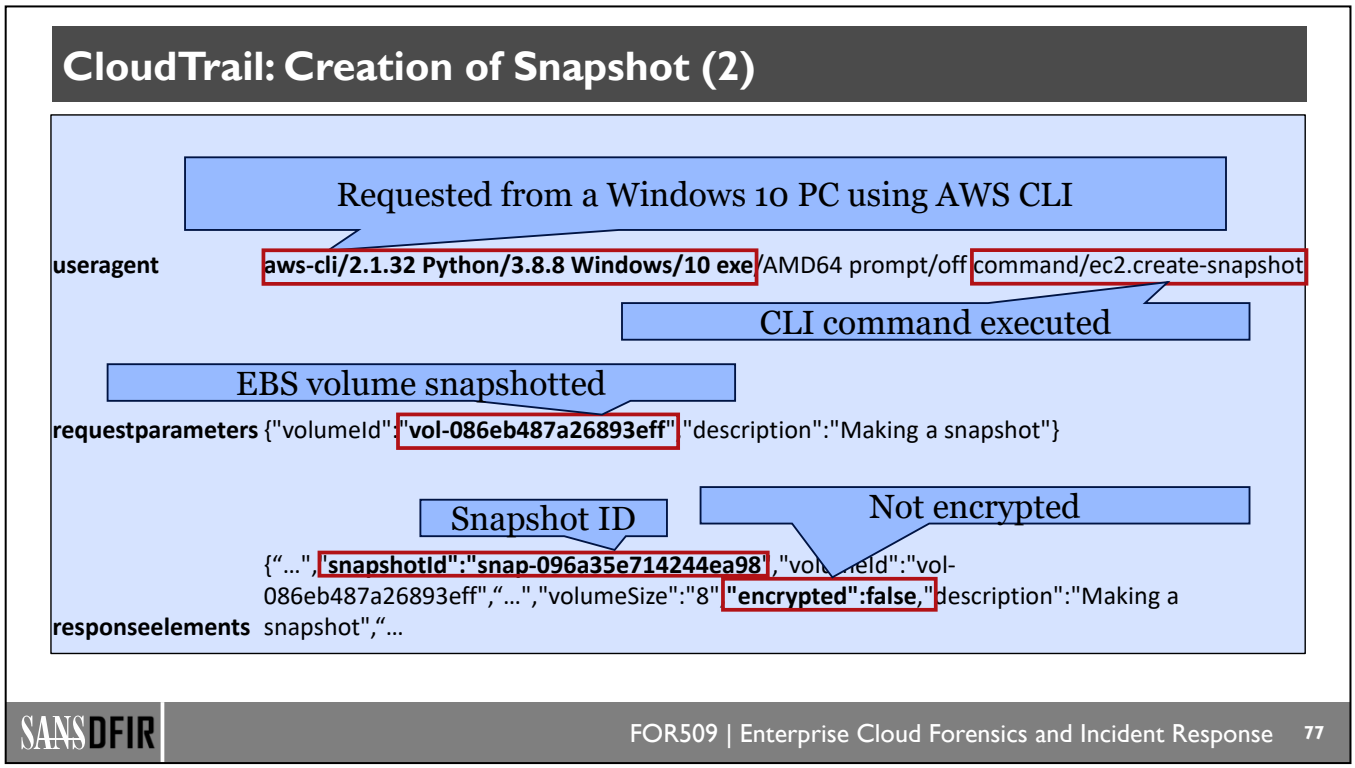
Notice this is the same accountID we saw in the root account ARN, but now it's associated with an IAM user account. That is because this IAM user is in the same cloud tenant, so the account number won't change. The account number would change if this CloudTrail log was from a different AWS account in the same org.

eventTime: When the snapshot was created.

eventSource: The event source is actually EC2. EC2 is the event source for this snapshot because the volume it's accessing is inside of EC2, and the snapshot will be stored in the EC2 service.

eventName: CreateSnapshot (eventName records the actual API call made).

Awsregion: Snapshots are regional, not global. Make sure you remember that and make a note of what region your snapshot is stored in. Otherwise, if you're in the wrong region, you won't find the snapshot you're looking for.



Here are some additional fields of interest from a CloudTrail record for creating a snapshot:

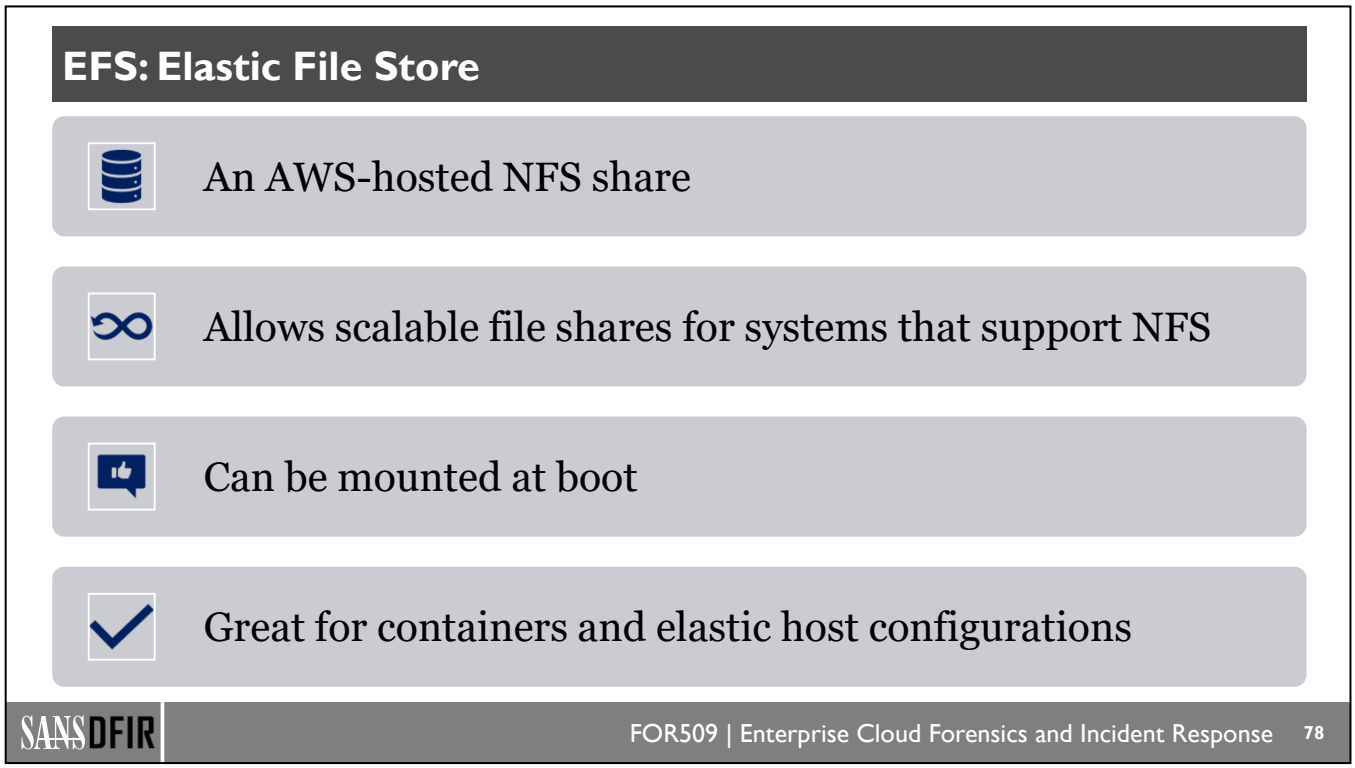
Useragent: In this example, you're going to see the user agent for the CLI. In this case, we made use of a Python-based CLI for Windows 10 that Amazon provides. Notice that, at the end of the useragent string, **it will also record the exact command that was executed in the CLI.**

You will not get all the parameters to the command, but you do get the command that would be executed that you passed in as part of the useragent string. If you have a threat actor who's iterating through things and you're trying to understand the bigger picture of what caused the ripple of commands and API interactions you're seeing recorded, these underlying CLI commands and their sequence could help.

Volumeid: This is the EBS volume that was snapshotted.

Snapshotid: This is the ID needed to retrieve the contents of the snapshot.

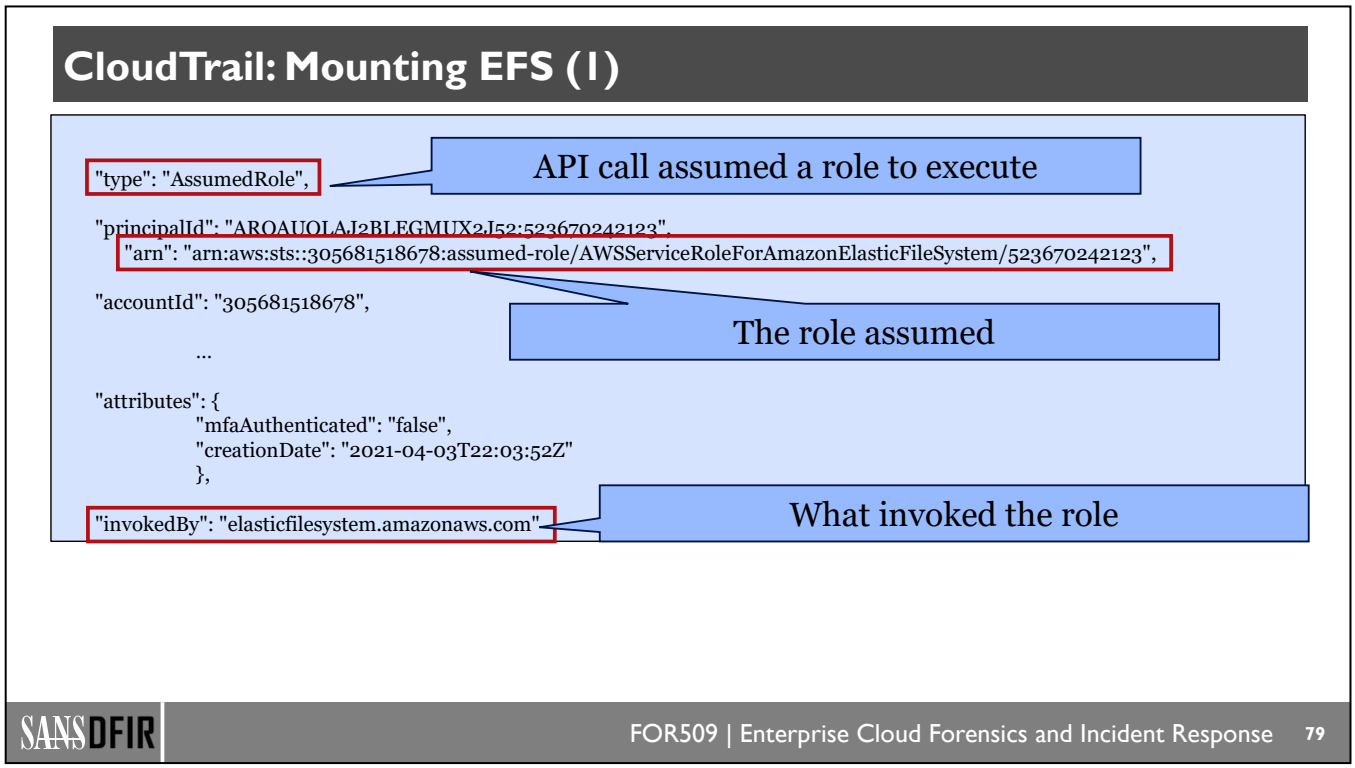
Encrypted: Indicates whether the snapshot was encrypted. Just remember that if the key was stored in the AWS Key Management service and the account that is accessing the snapshot has access to the key, then all the data will be decrypted transparently. We bring up this point to make sure you don't have the mistaken opinion that encrypted snapshots would not be accessible to a threat actor.



EFS[1] and EBS are very different: EFS is basically a network-hosted NFS share, while EBS is a dedicated block storage device. This means that multiple instances can all access a EFS share simultaneously, while only one instance can access an EBS volume at a time.

Many developers will create an AWS-hosted NFS share (EFS) and then attach it to multiple running EC2 instances. This way, for auto-scaling clusters and containers, they can have persistent data store. This allows multiple Linux instances and containers to read and write data to a central place, and then AWS will manage auto-scaling of EFS storage, and since EFS shares are regional, the data access speeds should be fast.

1. https://aws.amazon.com/efs/



Here, we see a CloudTrail event for creating an EFS share in AWS.

type: This lets us know that the action took on AssumedRole to complete. This is useful because you may have seen assumed roles in Lab 3.1, but they are not always malicious. In this case, our account is assuming the EFS role in order to create a network interface that will be bound to our subnet, which allows us to access the EFS share.

arn: This is the role being assumed.

invokedBy: This is the AWS service that invoked the EFS role.

CloudTrail: Mounting EFS (2)

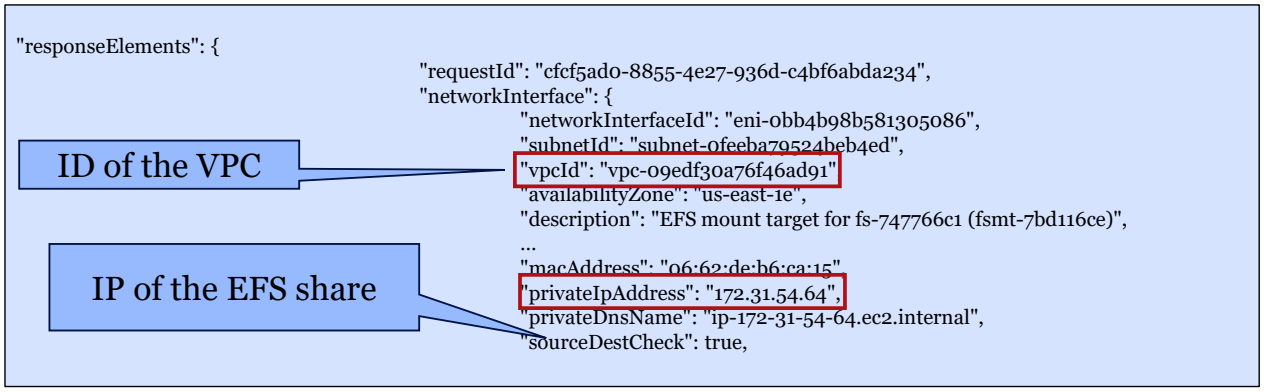
This is additional data found in the same CloudTrail entry we saw in the prior slide. Here we are highlighting additional fields that may be useful in an investigation and to understand what the log is telling us.

eventName: This is the actual API call made. In this case, it's showing that this log was created as the result of calling the CreateNetworkInterface API. This is a "create network interface", not "create EFS", because AWS maintains the overall EFS cluster. We are creating a network interface that allows the machines on this subnet to access the EFS cluster that AWS maintains, not a new EFS cluster.

awsRegion: This is the region where the EFS data should be located and where the network interface is located.

userAgent: This shows us that the user agent is elasticfilesystem.amazonaws.com, since that is what is making the call for us.

