541.1

Management Plane and Network Logging

SEC541.1 | Cloud Security Attacker Techniques, Monitoring, and Threat Detection

SANS

Management Plane and Network Logging

Welcome to SEC541.1: Management Plane and Network Logging.

TABLE OF CONTENTS

This page intentionally left blank.

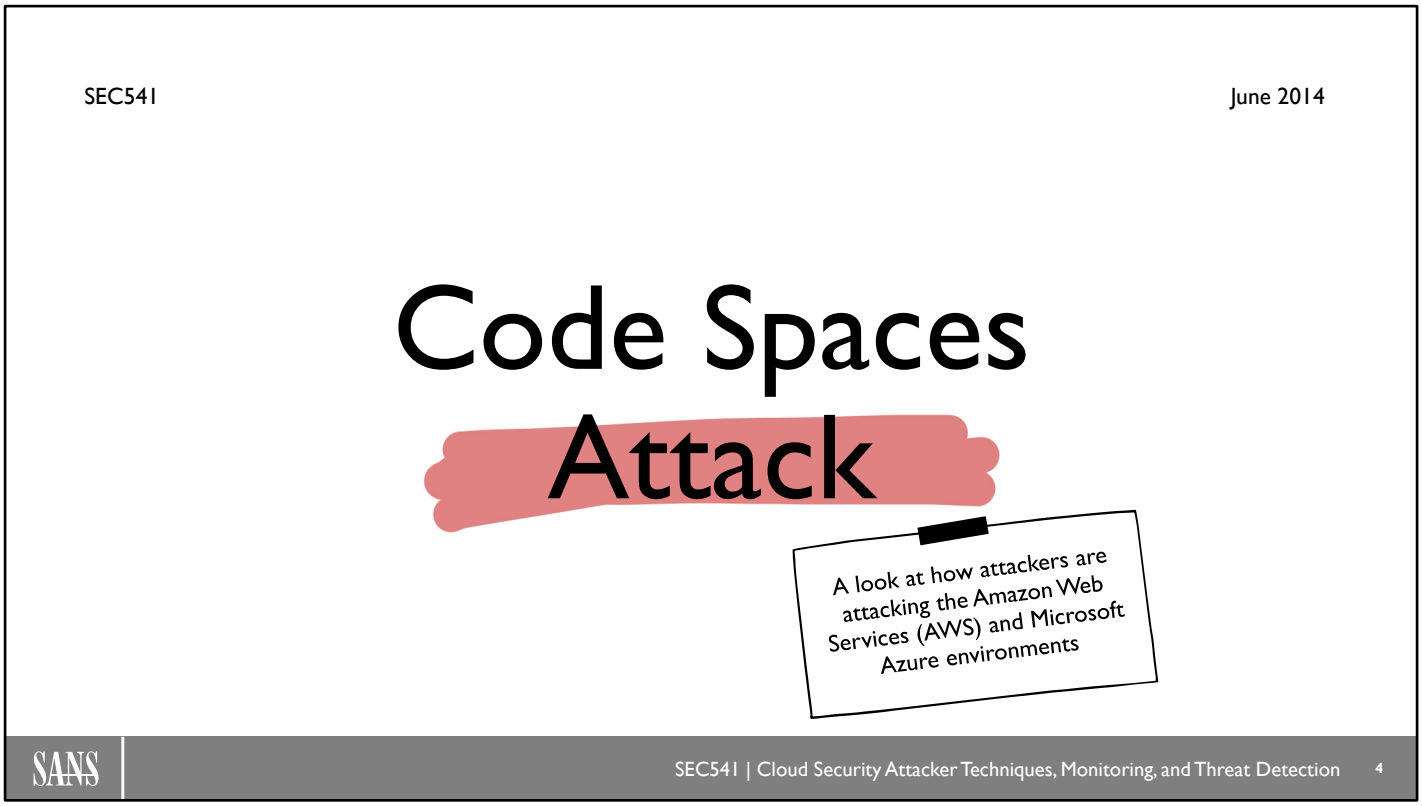
Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. **Code Spaces Attack**
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.



Often, security knowledge is a list of dos and don'ts, dry descriptions of steps to take, or large files listing suspicious IP addresses. Take the National Institute of Standards and Technology's 800-53 controls for example. Control ID of AU-11(1) is for "Auto Record Retention | Long-term Retrieval Capability". It states *Employ to ensure that long-term records generated by the system can be retrieved*. That sounds like a good idea. Keep logs for long term in case you need them. How long? That's up to the organization.

Why do we need those logs? What metrics should an organization use to decide how long is *long term*? What do we do with those logs? Who needs them? How detailed? How fast? These are the kinds of questions that you, as the security professional, will answer. You must make a case, convince your peers, boss, or the budget team of taking a particular course of action and expend time, money, and resources.

The authors of SEC541 believe this act of influence is more about story telling than spreadsheets of dry statements. A good mystery book, heist movie, or story of espionage and sabotage can help paint a picture in the mind, demonstrate risks, and answer the most important question….."so what?"

In SEC541, we will start every section major section with a real story of an attack against a company operating in the public cloud. We will walk through the attack, discuss how it could happen in your environment, and how you will be able to detect it and react to it.

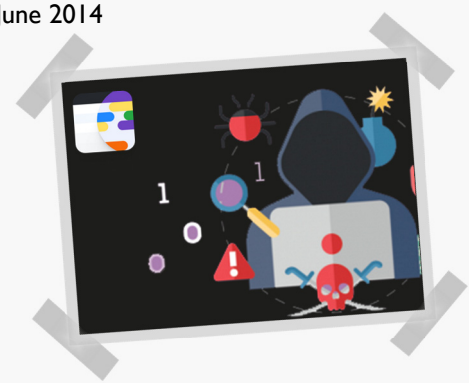
We will start off with a company called Code Spaces. Haven't heard of it before? We will get to that.

References:
https://csrc.nist.gov/publications/detail/sp/800-53/rev-5/final
https://www.esecurityplanet.com/networks/code-spaces-destroyed-by-cyber-attack/

Code Spaces Attack

Case Study

June 2014



A PaaS service for other developers, hosting development services

Initial Intrusion: Discovered administrative level cloud credentials

At Risk Infrastructure: S3 buckets and virtual machines for all customers

Code Spaces was a Platform as a Service (PaaS) company that helped developers build, test, and deploy their code. The company operated in the AWS cloud environment which may or may not be obvious to the end customers.

A PaaS cloud service is "*...to deploy onto the cloud infrastructure consumer created or acquired applications created using programming languages, libraries, services and tools supported by the provider".*

In other words, a PaaS gives developers a "platform" to do development without having to start from scratch. The developer has some of the decisions removed from them, so they can focus on what is unique about their application.

However, under the hood, the PaaS is operating on top of an Infrastructure as a Service (IaaS). This gives the consumer the ability to *"...provision processing, storage, networking and other fundamental computing resources where the consumer is able to deploy and run arbitrary software, which can include operating systems and applications".* https://csrc.nist.gov/glossary/term/infrastructure_as_a_service

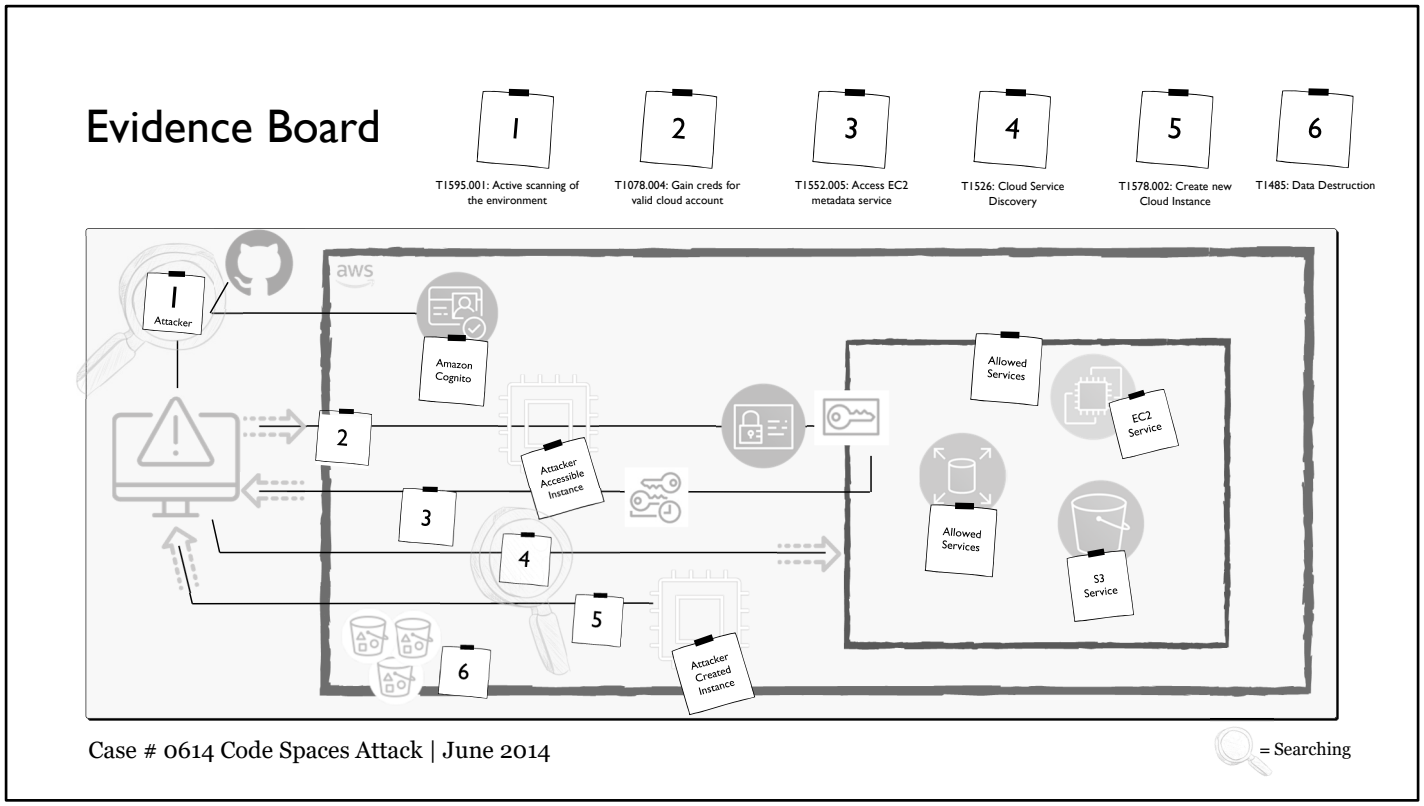
AWS and Azure are the two IaaS services that we will focus on this class. They are the top two, and most students seem to be dealing with both services, albeit at different levels. Code Spaces provided an important service in the marketplace. If a company can help a developer go from concept to operation quickly, allowing them to only focus on the app, that brings business value. Starting with just AWS or Azure, you have a blank slate, which is awesome and terrifying at the same time. But by building services that provide a lot of common services like container deployment, easy storage, test harnesses, or integration with important APIs like credit card billing can be a big win.

The initial intrusion that gave attackers access to the Code Spaces operated environment was through stolen administrative credentials. We do not always have the full story. Maybe it wasn't reported. Likely the attackers weren't caught so we never get that side of the story. For the Code Spaces attack, we aren't sure how they got the credentials.

The attackers were able to gain some level of access to the S3 buckets, virtual machines, and backup data in

the environment. In June 2014, when this attack happened, AWS and Azure did not have nearly as many services as they do today. Most organizations were moving data and VM computational systems from their on-prem environment to the cloud, operating them in a similar manner. (It wasn't until November 2014 that Lambda was introduced and started the serverless craze.)

Reference:
https://csrc.nist.gov/glossary/term/platform_as_a_service



Each story will start with the *evidence board*. The board helps us see the path the attacker took to gain initial access and move between resources and services inside the cloud environment. We break up each Case Study into multiple steps, taking each step one at a time. Each step is labeled 1, 2, 3, etc. on the evidence board, to show the resources in play.

At the top of the evidence board, we identify the MITRE ATT&CK technique used in that step. If you do not know much about MITRE ATT&CK yet, do not worry. We will introduce it in a bit.

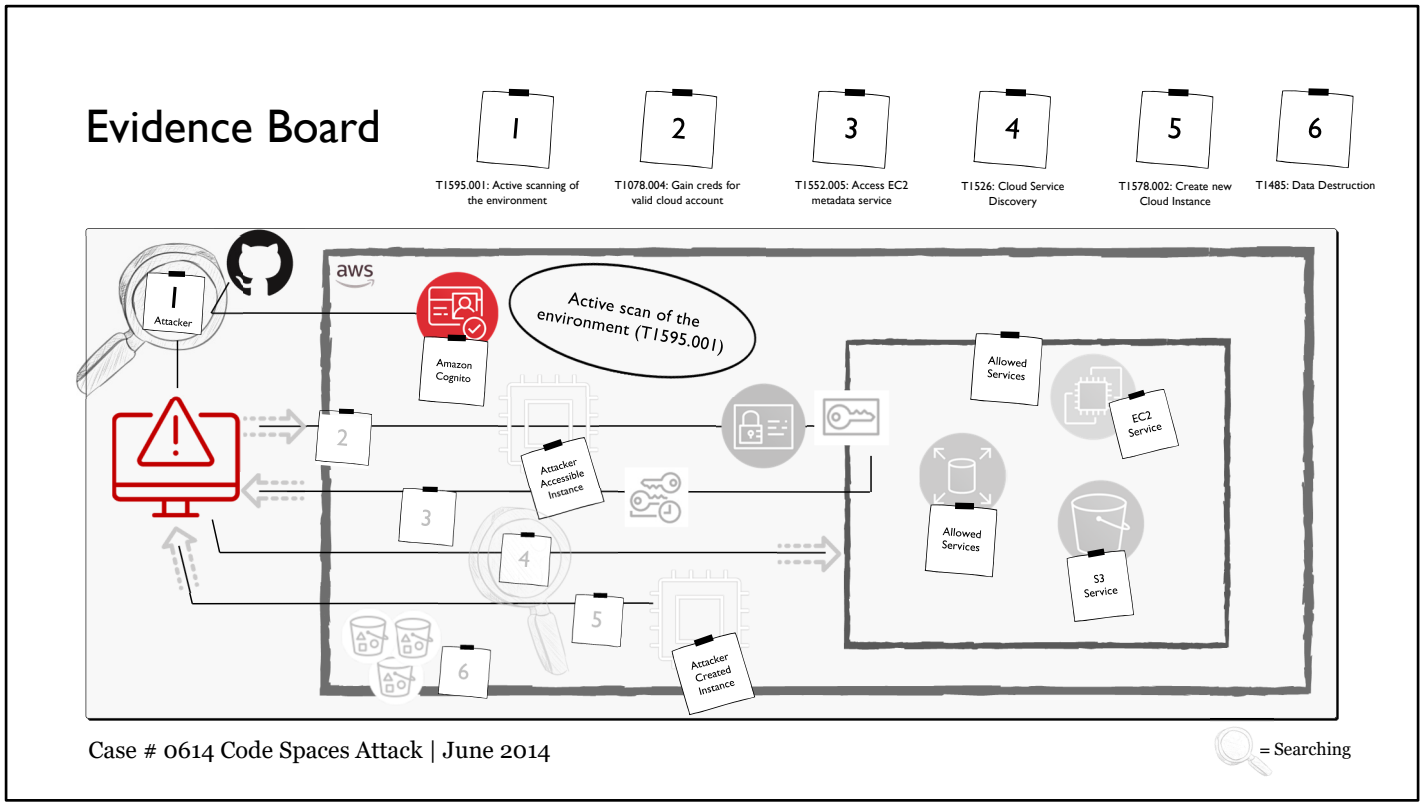
We may not have every detail perfect, or we have had to make an educated guess on the exact order of steps. This is to be expected. When you take these stories back to your organization to help demonstrate why a requested security change is needed, feel free to customize the story to fit your chosen cloud architecture.

We should also note that this class focuses on the customer side of the *shared responsibility model.* What we, as the customer of the AWS and Azure service, have responsibility over. The "Shared Responsibility" model is how cloud providers describe what they will take care of, and what we as the customer are left to manage.

We will not talk about attackers gaining access to the underlining AWS services through a flaw in CloudFormation (https://orca.security/resources/blog/aws-cloudformation-vulnerability/) or a critical vulnerability in Azure's Cosmos DB service (https://chaosdb.wiz.io/). These vulnerabilities are the vendors' responsibility to monitor, identify, and respond to if attackers have leveraged them. There is little we can do about it, except find another vendor. They are important, we should be aware and respond accordantly, but they are outside the scope of this class.

Let's dive into this attack.

Reference:
https://www.esecurityplanet.com/networks/code-spaces-destroyed-by-cyber-attack/



Just about every story starts off with scanning. We like to think about IP scans with Nmap[1], but scanning can be more OSINT focused. In today's world, an organization may leverage multiple trusted services to help build and deploy their applications. GitHub[2] and GitLab[3] are good examples. Attackers are monitoring public repositories looking for usernames, passwords, or tokens that are accidently checked into a repository. A careless developer accidently checks in a configuration file with secret information and robo scanners can monitor and use it.

GitHub can scan your environment and look for secrets. More information about *Secret Scanning*[4] on their page.

AWS provides a tool[5] that scans git repositories for secrets.

Truffle Hog[6] is another popular scanning tool.

These tools are designed for developers or security teams to scan their OWN projects. Attackers can easily adapt these tools to look for new GitHub repositories, especially if they target larger GitHub organizations.

Or, if an attacker is particularly interested in your organization, they may scan for S3 buckets that might be left open, using tools such as from Grayhat Warfare[7]. They have cataloged open buckets, and you can search for files with credentials in them.