CloudTrail: Mounting EFS (3)

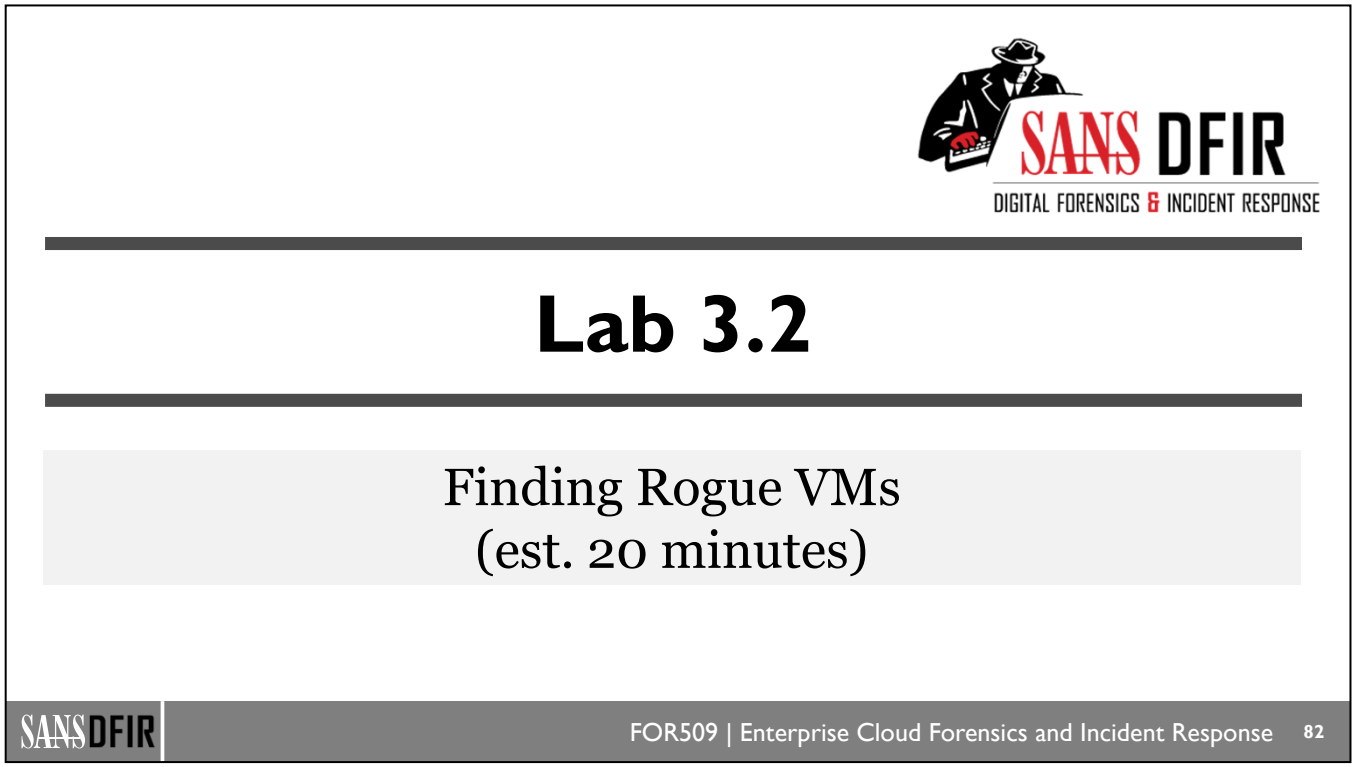


This is additional data found in the same CloudTrail entry we saw in the prior slides. Here we are highlighting additional fields that may be useful in an investigation and to understand what the log is telling us. This data is found in the very nested responseElements structure with the JSON CloudTrail entry. We do not pull out all of this underlying data within ELK. Instead, you would need to use a tool like jq to pull out this data.

vpcId: This is the ID of the VPC (virtual private cloud) network that you would need to reference to find traffic logs for this EFS share. We cover VPCs in the next section, but what you need to know is that flow logs are enabled on a per-VPC basis. In order to find out where flow logs that show traffic to and from this EFS share exist, you would need to know this VPC ID.

Why would we want to look at flow logs of EFS shares? If you believe that a threat actor has taken data from your EFS shares and exfiltrated it out of the network, then the flow logs from this specific VPC should allow you to see what traffic is occurring. Otherwise, the underlying instance may not have any evidence showing the activity since it would occur over the network itself and not on the host.

privateIpAddress: This is the IP address assigned to the EFS share. You would use this to identify EFS traffic flows in your logs.



This page intentionally left blank.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- VPC
- VPC Subnets
- Internet Gateways
- Load Balancers
- VPC Flow Logs
- VPC Flow Log Examples
- VPC Flow Log Storage
- VPC Flow Log Pricing
- Route 53
- **Lab 3.3**: VPC Flow Logs

This page intentionally left blank.

Virtual Private Cloud

- A collection of resources placed into a group
- A VPC can contain:
  - Subnets (Private/Public)
  - EC2 VMs
  - Network Security Groups
  - Internet Gateways
  - Load Balancers
  - Web Application Firewalls (WAFs)

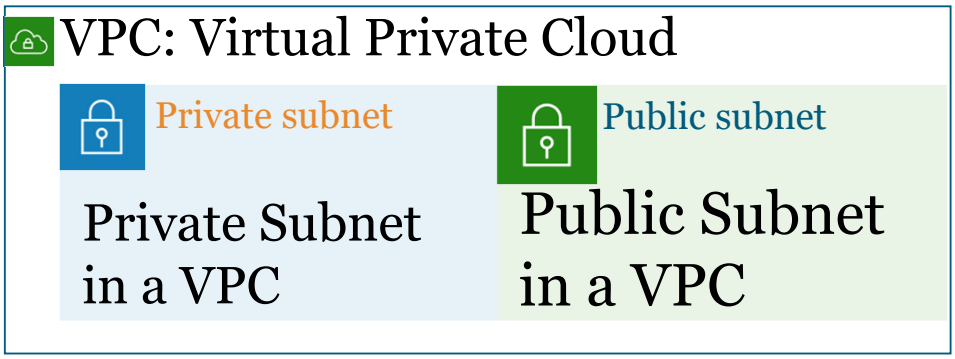
A **VPC**, which stands for virtual private cloud, is a grouping of resources that have or need network connectivity.

A VPC contains one or more of the following:

- **Subnets**: A subnet is an individual network segment within the AWS IP space. This is just like a subnet inside of a traditional on-premises network.
- **EC2 VMs**: The network interface of an EC2 VM must be assigned to a subnet, which is assigned to a VPC. EC2 instances within the same subnet can communicate without additional routing.
- **Network security groups**: These can be assigned at the subnet level and are basically AWS firewall rules for traffic entering or leaving your subnet.
- **Internet gateways**: These are connection points that route traffic between your VPC and the internet. Without one, you can't route traffic in from the internet or out to the internet.
- **Load balancers**: These are AWS-provided virtual network devices that allow you to intercept requests from external sources and route them to your instances to be able to handle large volumes of traffic. These come in different varieties that we discuss in the coming slides.
- **WAFs**: Web application firewalls (WAFs) are virtual security devices that inspect web application requests and filter out possible malicious content.

Read more about VPCs: https://aws.amazon.com/vpc/.

VPC Diagram



NOTE

VPCs are like a single network switch. Without any gateways they have no network access outside of the subnet.

This is a diagram meant to help you understand how subnets are grouped into VPCs.

Think of a VPC like a switch, except with the capacity to have 200 VLANs, or virtual LANs (a.k.a. subnets), working within them.

Each subnet can be configured differently, but they are grouped into VPCs on a per-project basis normally.

In this example, we are showing a public subnet and a private subnet in the same VPC. We do this to show how you can have two very differently scoped subnets within one VPC.

A private subnet has one of the private RFC 1918 address spaces applied (currently 10.x.x.x) to the AWS resources applied to the subnet. However, traffic, by default, can't leave the subnet. It can only communicate within the subnet, and there is no ability to access the internet from the private subnet, and in addition there is no way for internet traffic to reach the private subnet.

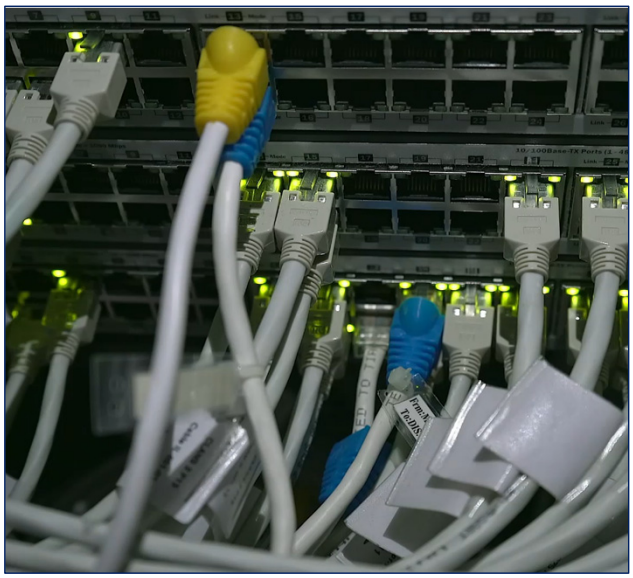
This is contrast to a public subnet that comes with an internet gateway configured by default that allows traffic to route to and from the internet.

Even though both of these subnets are within the same VPC, they can't route to the internet unless you specified some kind of gateway that would allow traffic to pass. This means each subnet is isolated from the internet without an internet gateway (which comes with a public subnet by default). So, what is the point of a VPC? The VPC is a logical grouping of subnets that allows you to have a project where you can have a private network of secure processing systems while also having the publicly accessible front end still grouped into the same VPC.

At a VPC level, you can define network security groups (firewall rules), mirror traffic to another VPC, record traffic in PCAP form to S3, and enable flow logs. You can also do additional security on the subnet level, but this logical structure lets you build a top-down security model.

VPC Subnets

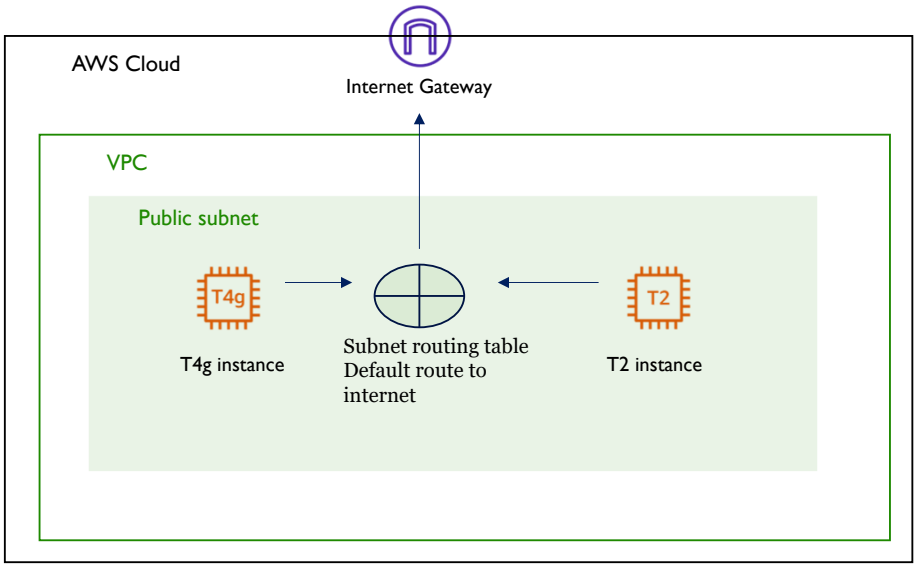
- Subnets within VPCs
  - Within a VPC you can have both public and private subnets
  - What makes a private subnet is no direct routing ability to the internet
  - What makes a public subnet is a default route that leads to the internet
  - Subnets can have different security policies and still live in the same VPC
  - VPCs are like VLANs; and each subnet has its own IP address space
  - You can create routes between VPCs; by default they are isolated



You can logically group your subnets in any shape that works for you. You could create an IR subnet and place all of your DFIR resources there so they can easily talk to each other. Conversely, you could create separate subnets—one for processing evidence that dumps data onto an EFS share that another analysis subnet would have access to for review. This is a way to virtually separate these network resources so you can easily maintain and secure them based on what their need is.

VPCs provide a way to group together up to 200 subnets into logical projects. For example, you could have an IR VPC, database VPC, and a VPC for a specific public application you are running. A small organization might just have one VPC for the whole company. The benefit of this logical grouping of subnets into VPCs is the ability to uniformly apply network security groups to those subnets that exist between them. This becomes even more interesting because you can dynamically change this configuration at any time to meet a need.

Internet Gateways



NOTE

Internet gateways allow for routing from VPC public subnets to the internet.

They also allow internet addresses assigned to your resources to be accessible.

This diagram is here to help you visually understand how internet gateways work.

The actual internet gateway is on the external perimeter of the AWS Cloud network. When you create an internet gateway in your subnet, you are creating a route between your public subnet and that internet gateway. Typically, this is the default route, meaning that all traffic not in this subnet or some other defined route will be sent to the internet. Note that this is not a NAT gateway, so your network security logs, flow logs, and host application logs will contain the actual external IP addresses that are being routed into your subnet. In addition, your network security group rules will define what traffic to and from the internet is allowed.

If you have a subnet without an internet gateway, it can't be directly accessed from the internet. If you have a private subnet with a NAT gateway, then it can reach the internet, but the internet can't route to it.

Read more about internet gateways:
https://docs.aws.amazon.com/vpc/latest/userguide/VPC_Internet_Gateway.html.

AWS Load Balancers

- AWS load balancers work one of three ways:
  - Network load balancer
    - Balances loads across multiple VPCs
  - Classic load balancer
    - Balances traffic by routing it based on weights or rotation
  - Application load balancer
    - Understands HTTP and can route requests between sets of web servers
- Load balancer logs are application-level logs. They get written to an S3 bucket you specify.

NOTE

If an AWS load balancer is in place, the external IP address of the attacker may not be passed on to the underlying instance you are analyzing.

This includes flow logs and application logs.

Make sure to get load balancer logs!

Amazon has three types of load balancers[1] they offer as virtual network appliances. Amazon maintains them but you can configure them.

Network load balancer: This load balancer takes a network request and then will route the traffic to different VPCs based on the configuration. The network load balancer can make sure to only send traffic to VPCs that are currently operational so that a failure in one VPC does not cause downtime to the user. The limitation is that a network load balancer only understands requests at the network transport level and not at the application level.

The classic AWS load balancer: This load balancer can understand network requests as well as deconstruct the application layer to interpret requests. You can use it to rotate requests to different web server pools in different VPCs. However, it is not as fully featured as the AWS application load balancer.

Application load balancer (ALB): This load balancer can actually interpret the requests and route things based on the contents of the application request so you can route traffic based on URL queries to specific servers. In addition, it has the full nginx backend for sophisticated configurations and integrations of web application firewall rules.

We discuss load balancers in this class because of the application and classic load balancers. If you are doing an investigation, and the compromised environment has one of these load balancers in the path of a request, your endpoint logs will be incomplete.

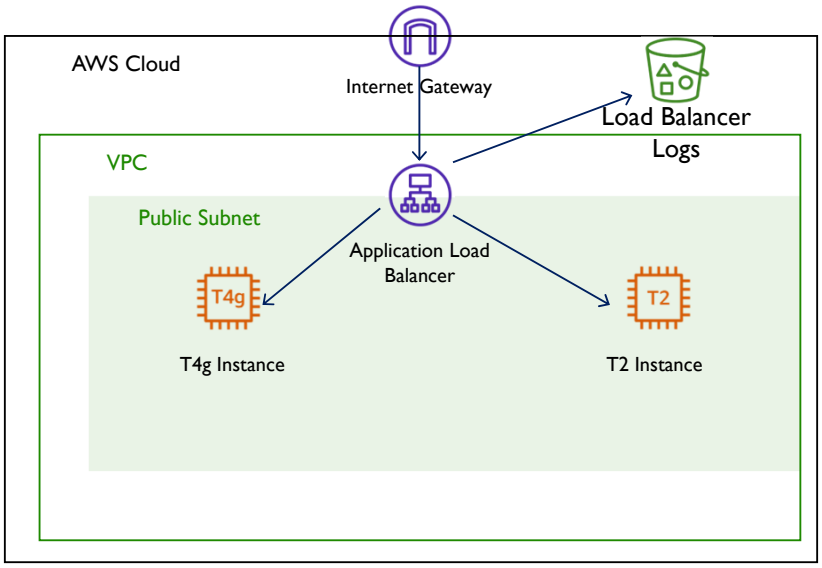
Why? The load balancer is what is accepting and recording the connections to the public source IP address and then proxying the connection internally to your VPCs. This means your VPCs will record the internal IP address of your load balancer rather than the actual source IP address accessing the load balancer.

1. https://aws.amazon.com/elasticloadbalancing/

In addition, the flow logs of the subnet where the load-balanced traffic is being sent to will not see the public source IP address; they will see the source IP address as the load balancer because of the same proxy behavior. You will need the load balancer logs in order to see the public IP address that was making requests to the application.

It's very common for load balancers to be used in large web applications, especially in front of your large clusters or auto-scaling group. If those kinds of systems get compromised and you want to figure out the source of the attack, there's a good chance that the initial logs you're going to get aren't going to be the logs that will help you find the source. Instead, they will show the attack, but they won't show where the attack is coming from. In order see the full logs and determine the actual source, you have to get the load balancer logs.

Load Balancer



NOTE

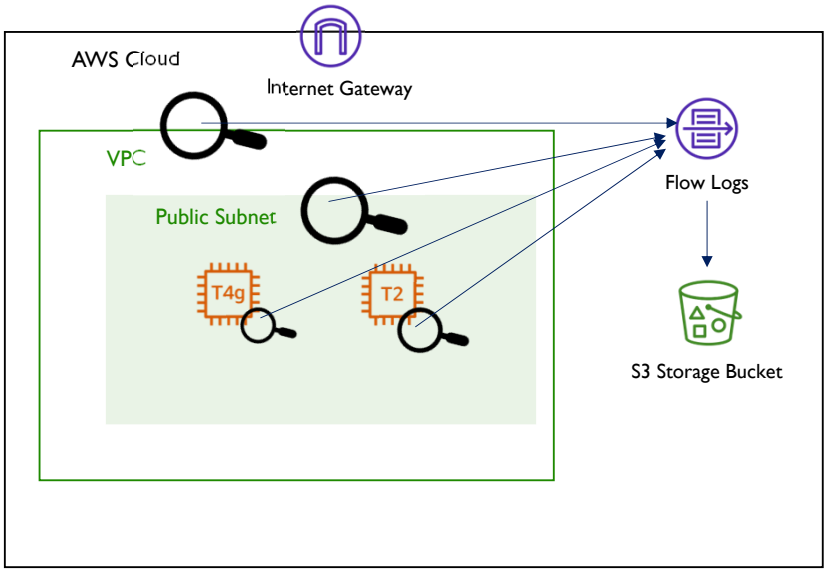
The load balancer is making the connection and proxying it to the instances behind it.

Without the load balancer logs, you will not get the original external IP address that made the request.

This is a diagram showing how the load balancer is receiving the request from the internet and then proxying the request to the instances.

The external user never makes a direct connection to the underlying instance; instead, the load balancer is proxying the requests as they come though.

VPC Flow Logs



NOTE

Flow logs can be enabled at the following:
- VPC
- Subnet
- Network interface

They put no load on your network and are collected out of band.

You can enable them at any time.

VPC flow logs[1] record a summary of what network traffic transited in and out of a specific location. They will capture items such as the source and destination network addresses and ports, but most importantly for our work, they will capture how much data was transmitted and how long the connection lasted.

In this diagram, we are looking at a single subnet within a VPC and all of the locations where we can enable flow logs to be recorded.

You can see that you can capture flow logs in the following places:

- **The individual network interface on an instance:** This will capture a summary of all traffic that transmitted to/from that instance but not the contents of the traffic itself.
- **The subnet level:** This will capture a summary of all traffic that transmitted in or out of the subnet but not the contents of the traffic itself.
- **The VPC level:** This will capture a summary of all traffic that transmitted in or out of the VPC but not the contents of the traffic itself.

The flow logs are stored in S3 buckets at an interval of 1–10 minutes, which you can configure. This means that the flow log system will monitor connections, and at the interval you have chosen (1–10 minutes) will write out the summary of what connections occurred in that interval as well as the amount of data and the duration for which they occurred. When you are creating your flows, you can choose to filter out certain types of traffic or ports or capture all of the connections occurring into the logs.

The most amazing part is that you can dynamically turn on flow logs to any part of the VPC. This means that mid-incident, you can enable flow logs if they didn't already exist, without making any changes to the environment that are observable to the threat actor or that cause any downtime.

1. https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html

VPC Flow Log Examples

version	2	The version of AWS Flow log
account-id	305681518678	The AWS account that this was generated in
interface-id	eni-06fa0119bda73bdc5	The virtual network interface that was contacted
srcaddr	209.17.96.130	The system that sent the connection
dstaddr	10.0.2.99	The system the connection was sent to
dstport	61087	The port that was on the destination
srcport	990	The port used by the sending system
protocol	6	The protocol number in use
packets	1	The number of packets in this connection
bytes	44	The number of bytes in this connection
start	1618617647	A UNIX timestamp of when this connection started
end	1618617703	A UNIX timestamp of when this connection ended
action	ACCEPT	What action the network security group took
log-status	OK	OK

NOTE

Flow logs record sessions every 10 minutes by default

These are the fields for an AWS VPC flow log.

The slide defines each of the parts of the log.

Every single connection within the configured time period that is within the flow log filter (by default, all connections) will have one entry no matter how much data passed across it. Only if multiple connections occurred during that interval between the two systems would you have multiple entries. If you had a very large transfer of data occurring over time, you may have to review multiple entries to determine the full scope of the data transfer if it lasted longer than the 1–10 minutes you specified for your interval.

Remember that flow logs summarize the connection, the amount of data transferred, and the duration of the connection. They do not log the contents of what was transmitted. In order to capture what was transmitted, you would have to enable traffic mirroring on the subnet.

You have two choices on where to send flow logs.

If you choose to store your flow logs in S3 buckets, it is $0.023 per gigabyte of flow logs captured. You can pay $0.023 for up to 50 terabytes of stored flow log data. This is on top of the cost to store the S3 data, but it's a small amount of money to pay to have this incredibly useful data generated.

If you choose to ship your flow logs to CloudWatch[1] and get them kind of integrated into their dashboards and visualization, then the cost goes up to $0.050 per gigabyte for the first 10 terabytes. That cost goes down to $0.05 a gigabyte once you exceed 10 terabytes in a single billing period. If your organization does not already have a system for storing, managing, searching for, visualizing, and alerting from flow logs, this may be an option for you to consider.

1. https://aws.amazon.com/cloudwatch/pricing/

Route 53: AWS DNS

- Amazon's Hosted DNS Service
- DNS Zone Query Logging
  - Log queries made to your hosted DNS domains
- Route 53 Resolver Query Logs
  - Log all DNS queries made within your VPC

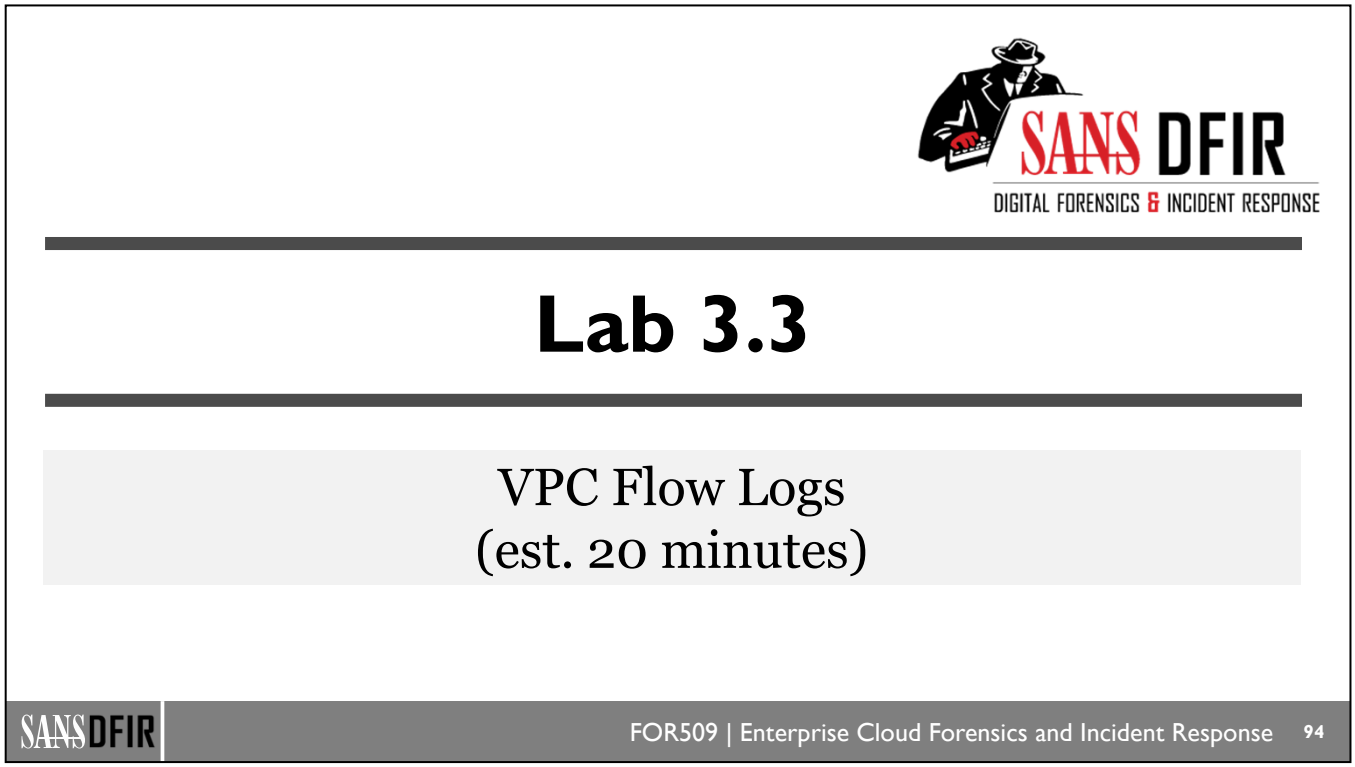
Route 53[1] is the AWS-hosted DNS service.

Why does this matter to DFIR? There are two possibly interesting sets of logs you could retrieve.

The first is DNS zone query logging. You can log all of the queries made against your AWS-hosted domains. This would allow you to find all of the resolutions a threat actor made across your environment.

The second, and likely more interesting, is resolver query logging. This allows you to log within AWS all of the DNS queries made within a VPC. If a threat actor compromised your tenant, you could track all of the DNS requests they made after they compromised a host. Even if the VM or container was deleted, you could still retrieve these logs for analysis.

1. https://aws.amazon.com/route53/



This page intentionally left blank.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- S3 Buckets
- S3 Buckets for Log Storage
- S3 Buckets Access Policies
- S3 Bucket Access Logs
- S3 Bucket Pricing
- **Lab 3.4**: S3 Analysis

This page intentionally left blank.

S3 Buckets

Those things you keep reading about in the news

A file share you can upload to with a web browser

S3 bucket names are universal

- That means only one customer can have an S3 bucket name in all of Amazon

S3 buckets[1] can be thought of as web-hosted file shares.

S3 buckets got a bad reputation in the security community because of how many S3 buckets were accidently set to public access. This meant that anyone who could find the name of the S3 bucket could iterate its contents. Many companies accidently exposed sensitive data this way, which is why you have read about S3 buckets so much in the news. There are now scripts written that will scan all of the S3 namespace looking for publicly accessible buckets. Some are run by bug bounty hunters looking for an easy commission; others are run by threat actors looking to steal data.

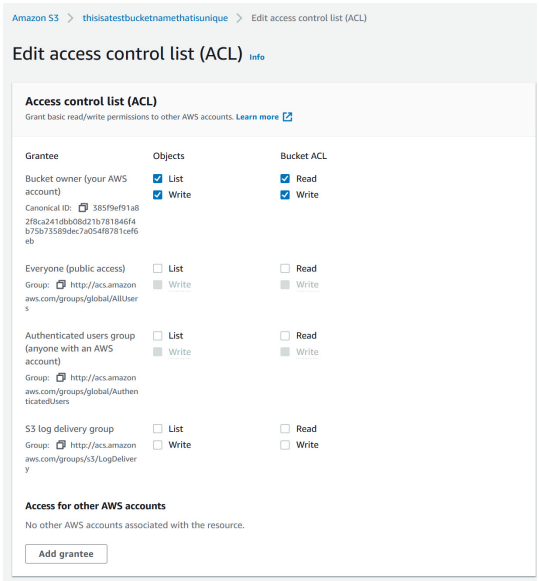
Today, AWS makes it much harder to accidently make a bucket public to the world. By default, all newly created buckets are locked down, requiring authentication to access their contents. In order to make a bucket public, you now have to change multiple security settings and set a custom policy on the bucket. AWS really wants to make sure you truly meant to make it public.

S3 bucket names must be unique for all of AWS, meaning only one entity, person, or company can register one bucket name. This is why we say they are universal: there can only be one S3 bucket named FOR509 in the universe. Now this can also open up opportunities for threat actors. There are threat actors who have named S3 buckets after legitimate companies as part of phishing attacks. There is no restriction on who can name the buckets and what you can name them, meaning a threat actor could make a bucket called "sansevals" and phish a lot of students.

1. https://aws.amazon.com/s3/

S3 Buckets: Restrictions of Access

- Public
  - Anyone can access the contents
- AWS User
  - Any AWS user in any AWS account can access
- Org User
  - A specified user or users can access the contents
- IAM Roles
  - Specific roles can access the contents
- Tokens in URL Strings
  - Anyone with the correct link, in the specified time, can access the contents



To see these kind of restrictions in action and the accidental security vulnerabilities they can create, check out http://flaws.cloud/.

There are two ways to access your S3 bucket. You can access it through the AWS API using calls like sync and cp to pull down data. These API calls will be recorded in CloudTrail like any other API call. You can also use the web interface to S3 to download files. If you use the web interface, those actions won't be recorded in CloudTrail but in the S3 server access log, which you will see in Lab 3.4. This log is not turned on by default, so make sure you do if you care about what's being hosted there.

Public: You can place this on a per-file basis or directory basis for the whole bucket. This means that anyone, anywhere without authentication can access what you marked as public.

AWS user: If you specify AWS user, this does not mean only users in your org can access this data; that's an org user. It means any user on AWS can access the contents of the bucket.

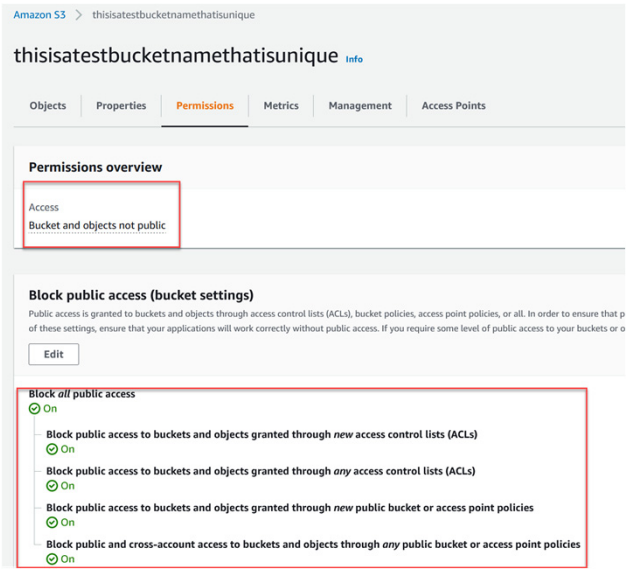
Org user: This means that, at least, the user has to be signed in to your org to access the contents of the bucket.