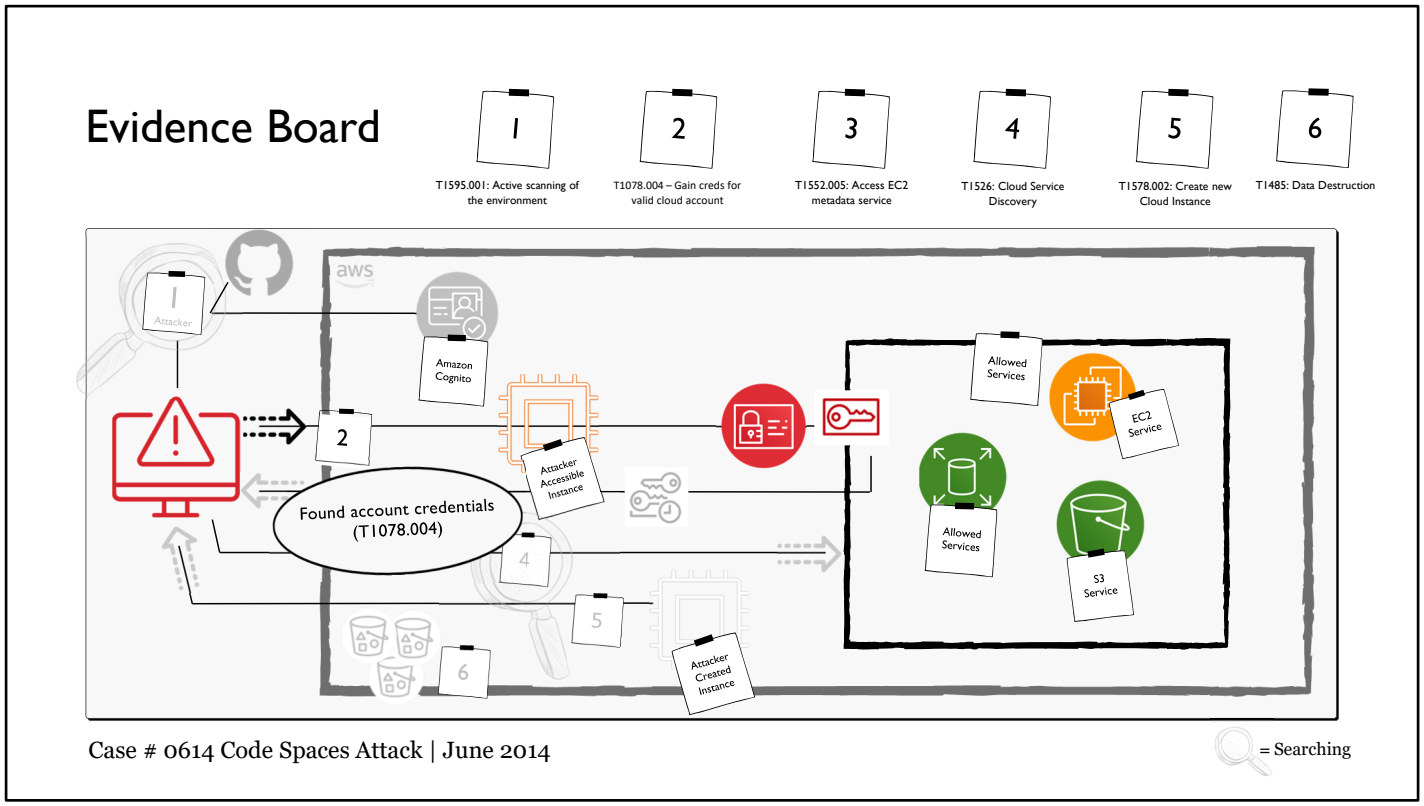
Other resources can be accidently exposed that may contain credentials. That snapshot of an EBS volume with your code and script with a token can be exposed publicly[8].

Just like AWS, the Azure Blobs can also be exposed to the internet with hidden goodies in it[9].
Sometimes, the scanning identifies resources that require a username and a password. The attacker can attempt to automatically guess what the credentials will be. We look at that on the next slide.

There is a different kind of scan we know that the victim suffered, and that was a Distributed Denial of Service

(DDoS) attack[10]. A denial-of-service attack will flood the victim with so much network traffic, that the services are overwhelmed, and they can not operate properly. When that attack is distributed among many different sources, then it can take an application offline. The Code Spaces services were hit with a DDoS attack that limited their operations to a certain point. Unfortunately for them, that was just the beginning.

[1] https://nmap.org
[2] https://github.com/about
[3] https://about.gitlab.com/
[4] https://docs.github.com/en/code-security/secret-scanning/about-secret-scanning
[5] https://github.com/awslabs/git-secrets
[6] https://github.com/trufflesecurity/truffleHog
[7] https://buckets.grayhatwarfare.com/
[8] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-modifying-snapshot-permissions.html
[9] https://www.cyberark.com/resources/threat-research-blog/hunting-azure-blobs-exposes-millions-of-sensitive-files
[10] https://attack.mitre.org/techniques/T1498/



Most initial access to a victim's cloud environment tends to fall into one of 3 categories.

Attacker found an unconfigured or default access vector
We spend money for AWS and Azure because we want to make something and run it, such as customer relationship management systems, hosted GitLab servers, or Kubernetes. In the light of the *shared responsibility* model, these services are the responsibility of the customer. They own the configuration and management. Sometimes, many times, they mess it up. Later in the class we will have a story about this attack.

Hacking an application
How many web applications are running in your environment? Probably a lot more than you can count. If you work in an organization where development teams own the life cycle of a project, the security team may not even know how many deployed resources are sitting out there. Every application is unique, developed by people who can make mistakes. Attackers can find vulnerabilities that not only give them access to that application but may open up the rest of the cloud environment. We will look at this attack in the class as well.

Attacker found or brute forced credentials
If an attacker can gain the username and password with some level of access to the cloud platform or some internet accessible facing management server, then they are able to perform whatever actions were granted to that account. Was it a read only account or an admin account? Can they steal secrets or spin up a crypto miner? As a detection and monitoring focused class, we are not going to dive into the depth of security controls. SEC488[1] is a great class for that. This is what the Code Spaces attackers did. Before they started using the stolen creds, they had to find it.

If an attacker finds the credentials, then they can walk in the front door. They may try and guess using commonly used passwords, WINTER2022. Automated guessing is called a brute force attack. In this class, you get to perform a brute force SSH attack and detect it against your environment.

There are other attack vectors that could be more effective but harder to accomplish, such as adding a backdoor to a shared resources or moving laterally from a trusted environment[2]. But, in this case, the attacker

discovered some credentials that gave them limited, but significant, access to the AWS environment.

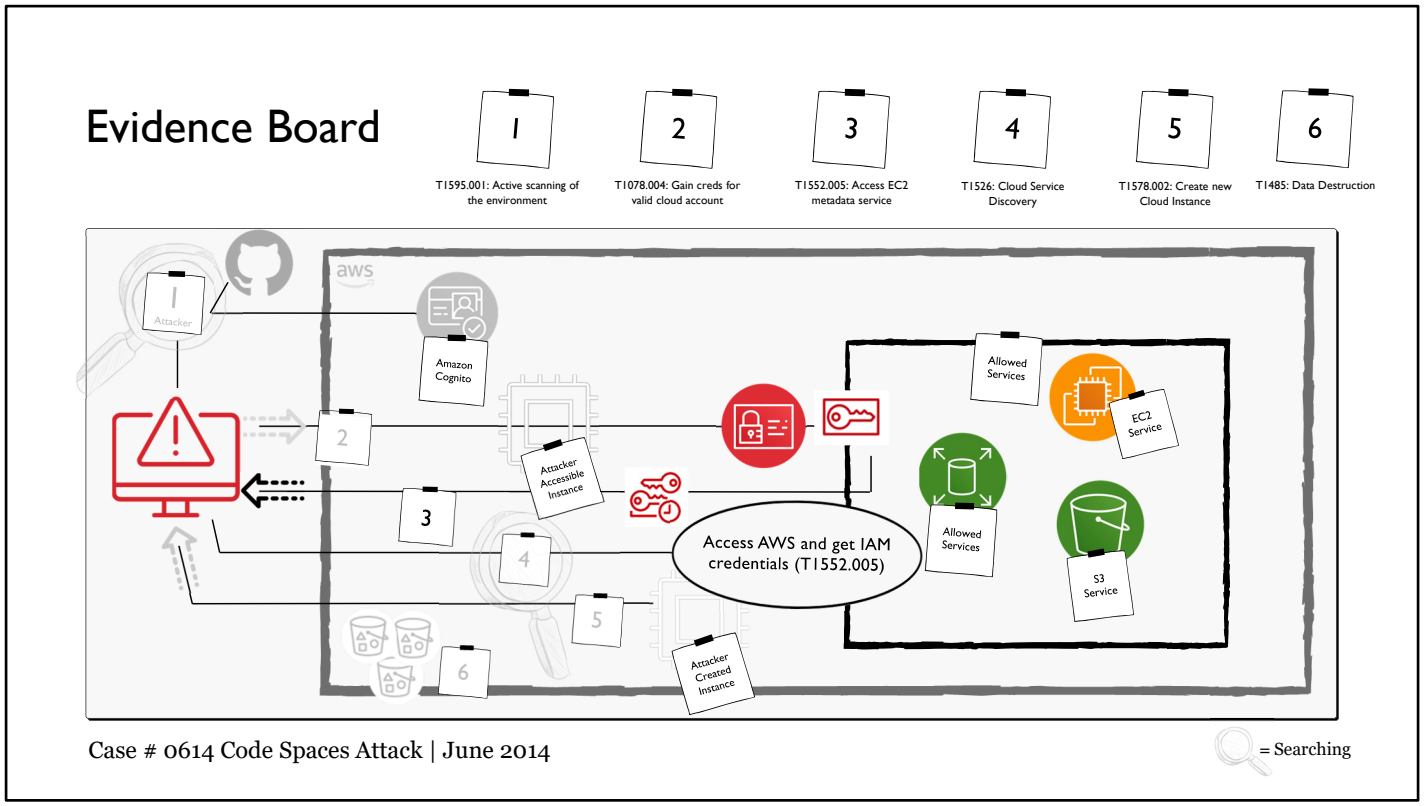
There is a whole host of attackers[3] just looking for a quick entry to an environment, ransom access, and then make a quick buck (or bitcoin).

It is not clear how the attacker gained the credentials. They may have been able to guess a username and password through brute force, or by making educated guesses[4]. One byproduct of automation, DevOps, and other "deployment by code" tools and capabilities is that access credentials are sometimes accidently checked into a repository[5]. This can happen if the wrong file gets pushed, or maybe it was a misconfiguration. GitHub and AWS do have bots running that will try and detect AWS account information being checked into GitHub[6], but are they faster than the ransomware team?

Once the attacker found the account credentials, the AWS accepted them as a valid user with a given level of privileges.

[1] https://www.sans.org/cyber-security-courses/cloud-security-essentials/
[2] https://tldrsec.com/blog/lesser-known-aws-attacks/
[3] https://krebsonsecurity.com/2021/05/a-closer-look-at-the-darkside-ransomware-gang/
[4] https://nordpass.com/most-common-passwords-list/
[5] https://www.theregister.com/2015/01/06/dev_blunder_shows_github_crawling_with_keyslurping_bots/
[6] https://www.dannyguo.com/blog/i-published-my-aws-secret-key-to-github/

Reference:
Ransomware Webcast by SANS - https://www.sans.org/mlp/ransomware/



When an attacker gains access to a cloud environment, those credentials could be long lived or temporary. In the case of Code Spaces, the attacker was able to continuously log into the environment, so likely a more permanent credential. Long term credentials are a typical way to allow resources not part of the cloud environment to gain access. Usually the resources are people, but it could also be local virtual machines. These are high risk areas and should be protected at all costs.

Humans logging into any kind of infrastructure should always use *multi-factor authentication (MFA).* AWS and Azure have published articles on how to implement MFA properly. An MFA method would have stopped this attack in its tracks.

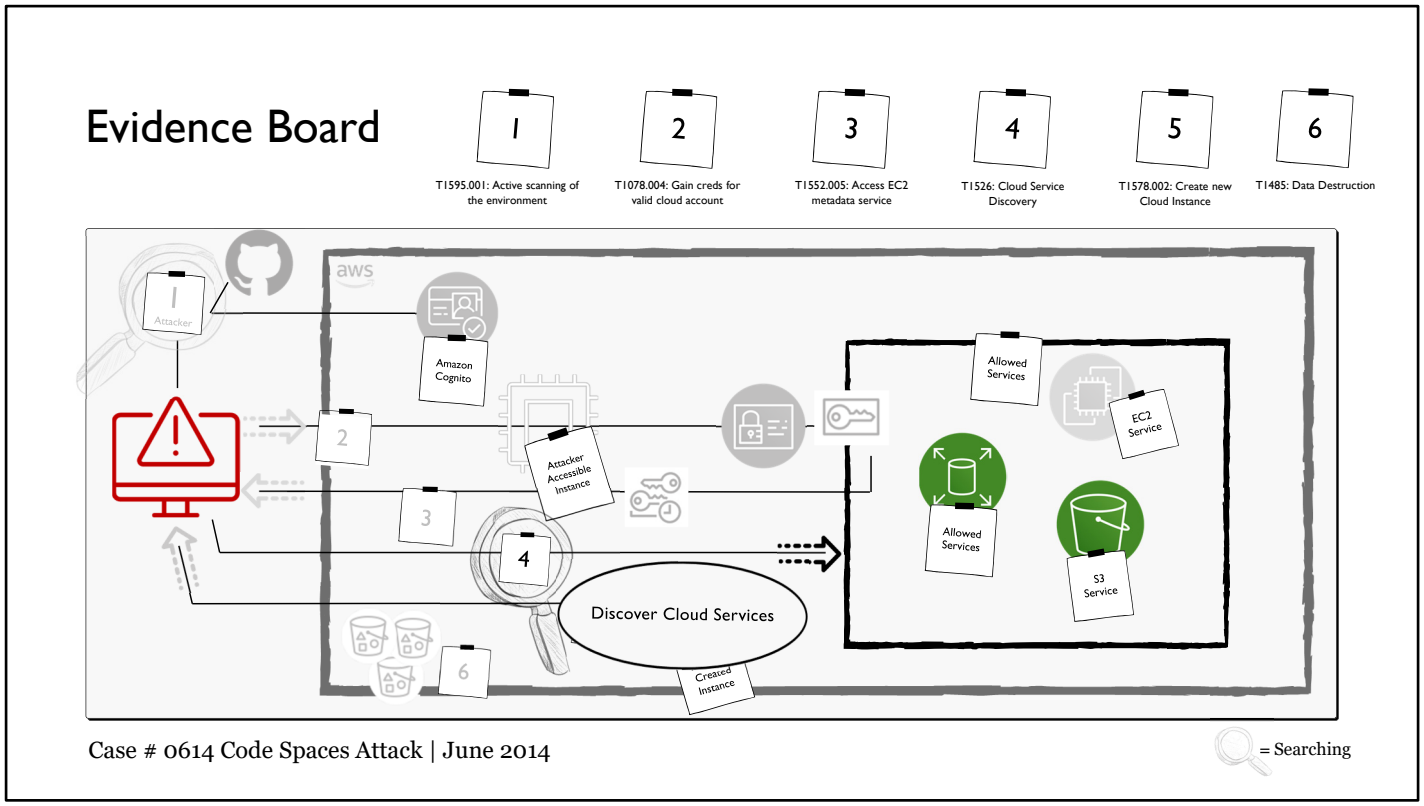
In an AWS environment, long term authentication usually means username and password is sent to an authenticator service. The authenticator evaluates the provided information and returns credential information. In the AWS environment, to use the API, a key ID and secret key may be used by the AWS CLI to access the API. The AWS authenticator will return back a token that is temporary and used to prove that you have been authenticated.

A key ID could be used by a human, or by a computational system such as an EC2 which has been given an IAM Role to access S3 buckets.

The attacker gained access to the environment, likely providing a guessed or found username and password credential, and the AWS authenticator returned the token information. Now, all activity the attacker takes is tracked with that key ID.

Next, the attacker needs to learn what they are authorized to do with these credentials.

References:
https://aws.amazon.com/iam/features/mfa/
https://docs.microsoft.com/en-us/azure/active-directory/authentication/concept-mfa-howitworks



Once the attacker has gained some level of access to the environment, their next question will be "What can I do with this?". In the case of Code Spaces, the attacker started poking around.

In a traditional on-premise environment, the attacker might have had to conduct a network scan to see what systems are running, and what accesses are presented on each host, such as Remote Desktop Protocol (RDP), File Transfer Protocol (FTP), or Server Message Block (SMB).

This is still a valid and viable method to perform a "Discovery" tactic. It's focused on what the cloud customer has installed and running in their environment on their virtual machines.

But the cloud companies provide their own built-in Cloud Discovery Services. According to "The NIST Definition of Cloud Computing", there are five tenets of a cloud computing service: on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service.

In order to satisfy on-demand self-service and rapid elasticity, the cloud service must make it easy to automatically perform actions on the cloud services. We are using the cloud because we want to automatically spin up thirty servers at a moment's notice. This means we need easy ways to ask the cloud service what is running, how is it configured, and maybe make changes.

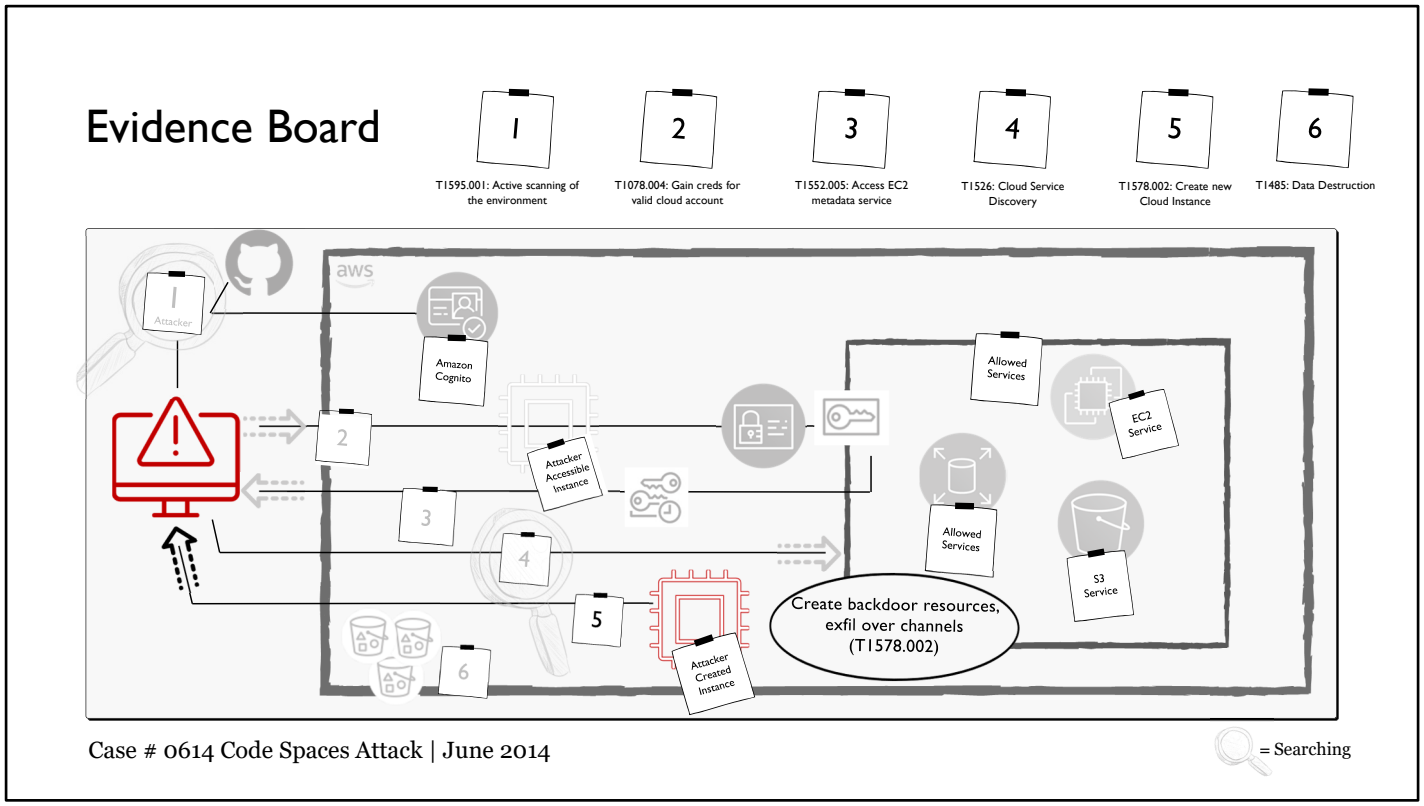
The attacker would have performed a Cloud Discovery operation using their stolen credentials, and they found that they had access to the "EC2 Control Panel". The IAM role assigned to that person's stolen credentials allowed them to not only view resources, but to create them—or destroy them!

The EC2 control panel provided the attackers access to the some EC2s, S3 buckets, EBS snapshots, and all AMIs.

The goal of this attacker was to make some cash, in some crypto form. The attacker was leaving messages for the company to contact a Hotmail address. When Codes Spaces replied, the attacker demanded payment to resolve the DDoS attack.

A nice feature of AWS and Azure is their DDoS protections, which do come with a premium cost. However, the attackers had detailed inside knowledge of the EC2 deployments of this account, which they very likely leveraged to focus the effect of the DDoS.

References:
Remote Desktop Protocol: https://docs.microsoft.com/en-us/troubleshoot/windows-server/remote/understanding-remote-desktop-protocol
File Transfer Protocol: https://datatracker.ietf.org/doc/html/rfc959
Server Message Block: https://www.sans.org/reading-room/whitepapers/detection/paper/37472
NIST SP 800-145: https://csrc.nist.gov/publications/detail/sp/800-145/final



The company knew their control panel had been accessed. The company determined that they were in the clear since the intruder did not have access to private key information, so rather than pay the ransom, the company changed the password they used. However, the attackers had more access than was assumed. The attackers had already used their accesses to create new resources to act as backdoors. They gave themselves other means of gaining access even if the passwords were changed.

Threat modeling is the process of understanding the threats and potential vulnerabilities of your organization; understanding what kind of attacker might go after your particular environment. Hospitals and city municipal environments are targets for ransomware attacks, while producers of internet/software technologies might be vulnerable to "supply chain" attacks. Understanding what the attacker might attempt against your environment might help focus your detections, analytic capabilities, or your security improvements.

This attacker seemed a bit of both. Their real goal was ransom, they provided the Code Spaces team with a demand for money. The data that Code Spaces had was likely not that interesting to the attackers. If your company is a financial organization, or a government contractor, then the attacker will operate very differently.

Typical ransomware attacks will lock the victim out of access, or encrypt the computers and sell back the decryption tool. These attackers leveraged their good fortune of access to the EC2 console to build backdoors into the environment. They were able to create new services, likely EC2s with its own SSH access, so that the attackers could gain back access if they were discovered.

They may have done much more. In 2014, S3 buckets were trivial to turn from "protected" to "publicly hosted website" with a configuration change. S3 buckets were leaking everywhere. Today, you have to jump through a few hoops and there are multiple ways to lock down an S3 bucket. EBS volumes, Lambda function, even EMR could be made public. A company that has sensitive data, such as a government contractor or a financial institute, should architect their environment so that sensitive resources are completely separate from public facing resources, and any changes sends off warning bells.

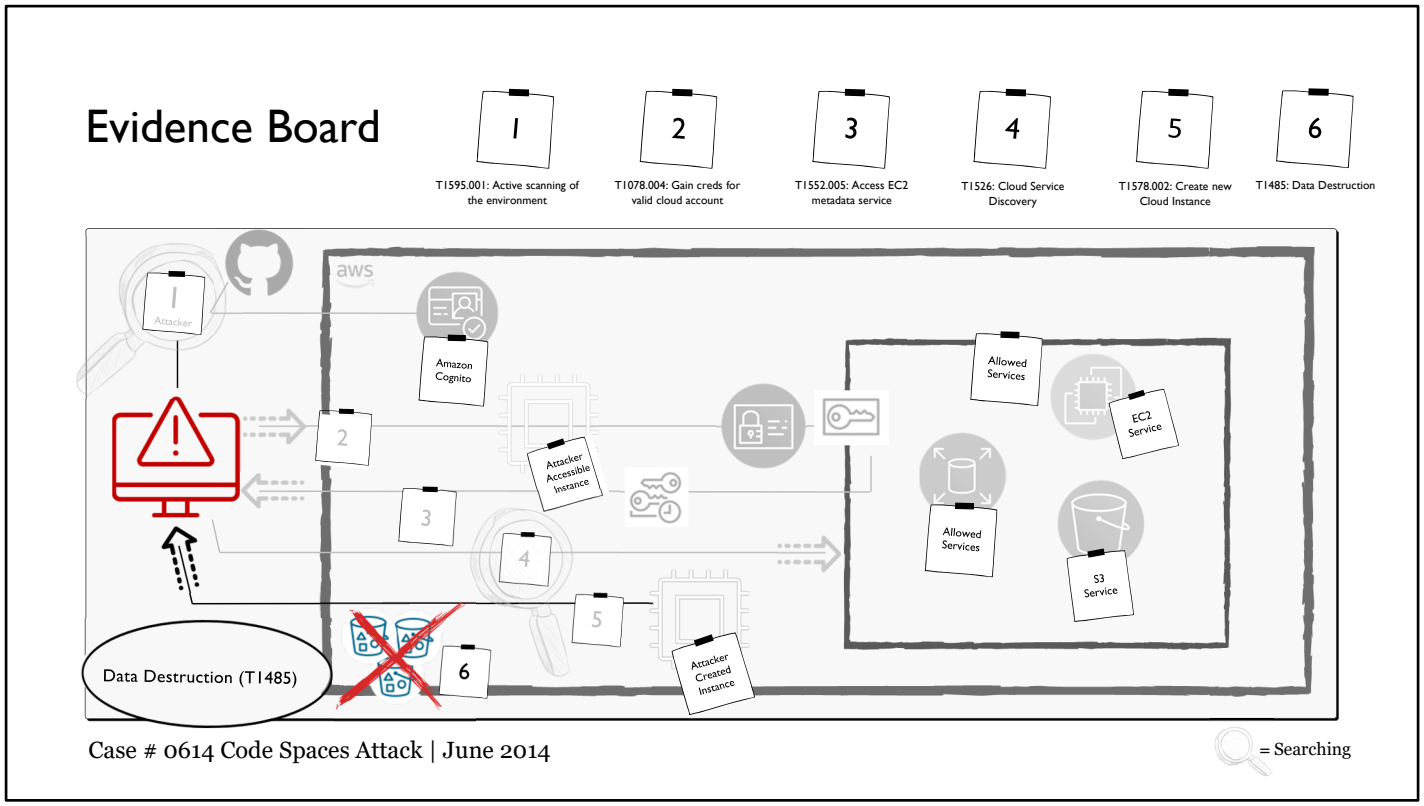
In the case of Code Spaces, we do not have knowledge that resources were made public or exfilled; we will see that another day.

MITRE ATT&CK technique T1578.002 is a defense evasion tactic created by spinning up new cloud instances and new resources inside the environment that may not be noticed[1].

Note: SANS has a set or resources dedicated to ransomware[2].

[1] https://attack.mitre.org/techniques/T1578/002/
[2] https://www.sans.org/mlp/ransomware/

References:
Threat Modeling Paper: https://www.sans.org/reading-room/whitepapers/threats/paper/38955
Ransomware Attacks: https://healthitsecurity.com/news/fbi-probing-2-hospital-ransomware-attacks-hackers-remove-health-data
SolarWinds: https://www.npr.org/2021/04/16/985439655/a-worst-nightmare-cyberattack-the-untold-story-of-the-solarwinds-hack
Public AWS Resources: https://github.com/SummitRoute/aws_exposable_resources
MITRE ATT&CK technique T1578.002: https://attack.mitre.org/techniques/T1578/002/



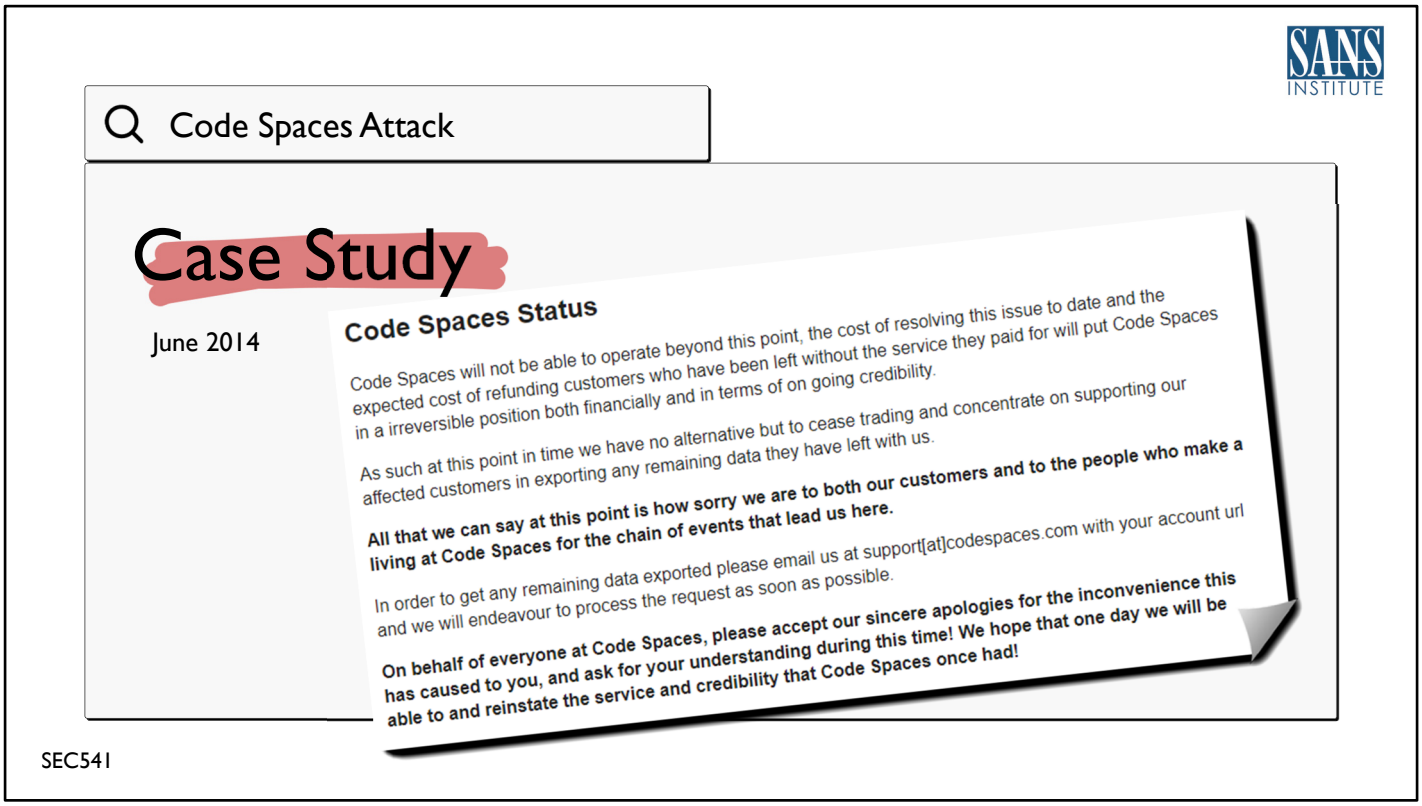
With those backup accesses, the attackers were able to regain control over the cloud's data management plane. They could manipulate the resources they created, and other customer resources as well. When the attacker saw Code Spaces attempting to regain control, they simply used their backdoor resource, likely undiscovered by the company, to delete all the EC2s, block storage, and S3 buckets. This was all of the customer's data. Not only their working data, but all backups as well.

Data destruction (T1485) is a particularly nasty attacker tactic. The rise of ransomware has automated the '*encrypt for cryptocoin*' model that is targeting businesses across the verticals. This was a worse case situation for Code Spaces, causing them to have to shut down. A number of security best practices would have made this attack harder. We should consider that this attack happened in 2014, and Azure and AWS have since created a number of security frameworks and improved services. For instance, backups should always be kept in a different account, so that if access is gained in the prod account, they cannot manipulate the backup account.

In this class, we are interested in figuring out how to detect these families of attacks, monitor for nefarious activity, and build investigative practices. Blog posts will tell you to just lock down the entire network and do all your security controls perfectly. We don't live in a perfect world, and every security team has to live in the balance. Knowing what attacks are likely and the gaps in your security controls will help pinpoint how and where to monitor for attacker techniques. As your environment matures and you close some gaps, you can redirect monitoring.

For instance, it is easy to write off the Code Spaces story just by saying "Multi-factor authentication". That likely would have kept the attacker from gaining access to the console. But as we talked about in previous slides, there are a number of attack vectors. Closing one door just leaves 50 more open for attack. Cloud environments are constantly changing. Developers are always needing new tools and capabilities. Security teams are always trying to keep up. There will be gaps. Find the gaps and start looking for the attack.

Reference:
https://attack.mitre.org/techniques/T1485/



Code Spaces ended up throwing in the towel when they realized customer data was gone and unrecoverable. They went out of business. This message was on their website, pulled from the web.archive.org[1].

On the page, they list what they lost.

- Apache Subversion code repositories[2]
- All Git repos were available for export, but all backups and snapshots were gone
- All VM machines
- All EBS volumes with database files

[1] https://en.wikipedia.org/wiki/Apache_Subversion
[2] https://web.archive.org/web/20140624170450/http://www.codespaces.com/

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. **Course Overview**
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Welcome (1)

Who should take this class?

- Those who want to learn how to analyze and detect attacks happening in AWS and Azure
- Those who want to design AWS and Azure infrastructure so that it's more secure and investigable
- Those with hands-on experience with AWS or Azure

SEC488 Cloud Security Essentials is a great intro to cloud security.

This class welcomes everyone who is part of the cloud security circle of life. We are seeing the lines blur between architecture, SOC analysis, hunting, compliance, and operations. Infrastructure as a Service cloud providers give everyone the ability to build, create, and operate the entire stack, from application down to network. It is up to the organizations to determine how to break up that responsibility.

Some organizations use a traditional separation of duties, where developers build an application and throw it to the IT operations organization. In these organizations, you may see a dedicated security team responsible for deployment and monitoring of the cloud IT resources. The downside for these teams is that they may miss out on the speed to deployment that the cloud can provide, keeping the more traditional linear approach to deployment, with formal gates setup.

On the other end of the spectrum, the commercial cloud offers a new opportunity for development teams to own their entire product stack. The downside for these teams is that they may create different deployment and security control requirements, opening themselves up for undetected security vulnerabilities. Typically, these organizations will start building security teams later, who have the ability to recommend, rather than enforce security requirements.

We bring this up because it may not be a dedicated security team who is responsible for detecting and responding to threats. Security teams may be responsible for performing the actions, but the developers and architects need to create solutions that are monitorable.

This class assumes the students have a working knowledge of either AWS or Azure. Understanding cloud concepts, such as the shared responsibility model, virtualized networking, basics of identity management, and deploying core services are the foundation to monitoring them.

SANS SEC488 is a fantastic intro to Cloud Security and is authored by one of the SEC541 authors. Information about SEC488 can be found at https://www.sans.org/cyber-security-courses/cloud-security-essentials/.

This class focuses on the two main cloud service providers, AWS and Azure. Our in-class discussions will look at a concept, then discuss the AWS and Azure specifics. When we go to demonstrate the material in the labs, we had to pick a cloud platform, so we use AWS. However, all of Section 4 is Azure focused, including the labs. While lectures are 50/50 AWS and Azure, the labs are more 75/25 AWS to Azure for that reason.

Welcome (2)

What you need for this class:

- A computer with a web browser to get to the AWS Web Console.
- For OnDemand students, you will need to have an AWS account with full access. Steps to setup your account can be found in Lab 0 of the workbook.
- For Live/Live Online classes, SANS will provide you with an account for the class labs during our course. We have provided instructions for performing the labs in your own AWS environment after you leave the class.

For OnDemand students, you will need to provision an AWS account for your Labs. Follow the instructions in Lab 0.

For live students, your account will be provisioned for you, with connection details in the My Labs portal.

All labs for this class are happening in the cloud account, so your computer will only need a modern browser to do the labs.

More details are on the class site:
https://www.sans.org/cyber-security-courses/cloud-security-monitoring-threat-detection/

What This Course *IS NOT*

- Not a deep dive into all cloud service providers (CSPs)
- Not a cloud architecture or operations class

Instead, we will teach you…

- How to monitor and investigate AWS and Azure
- How data is really generated, stored, and used in AWS and Azure
- The building blocks you need to build a robust threat program

What This Course *IS NOT*

When you go to most vendors for a cloud training class, you will be getting a how-to for specific services. "Let's spend 30 minutes talking about how to use S3."

This class is not an architecture or sysops class. We will not be learning how to effectively manage your elastic environments. Instead, we will be looking at the types of attacks we observe against commercial cloud environments, and understand how we might detect, analyze, or investigate these types of attacks.

You will also get the information you need to understand how to detect tomorrow's new attack or handle the newest AWS or Azure service released next year.

Section 1: Management Plane and Network Logging

- We will look at the management plane of AWS and Azure, how we can use it for investigations, and then look at network logging.
- Major topics include:
  - Course Overview
  - MITRE ATT&CK and Definitions
  - API Logging
  - Parsing JSON
  - Cloud-Native Logging Services
  - Network Flow Logging
  - Capturing Raw Network Traffic

This page intentionally left blank.

Section 2: Compute and Cloud Services Logging

- We will investigate how compute service logging works in the cloud, focusing on containers and virtual machines.
- Major topics include:
  - Tesla Attack
  - Host Logs
  - Log Agents
  - Containers
  - Managed Container Services
  - Cloud Service Logs

This page intentionally left blank.

Section 3: Cloud Service and Data Discovery

- We first look at the cloud metadata services. Then we will investigate inventory and security services in the cloud vendors.
- Major topics include:
  - Capital One Attack
  - Metadata Services and GuardDuty
  - Cloud Inventory
  - Data Discovery
  - Vulnerability Analysis Services
  - Data Centralization Techniques

This page intentionally left blank.

Section 4: Microsoft Ecosystem

- We will spend all of Section 4 focusing on Microsoft, and we get two attack use cases in one section!
- Major topics include:
  - Malwarebytes Attack
  - Microsoft 365
  - SolarWinds Attack
  - Azure Active Directory
  - Storage Monitoring
  - Detection Services
  - Network Traffic Analysis

This page intentionally left blank.

Section 5: Automated Response Actions and CloudWars

- We first look at automation in the cloud, and how we can use it for our investigations.
- Major topics include:
  - Automated Response Actions
  - IT Ops Workflows
  - Security Workflows
  - Constructing Response Actions

This page intentionally left blank.

CloudWars!

CloudWars

- This is a **friendly challenge** to put your cloud knowledge and skills to the test.
- You will be quizzed and tested against AWS and Azure cloud environments.
- Do not worry, it is not just digging through logs.
- In a live class, most points earns a coin, but the most important thing is to **LEARN** and **HAVE FUN!**

At the end of class, you will be participating in CloudWars. Do not think about it as a competition, the points are not as important as the learning.

You will work through a set of challenges, based on any of the information from class.

CloudWars also comes with a hint system, so if you are stuck, you can get that hint to get help you out.

It will be fun!

Monitoring and Threat Detection

As we've come to realize, the idea that security starts and ends with the purchase of a prepackaged firewall is simply misguided.

-Art Wittmann

Let's start off by defining some terms we will be using throughout class. It is all about observation. Seeing what is happening in your environment and being able to tell if what you observe is normal or abnormal. This can be difficult in ever-changing environments. Some organizations give logs to a tech and say, "go hunt", but without the context of what is normal and what is abnormal. Digging through piles of logs without a systematic process or goal can lead to wasted time and resources. First, let's look at two words we will be using a lot.