IAM roles: You can specify specific IAM user or any IAM roles that can access the contents of the bucket. This is useful if the bucket is being used for some kind of automation.

Token: If you provide access to the S3 bucket using a custom URL this is really an STS token or SAML token embedded within the request that provides authentication for a limited period of time. In the IAM section we talked about SAML tokens where you can specify and generate dynamically a token that provides configurable read/write access to a resource (like an S3 bucket) for a period of time you specify. This is a great way to be able to push data in and out of an environment if you don't want to provide some kind of permanent access to an outside application or an untrusted environment.

S3 Buckets: No Longer Public by Default (1)

- When created, S3 buckets default to restrict public access
- Clear wording is shown to reflect if the bucket, or its contents, is public



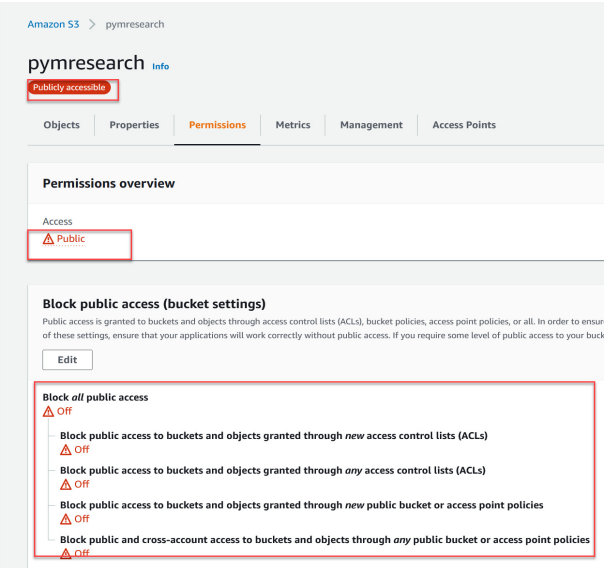
These are the default permissions for a S3 bucket when it's created in AWS today.

The first red box highlights the permissions overview. This will give you a summary showing if any part of the bucket is public.

The second red box highlights the breakdown of the default enforcement that blocks public access. Notice that this specifies on four different levels how public access is being restricted.

S3 Buckets: No Longer Public by Default (2)

- This is a publicly accessible bucket
- Notice all of the warnings displayed on the screen



This is what it looks like if an S3 bucket is set to public access.

Notice that it lets you know the contents are public in three places:

1. Under the name of the bucket is a large red tag visible at all times, stating it is publicly accessible.
2. The permissions overview has a large red warning symbol letting you know the contents are public.
3. Each of the block settings is warning you that it is off.

It is very hard to not know you have a public S3 bucket today.

S3 Buckets: Cornerstone of AWS IR

- S3 buckets can store
  - Snapshots
  - NetFlow logs
  - Load balancer logs
  - CloudTrail logs
  - S3 audit logs
  - Anything!

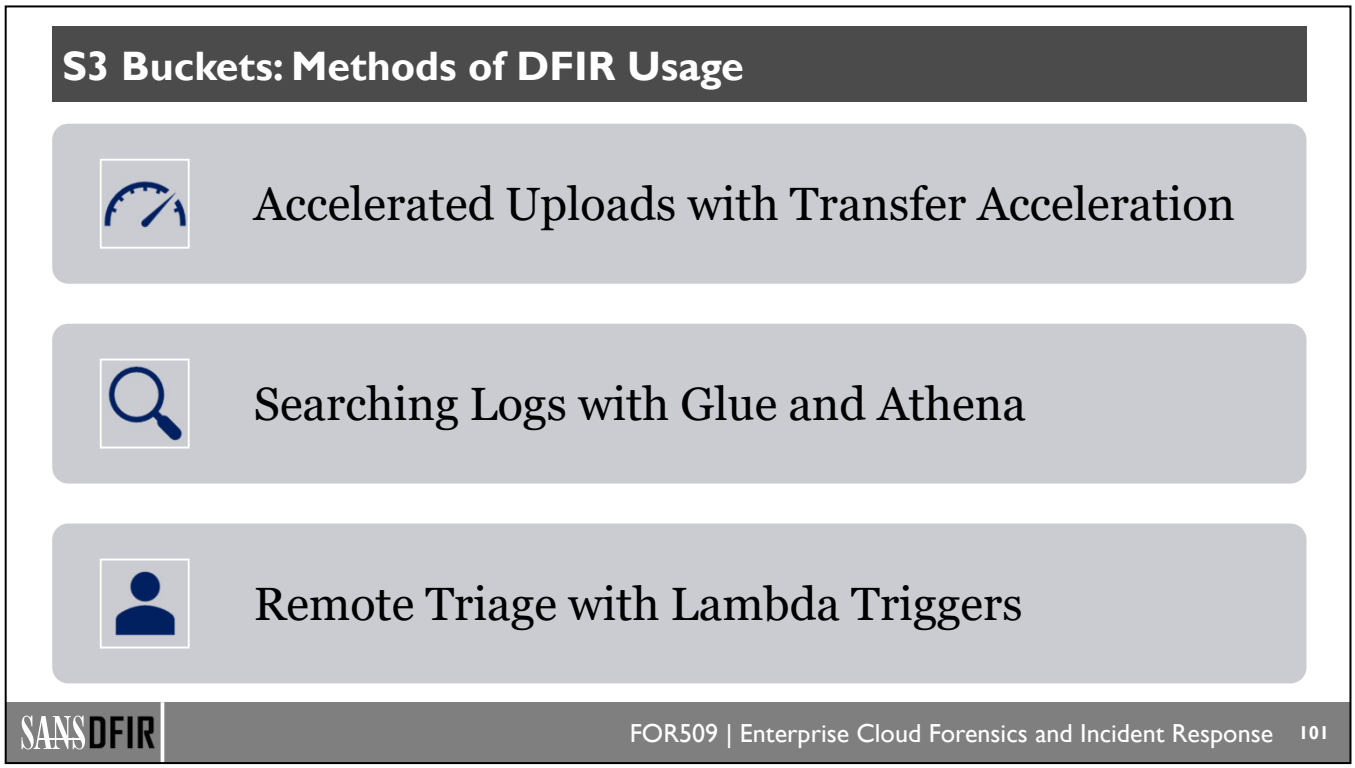
So, what are the things S3 bucket store for forensics?

We can store snapshots, NetFlow logs, load balancer logs, CloudTrail logs, S3 server access logs, and more—and we can dump them into a bucket. You can pretty much put anything in a bucket. This is one of the ways that AWS is really different from Azure. In the next section we're going to talk about the fact that in Azure, data is stored in kind of a blob, because, in the back end, everything is stored in the database. On the other hand, in AWS with S3 buckets, everything is a file. It's just a file that can grow to arbitrarily large sizes.

Intelligent tiering means how quickly the data can be accessed and how many duplicates of the files are stored in data centers across the region for redundancy.

The cheaper the storage fee,[1] the lower the data retrieval speed, and the less fault tolerance that exists for the data. You can set your data to automatically move down the access tier list toward cheaper storage over time as the logs age out, until you eventually mark them for auto-deletion.

1. https://aws.amazon.com/s3/pricing/



What are some of the advantages of using S3 in DFIR investigations?

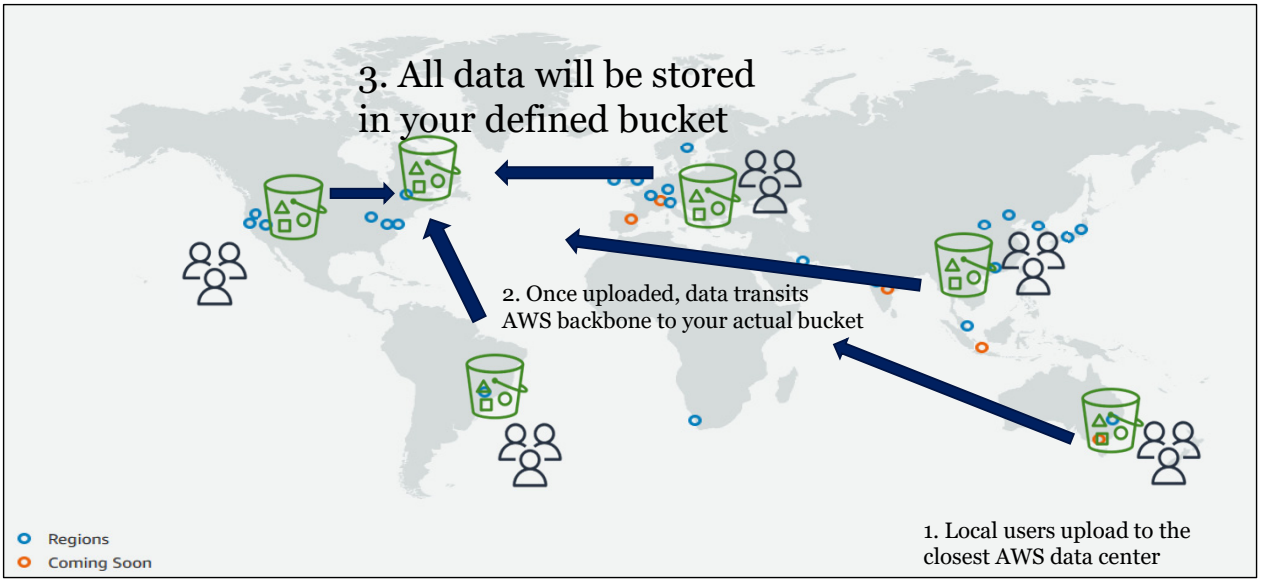
The first thing we should talk about is Transfer Acceleration. When this is selected for your bucket, it provides a URL that contains a DNS alias for your bucket rather than the region where the bucket was created. When someone else around the world goes to access that bucket, it will automatically look up their closest region and data center, and then redirect the access to the fastest data center. If that user was uploading some kind of evidence, the data uploaded will then go through the Amazon back end and back to the region you specify the bucket.

If you're doing global triage, collection, and transfer, this feature can really speed up your data collection. And it's something I would highly recommend if you're dealing with global issues. Transfer accelerated uploads cost $0.08 per GB, which for global collections, when you are bringing data into the cloud for analysis, is more than worth it.

If you are putting large amounts of logs in your S3 buckets (such as CloudTrail logs and flow logs), you could search all of them in place with an AWS service called Athena, which we talk about later in this section. Glue is a service that puts some kind of structure around the data being searched, and Athena lets you use SQL-like queries against data that is sitting in your S3 buckets.

Lastly, there's remote triage with Lambda triggers. We will talk more about this when we get to the automation section. The idea is that every time a file is uploaded to an S3 bucket, you can subscribe to that event and have AWS execute a "Lambda" function, which would then perform some set of actions against the uploaded data. In this way, you can begin to automate the processing of uploaded triage data.

AWS Transfer Acceleration



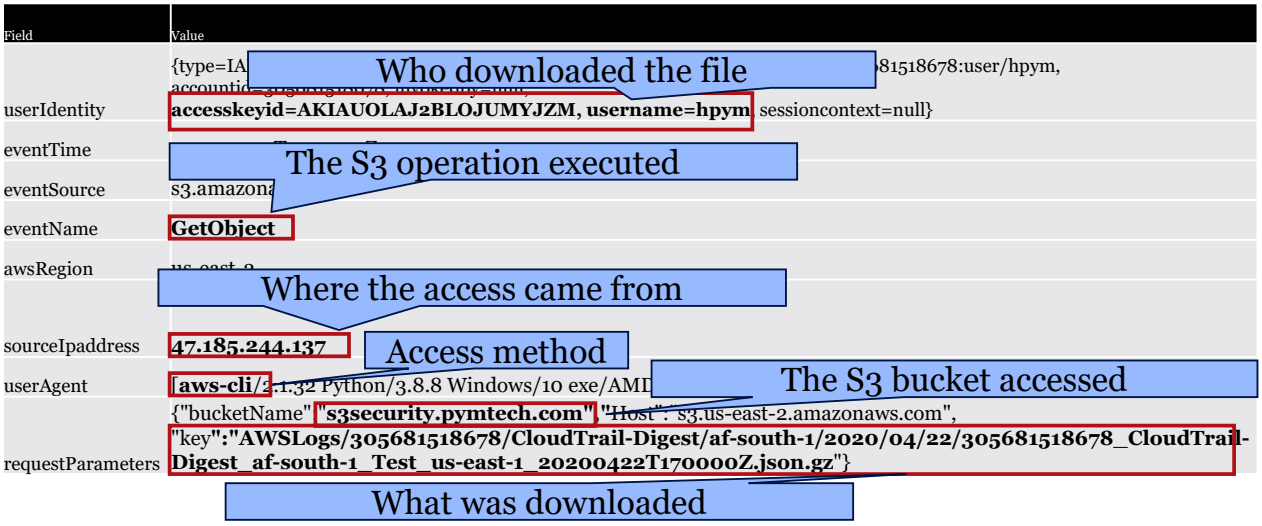
AWS Transfer Acceleration[1] allows you to speed data collection when you need to have global reach.

It allows users around the globe to always reach their closest data center when uploading or downloading data to your S3 bucket.

In the case of an upload, the data will first go up to the local data center, for fastest speed, and then replicate to your chosen bucket through the AWS backend fiber network.

1. https://docs.aws.amazon.com/AmazonS3/latest/userguide/transfer-acceleration.html

S3 Bucket Access in CloudTrail



If someone is accessing S3 buckets via the AWS API/CLI, you will get CloudTrail logs. Otherwise, if they access them through the web interface, you will need to look at the server access logs. We will review those logs in Lab 3.4.

userIdentity: The authenticated user in AWS who made the request.

eventName: The operation that occurred (in this case, GetObject, which will download a file from S3).

sourceIpaddress: Where the access occurred.

userAgent: In this case, the AWS CLI was used for our Windows 10 system.

requestParameters: Here you can see the bucketName we accessed is specified as s3security.pymtech.com, and the key lists out the full path within the bucket of the file that was downloaded.