The definition of **monitoring** is to observe and check the progress or quality of something over a period of time; keep under systematic review. We are collecting data over time and seeing anomalies, changes, or hitting some threshold that we have pre-determined to be worthy of investigation. Take the fuel gauge in a car. We are monitoring the amount of fuel in the tank as that amount changes over time. When it hits a certain threshold, a light comes on to tell us we need to go fill it up.

The definition of **detection** is the action or process of identifying the presence of something concealed. Detecting is the act of uncovering. We are specifically homing in on the phrase "*identifying the presence of something concealed*". An attacker wants to keep themselves hidden. They may be terrible at hiding, but that's beside the point. In order to ***detect the threat*,** we must be able to observe the operation of the environment and determine when there is a change. Our infrastructure is always changing, and our security knowledge is always improving. (We hope it is!) We will learn new things, detect new attacks, and move those detections into our automated monitoring services.

Definitions are from the Oxford Dictionary: https://www.oed.com/

Types of Detections

There are several ways to categorize types of detections:

- Atomic: Detecting specific indicators that are known bad. IP, domains, and file hashes are common.
- Statistical: Looking for outliers in logs such as most talkative, uniqueness in environment, anomalies
- Behavioral: Tracking across multiple logs, usually related but different, and identifying a set of actions that together hit a threshold

Not all detections are created equal, and some hidden threats are easier to spot than others. It's all about what data must be collected, and what formula is needed to determine that a threat is detected. We like to look at the art of **detection** in three categories, based on the type of data needed to be collected and how universal the detection algorithm is. Each of these types of indicators have pros and cons, which we will discuss.

Atomic

Atomic indicators are the easiest method of monitoring for indicators of an attack or compromise, if you have the right data. An atomic indicator is a single piece of information that can be used to determine good or bad. Take file hashes for an example. Every file on your computer can be hashed, or a unique fingerprint of a file, and compared to a list of known malware fingerprints. Antivirus companies make a living gathering known bad malware hashes and shipping them to endpoint products running on your computer. Network IPs and domains are network-based atomic indicators that can be compared to collected network logs, detecting potential communications. Lenny Zeltser publishes a list of freely available bad IP's, domains, and URLs on his site[1].

Atomic Indicators are clear cut. "These files are known malware and bad so automatically delete them from your computer". No ambiguity. This is a data collection and delivery problem. According to one research organization, there are 10's of millions of new malware, or malware variants, created every month (https://www.av-test.org/en/statistics/malware/). Buying a malware application that gathers atomic indicators and sends them their endpoint products is relatively easy. The disadvantage of these atomic indicators is that they have to have already been observed, analyzed, determined to be a known bad, then deployed to all customer endpoints before they can be effective. New or altered malware will slip past those detections.

Check out this SANS article on "*Obfuscation and Polymorphism in Interpreted Code*"[2].

Note: This industry grew up in a world where host computers and servers can operate for years. What happens in your cloud environment when a server lives for an hour? Rather than upgrade your running virtual machines, you upgrade and patch your base image and redeploy all new servers over the course of 24 hours. Is that malware program still as useful? It is up to your organizations to determine how to protect those endpoints.

Or the attacker may use the hosts tools against them. Living off the Land is such a technique, were the attacker uses the powerful host tools themselves rather than deploying custom malware. In our upcoming discussion, we will see how an attacker can do a significant amount of damage with a single AWS or Azure command line tool[3].

We need to add new detections to our security programs.

Statistical
Statistical detections look for outliers, changes, or disruptions that look different than the baselines. This is tricky, because all the words in that previous sentence is up to you to decide. How do you create a baseline, what does "outlier" mean in your case? The formula is very dependent on the architecture and processes of your organization.

Let's say you have 1,000 web servers running over the course of the week. If one of them gets hacked and a new piece of malware is dropped, one that you have no atomic indicator for, then you could look across all your files to find one that is unique. You create a baseline, such as all the common files, then you grab all the unique files. This is an indicator but isn't a slam dunk case of malware. It could be a configuration problem, or a weird file that the application created in the wrong directory. It could be something to explore.

Another example is more cloud specific. All of your systems run in the United States regions. Doing some analysis, you see a single VM running in a European region. Is this a mistake, or an adversary's hidden Bitcoin miner? Network traffic is another place where statistical analysis can be used. In this class, we will show you how to collect some network traffic across your environment. You could analyze the data and find that one of your web servers is talking more than the others. Misconfiguration or data exfiltration?

Behavioral
Behavioral detections are even more tricky. A behavioral detection usually requires a number of different logs, from different telemetry sources, that tell a story of questionable activity. Network traffic showing a brute-force attack may not be a cause for concern (it's happening all the time). However, a brute-force attack that is followed by a host calling out to an unknown IP address? That sounds like it could be a successful drive-by brute-force attack, allowing the attacker to gain access, then dropping a piece of malware that is calling back to their command-and-control service. There has been a lot of research into true behavioral analytics, and some security companies with advanced network or host-based intrusion systems leverage behavioral analytics, but it is hard to rely on automated behavioral analytics because the chain of events are so unpredictable.

[1] https://zeltser.com/malicious-ip-blocklists
[2] https://www.sans.org/white-papers/37602
[3] https://www.sans.org/webcasts/wolf-sheeps-clothing-dissecting-living-land-techniques-114615

What is Threat Hunting?

The proactive evaluation of the infrastructure operations to detect a threat beyond the deployed security tools

Or digging through logs to identify someone messing with your infrastructure

Let's talk about threat hunting for a minute. Threat hunting is a method to detect an adversary behavior. Just randomly picking out logs and staring at them will produce little revelations. We need a starting point, some action or indicator to look for. Then, we will build a systematic method to identify those behavioral indicators.

"Hunting" conjures the idea that you will have to go out there and find the adversary activity happening in your environment. It's not easily detected by automated security; threat hunting focuses on trying to identify active threats in your environment. Current or past security events that have not been triggered by automated security systems, are not easily visible, and require an investigation to find them.

By starting with t*hreat*, we are looking at what other actors have done before, understanding their methods and how vulnerable we are to those methods, and then investigating our own environment looking for those behaviors.

Despite the prolific use of the buzz word, threat hunting is not the only way to approach detection and monitoring in your environment, but it is a good starting point for finding the most evasive attacks that have evaded your atomic and statistical indicators.

What is Threat Hunting?

We will approach Threat Hunting in this class as a process of three main steps.

- Build a hypothesis. What potential threat should we investigate, that is likely to occur, that we may not detect automatically?
- Conduct an investigation. Leverage the tools we have in AWS to determine if a threat is active.
- Propose improvements. Threat hunting can be difficult in certain environments. We may be able to improve the detection or mitigate the threat.

We break down threat hunting into a 3-step process:

Build a hypothesis
A threat hunter cannot effectively do their job without knowing what they are looking for. Just as a seasoned bargain hunter must know what stores have the best sales on a particular day, the threat hunter must know what techniques attackers could use, and how likely it is they will use them against their own environment. That can be tough without a dedicated threat research team, but we will show you resources that you can use.

We can take our Code Spaces story from earlier. We see a company that was attacked, and we understand HOW it was likely attacked. Now, we start asking ourselves, can we detect a brute-force attack? Do we have logins to our console coming from unknown locations? Can we scan inventory and determine if an unauthorized resource has been created? The hypothesis is "Attackers gaining access to our environment can spin up backdoor resources, and we can inventory our accounts and detect that." We then need to go ask some questions such as:

- Would we be a target for ransom attack? (Today, it seems that answer is aways "yes".)
- Do we have access to the data we need to determine what the inventory is?
- Can we determine if unauthorized or unplanned resources are spun up?
- If we find them, can we track down who the owner is?

Conduct an investigation
Now that we have a plan, we start investigating. We have a number of methods to get the data we need. We probably first start clicking through the cloud's management console. It's the easiest way to see data. We quickly realize that for decent size environments, that will be quickly unmanageable. So, we start using command lines or SDKs to pull the data we need, analyze it, sift through it, and determine if we can see activity and what it looks like.

If we don't detect some activity that can mean either the attack didn't happen, or we are not able to detect it. False negatives can be dangerous. This is one of the reasons Red Teams have gained popularity in recent years. Having a team perform the attack, recording WHAT they did, then measure the effectiveness of the

threat hunting is the best way to validate a hypothesis. The course authors have brought that Red Team mindset to this class. Many of the labs will start off with you performing the attack, then guiding you through the hunting and detection of that attack in a controlled environment. The course authors hope you take these methods and test them in your organization's larger testing or production environments.

Propose improvements
We get sad when we see security teams have to work around difficult architecture, lacking data collection, or worse have too much data lacking context. You may create a hypothesis, perform the investigation, and determine you have no viable way of detecting this attack. The architecture teams need this feedback in order to improve the environment. Create feedback loops to improve architecture based on performance, operations, AND monitoring.

Reference:
https://www.sans.org/cyber-security-courses/red-team-exercises-adversary-emulation/

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. **EXERCISE: Deploy Section 1 Environment**
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Lab 1.1 | Deploy Section 1 Environment

Exercise Duration: 30 Minutes

Objectives

In this lab, we will set up our core infrastructure for use in the rest of the class.

- Create the Inspector Workstation, which is our virtual machine for all labs
- Create SSH Keys for EC2 access
- Set up AWS services
- Deploy the base infrastructure using AWS CDK

Our first lab is deploying part of the environment for the week, and generally kicking the tires to ensure that everything is setup properly in your AWS account.

There are two main Infrastructure as Code stacks in use today. AWS has CloudFormation tools to make that easier to use, such as CDK. And HashiCorp has the multi-cloud support service, Terraform. You will use both in this class, giving you a taste of each. This section will be built using AWS's Cloud Development Kit (CDK). More details of this service are in the lab.

First, you will log into your AWS Account and connect to the Inspector Workstation. This virtual machine is the student workstation that will be used throughout the course.

Next, you will create SSH keys for logging into your EC2 instances.

Next, we turn on some AWS services we will be using throughout the class.

Finally, we will kick off building out our lab infrastructure.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. **MITRE ATT&CK and Definitions**
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Threat Intel to Build Hypothesis

A supposition or proposed explanation made on the basis of limited evidence as a starting point for further investigation.

-Oxford Dictionary

A hypothesis is the real starting point in the scientific method. This is where you come up with an explanation, or supposed truth, that needs to be investigated.

In our threat hunting process, we will create a hypothesis that states, "What threat techniques may be used against our environment, and do we have the information to detect it?" Our starting point can be based on our one of the top three initial access vectors. Or the starting point may be a statistical anomaly in our network traffic.

Whatever the starting point might be, we need intelligence to tell us about attacker techniques, the business vertical or infrastructure that is being targeted, and the artifacts that we need to collect to detect it. That is where Cyber Threat Intelligence will come into play.

Cyber Threat Intelligence (1)

Cyber Threat Intelligence (CTI) is information that helps an organization understand threats, detect vulnerabilities, and help prioritize.

This includes:

- The nature of the threat. How is the attack used, by whom, what business verticals are at risk?
- The vulnerability itself. What does it look like, what are the artifacts if an attack occurs, how can we identify them?
- Specific ways to detect or mitigate those threats

Cyber Threat Intelligence is information that your org can use to start building your threat hypothesis. What vulnerabilities are there in my environment? What kinds of threats are using those vulnerabilities? How easy is it to perpetuate those attacks?

Sometimes, we think of CTI simply as a stream of known bad IP addresses or file hashes. This is good information that your SOC or security team can use for blocking or monitoring. For threat monitoring and detection, more details about the nature of the attacks are necessary. What are the most prevalent web attacks? How do attackers take advantage of S3 misconfigurations? If an attacker has exploited our management plane, how could I tell?

CTI could be paid-for services, or maybe some specific websites that track AWS vulnerabilities. However, with the nature of the cloud evolving quickly, it usually requires the threat monitoring and detection team to collect information from a number of different sources, all focusing on different kinds of attacks.

Let's look at a few CTI resources that we will leverage in this class.

Cyber Threat Intelligence (2)

Threat intel feeds tend to focus on attacker IPs, domain names, or malware hashes. We assume you are using one of those.

Cyber Threat Intelligence can look beyond atomic indicators.

- MITRE ATT&CK Matrix
- OWASP Top 10
- CIS Critical Security Controls

Threat intel is usually thought of as a feed of data that can be automatically applied to search, filter, or block because they are known to be bad actors. However, how can we learn about real cyber threat intelligence? How can we learn about what attackers are doing, how they are doing it, and how we can search for it?

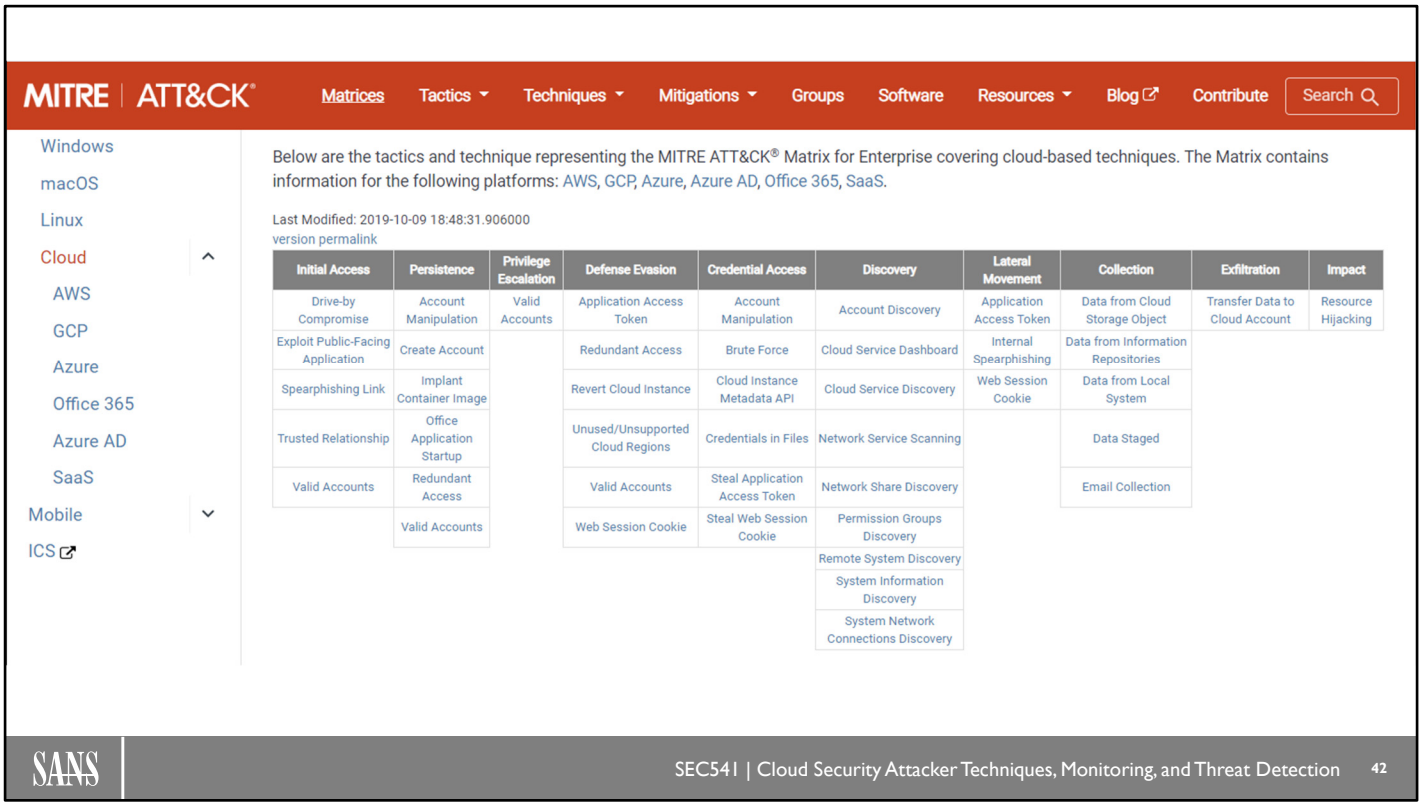
We will look at three pieces of Cyber Threat Intelligence data that are free and are in use throughout the community, but there are many others. Most security product companies integrate threat intel data with their product. Think about the traditional malware detection tool. You run an agent on your computer, and it monitors the hash values of files that are running, dropped, or surveys on the disk. If it detects a file that matches a known hash signature, it's detected as "bad" and can be removed.

This goes for network analysis tools as well. Look at the Snort[1] software that monitors network traffic and detects potential threats based on a set of rules. The security team can decide which Snort rules to implement, and even write their own, but they are largely based on atomic indicators (https://www.snort.org/rules_explanation).

Throughout this section we will be discussing threats from the following three threat intel sources:

- MITRE ATT&CK[2] Matrix
- OWASP Top 10[3]
- CIS Critical Security Controls[4]

[1] https://www.snort.org
[2] https://attack.mitre.org
[3] https://owasp.org/www-project-top-ten
[4] https://www.cisecurity.org/controls



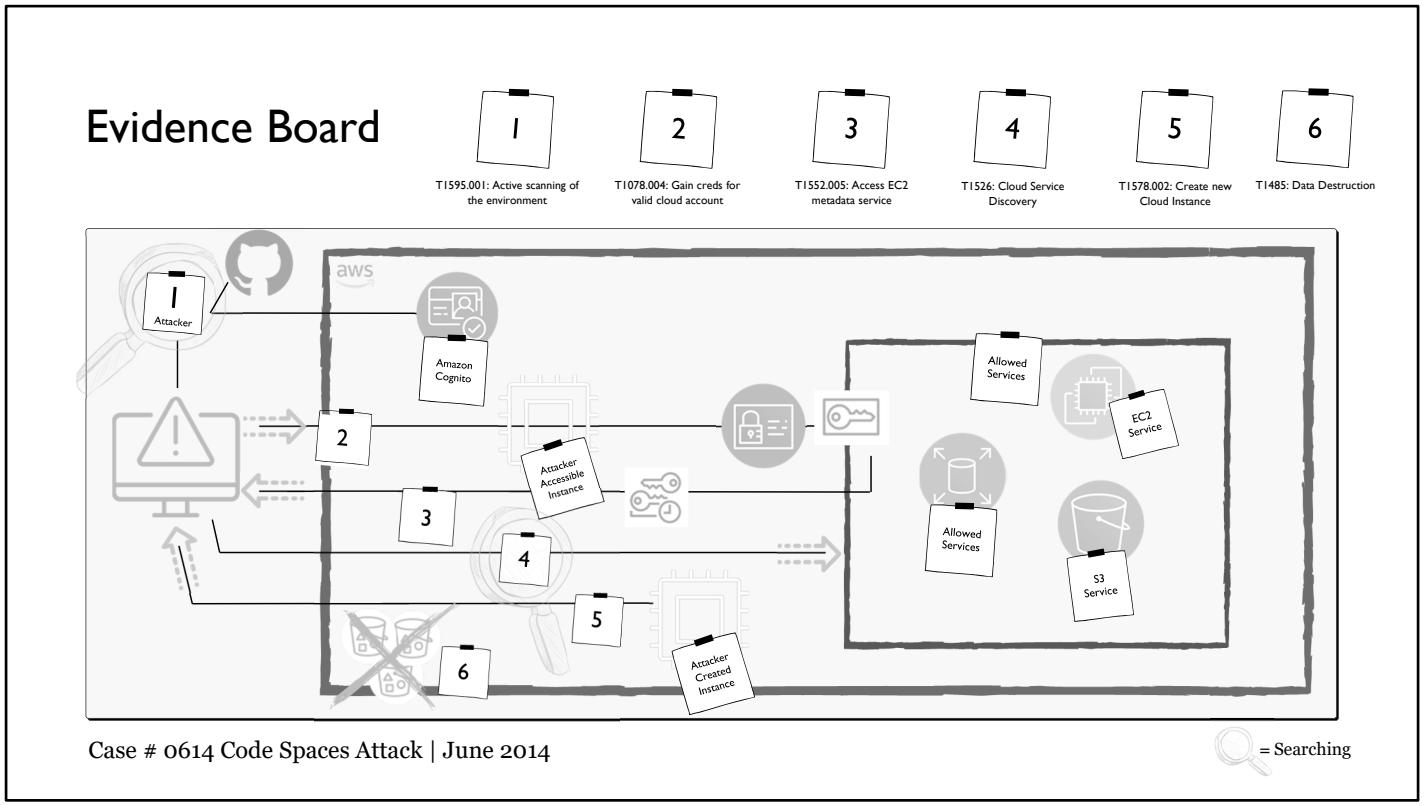
MITRE is a federally funded research company that has a research and development center that focuses on cybersecurity. One of their many contributions to the public is the MITRE ATT&CK Framework. ATT&CK, which stands for Adversary Tactics, Techniques, and Common Knowledge, attempts to bring order to the chaos of cataloging and describing how attackers operate in infrastructure. MITRE has multiple frameworks that focus on different environments, including one specifically for cloud infrastructure[1].

Across the top of the slide are the different tactics: Initial Access, Persistence, Privilege Escalation, Defense Evasion, Credential Access, Discovery, Lateral Movement, Collection, Exfiltration, and Impact. Think of these as the "goals" of the attacker. An attacker wants to gain "persistence" in your organization. Going down that column are the techniques that have been reported that an attacker will use. If you click one, such as "Create Account", you will get details of what that technique looks like, how to detect it, and how to mitigate it. Generally, the information is helpful. There is also a list of reference materials to read to determine more details. Each of the techniques have a unique identifier, like T1136.

What's so great about ATT&CK is how other companies are leveraging ATT&CK to better communicate in reporting. If a company says that a particular attacker is using technique T1136, you are able to reference back to the MITRE ATT&CK framework to read more about it.

The framework is written from the perspective of an attacker and their actions. If you want to contribute, you are able to submit new techniques, or suggested changes, right to MITRE.

[1] https://attack.mitre.org/matrices/enterprise/cloud



In SEC541, we will be referencing MITRE ATT&CK throughout the course, and specifically highlighting the attack techniques during the initial attack walk through.

Across the top of the slide is each step, with the ATT&CK technique name being referenced. For attack number one, the attack technique is T1595, and the sub-technique is 001. Active scanning of the network can be seen in technique T1595[1].

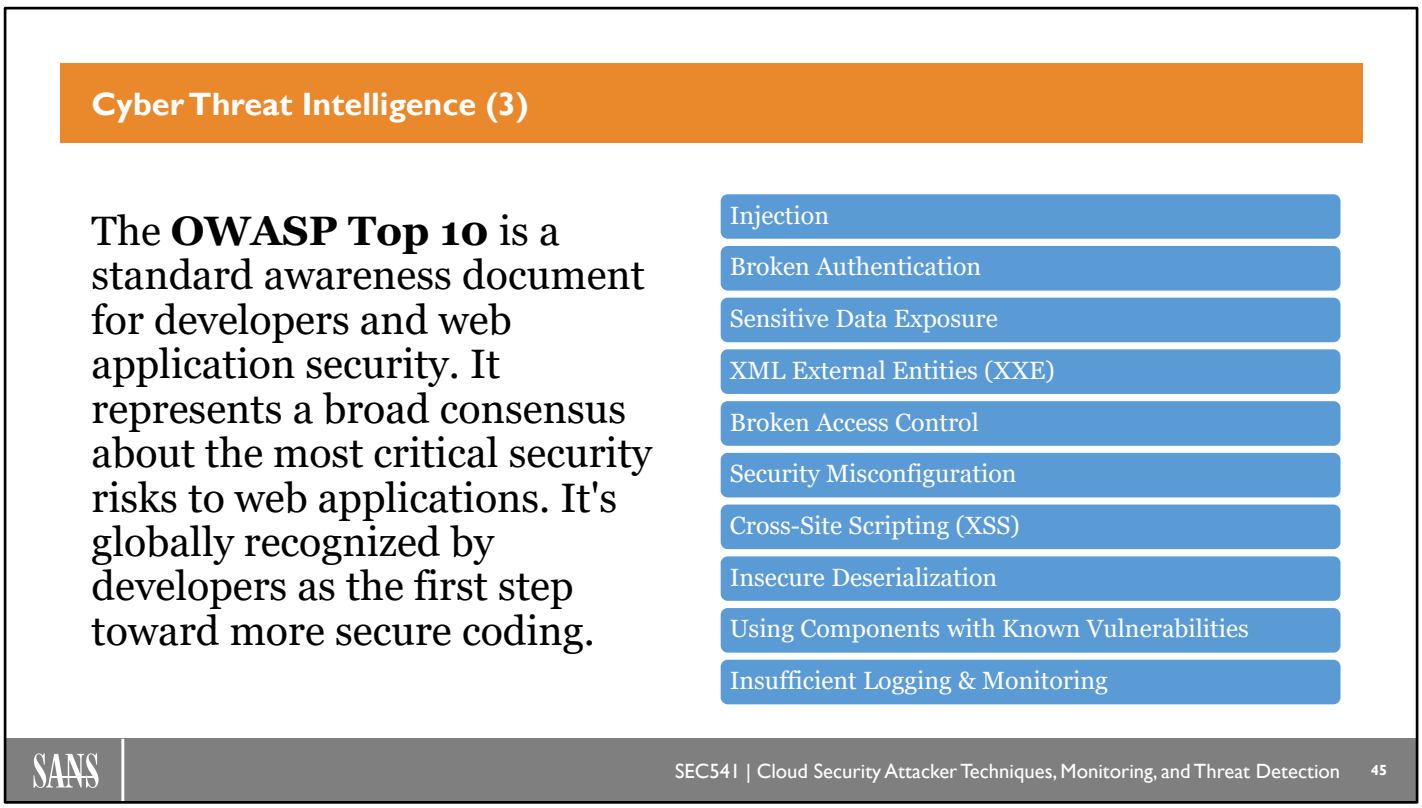
You will notice that attackers reuse similar attack techniques. Although there are any number of combination of attacks that could be used, most attacks have at least these three common steps:

1. Attacker gains access to an environment, usually either through stolen credentials, default access through some management system, or through exploitation of vulnerable applications. In this class, you will do all three against your own environment. Sure, there are other, more complex and sneakier attacks. MITRE has a list of about twenty different techniques in its "Initial Access" tactics, but those are the most common.
2. Once gaining access, attackers will leverage the stolen credentials or misconfigured access to look around and see what they can do through Discovery[2]. For cloud environments, that usually means using the cloud credentials taken, and scanning the environment, usually first through the cloud service provider's API, but maybe also network and system scanning. In a cloud architecture, if it follows modern design patterns, the execution system the attacker initially gains access to may not be all that interesting. Data is probably in S3 buckets, the database service, or maybe a cloud managed data storage system.
3. Once they have looked around and found what they want, then they have to do something, such as exfil data[3] or try to gain further access through Persistence[4], Privilege Escalation[5], or Defense Evasion[6].

[1] https://attack.mitre.org/techniques/T1595/001
[2] https://attack.mitre.org/tactics/TA0007
[3] https://attack.mitre.org/tactics/TA0010
[4] https://attack.mitre.org/tactics/TA0003
[5] https://attack.mitre.org/tactics/TA0004
[6] https://attack.mitre.org/tactics/TA0005

There are infinite possibilities, but these are the most common attack vectors.

Determining which techniques to focus on may also be a byproduct of your environment. Are your virtual machines highly scalable, and thus do not live long? Do your virtual machines have permissive permission policies or are they built with least privilege? Are containers long running or do you use Function as a Service? These architecture and operational decisions drastically change the expected behaviors, and thus the detection of abnormal behavior. Most organizations have a little of both.



The Open Web Application Security Project (OWASP) is a nonprofit foundation set up to improve the security of software. Every few years, OWASP publishes the "Top 10 Web Application Security Risks" based on observed attacks in the wild[1].

This top 10 list details the attack basics, how it works, and how to mitigate it. The importance of web technology only grows as organizations move into the cloud, and more and more compute systems are using web technologies that could be vulnerable to these OWASP Top 10 techniques.

As stated in the OWASP Top 10 page, the top attacks are described below:

1. **Injection:** Injection flaws, such as SQL, NoSQL, OS, and LDAP injection, occur when untrusted data is sent to an interpreter as part of a command or query. The attacker's hostile data can trick the interpreter into executing unintended commands or accessing data without proper authorization.
2. **Broken Authentication**: Application functions related to authentication and session management are often implemented incorrectly, allowing attackers to compromise passwords, keys, or session tokens, or to exploit other implementation flaws to assume other users' identities temporarily or permanently.
3. **Sensitive Data Exposure**: Many web applications and APIs do not properly protect sensitive data, such as financial, healthcare, and PII. Attackers may steal or modify such weakly protected data to conduct credit card fraud, identity theft, or other crimes. Sensitive data may be compromised without extra protection, such as encryption at rest or in transit, and requires special precautions when exchanged with the browser.
4. **XML External Entities (XXE):** Many older or poorly configured XML processors evaluate external entity references within XML documents. External entities can be used to disclose internal files using the file URI handler, internal file shares, internal port scanning, remote code execution, and denial of service attacks.

[1] https://owasp.org/www-project-top-ten/

5. **Broken Access Control:** Restrictions on what authenticated users are allowed to do are often not properly enforced. Attackers can exploit these flaws to access unauthorized functionality and/or data, such as access other users' accounts, view sensitive files, modify other users' data, change access rights, etc.
6. **Security Misconfiguration**: Security misconfiguration is the most commonly seen issue. This is commonly a result of insecure default configurations, incomplete or ad hoc configurations, open cloud storage, misconfigured HTTP headers, and verbose error messages containing sensitive information. Not only must all operating systems, frameworks, libraries, and applications be securely configured, but they must be patched/upgraded in a timely fashion.
7. **Cross-Site Scripting (XSS)**: XSS flaws occur whenever an application includes untrusted data in a new web page without proper validation or escaping or updates an existing web page with user-supplied data using a browser API that can create HTML or JavaScript. XSS allows attackers to execute scripts in the victim's browser which can hijack user sessions, deface websites, or redirect the user to malicious sites.
8. **Insecure Deserialization**: Insecure deserialization often leads to remote code execution. Even if deserialization flaws do not result in remote code execution, they can be used to perform attacks, including replay attacks, injection attacks, and privilege escalation attacks.
9. **Using Components with Known Vulnerabilities**: Components, such as libraries, frameworks, and other software modules, run with the same privileges as the application. If a vulnerable component is exploited, such an attack can facilitate serious data loss or server takeover. Applications and APIs using components with known vulnerabilities may undermine application defenses and enable various attacks and impacts.
10. **Insufficient Logging & Monitoring**: Insufficient logging and monitoring, coupled with missing or ineffective integration with incident response, allows attackers to further attack systems, maintain persistence, pivot to more systems, and tamper, extract, or destroy data. Most breach studies show time to detect a breach is over 200 days, typically detected by external parties rather than internal processes or monitoring.

Cyber Threat Intelligence (4)

CIS Critical Controls and Benchmarks v8

1. Inventory and Control of Enterprise Assets
2. Inventory and Control of Software
3. Data Protection
4. Secure Config of Enterprise and Software
5. Account Management
6. Access Control Management
7. Continuous Vulnerability Management
8. Audit Log Management
9. Email and Web Browser Protections
10. Malware Defense
11. Data Recovery
12. Network Infrastructure Management
13. Network Monitoring and Defense
14. Security Awareness and Skill Training
15. Service Provider Management
16. Applications Software Security
17. Incident Response Management
18. Penetration Testing

Okay, though they are not normally thought of a source of Cyber Threat Intelligence, the CIS critical controls and benchmarks are controls that you should implement to make yourself more secure.

But think about it! If your environment is a mess, without security controls and secure system configuration, then detection will be nearly impossible. We need a clean environment so bad behavior stands out.

The Center for Internet Security (CIS) is another non-profit organization that provides a rich set of guidance in many aspects of computing. Two cloud focused products are the CIS Critical Controls and the CIS Benchmarks.

The CIS Critical Controls v8 are a set of eighteen controls that organizations should have in place to better protect their infrastructure. For threat detection, these controls are useful when doing threat modeling. If an organization is not implementing a particular control, they have an idea of a vulnerability in their infrastructure they can go investigate. The "CIS Controls Cloud Companion Guide" demonstrates how to apply those controls in a cloud environment[1].
The "CIS Benchmarks" for AWS details specific ways to audit security controls, and specific controls and how they should be implemented[2].
CIS is a wealth of knowledge in how to secure your environment, and this can help prioritize where to focus monitoring efforts.

[1] https://www.cisecurity.org/white-papers/cis-controls-cloud-companion-guide
[2] https://www.cisecurity.org/benchmark/amazon_web_services

Cyber Threat Intelligence (5)

Some attack techniques are the same in the cloud as on-prem:

- SSH brute force to gain access to a web server
- Perform website traversal attack to run code on an EC2 based website

Some attacks might have similar goals, but look different in cloud:

- Performing a discovery of cloud services while operating on a hacked website
- Stealing another workload's credentials by stealing their STS token

There are a number of threats that we might see in a cloud environment.

SSH brute force is technique T1110, and involves continuous guessing of SSH credentials until an attacker has guessed the username/password correctly and is able to log in. Virtual machines in cloud environments are typically managed through SSH. It is never a best practice to use a username/password for SSH credentials, but it's still done[1].

OWASP path traversal is part of the OWASP Top 10. An attacker attempts to access files and directories that are stored outside the web root folder by adding "../" sequences to the web path[2].

Cloud service discovery is number T1526 in the MITRE ATT&CK framework. This technique leverages the cloud environment management platform to discover what services, capabilities, or resources have been applied. For instance, with the proper credentials, an attacker could ask AWS to return a name of all S3 buckets. That bucket name could be targeted to retrieve sensitive information[3].

In addition to SSH keys, AWS uses secure keys and tokens to allow people and computer systems to access the AWS infrastructure and other systems. Sometimes, these keys get checked into source code, left on developer environments, or are freely accessible on a compromised systems. There are a couple of techniques that describe how to obtain creds in tactic TA0006[4].

[1] https://attack.mitre.org/techniques/T1110/
[2] https://owasp.org/www-community/attacks/Path_Traversal
[3] https://attack.mitre.org/techniques/T1526/
[4] https://attack.mitre.org/tactics/TA0006/

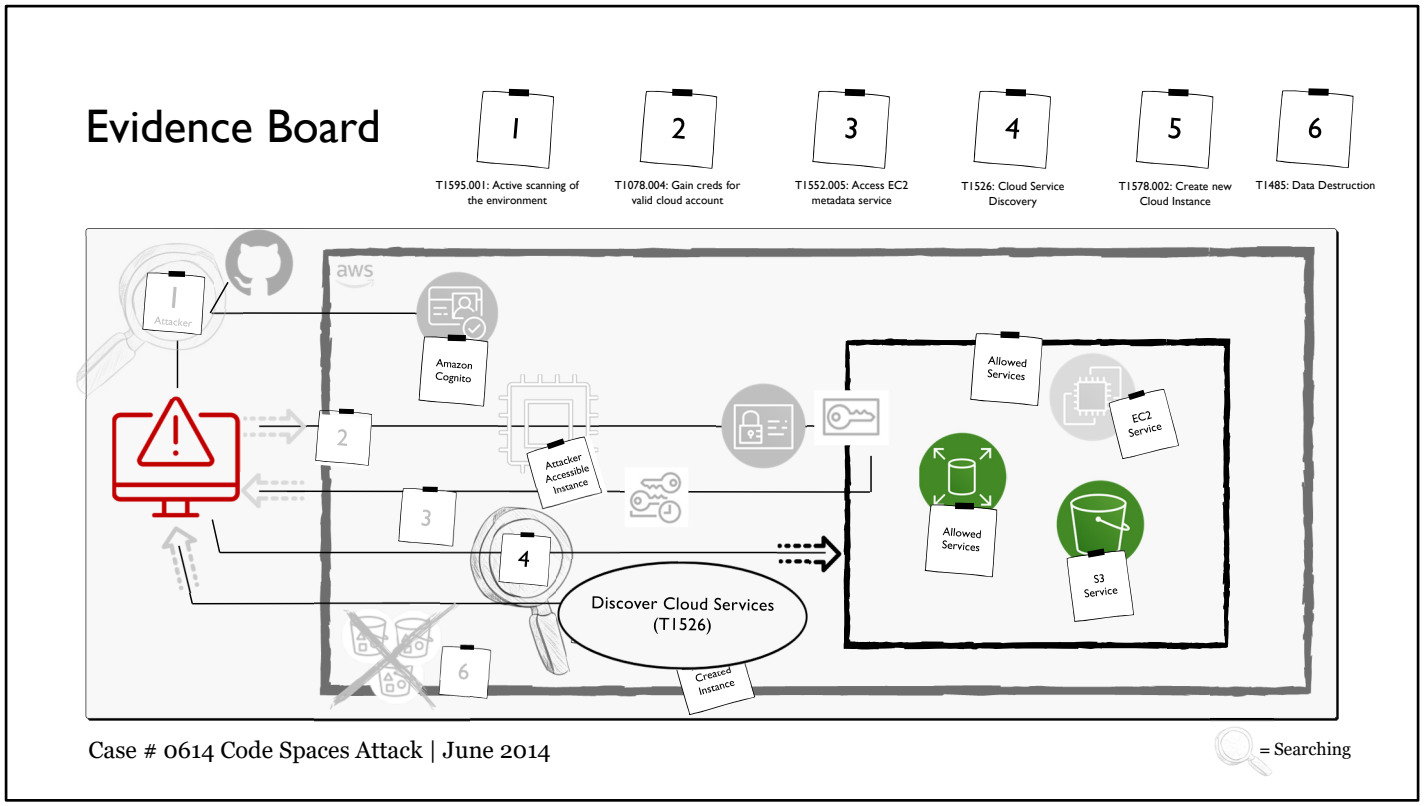
Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. **API Logging**
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.



One of the first things an attacker will do when they gain access to an EC2, or other compute services in an environment, is to try and figure out what services are available to them. MITRE ATT&CK T1526 is the Cloud Service Discovery[1].