S3 Bucket Access in Server Access Logs

Field Name	Data
Bucket Owner	385f9ef91a82f8ca241dbb08d21b781846f4b75b73589dec7a054f8781cef6eb
Bucket	pymresearch
Time	[01/May/2021:
Remote IP	5.62.19.43
Requester	-
Request Id	MMAVY5K6DKSH6VNY
Operation	REST.GET.OBJECT
Key	PymPacket.doc
Request-URI	"GET /Pym HTTP/1.1"
HTTP Status	
Error Code	
Bytes Sent	75776
Object Size	75776
Total Time	116
Turnaround Time	114
Referrer	-
User-Agent	Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:88.0) Gecko/20100101 Firefox/88.0
Version Id	-
Host Id	GmwhWqHRvb7QM5Uz+FSxUtz9sqtKW6rcy4JqUyX6YxGXJmAkrHP7m9zWVTiG4eEXSKqM2SuEUhY=
Signature Version	-
Cipher Suite	-
Authentication Type	-
Host Header	pymresearch.s3.amazonaws.com
TLS Version	-

Which IP downloaded the file

The S3 operation executed

What was downloaded

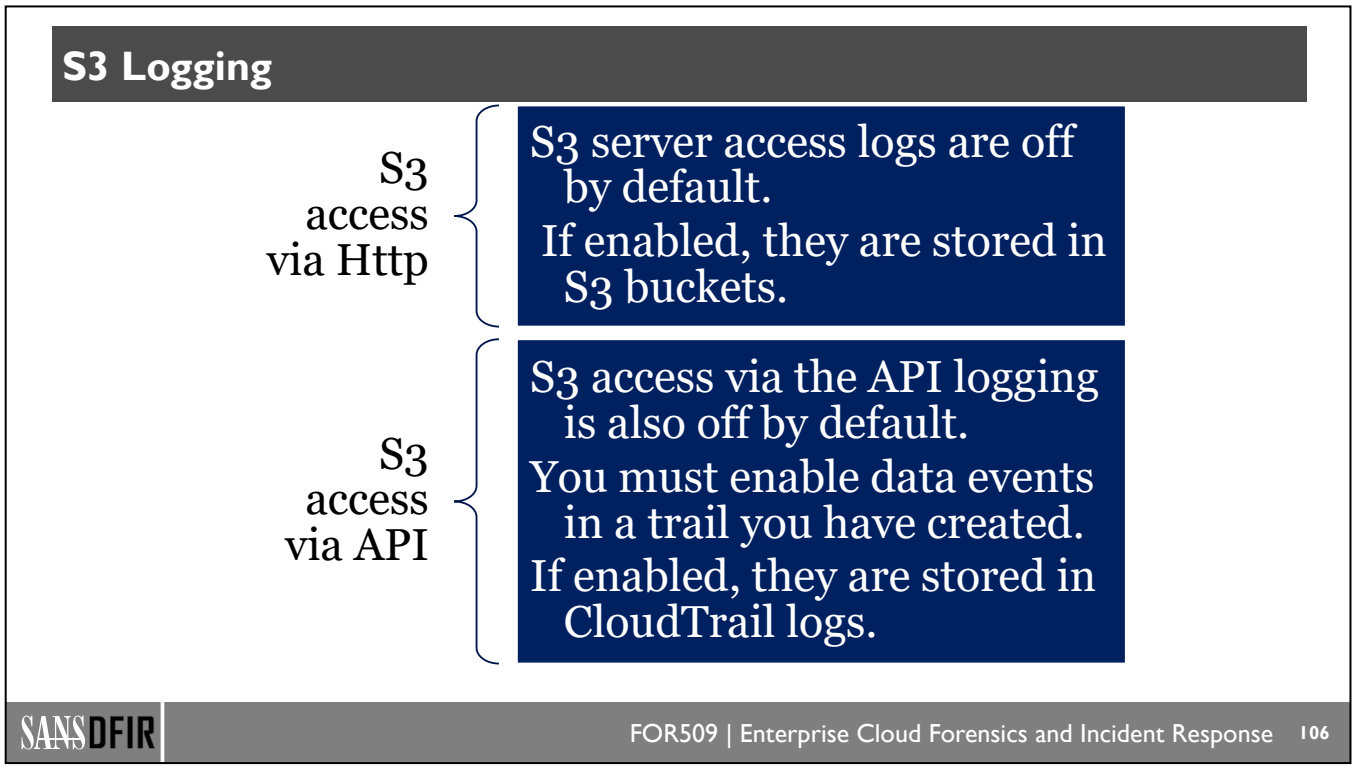
How much was transferred

The S3 bucket accessed

Bucket Owner: The canonical user ID of the owner of the source bucket. The canonical user ID is another form of the AWS account ID.
Bucket: The name of the bucket being accessed, without the full host name
Time: The time and date the request was processed
Remote IP: The IP address that made the request
Requester: If authenticated, this will record the AWS account/key/role that made the access
Request Id: A unique string used to identify this specific request
Operation: Which request was made
Key: What file was requested
Request-URI: The full URI that was passed in
HTTP Status: The status code (200 means success)
Error Code: If there was an error, error codes would be displayed here
Bytes Sent: The number of bytes sent for this request
Object Size: The total size of the object being requested
Total Time: The time in milliseconds it took to complete the request from the S3 web server's perspective
Turnaround Time: How long in milliseconds AWS took to process the request
Referrer: What URL was previously visited that linked to this bucket. In this case this was a direct access.
User-Agent: The user agent that was supplied by the user/tool that made the request
Version Id: If the operation has a version id, it will be displayed here
Host Id: The extended request id
Signature Version: If the request was authenticated a version of that signature version used would be displayed here

References:
https://docs.aws.amazon.com/AmazonS3/latest/userguide/ServerLogs.html
https://docs.aws.amazon.com/AmazonS3/latest/userguide/LogFormat.html

Cipher Suite: If the request was over TLS, it will display the cipher suite used
Authentication Type: If the request was authenticated it will display what type of authentication was used
Host Header: The endpoint used to connect to S3, normally the full bucket host name
TLS Version: If TLS was used, it will display the version

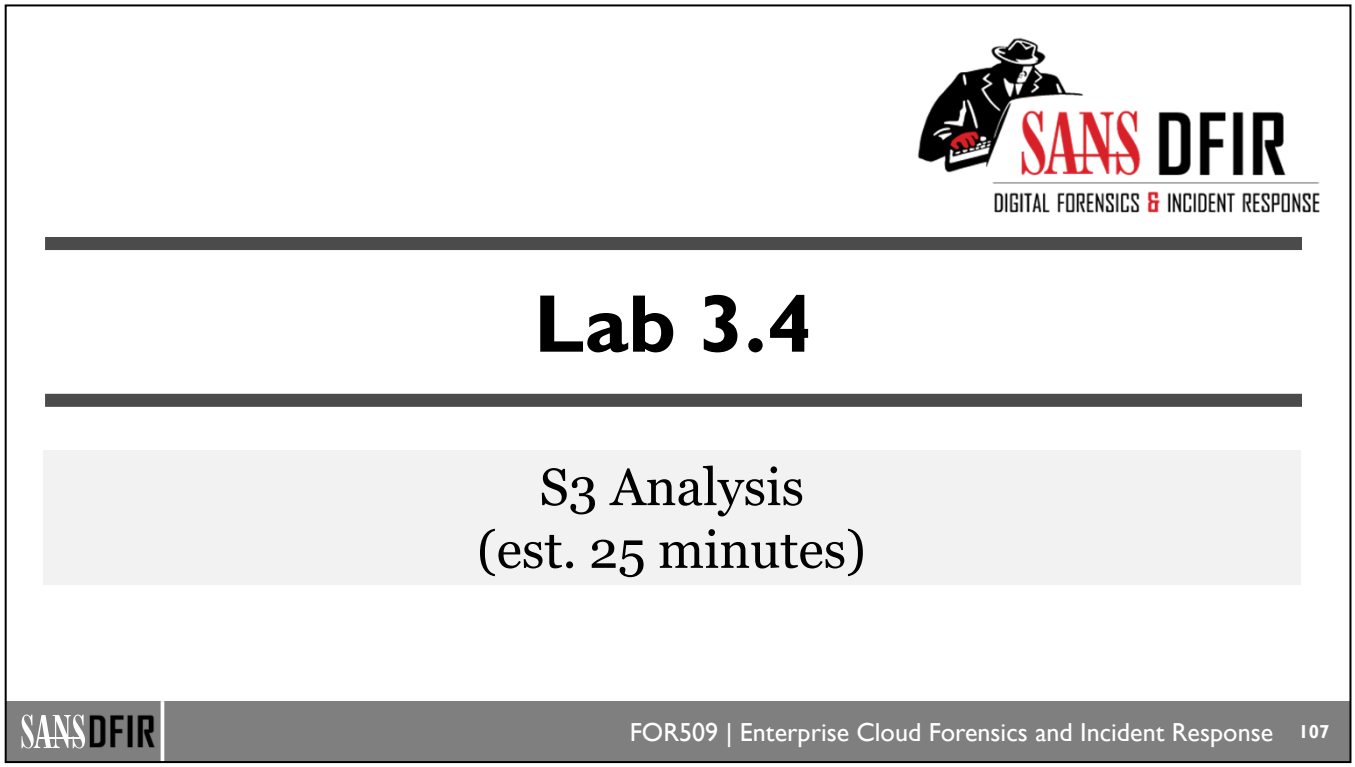


This slide is here to confirm the understanding of the two different kinds of S3 logging available.[1]

If the access to the S3 bucket was made through the web interface to S3 then server access logs must be enabled to record those accesses. These entries will never appear in the CloudTrail logs as the request was received by the S3 application and not the AWS control plane that generates CloudTrail events. Server access logs will be sent to S3 if they are enabled.

If the access to the S3 bucket is made through the API it can be captured in CloudTrail logs, if you've created a trail that includes S3 data events. The default CloudTrail that AWS maintains for you does not have data events.

1. https://docs.aws.amazon.com/AmazonS3/latest/userguide/logging-with-S3.html



This page intentionally left blank.

FOR509.3: Amazon Web Services (AWS)

Section 3.1: Understanding IR in AWS

Section 3.2: Networking, VMs, and Storage

Section 3.3: AWS Native Log Searching

Section 3.4: Event-Driven Response

Section 3.5: In-Cloud IR

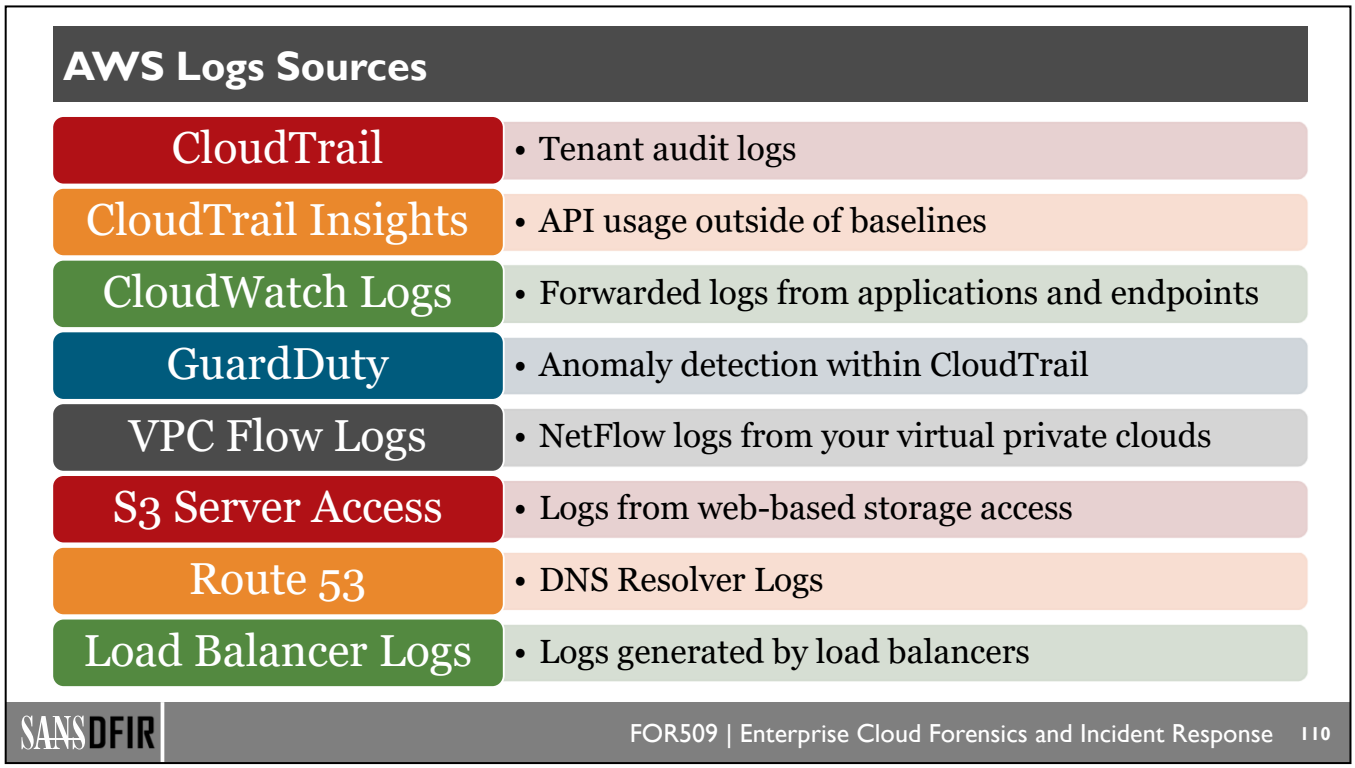
This page intentionally left blank.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- AWS Log Sources
- AWS Athena
- AWS Glue
- AWS Glue Syntax
- AWS/Glue Pricing
- AWS Detective

This page intentionally left blank.



CloudTrail: What we have been discussing the most in this section.

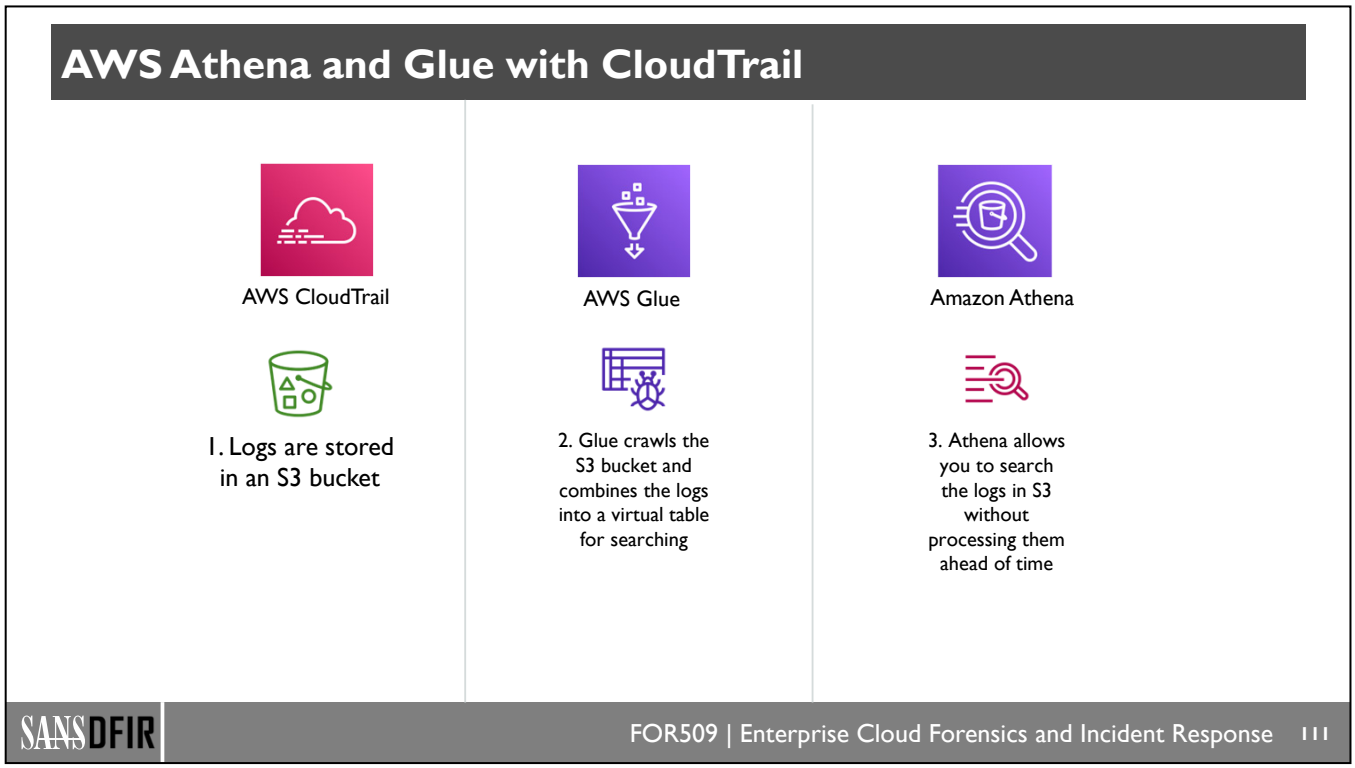
CloudWatch Logs: A feature of CloudWatch that lets you take application logs from your underlying systems and ship them directly to CloudWatch for alerting and aggregation metrics. The original usage of this feature was not for security or detection. The original usage was to be able to do things like track errors and uptime metrics.