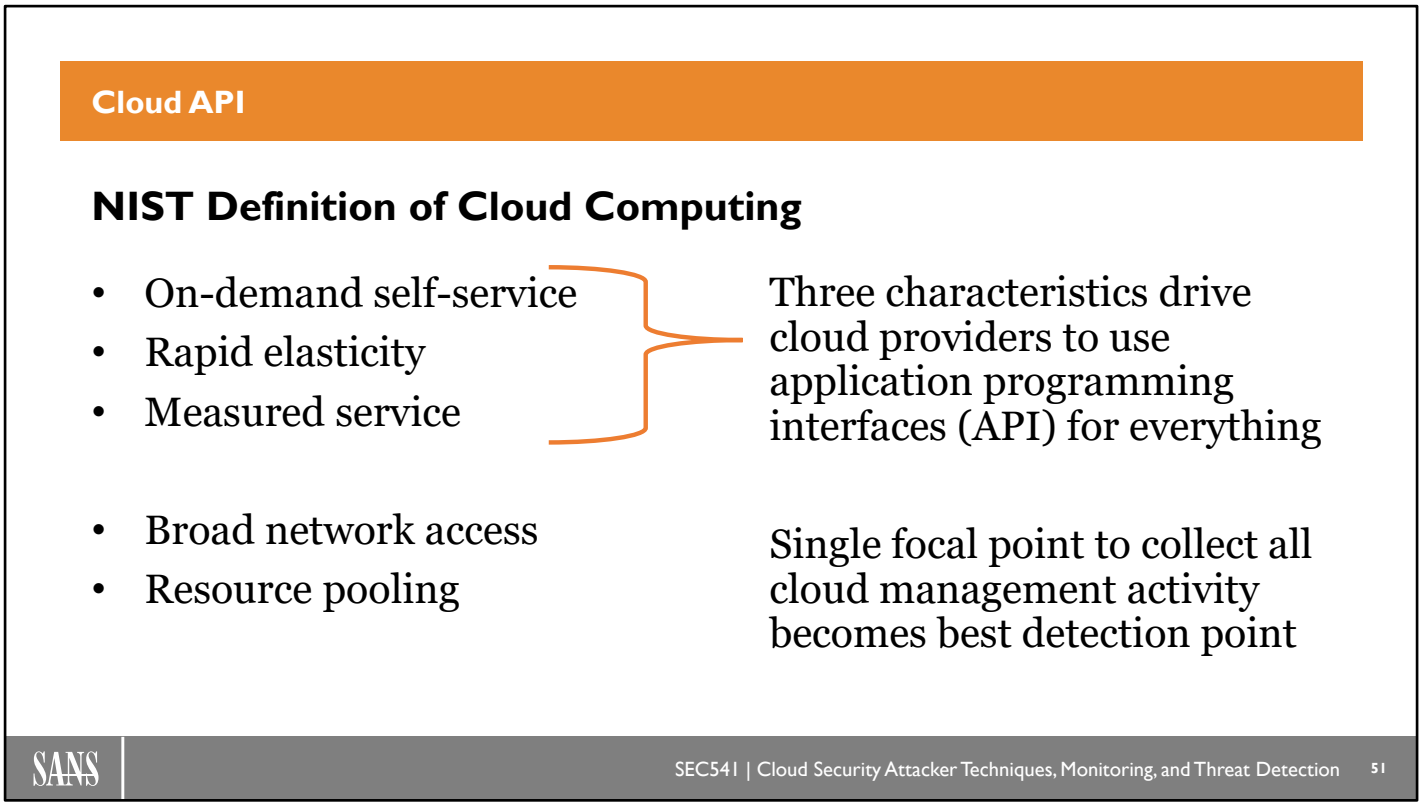
An enumeration of the cloud services running, or available, is visible from the access point. *How* they perform the discovery attack may be different from PaaS to IaaS to SaaS, but the goal is the same: "What can I do from here?".

As soon as services are discovered, it's time to dive deeper into the Cloud Infrastructure Discovery[2].

In an IaaS environment, you may have access to users, virtual machines, databases, or containers. In the case of Code Spaces, the attacker was able to gain some level of console access, which could have allowed them to click through the WebGUI or run some commands.

All this discovery has to go through the Cloud Management API. We will try and leverage that to our detection advantage.

[1] https://attack.mitre.org/techniques/T1526
[2] https://attack.mitre.org/techniques/T1580



The "NIST Cloud Computing Standards Roadmap" was commissioned by the US government to support the secure and effective adoption of cloud computing in mission critical environments. The USG Cloud Computing Technology Roadmap is over 100 pages of standards, descriptions, guidance and use cases about adoption of cloud services. Much of US government's understanding and terminology, security approach, and cloud purchasing power starts with this roadmap[1].

The roadmap is almost too much information, but it does break down five essential characteristics that describes a “cloud computing” model, three service models, and four deployment models.

The five essential characteristics distinguish a “cloud model” from a “shared hosting model” or one of the predecessors to cloud computing.

- **On-demand self-service:** A consumer can provision the computing resources automatically, without requiring another human interaction for each service provider. Once the cloud company has your credit card, you have the ability to spin up whatever expensive service that cloud company offers, at any time you would like. This level of self-service is usually through a management console or through some type of direct computer interaction through a standard application programming interface (API). Standardization is important to allow customers to automate changes to their environment. This automation is especially necessary in Infrastructure as a Service offerings, such as AWS and Azure, where customers can build complex and rapidly changing infrastructure components, especially to support the next essential characteristic.

[1] https://www.nist.gov/system/files/documents/itl/cloud/NIST_SP-500-291_Version-2_2013_June18_FINAL.pdf

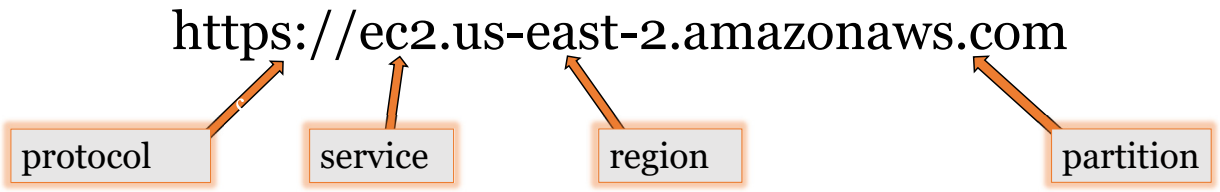
- **Rapid elasticity:** Resources are provisioned and released as needed, when needed, automatically and usually in response to some trigger. This is one of the main selling points of a cloud service. In a traditional data center, deployment at your organization could require months to decide, acquire, and stand-up virtual machines to meet some intended growth. It would be much better to just rent that compute system when it is needed. Better yet, you likely have times of the day that you may need more compute than others. For instance, an online tax preparation company is really busy from January until May, but its business case really scales down afterward.
- **Measured service:** Cloud services monitor and measure everything that is happening—that is how they get paid. Resource usage can be monitored, controlled, reported, and charged to your credit card in a fraction of a second. From a security standpoint, those measurements are available to us to monitor what is happening and who caused it. The API logs are a prime example. The cloud provider is doing authentication and authorization of you and any resources that are performing an activity in the environment. For example, a virtual machine has to pull a configuration file from an S3 bucket. The virtual machine is granted permissions through the cloud provider's identity access management service. When the EC2 wants to pull the S3 data, a set of API calls are made to get the right credentials, tell AWS to get the bucket content, and then retrieve that bucket content. Those interactions can be tracked through the API services.
- **Broad network access:** The service is available over the network and provided through a host of platforms, such as computer, phone, or API.
- **Resource pooling:** The cloud provider resources leverage some type of multi-tenant model, usually through a form of virtualization. Your environment is not just yours but is shared with everyone around you at some level. Modern cloud companies have taken great pains to ensure virtual segmentation in order to attract the more lucrative clients (banks with cash).

Elastic services available on-demand and measured to a granular level means we have logs that we will use to detect and track attackers as they move through our network.

AWS API (1)

CloudTrail tracks Application Programming Interface (API) calls to AWS environment and gives us tools to interact with the cloud.

Each AWS service has a set of REST API calls to that service. Most services are regional, but some services are global.



All commands to the AWS environment are through their Application Programming Interface (API). This means every click on the management console generates an API call and every CLI command we run hits the same API commands. We do not need to go so far into the weeds of the API protocol to write our own exchanges, however understanding the API and the AWS service will help us analyze the logs they generate. Threat detection by going beyond the managed security tools works best when you really understand HOW the service is working underneath. Running surface level tools can help, but we need to customize, analyze, and pull out the anomalies that are so far undetected. Let's look at the API, the heart of how stuff gets done in the cloud.

Every AWS service, like EC2, S3, etc., has a separate endpoint that listens for web-based REST calls. REST, which stands for Representational State Transfer, is a software architecture that provides a set of overarching constraints for interacting with a web service. Payloads could be HTML, XML, or JSON.

Most AWS service endpoints are tied to a particular region and interacting with those endpoints will only affect those resources in that region. For instance, ec2.us-east-2.amazonaws.com will only interact with EC2 instances in US-East-2 (Ohio). Global endpoints for global services, like IAM or S3, are not tied to a particular region.

protocol://service-code.region-code.amazonaws.com
https://ec2.us-east-2.amazonaws.com

https protocol is an application-level protocol (see OSI model) that was designed primarily for websites and web browsers to exchange information. Much of the AWS and Azure communications is built on this protocol.

EC2 service is how AWS separates its resources and actions. EC2 is the service from managing customer networking services and virtual machines. The S3 service manages S3 buckets. There are over 200 services for AWS with new ones updated all the time.

us-east-2 region denotes the data center where this service and resources reside. Most resources you work with, such as an EC2 or running container, will live only in a single region. Separation of resources by regions allows customers to decide where their data and resources will reside, and also build redundancy in applications if a particular region were to become inoperative. For threat detection, we typically think about regions as we try to pull logs across our infrastructure, across these regions, into a single analytic environment. AWS and Azure are updating their data collection services to make this more robust, but we may have to manage it ourselves.

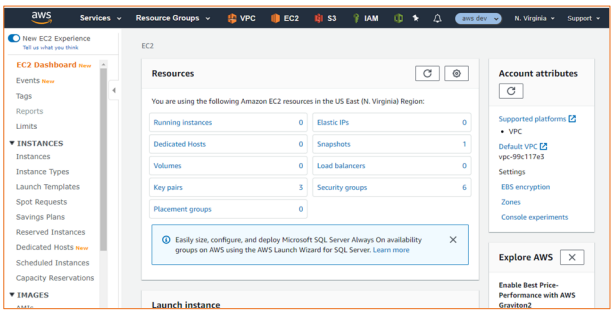
amazonaws.com partition is always amazonaws.com for commercial environments. AWS and Azure has some special partitions for China or government only resources. The service code may work differently in these partitions. In this class, we will just look at the commercial regions

References:
API Requests - https://docs.aws.amazon.com/AWSEC2/latest/APIReference/making-api-requests.html
AWS Regions - https://aws.amazon.com/about-aws/global-infrastructure/regions_az/
Azure Regions - https://docs.microsoft.com/en-us/azure/availability-zones/az-overview

AWS API (2)

The AWS Management Console is usually the first, and easiest, way to interact with AWS services.

- Provides wizards that automate or speed up the process of creating, editing, and deleting AWS services
- The AWS Management Console provides good visualizations of AWS services
- It's sometimes easier to see how individual AWS resources are related to each other
- It's not great for automation, scaling, or advanced filtering



The AWS Management Console[1] provides a set of user interfaces, wizards, and tools to help us make sense of the AWS resources. Remember, the management console is hitting the REST APIs we discussed in the previous page. Any create, read, update, delete and list of AWS resources is done by the web application making a specific call to the service API in a region.

Management console makes things easier
The AWS management console has wizards to help customers build a set of AWS resources that might be tied together. For instance, when you create a virtual machine in AWS, it takes you through a series of screens to ask you what instance type to use, what size EBS volume, creating a security group and even a new keypair. The keypair and security group are different resources than the virtual machine. When you select "submit" on that final screen, the wizard is creating multiple resources on your behalf. The wizards are helpful, especially if you are new to a cloud resource and are not familiar with its details.

The management console is also helpful when visualizing data about multiple, related resources. Back to our virtual machine example, when you select a particular virtual machine and look at its details, you will see the virtual machine, the subnet it's in, the key pair name, the EBS volume information, and more. Each of those elements are separate resources that the management console application queried for you and rendered as if they were a single item.

The problem with the management console
From an operational standpoint, your organization will need to quickly move from clicking a management console to automating through the API, via the SDKs or CLI. Why have a human create the same VM every time a new developer joins the team when you can automate it with a script.

The management console also hides the true nature of the resources, the APIs, and how they actually work together. This information is necessary when hunting through logs to detect a nefarious activity. It is the author's recommendation that if you are responsible for security detection or threat hunting, then get familiar with the CLI and SDKs.

Log analysis in the Console
AWS and Azure have built, and are improving, analytic services in their cloud environment. These purpose-built tools will allow you to run queries across collected logs to find nefarious activities. They are usually easier to setup than a fully featured query tool like Splunk, but likely are not as full featured. But don't worry, we discuss these tools throughout the class.

[1] https://console.aws.amazon.com/

AWS API (3)

The AWS CLI is a unified command line interface to the individual REST APIs of the service nodes.

- AWS CLI v2 is based on Python 3.7.x and 3.8.x
- AWS commands follow a similar pattern

```
cyber$ aws ec2 create-key-pair --key-name "test"
```

- The CLI is good for creating auditing scripts, limited automation, and quick commands
- Each command through the CLI creates a set of REST API calls to the AWS service endpoints

The AWS CLI is a Python tool that makes it easier to interact with the AWS API without having to deal with REST interfaces. Each command roughly translates to an AWS call. The results are translated and displayed in JSON, text, YAML or in a table format. The AWS CLI version 2 is now considered the standard version, but version 1 of the CLI is still available and is being updated at the moment.

The AWS CLI commands follow a similar pattern.

```
$ aws ec2 create-key-pair –key-name “test”
```

- “`aws`” is just the program we are running. Version 1 of the AWS CLI was a Python package you downloaded and installed. Version 2 is a standalone executable that has the Python runtime embedded. This makes it so much easier to install and run, without having to deal with Python versioning. The AWS CLI will validate the service node.
- “`ec2`” is the service endpoint. When you configure the AWS CLI, you specify the region, such as “`us-east-1`”. Remember from two slides ago, in https://ec2.us-east-1.amazonaws.com, the us-east-1 is the region location. The AWS CLI doesn’t validate the region. If you were to append “`--region fakeregion`” to the command, it would try and connect to ec2.fackregion.amazonaws.com.
- “`create-key-pair`” is the command we are going to run.
- “`--key-name 'test'`” represents a set of key value pairs to provide parameters to the API call.

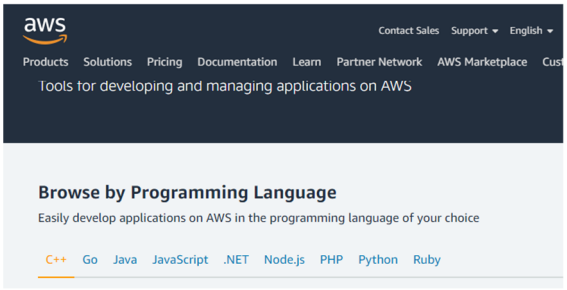
The AWS CLI validates that the endpoint, command, and parameters are valid. If it is, it will use the Botocore SDK to convert the command to an HTTP REST call and interact with the service.

Azure has a CLI and PowerShell tools for command line interactions. These can be found at https://azure.microsoft.com/en-us/downloads/.

AWS API (4)

AWS provides a number of software development kits (SDKs) that can be used in a program to automate and manage.

- The SDKs can drastically improve automation, scaling, and custom tooling
- SDKs are used in Lambda and other cloud-based computations
- The AWS SDK for Python is Boto3, and is built on a low-level interface called Botocore
- AWS provides a number of CLIs specific to certain services such as ECS, AMI, and serverless deployments



We don't interact directly with the API when we are creating, listing, describing, or deleting resources. Rather, we will use one of the software development kits (SDKs) created and maintained by the cloud providers to support development.

For AWS, there are SDKs for C++, Go, Java, JavaScript, NET, NodeJS, PHP, Python and Ruby.

The Python SDK is probably the most used. Botocore is a low-level SDK that interacts directly with the AWS service nodes. Boto3 is another Python library that is built on Botocore and provides a higher-level abstraction. Boto3 makes it very simple to interact with AWS services without having to worry about the REST connections. Boto3 is typically what Python developers use.

The authors' opinion is that Boto3 is by far the simplest way of interacting with AWS services when programming. But a quick look at the AWS lab repos on GitHub shows AWS teams writing code in shell, JavaScript, and even Rust.

Azure SDK's come in .NET, Java, JavaScript/TypeScript, Python, Go, C++, C, Android and iOS.

Reference:
https://azure.microsoft.com/en-us/downloads/

AWS API (5)

$ aws ec2 describe-instances

```
POST / HTTP/1.1
Host: ec2.us-east-1.amazonaws.com
Accept-Encoding: gzip, deflate
Content-Type: application/x-www-form-urlencoded; charset=utf-8
User-Agent: aws-cli/2.0.6 Python/3.7.3 Linux/4.4.0-18362-Microsoft
botocore/2.0.0dev10
X-Amz-Date: 20200609T214941Z
Authorization: AWS4-HMAC-SHA256
Credential=AKIAW7BE7ZEKMHXWPBNA/20200609/us-east-1/ec2/aws4_request,
SignedHeaders=content-type;host;x-amz-date,
Signature=6d2a93a0e9b3ba71488088b7aa35de3b04b666e280d7a01db6583bf084951ff9
Content-Length: 43
Connection: close

Action=DescribeInstances&Version=2016-11-15
```


Let's take a look at the REST API traffic when we use the AWS CLI to communicate with the EC2 service's api in us-east-1, the Northern Virginia region. Running *aws ec2 describe-instances* will return all the instances in the region. If we do this in our command line tool right now, it will likely come back in detailed JSON data.

The *ec2* directive in the command line tells the CLI tool to connect to the ec2 service in the current region, which is us-east-1.

It's a REST call, with an "Action" of "`DescribeInstances`". If you wanted to see details of this command, it is available at https://docs.aws.amazon.com/cli/latest/reference/ec2/describe-instances.html.

Or look at the Boto3 code at https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/ec2.html#EC2.Client.describe_instances.

"Version" is the latest version number for this API.

For this particular call, it's a POST call. Some queries to the API might be a simple "`GET`".

Sometimes, that AWS CLI command "`describe-instances`" is similar to the REST API action value, but not always. Knowing this value of "DescribeInstances" is important, because when AWS tracks and logs commands to the API, that is the value. Not *describe-instances*. As we will see in a later section, it can take some sleuthing to dive into the logs that track these API calls.

AWS API (6)

$ aws ec2 describe-instances

```
HTTP/1.1 200 OK
x-amzn-requestid: 826438a7-9cd0-4cb0-a978-8370ffa2ff26
Content-Type: text/xml;charset=UTF-8
vary: accept-encoding
Date: Tue, 09 Jun 2020 21:49:40 GMT
Server: AmazonEC2
Connection: close
Content-Length: 64318

<?xml version="1.0" encoding="UTF-8"?>
<DescribeInstancesResponse xmlns="http://ec2.amazonaws.com/doc/2016-11-15/">
    <requestId>826438a7-9cd0-4cb0-a978-8370ffa2ff26</requestId>
    <reservationSet>
        <item>
            <reservationId>r-0cd774cbade52759a</reservationId>
            <ownerId>478966434068</ownerId>
            <groupSet/>
            <instancesSet>
                <item>
                    <instanceId>i-0d1c993c873a7d78e</instanceId>
```


Here is the return from the server. It is XML data that will contain all the instances. The AWS CLI will take this returned data and transform it into JSON, YAML, a table, or text and print to the screen.

The SDKs are facilitating these REST API calls. Why are we talking about API calls? They are important because ALL activity through the AWS and Azure environment is happening through these API calls. Although builders and operators may not care about this level of detail, as you are building threat detections and trying to determine what is happening in the cloud environment, you will need to understand how the underlining communications are happening.

All AWS and Azure commands are going through an API, and most of those API calls are being logged and are available for us to analyze. Want to know who created a new instance and when? Next, we go look at the logs.

CloudTrail (1)

CloudTrail lets you retrieve most API calls:

- Some API calls are tracked, but not all: Trusted Advisor, Import/Export, DeepComposer, DeepLens, etc.
- CloudTrail stores API calls for ninety days
- CloudTrail can be configured to store API calls to S3 buckets
- The data is stored in JSON format for each extraction, translation, and manipulation by standard tooling
- The CloudTrail logs are also accessible by querying the AWS API, just like interacting with any AWS service

AWS is collecting the metadata about most API calls made to the AWS services. CloudTrail is the name of the AWS service that serves you those logs for analysis or storage.

Most API calls are tracked, but not all of them. Double check to make sure that the technique you are detecting will generate the appropriate logs in CloudTrail. There are a number of reasons why an API call is not tracked in CloudTrail. Non-public APIs only usable by the GUI may not track in CloudTrail. This includes DeepComposer and Trusted Advisor. Some services might be newer and just have not gotten into CloudTrail. This does not happen often anymore, but it does happen.

There have also been reports of security researchers getting around CloudTrail tracking some API calls. AWS usually identifies and remedies those bugs quickly. For instance, one researcher discovered that if the AWS service used the JSON 1.1 protocol, the API returned a unique error code, and the resource associated was set to a certain value, a number of AWS API actions would not be tracked in CloudTrail[1].

By default, logs are stored and retrievable through CloudTrail for ninety days. You do not have to turn this on or set it up. CloudTrail is always providing 90 days worth of logs, with limited querying tools.

Most organizations will want to save their CloudTrail logs for a longer time, or possibly pull together logs from multiple accounts across regions to a centralized location. CloudTrail can be setup to store the logs in an S3 bucket. This is called a “CloudTrail Trail”, which is a bit confusing.

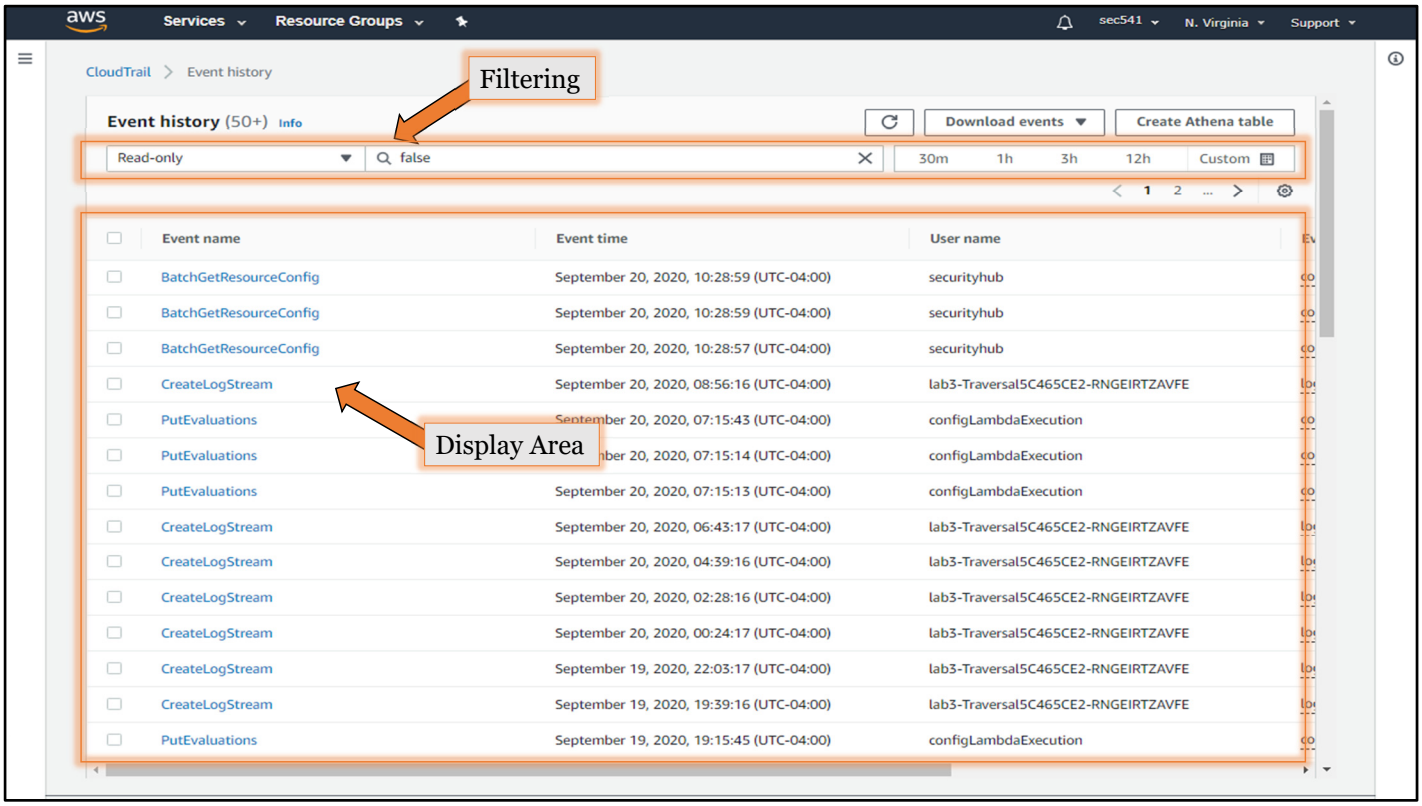
Each log, or the metadata from a unique API call, is stored as JSON data. This makes it easy to extract and evaluate with tools and programming languages. As we will see later in the class, this also helps when using AWS analytic services.

In our example, a single CloudTrail log for *DescribeInstances* will be stored in CloudTrail. That log will describe who made the call, when, what was requested, and if the response was successful. CloudTrail will not, however, capture the results of the *DescribeInstances*, the listing of EC2s in our region. Only the metadata surrounding the call that was made.

The CloudTrail logs available for ninety days are retrievable with an AWS call to the CloudTrail API. A security team can jump into any account and query for activity over the last 90 days without any setup and log capturing. Although not the most robust method, it is a quick way to start.

[1] https://frichetten.com/blog/aws-api-enum-vuln

Resources:
CloudTrail details - https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-aws-service-specific-topics.html
Services unsupported by CloudTrail - https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-unsupported-aws-services.html



The main dashboard page gives you easy access to view the CloudTrail events.

The filter boxes will let you filter on a single parameter. Next to the filter boxes you'll find a section that lets you filter during a timeframe.

The “Download events” section will let you quickly extract the events on this screen in either JSON or CSV format.

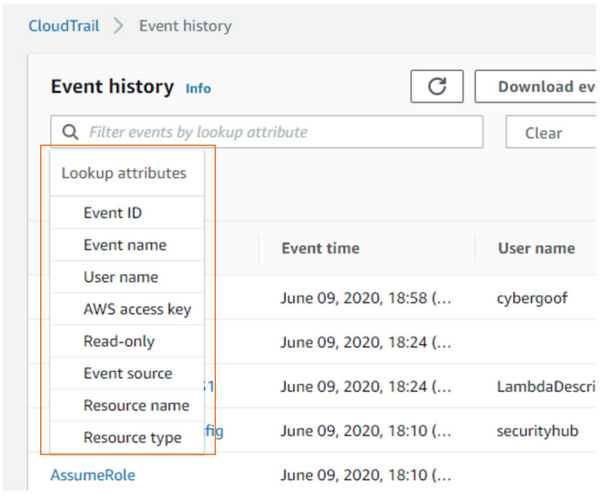
Athena is an AWS service that lets you query data in a SQL format. We will be diving deep into Athena in the next section of the course.

The gear icons allow you to alter how the data is displayed on this page by selecting the columns and number of logs per page, and even converting between local and UTC.

Course authors suggestion: The AWS will sometimes show UTC time and will sometimes show local time. However, the data itself is actually stored in UTC. It’s usually just best to make it all UTC and know how to convert to local time. Otherwise, it gets confusing. I like getting a couple of cheap clocks for the wall where the security or IT ops team sits—one for local time, and one for UTC.

CloudTrail (2)

On the "Event history" page, you can quickly filter based common field.



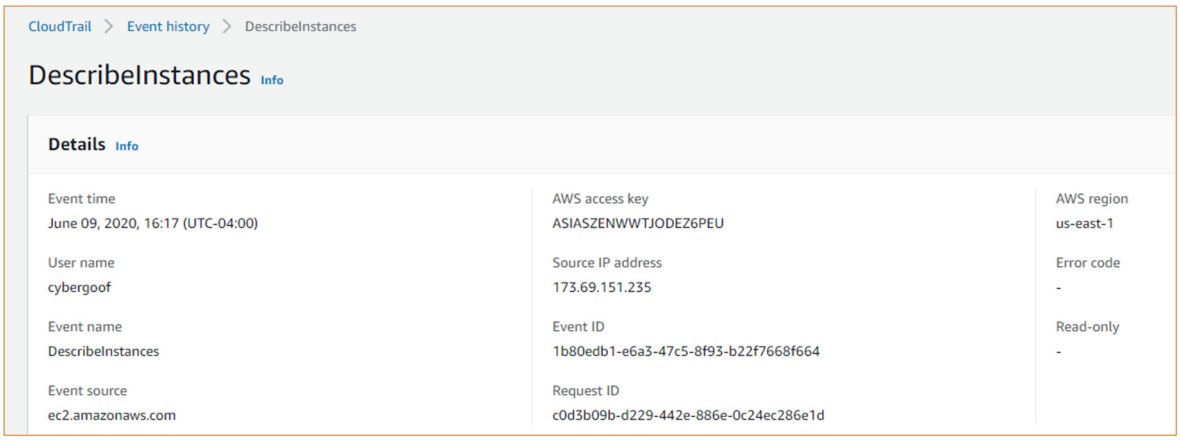
As we saw on the previous page, the dashboard provides some easy, rudimentary filtering capabilities. Unfortunately, through this UI, you can only filter by one dimension at a time. The AWS CLI command for *cloudtrail lookup-events* has a bit of an explanation. The management console is using this field to provide --*lookup-attributes*, which only allows for a list of 1 value, at this time.

- `EventID` if you already know the `eventID` you want to search for. `EventID` is something like `faa1dec7-5fc9-4318-b7c0-f210f348acac`.
- `Event name` is the action that occurred, such as "`ListBucket`", "`DescribeInstances`", and "`UpdateTrail`".
- `User name` is the user who created the action. This might be an IAM user in your account, a cross account role, or the AWS resource that performed the action.
- `AWS access key is the key that was` used to perform the action. This is great when you need to track down access key usage across your environments.
- `Read-only` is set to true or false. An event is read-only if it didn't make any changes to resources. Describe EC2s as `read-only` when doing an investigation. There will be lots of read-only events that you won't care about. This lets you look for creation, update, or deletion events.
- `Event source` is the service you are filtering on: EC2, WAF, etc.
- `Resource name` will be the name of the resource if the action created, edited, or deleted a resource. If you create an IAM role, then the `resource name` will be the role that was created.
- `Resource type` is the object type in the event. When creating an IAM Role, the `resource type` is "AWS::IAM::Role".

Reference:
https://docs.aws.amazon.com/cli/latest/reference/cloudtrail/lookup-events.html#options

CloudTrail (3)

Earlier, we ran "DescribeInstances" and saw the REST call. This is the CloudTrail event.



In the CloudTrail dashboard, click an event to see its details. In our "`DescribeInstances`" example, we can see the details for that event. Common parameters are shown in this display window.

`Event time` is given in local time. Frustratingly, you are not able to set UTC time on this page.

`User name` is the AWS username of the IAM user or role that performed the action. This `username` is part of a JSON object called *`userIdentity`* as described on the reference page[1].

`Event name` is a key value to pivot off when doing threat analysis and investigations. What are all the possible `event names`? Or better yet, what are each of the event names for each service? Good question—I don't have a great answer. Web searches have shown write-ups or blog posts of all the event names, but those are only snapshots in time. The "event" is the action that was taken to do some command. The IAM policy creation wizard can make it easy to go figure out what "write" commands for EC2 are possible. The CloudTrail filter page (previous screen) can also let you use the drop-down to do a filter. However, I have found that the filter page performs poorly at times. It can be frustrating.

`Event source` is the AWS service that the request was made to.

`AWS access key` is the ID of the access key used to sign the request. This could be a temporary security credential.

`The source IP address is the IP address` of the executor of the command.

`The request ID is t`he value that identifies the request. This ID is generated by the service being called

`The event ID is` GUID generated by CloudTrail to identify each event.

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

`Read-only` identifies if this operation is a read-only operation. This is a good value to filter on, since there will be many times you don't want to look at all the reads, and there will be a lot.

Reference:
https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-record-contents.html?icmpid=docs_console_unmapped

CloudTrail (4)

CloudTrail records a single JSON object for every API call.

The results of the command are not recorded.

On the details page, you can see the CloudTrail JSON data in full. CloudTrail captures a significant amount of information about the API call itself, but not the actual data that might have been returned. For instance, when we queried the AWS environment with `DescribeInstances`, we saw the resulting display of all the instances in our environment. That instance information is not available in the CloudTrail—just that a user queried for instance information (it's the metadata about the call).

The CloudTrail record has a set of standard fields such as `eventTime`, `eventSource`, and `awsRegion`. Others are optional depending on the API call[1]. When working with CloudTrail logs, either detecting special events or developing code that triggers off CloudTrail, the best way to know the data that is provided in a particular log is to just generate the log.

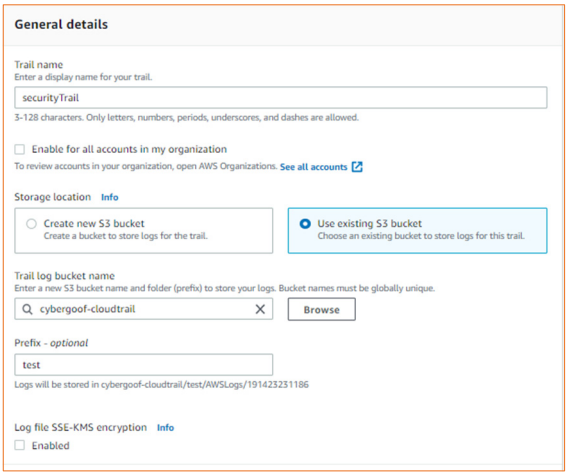
There are non-API calls that are collected by CloudTrail[2]. `AWS Service Events` and `AWS console sign-ins`[3].

[1] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-record-contents.html
[2] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/non-api-aws-service-events.html
[3] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-aws-console-sign-in-events.html

CloudTrail (5)

A "trail" can be used to identify a way to save CloudTrail object logs to an S3 bucket:

- Saved to a specific bucket
- A prefix for all the stored logs
- Encryption
- SNS notifications when log arrives
- What types of API calls? Management events, data events, insight events
- Read/write/both events



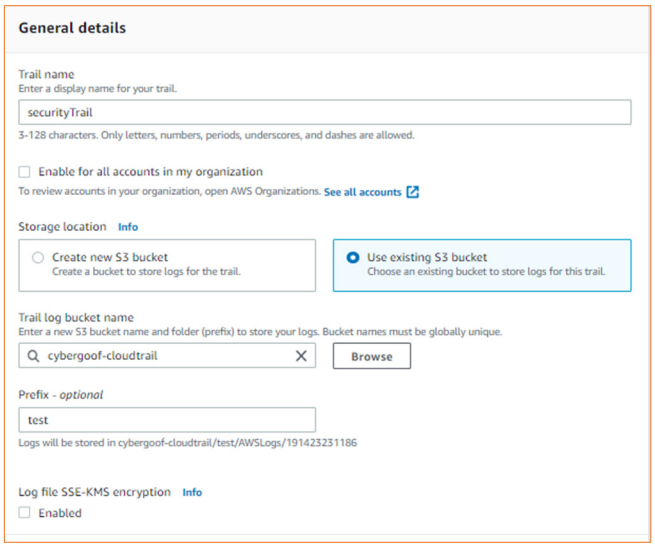
By default, CloudTrail is storing ninety days of management events. A "trail" is an AWS resource in the CloudTrail service that describes what kinds of events to save, the bucket to save it to, and potentially other actions when an event is generated. When creating a trail, there are some parameters that can be applied to manage how events are saved.

- A bucket is specified that the events are stored in. You only pay standard S3 storage prices.
- A *trail* could store events with a prefix for easier finding. For instance, if we create a trail specifically for security purposes, we could prefix all logs with "security".
- Events can be stored with either S3 standard encryption, or with a KMS key. One big advantage of the KMS key is that you can limit who could access those logs to only those who have access to that key.
- An SNS topic message can be created every time an event is created.
- The default events captured by CloudTrail are management events. There are also data events and insight events, which are discussed in two slides.
- In a Trail, you may want to focus just on read events, write events, or both events.

CloudTrail (6)

Limitations:

- Five trails per account per region
- Delivery to CloudTrail within fifteen minutes of API call
- Delivery to S3 bucket every five minutes



When creating a trail, there are some limits that you have to be aware of, especially that trails are not real time. Any alerts you setup will have a delay.

There is a limit of five trails per account per region. Security teams will typically use one trail to send data to a security team owned bucket for log analysis. This trail should be deployed with every account, and turning off that trail is denied. Management events will now show access to S3 buckets, and that data might be necessary for regions and accounts with high-risk buckets, such as ones that hold credit card numbers. A second, more customized tail may be necessary for monitoring these high priority buckets. Development teams will likely want to create a trail to send to CloudWatch for rule eventing. Well, that is three trails already, only two left. AWS pushes an elastic mindset with its resources, but Trails do not.

The API service will deliver the log to CloudTrail within fifteen minutes. It is not real time.

Events themselves are bundled up into compressed formats every five minutes. It could take nearly twenty minutes from the time an event happens until it is dropped into the S3 bucket—usually faster, but it can take some time.

We will discuss using CloudWatch for faster event driven operations, but for collecting logs, it will not happen immediately.

CloudTrail (7)

1. Management events
   - Typical AWS API calls dealing with resources
2. Data events
   - Operations performed within a resource
   - High volume, noisy: S3 objects, Lambda invocation
3. Insight events
   - AWS service that monitors "Write Management Events" and detects abnormal activity

There are three different types of events that CloudTrail will capture.

Management events are the default events captured by the default CloudTrail settings. These events describe API calls to the AWS services, which typically are performing the actions of create, describe, update, list, or delete of AWS resources themselves. When conducting threat analysis and detection, these are the most common CloudTrail events we will be working with. We want to know when resources have CHANGED. Actions such as identity access policy changes, turning off the security trail, or creating BitCoin miners in an unused region are big giveaways of nefarious activities.