CloudWatch Logs Insights: Used for metrics and patterns within your application logs if you choose to ship them to CloudWatch Logs.

GuardDuty: We looked at GuardDuty earlier in this section. This is a machine learning–based expert system that finds possible intrusions and highlights bad practices.

VPC Flow Logs: Network flow logs of connections occurring within your VPCs.

S3 logs: Server access logs that record web-based storage access of data in your buckets.



This is an example of how you can use Glue and Athena together to search your CloudTrail logs while they are still sitting in an S3 bucket.

Glue will crawl through the bucket. For every CloudTrail log it finds, it will add that log and the basic metadata into a virtual table for searching in Athena.

When you need to, you can then search all of the CloudTrail data in your S3 bucket without having to load it into a database or process/convert it. Athena allows you to perform SQL-like queries across all of the logs that Glue crawled.

Athena is very affordable but is priced on a per-usage basis.[1]

For every successful search you perform in Athena, you will be charged $5 per TB of data searched. Depending on your data set, this could be a large amount of money, so be sure to check to see how much data you are searching first. Also, the results of your query will be stored in an S3 location, which means you will be charged to store the results. If you are bringing back very large data sets as a result of your query, you will want to delete those out of S3 to prevent being charged for them.

Glue is charged at $0.44 per hour (with a $0.04 minimum per crawl), which represents the time a 4 CPU/16GB RAM EC2 instance takes to crawl the data. It's called "Glue crawling" because Glue will walk through the S3 bucket you specify to find and bring back all the record locations for Athena to search.

1. https://aws.amazon.com/athena/pricing/

AWS Glue Syntax

```
CREATE EXTERNAL TABLE
[TABLE_NAME] (
   eventVersion STRING,
   userIdentity STRUCT<
      type: STRING,
      principalId: STRING,
      arn: STRING,
      accountId: STRING,
      invokedBy: STRING,
      accessKeyId: STRING,
      userName: STRING,
      ...
```

- Glue allows you to format, convert, enrich, and combine data sources for Athena to search
- Glue creates the tables/partitions that Athena will search across
- Glue can define schemas like date and time against specific log fields
- CloudTrail has a predefined Glue data set

This is an example of what Glue[1] syntax looks like.

We are creating an external table, meaning the data is hosted outside of Glue/Athena.

We are specifying a name for the table.

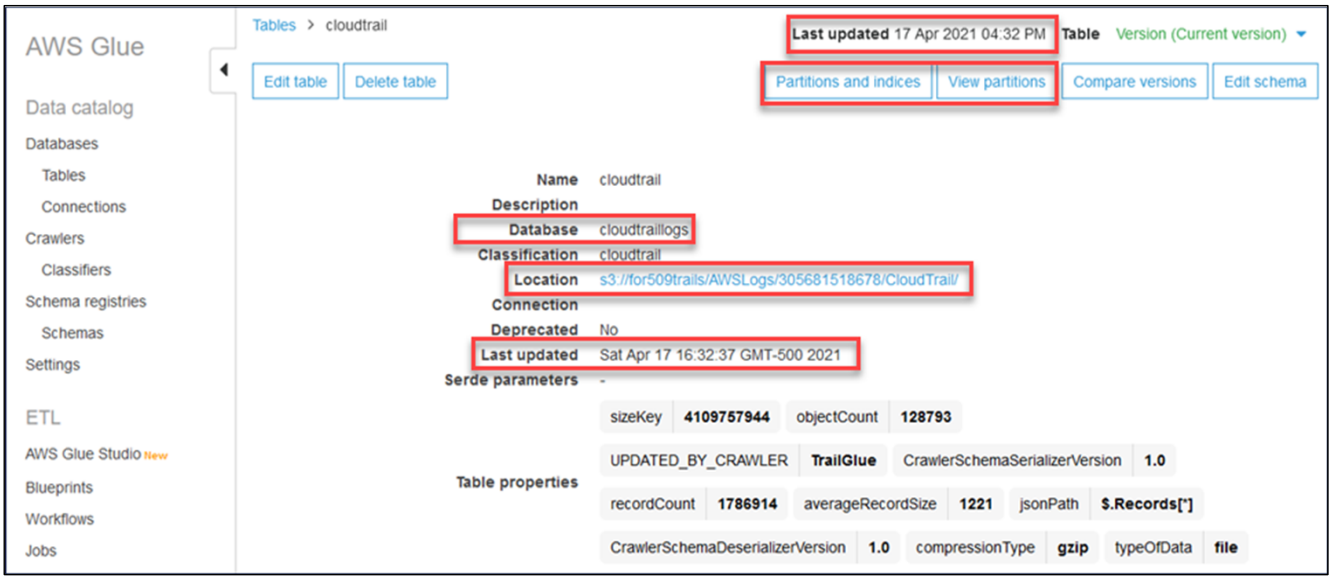
The rest of data shown is given as two examples.

The first, eventVersion STRING, indicates that the first field in the log entry "eventVersion" should be treated as a string for searching.

The second, userIdentity, is being defined as a structure. This allows us to then break down the individual JSON sub-elements into individual elements for searching within Athena, each with its own type defined. So, for instance, arn is followed by : and then the word STRING, which is telling Glue to take the sub-element userIdentity.arn and treat it as a string when searching.

1. https://aws.amazon.com/glue/

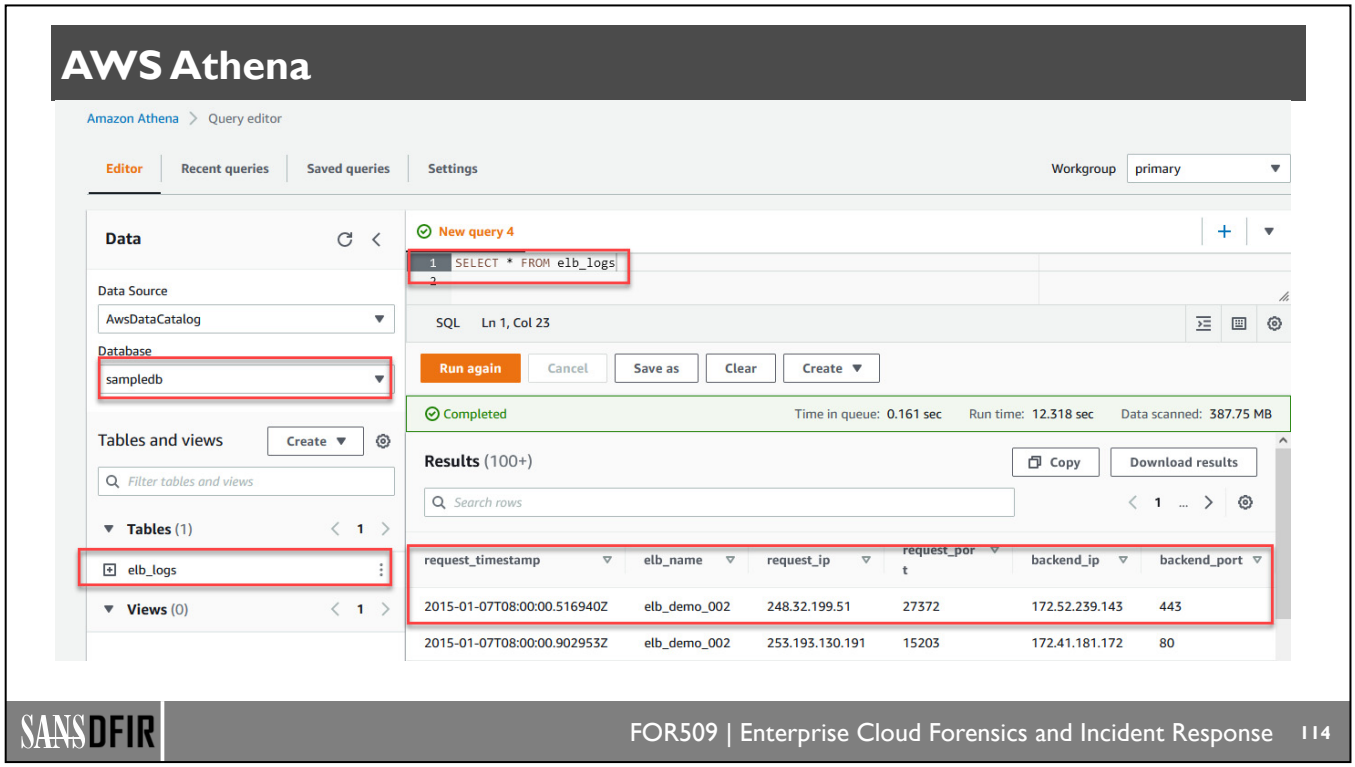
AWS Glue Interface



This is the interface to Glue.

As you can see, Glue has made a table for Athena to search that points to an S3 bucket that contains our CloudTrail data.

If you wanted to make a new table, you'd need to create a crawler and specify the fields that Athena will be searching with data types. For CloudTrail, AWS actually provides a Glue template that will allow you to automate the log ingestion, schema definition, and partitioning of individual logs into one table.



Athena[1] is an AWS service that allows you to search data that is resident within your S3 buckets as if it was in a SQL database.

In this screenshot, you can see that we are using a SQL query against all of our CloudTrail logs stored in our S3 bucket to retrieve file downloads made against our S3 bucket via the AWS API.

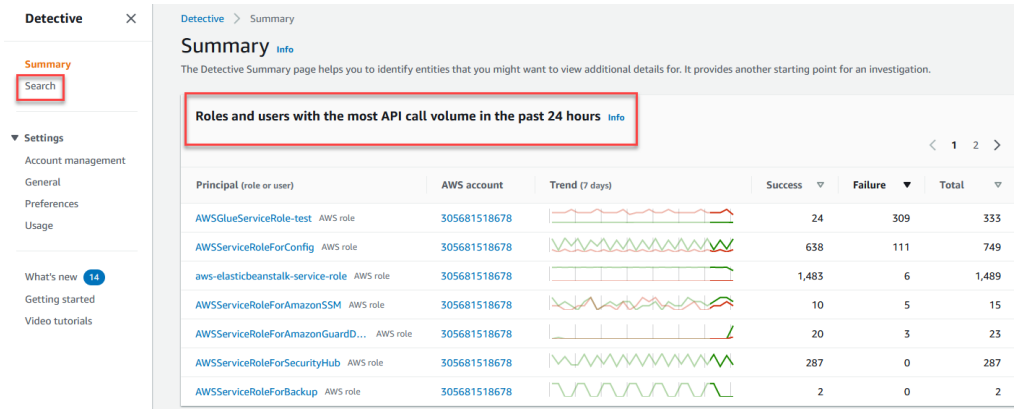
Athena allows you to search terabytes of data stored in your S3 buckets without having to first process and load it all into something like ELK. In addition, you can schedule Athena queries to run at regular intervals to perform regular threat hunts as new logs get written into your CloudTrail buckets.

However, Athena does require some help in figuring out how to turn all of the individual CloudTrail logs into a single table and how to deal with the individual fields. For this, we need Glue, which will provide us the ability to define schemas and partitions to bring in the data for searching.

1. https://aws.amazon.com/athena/

AWS Detective (1)

Detective highlights potential issues and allows hunts



If you are looking to get the most insights and investigative help from AWS service within your investigation, there is AWS Detective.[1]

AWS Detective will go beyond GuardDuty and combine all of your available log sources to highlight possible incidents occurring within your organization.

This is not a SIEM; SecurityHub is more like a SIEM than AWS Detective. Instead, this is more of a threat-hunting portal to begin looking for signs of intrusion or to expand on what you already know.

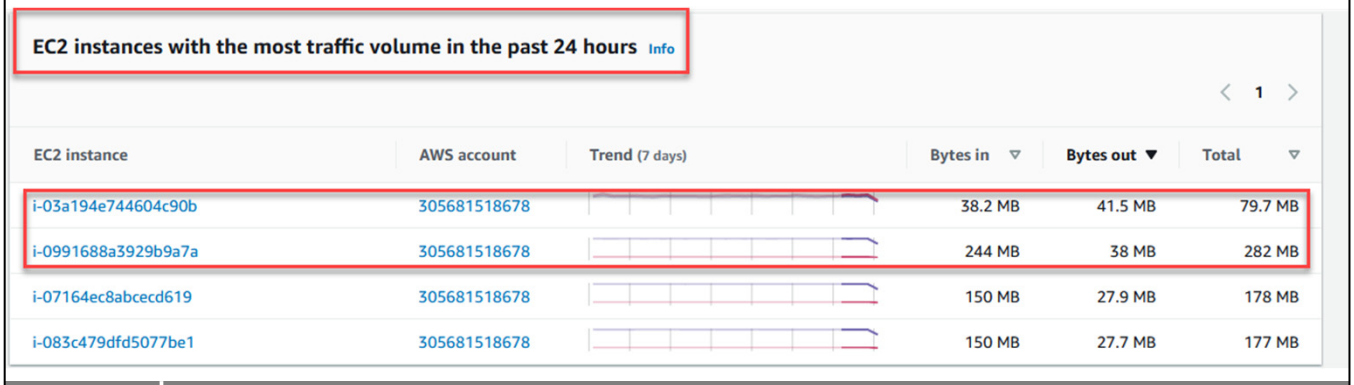
Pricing for AWS Detective:
Ingested Logs per GB

<= 1,000GB	$2/GB
<= 5,000GB	$1/GB
<=10,000GB	$.50/GB
>10,000GB	$.25/GB

1. https://aws.amazon.com/detective/

AWS Detective (2)

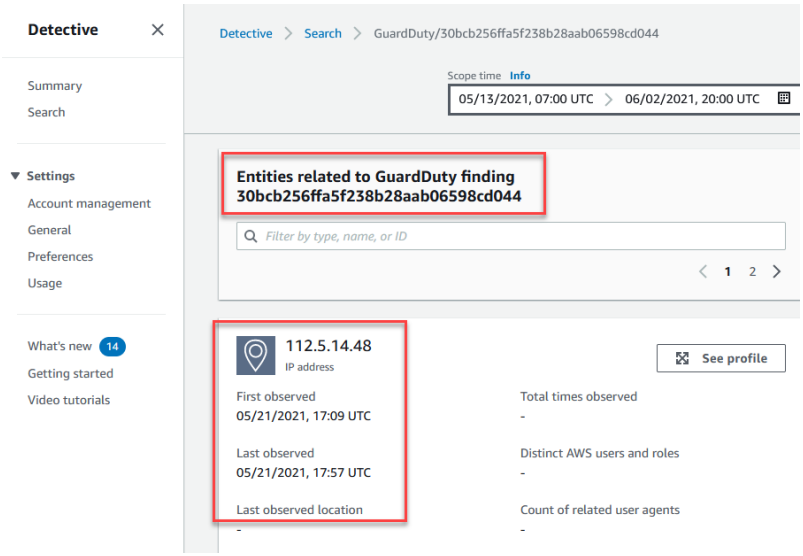
It highlights trends and patterns for your review



Detective highlights trends, abnormalities, and new occurrences to allow you to quickly find anomalous activity as well as hunt in your environment. In this example we see which ec2 instances had the most network traffic in the last 24 hours. While it is not pointing out any known incidents, the data is displayed for a human to determine if the amount of traffic aggregated over that time period is out of the expected norm. Large traffic spikes in a day could indicate that data is being exfiltrated out of your AWS tenant.

AWS Detective (3)

- Search for items like:
  - GuardDuty Alerts
  - IP Addresses
  - S3 Buckets
  - User Agents
  - Etc.
- In this example we are looking at all the SSH brute force attempts detected by GuardDuty



Detective allows you to search across multiple combined log sources to get a larger view of your AWS account. It includes AWS security services like Amazon GuardDuty, Amazon Macie, and AWS Security Hub as well as partner security products can be used to identify potential security issues, or findings. It brings in data from multiple data sources such as Virtual Private Cloud (VPC) Flow Logs, AWS CloudTrail, and Amazon GuardDuty.

AWS Detective isn't cheap, but it allows you to see an enriched and unified view of your security data.

FOR509.3: Amazon Web Services (AWS)

Section 3.1: Understanding AWS

Section 3.2: Networking, VMs, and Storage

Section 3.3: AWS Native Log Searching

Section 3.4: Event-Driven Response

Section 3.5: In-Cloud IR

This page intentionally left blank.

Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- Lambda Functions
- Lambda Examples
- Lambda Pricing
- Step Functions
- Step Functions Pricing
- Event Triggers
- Event-Driven DFIR Automation

This page intentionally left blank.

Lambda Functions

- Lambda is AWS's version of serverless functions
- Lambda functions can be written in several languages:
  - Java
  - JavaScript
  - Go
  - PowerShell
  - Node.js
  - C#
  - Python
  - Ruby

Event-driven response relies upon triggers of actions from cloud events and serverless functions.

Serverless functions in AWS are called Lambdas.[1]

Lambda is not a programming language but the name of the underlying service. You can write a Lambda in any programming language you like. For instance, you can currently write Lambdas in:

1. Java
2. JavaScript
3. Go
4. PowerShell
5. Node.js
6. C#
7. Python
8. Ruby

In Azure, this is called an Azure function; in GCP, it's a Cloud Function or part of Pub/Sub.

The big limitation of Lambdas is that they must complete execution in 15 minutes, maximum, or else they are killed. So, as long as you can guarantee your process will finish in 15 minutes or less, you can make use of Lambdas for your needs. When we look at the pricing, you'll understand why you might want to do this.

Lambdas are incredibly cheap to run and require no management on your part to keep them running.

You specify the trigger, or trigger them manually, and AWS will spin up the container; run the code and then destroy the container as many times as the function is called.

1. https://aws.amazon.com/lambda/

So rather than you paying for an EC2 instance to be running and waiting to receive requests or paying the money to have a whole cluster of auto-scaling resources, it's 20 cents for 1 million executions.[1]

Examples of triggers you could automate reactions to include but are not limited to:

- GuardDuty events
- When a file is created in S3 buckets
- Traffic spikes
- Specific CloudTrail events
- CloudWatch Logs events

1. https://aws.amazon.com/lambda/pricing/

Isolating Compromised Hosts

- Need to isolate a compromised host?
  - Virtual firewalls can be configured on demand
  - Lambda to create an isolated security group:

```
def createSecurityGroup(groupName, descriptionString, vpcId):
resource = boto3.resource('ec2')
securityGroupId = resource.create_security_group(GroupName=groupName,
         Description=descriptionString, VpcId=vpcId)
         securityGroupId.revoke_egress(IpPermissions= [{'IpProtocol': '-
1','IpRanges':    [{'CidrIp': '0.0.0.0/0'}],'Ipv6Ranges':
[],'PrefixListIds': [],'UserIdGroupPairs': []}])
         return securityGroupId.
```

  - Lambda to change EC2 security group to new isolated NSG:

```
def modifyInstanceAttribute(instanceId,securityGroupId):
         response = ec2Client.modify_instance_attribute(
                  Groups=[securityGroupId], InstanceId=instanceId)
```


If you want to isolate an instance, you can do that by changing the network security group it belongs to. You would change it to the network security group you have created that allows no traffic in or out.

On the slide are the Lambda functions[1] you would put into your automation step functions to isolate an instance based on some trigger. The first function would create the isolation network security group; the second would use the retuned value to place the instance you specify into that isolation group.

1. https://aws.amazon.com/blogs/security/automate-amazon-ec2-instance-isolation-by-using-tags/

Event Triggers

Lambda functions can be triggered with AWS events

Examples of events include:

- S3 uploads
- CloudTrail events
- EC2 changes
- EFS changes
- Amazon config changes

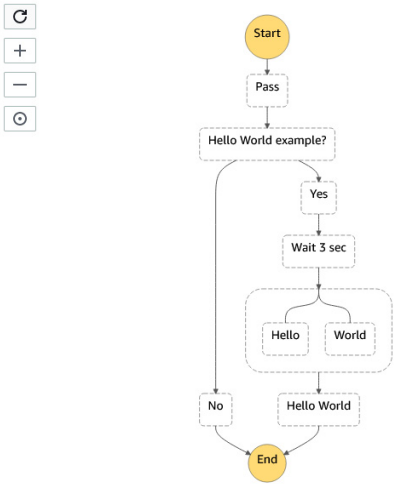
Almost any internal AWS event can trigger[1] a Lambda, including other Lambdas. In addition, you can begin a step function with a manual execution or an external source.

For instance, you could have your on-premises SOAR or SIEM start a step function automation to respond to an alert.

1. https://dashbird.io/blog/what-are-aws-lambda-triggers/

Step Functions

- Step Functions allows you to chain Lambda functions into a workflow where the prior output is the input for the next step.
- This allows you to divide your workflow into fast and cheap running Lambda functions and build a larger IR automation tool.
- Remember that Lambdas can run for a maximum of 15 minutes before they are killed.



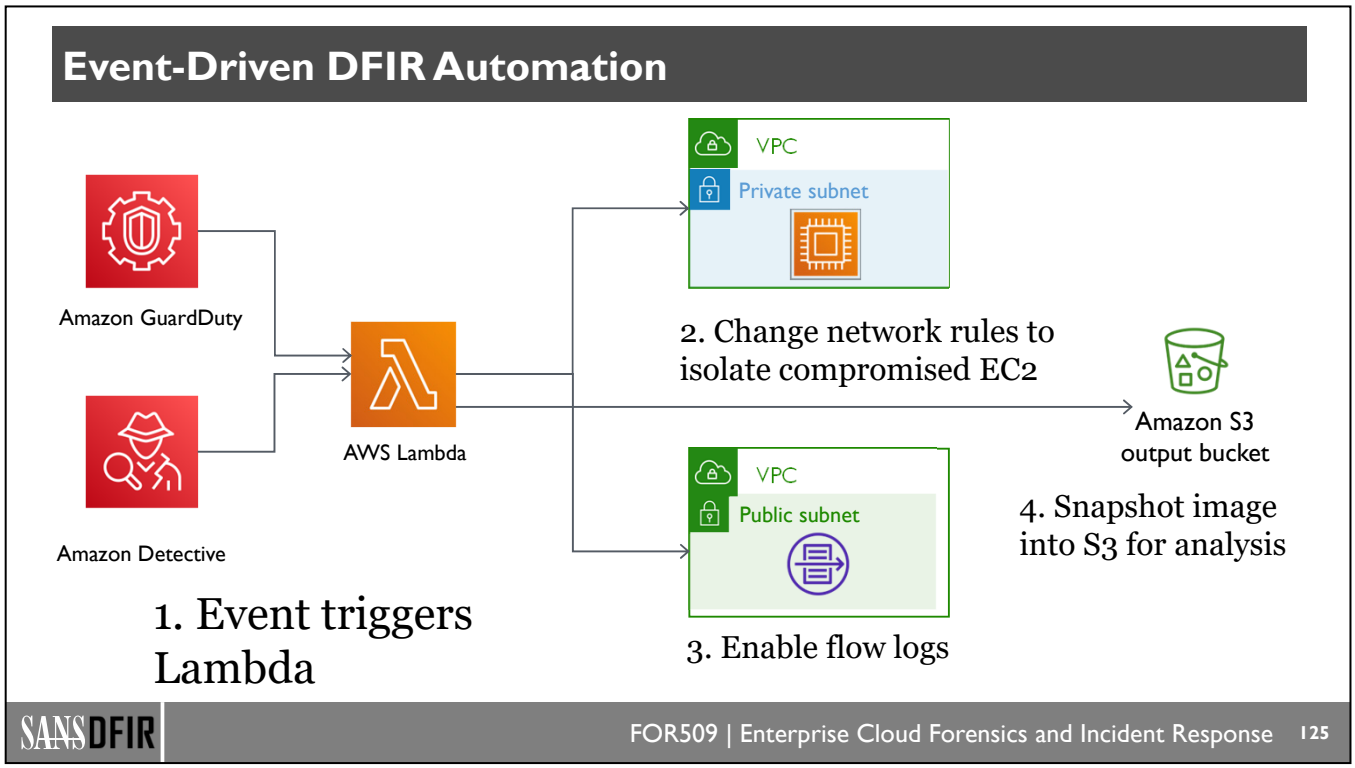
Step Functions[1] allows you to create a workflow of Lambda functions.

The UI for Step Functions allows you to put in a branching set of Lambda functions and evaluate the results of each previous Lambda execution to determine what step should be taken next. Doing this, you can map out your workflow that you previously accomplished in a container or EC2 instance into a series of Lambda scripts.

Why would you want to run Step Functions over a container?

The cost![2] The only addition costs involved are for evaluated Lambda state transitions after the first 4,000 each month. Even then, after the first 4,000 you're only charged at $0.025 per 1,000 state transactions, meaning if you take the time to build your DFIR workflow into Lambdas and Step Functions, you can create some awesomely powerful and very cheap systems.

1. https://aws.amazon.com/step-functions/
2. https://aws.amazon.com/step-functions/pricing/



In this diagram, we are looking at an example of a workflow for one specific event-driven DFIR automation chain.

Starting on the left, we have event sources; in this case, we have two: GuardDuty and Detective.

We then have a Lambda, which can be triggered by events generated from either source. It will then perform three actions:

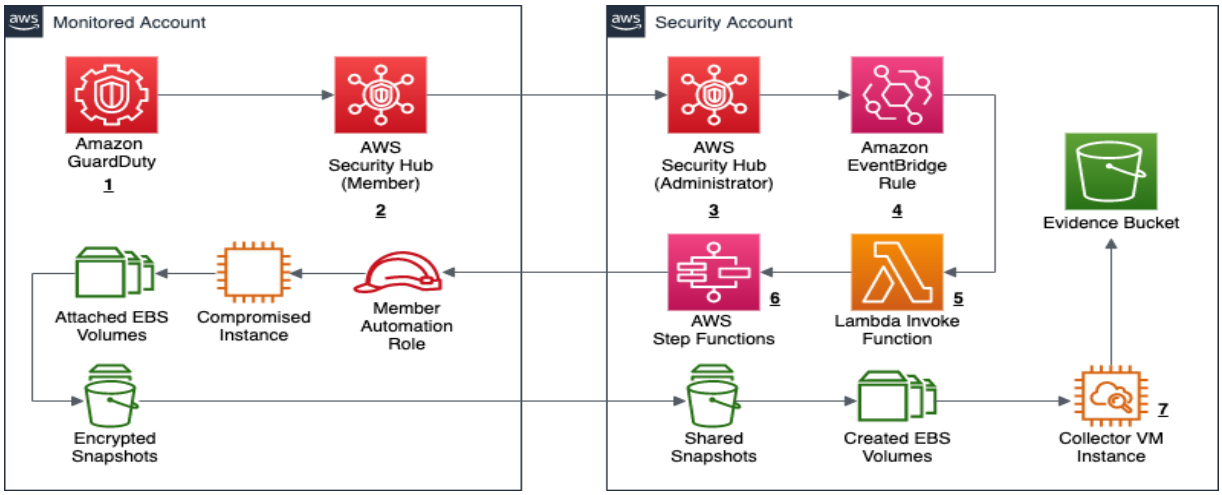
1. It will change the network security group rules to isolate the EC2 instance specified in the GuardDuty alert or the Detective alert.
2. It will then push a disk image to S3 for storage and analysis.
3. It will check to see if flow logs are enabled in that VPC and enable them if they are not already there.

This is just one example of an event-driven automation we could create to rapidly triage events as they are detected.

Watch Ryan Tick's talk about disk imaging in AWS with Lambda[1] or read AWS's posted solution for forensic disk collection.[2]

1. https://www.youtube.com/watch?v=CR4_a-TO_gw
2. https://aws.amazon.com/blogs/security/how-to-automate-forensic-disk-collection-in-aws/

AWS Forensic Disk Collection Automation



https://aws.amazon.com/blogs/security/how-to-automate-forensic-disk-collection-in-aws/

The above is a diagram from AWS's own automation solution for forensically imaging snapshots.

You can see the following automation chain:

1. A GuardDuty or other alert is triggered
2. SecurityHub service receives the alert and forwards it to a cross-org account set up for security
3. The SecurityHub service in the security account receives the alert
4. The EventBridge service then invokes the lambda function to begin the disk imaging
5. A lambda function receives the instance information from the EventBridge service and starts a step function with the details
6. The step function fires off a series of stateful lambdas to:
    1. Assume the role in the monitored account
    2. List the attached EBS volumes on the compromised instance
    3. Create encrypted snapshots for each volume
    4. Share those snapshots to the security account
    5. Restore the snapshots to volumes within the security accounts
    6. Attach the volumes to "collector" VMs that will create the full disk image
    7. Store the created images within an S3 bucket

FOR509.3: Amazon Web Services (AWS)

Section 3.1: Understanding IR in AWS

Section 3.2: Networking, VMs, and Storage

Section 3.3: AWS Native Log Searching

Section 3.4: Event-Driven Response

Section 3.5: In-Cloud IR

This page intentionally left blank.

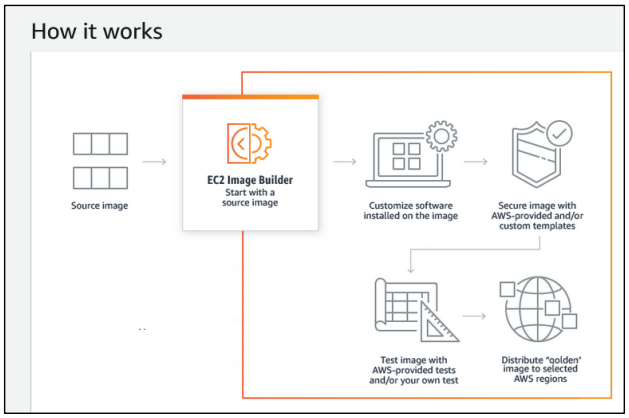
Amazon Web Services (AWS) Roadmap

- 3.1: Understanding IR in AWS
- 3.2: Networking, VMs, and Storage
- 3.3: AWS Native Log Searching
- 3.4: Event-Driven Response
- 3.5: In-Cloud IR

- Creating IR VMs
- AMI and IR
- In-Cloud vs. On-Prem
- Downloading AMIs
- AWS Systems Manager
- Capturing Linux Memory
- Capturing Windows Memory
- ECS, EKS
- **Lab 3.5**: Tracking Lateral Movement

This page intentionally left blank.

Creating IR VMs



Amazon makes it easy to create a custom IR VM

The EC2 Image Builder lets you pick a base image (Windows, Linux, etc.)

You can then configure the base system and upload an image as a template for your organization to deploy on demand anywhere in the world

Don't want to use the base Amazon images?

You can create your own custom AMI (Amazon Machine Image) to base your DFIR workstations on.

Using the AWS EC2 Image Builder,[1] you upload an ISO of your choice with the base operating system. Next, you get to customize what software you want installed on it.

Once you're done, this is now a base EC2 template you can use to deploy DFIR workstations anywhere in the world.

1. https://aws.amazon.com/image-builder/

EC2 Instances and IR

EC2 instances can be examined in two ways:

Within AWS:

- You can snapshot the running system and attach it to a DFIR AMI that is running
- Run all your tools against the snapshots
- Snapshots are basically raw disks that can be accessed like a DD image

Outside of AWS:

- You can snapshot the running system and then download the data
- You can power off the running system and download the EBS volume
- The time to download could be longer than it takes to do the examination within AWS

We would always recommend doing EBS analysis within the cloud.

The time and cost of taking data out of the cloud really does slow things down. Within AWS, you can snapshot a disk at any time, attach it to another DFIR EC2 instance, and begin your examination right away. You can choose to make a forensic image of the snapshot or just access it like a raw disk.

If you have to do your EBS analysis outside of AWS, then you have a couple of options in bringing down the data. We would almost always recommend doing a snapshot and then downloading the data. Depending on the size of the volume, though, your total transfer time could be longer than the actual analysis!

In-Cloud vs. On-Prem AMI Examination

In-Cloud:

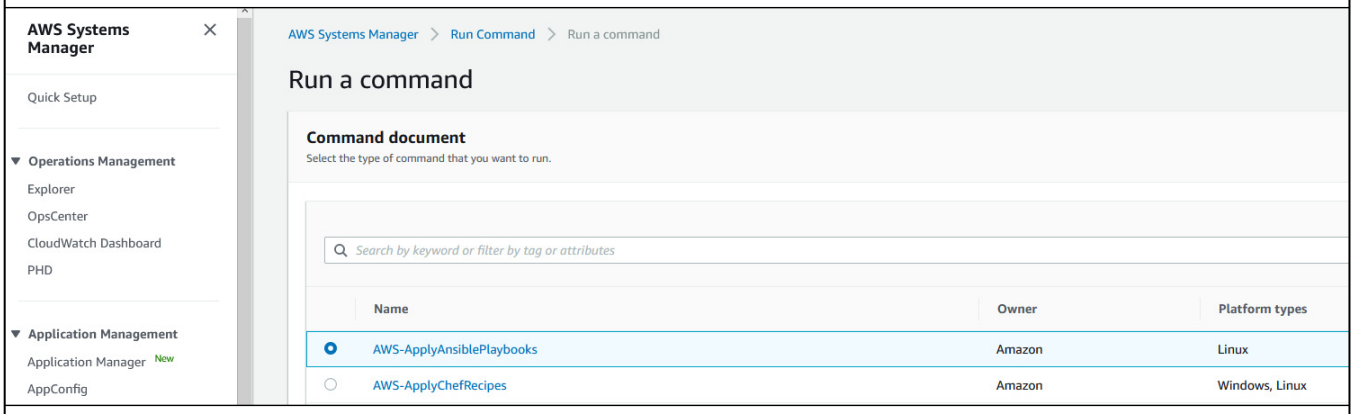
- Pros:
  - Fastest time to access data
  - Able to process and load data using Amazon-provided services:
    - Elastic auto-scaling clusters
    - Athena for log searching
    - Containers for scalable processing
  - On-demand DFIR lab anywhere in the world
  - Automated chain of custody through access logging
- Cons:
  - Cost for every examination
  - If the entire organization is compromised, your DFIR account may be too

On-Prem:

- Pros:
  - After initial investment in hardware, software, and people, there is no cost per incident to run the DFIR systems
  - Isolated labs can be outside of compromised cloud environment
- Cons:
  - Time to download cloud data
  - Time to process data
  - Data speeds can be slower on international transfers
  - Infrastructure maintenance costs

This page intentionally left blank.

AWS Systems Manager



In Lab 3.5, you will see CloudTrail logs from AWS Systems Manager[1] (a.k.a. SSM).

SSM is great in DFIR for two reasons:

1. From the API or CLI, you can run commands in a root shell on any instance you have the role to access. Using this you could use SSM to deploy triage scripts, execute commands when a system is isolated, or act as part of an automation chain like we saw in the last section.
2. You can execute commands on remote systems without exposing credentials to a threat actor on the system. SSM receives the requests and executes as the user it's running as (root/administrator) without needing credentials passed to it.

SSM comes standard with all Amazon Linux 2 EC2 instances. For any other instance, it does have to be configured. You configure SSM within the EC2 dashboard, making sure you have the right IAM roles established for it. If you've ever used EC2's "Instant state connect" feature, that's really SSM.

There will be times when SSM is configured on a system but simply not running; in those cases, you will have to restart the service.

1. https://aws.amazon.com/systems-manager/

Capturing Linux Memory

Margarita Shotgun

- https://github.com/ThreatResponse/margaritashotgun
- Logs in to multiple running EC2 instances at the same time via SSH and captures memory
- No longer updated

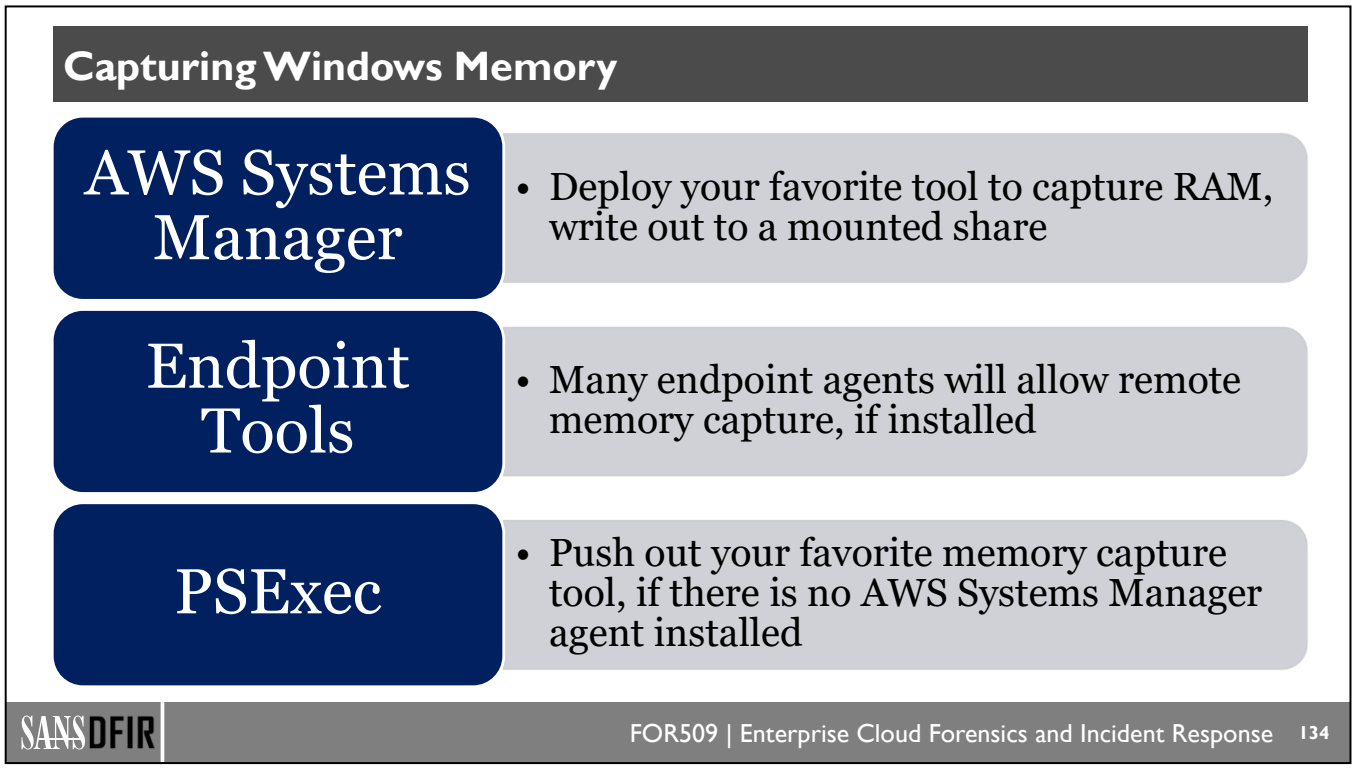
AVML and AWS Systems Manager

- AVML is a replacement for LiME for Linux memory captures
- AWS Systems Manager allows you to execute commands across instances that have the system manager agent installed
- AVML can be statically compiled, meaning it does not require compromised systems have kernel headers

Margarita Shotgun is no longer maintained but can still work. The major issue with this approach is that you have to have the correct LiME module to perform the capture. This means for every system you plan to target you need to know the LiME module name prior to acquisition.

AVML[1] being executed through SSM has no such restrictions. AVML (Acquire Volatile Memory for Linux) is created and maintained by Microsoft. This tool is statically compiled and works without kernel drivers. We've used AVML on production Linux systems with up to 4TB of RAM and high usage without issues. In our scenario here, you would have AVML in an S3 bucket and then execute SSM to copy down the executable, run AVML to capture the system RAM, and then copy the memory image back up to S3.

1. https://github.com/microsoft/avml



Windows memory capture is a much easier problem.

You can use the tool of your choice. Here, we are listing three options for deploying the memory collection tool to your instance.

EKS: Elastic Kubernetes Service

- The real Kubernetes:
  - Amazon also provides a managed Kubernetes service called ECS
  - Can spin up any number of containers from the container registry

EKS[1] is the full version of Kubernetes that you can use within AWS.

From a DFIR perspective, there is no real difference between ECS and EKS.

1. https://aws.amazon.com/eks/

Most Common Container Investigations

Vulnerable applications from the AWS Marketplace

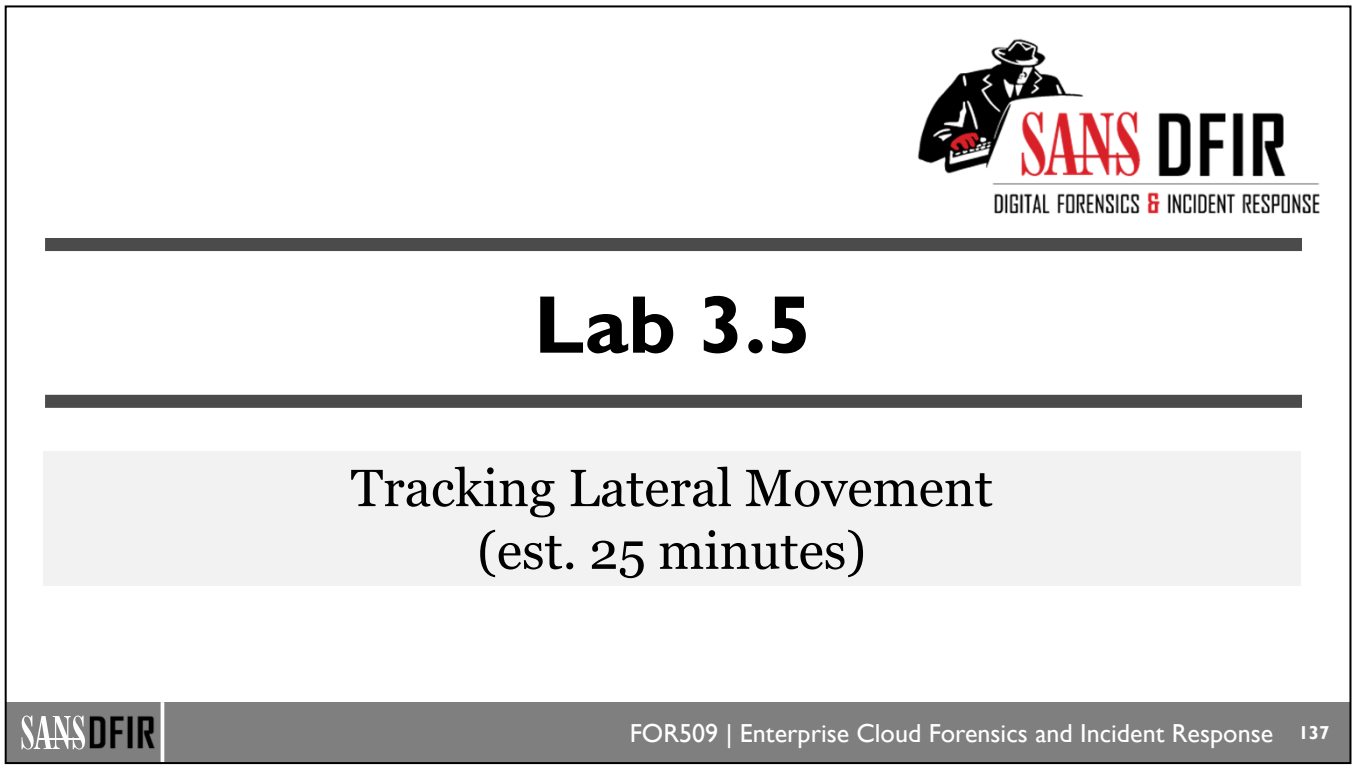
- Many developers will launch preconfigured container applications from the AWS Marketplace
- Many of these preconfigured applications are out of date and can be launched with known exploitable services

Stolen IAM roles through the metadata service

- Once an attacker has access to the container environment, they can reach the metadata service, unless the NSG prevents it
- Always available at http://169.254.169.254/latest/meta-data/
- Once the metadata service is accessed, IAM roles and credentials applied to the instance can be stolen
- This is true for v1 or v2 of the metadata service if the attacker is local on the system

The most common container investigations we encounter have to do with developers using templated clusters from the AWS Marketplace. Many developers don't realize that some of the projects that are available there for free are not maintained and do not come fully patched. Many times, they get shipped with instances that will be vulnerable once launched, and some nice threat actor is waiting to exploit them.

The second most common threat we see from within compromised containers will be seen in Lab 3.5. Threat actors are "AWS aware" and realize that the second they get access to anything that has an IAM role attached and is running in AWS, they can simply make a query to the metadata service URL and retrieve STS tokens that allow them to impersonate the service. This is difficult to detect when it's occurring as the metadata service is actually running within the AWS hypervisor and requests sent won't turn up in CloudTrail or VPC flow logs.



This page intentionally left blank.

Course Resources and Contact Information

Here is my lens. You know my methods. -Sherlock Holmes

AUTHOR CONTACT
David Cowen
dlcowen@gmail.com
Twitter: @hecfblog

SANS INSTITUTE
11200 Rockville Pike, Suite 200
North Bethesda, MD 20852
301.654.SANS(7267)

DFIR RESOURCES
digital-forensics.sans.org
Twitter: @sansforensics

SANS EMAIL
GENERAL INQUIRIES: info@sans.org
REGISTRATION: registration@sans.org
TUITION: tuition@sans.org
PRESS/PR: press@sans.org

This page intentionally left blank.