Creating or updating resources can be a tell-tale sign of a problem, but what about access to sensitive data? Management Events capture API actions on the S3 buckets themselves, but not the data inside S3 objects. **Data events**, however, track API actions on the S3 objects with information such as AWS account of caller, API call, and other details. It can be applied to specific S3 buckets or all buckets in the account. Data events can also be used to track Lambda invocations. The problem with data events is the huge volume of data. Should you track every time an object is read from any S3 bucket? Some organizations want to track that data, but you are paying for the log storage. When using tools like Athena to analyze, or Splunk to ingest, having that much unnecessary data can increase cost, not to mention the frustration of users. For data events, it is the authors' recommendation to create a separate trail for high-risk data, or accounts that have high risk data. Do not bother monitoring object reads on the marketing team's S3 bucket.

AWS has a concept called ***Insight events*.** These are not API events, but second order analytics generated by AWS based on monitoring of CloudTrail logs. They identify abnormal activities, more focused on operations concerns rather than security issues.

CloudTrail in Practice (1)

If an attacker tries to discover cloud services, can we detect it?

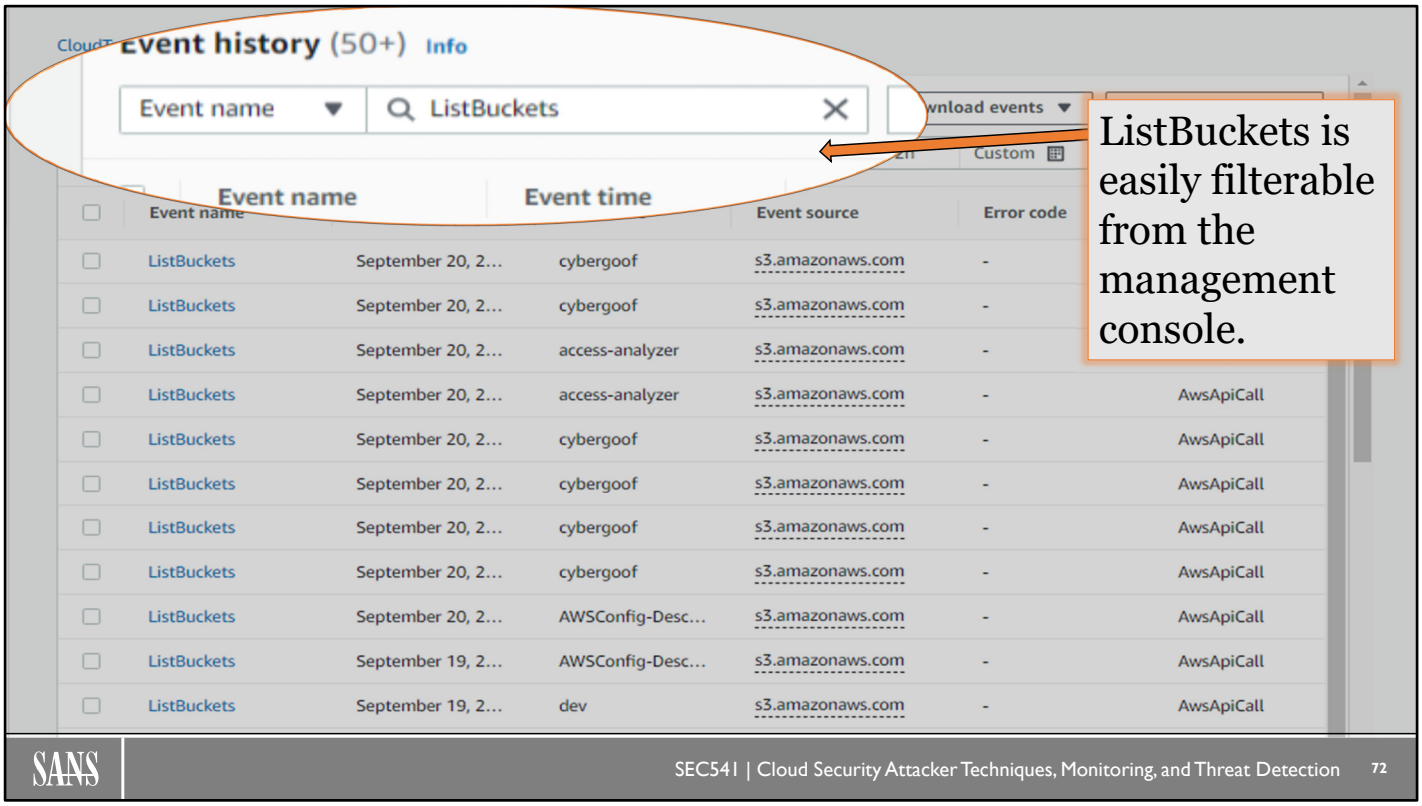
- Does CloudTrail give us the right information?
- How would we query it?
- Could there be false positives and false negatives?

Can we detect? aws s3 ls

Looking back at our attacker, we want to determine if we can detect a Cloud Service Discovery attack. The first thing the attacker will do is start looking for juicy targets. Let's focus in on a particular action, listing all S3 buckets.

Our opening hypothesis is that we can detect when someone performs a *ListBuckets* command to the S3 service API. Can the CloudTrail logging service give us that information? Can we prune away the false positives, or normal listing bucket activities? How hard is it to gather the information, and what tools do we need to perform the query? Does CloudTrail have enough information to take action? Otherwise, it may not be worth it.

The first thing we can do is generate the log ourselves, so we are aware of how the CloudTrail log looks. In the command line, the command is simply *"aws s3 ls"*, which will do a ListBuckets.



First, we will use the use the CloudTrail Event History filter to find what we need. We can search for the event all *ListBucket* events in the last 30 minutes.

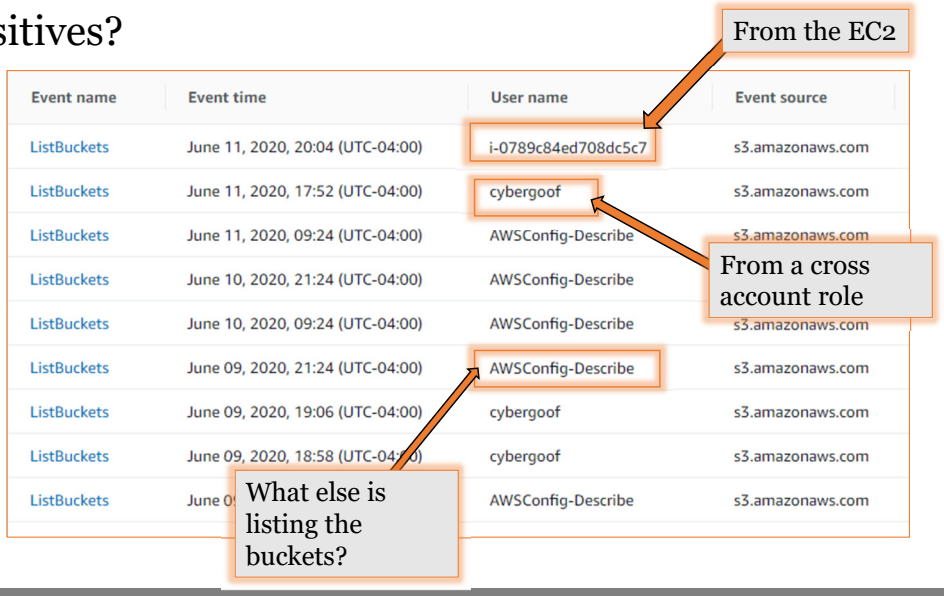
This filtering of event history allows the user to filter based on access key, event ID, event name, event source, if it's read-only, the resource name, resource type, or username. This limited filtering is okay for an initial look at the data, but it is hardly sufficient for a threat detection scenario. Want to filter on ListBuckets from a particular user? We will talk about how to do that soon.

References:
https://docs.aws.amazon.com/cli/latest/reference/s3api/list-buckets.html
https://docs.aws.amazon.com/AmazonS3/latest/API/API_ListBuckets.html

CloudTrail in Practice (2)

Do we have false positives?

False negatives?



Event name	Event time	User name	Event source
ListBuckets	June 11, 2020, 20:04 (UTC-04:00)	i-0789c84ed708dc5c7	s3.amazonaws.com
ListBuckets	June 11, 2020, 17:52 (UTC-04:00)	cybergoof	s3.amazonaws.com
ListBuckets	June 11, 2020, 09:24 (UTC-04:00)	AWSConfig-Describe	s3.amazonaws.com
ListBuckets	June 10, 2020, 21:24 (UTC-04:00)	AWSConfig-Describe	
ListBuckets	June 10, 2020, 09:24 (UTC-04:00)	AWSConfig-Describe	s3.amazonaws.com
ListBuckets	June 09, 2020, 21:24 (UTC-04:00)	AWSConfig-Describe	s3.amazonaws.com
ListBuckets	June 09, 2020, 19:06 (UTC-04:00)	cybergoof	s3.amazonaws.com
ListBuckets	June 09, 2020, 18:58 (UTC-04:00)	cybergoof	s3.amazonaws.com
ListBuckets	June 0	AWSConfig-Describe	s3.amazonaws.com

Wow, a lot of things can list the buckets, not just human interaction. The User name of *cybergoof* is the author performing the action. Not just through the AWS CLI, by running *aws s3* ls, but just visiting the S3 page in the management console will generate ListButckets. Is this legitimate? It is, and it might be difficult to tell if a user is legitimately performing a ListBuckets, or if an attacker has stolen the credentials of the *cybergoof* user.

AWS services are also performing ListBuckets. The AWS Config is always looking at resources. We need to ignore these commands, as they are not likely an attack.

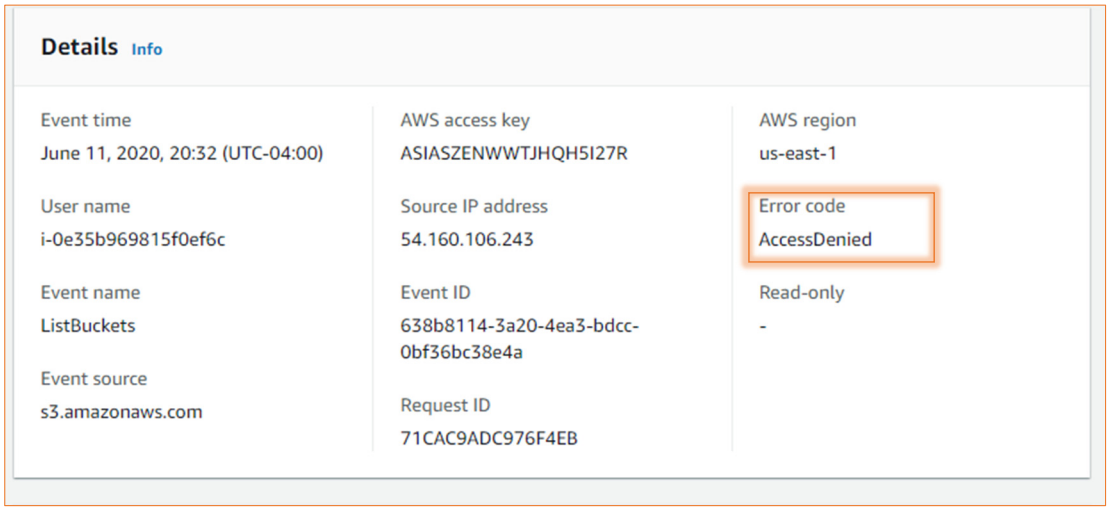
We also see an EC2 is performing a ListBucket. Is that normal practice? Would a virtual machine running on a web server want to list all the buckets in an account? Likely not. Certainly not in a production environment. If a web server is supposed to read data from a bucket, it will already have been configured for the specific bucket to read. The security team has to know the purpose of the resources deployed, what is "normal", or at least be able to engage with those in the know. So much is happening in the cloud environment —it's always changing—but somehow the security team needs to know what the dev and architecture teams intend.

This EC2 warrants further investigation.

This is the tricky part of threat analysis. You are investigating attacks that are not detected through your security tools because their behavior may not necessarily be harmful. You are detecting bad behavior hiding in normal operations. Determining that your EC2s should not be doing S3 bucket listing as part of normal operation will help you start to pinpoint this unusual behavior.

CloudTrail in Practice (3)

What about a denied request?



When we click into the details of this particular request, we see we have an "Error code" of *AccessDenied*. This tells us that an API call was attempted, specifically a *ListBuckets* from the user *i-0e35b969815f0ef6c*, which is an EC2 instance. That particular EC2 has an IAM role attached, but that IAM role's policy does not allow for *ListBuckets*. So, the CloudTrail has logged it as an *AccessDenied*.

There may be a number of reasons that an EC2 would make a failed EC2 call. Maybe that EC2 is a developer or admin instance acting as a bastion host. The developer or admin, a human, is running commands that end up hitting the permission boundary. This happens all the time. It's important for the investigator to know what kind of resource they are investigating.

It could be a machine running code that is performing the unapproved command. This can happen all the time when developing code in the cloud environment while trying to manage least privilege. Maybe the code needs to perform some action and it was accidently denied. This is why it's important to have separate accounts for development than production. What might slide in development may require a detailed investigation in production.

We know that this is a web server, and it is running in production. So, what could be potential reasons that the instance is making unexpected, and denied API calls? Web servers that are hosting customized applications to the general internet should be considered high risk, so this is probably a good place to start an investigation.

We likely will want to start investigating the web server itself. Maybe take a snapshot of it to preserve what is on it, but we will talk about those types of forensic analysis later in this class.

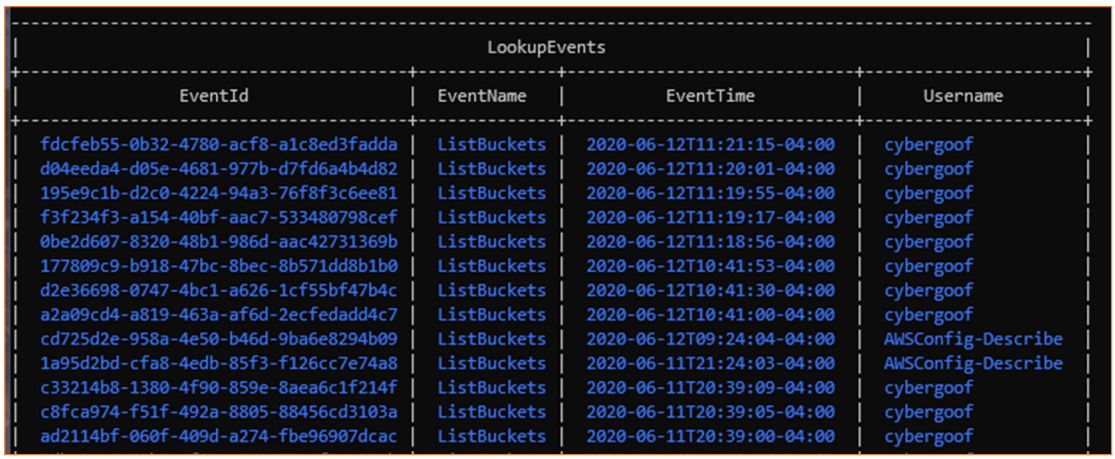
Thinking about a web server making a denied API call, we will likely discover 1 of 3 scenarios have happened:

1) The production application is deployed with missing permissions set. It may not be immediately obvious, or it would have been caught in testing, but this will likely cause a failure in the future. Time to call the developers or ops team.
2) The application is making unnecessary API calls. Maybe the developers made a mistake? But this is a good time to send the code back. We discussed earlier that we need to be able to provide input to architects and developer teams. All teams are needed to make the AWS environment easier to monitor. Identifying noise, like applications creating invalid API calls, will make it harder to detect real problems.

3) An attacker may be using this web server to perform a Cloud Service Discovery attack. It is healthy for the threat detection teams to be a bit paranoid, so we will assume that is what is happening. If an attacker gains access through a web exploit, or through the front door of a deployed managed service, then they will be operating as that resource when they start performing enumeration.

CloudTrail Through CLI

```
$ aws cloudtrail lookup-events --lookup-attributes
      AttributeKey=EventName,AttributeValue=ListBuckets
```



```
-------------------------------------------------------------------------------------------------------------
|                                                LookupEvents                                               |
+--------------------------------------+-------------+---------------------------+-----------------------+
|              EventId                 |  EventName  |         EventTime         |       Username        |
+--------------------------------------+-------------+---------------------------+-----------------------+
|  fdcfeb55-0b32-4780-acf8-a1c8ed3fadda |  ListBuckets |  2020-06-12T11:21:15-04:00 |  cybergoof            |
|  d04eeda4-d05e-4681-977b-d7fd6a4b4d82 |  ListBuckets |  2020-06-12T11:20:01-04:00 |  cybergoof            |
|  195e9c1b-d2c0-4224-94a3-76f8f3c6ee81 |  ListBuckets |  2020-06-12T11:19:55-04:00 |  cybergoof            |
|  f3f234f3-a154-40bf-aac7-533480798cef |  ListBuckets |  2020-06-12T11:19:17-04:00 |  cybergoof            |
|  0be2d607-8320-48b1-986d-aac42731369b |  ListBuckets |  2020-06-12T11:18:56-04:00 |  cybergoof            |
|  177809c9-b918-47bc-8bec-8b571dd8b1b0 |  ListBuckets |  2020-06-12T10:41:53-04:00 |  cybergoof            |
|  d2e36698-0747-4bc1-a626-1cf55bf47b4c |  ListBuckets |  2020-06-12T10:41:30-04:00 |  cybergoof            |
|  a2a09cd4-a819-463a-af6d-2ecfedadd4c7 |  ListBuckets |  2020-06-12T10:41:00-04:00 |  cybergoof            |
|  cd725d2e-958a-4e50-b46d-9ba6e8294b09 |  ListBuckets |  2020-06-12T09:24:04-04:00 |  AWSConfig-Describe   |
|  1a95d2bd-cfa8-4edb-85f3-f126cc7e74a8 |  ListBuckets |  2020-06-11T21:24:03-04:00 |  AWSConfig-Describe   |
|  c33214b8-1380-4f90-859e-8aea6c1f214f |  ListBuckets |  2020-06-11T20:39:09-04:00 |  cybergoof            |
|  c8fca974-f51f-492a-8805-88456cd3103a |  ListBuckets |  2020-06-11T20:39:05-04:00 |  cybergoof            |
|  ad2114bf-060f-409d-a274-fbe96907dcac |  ListBuckets |  2020-06-11T20:39:00-04:00 |  cybergoof            |
```


The management console gave us minimal ability to filter for events. We can replicate that visualization on the command line.

As we said before, the filtering on that management console page used a parameter *--lookup-attributes* to tell the API which logs to bring back based on a single dimension.

AWS CLI commands have the same basic building blocks.

aws is the application name
cloudtrail is the name of the API service
lookup-events is the command

After the app, service and command, it is a set of key value parameters with the key denoted by double dashes.

For this command, the key is *--lookup-attributes* and the value is this long string *AttributeKey=EventName,AttributeValue=ListBuckets*

The attribute type we are going to look up is *EventName*, and the value of that attribute is *ListBuckets*

Keep in mind, this API call is pulling data from our default CloudTrail service that is keeping logs for 90 days. If you want to access a "trail" of last year's logs, then you will need another service. We will talk more about that in the class.

Lookup events can look from the following attributes: `access key, event ID, event name, event source, read only, resource name, resource type, username.`

The `--lookup-attributes` command is not an adequate filter for analysis. Technically, the "`lookup-attributes`" is a list, but there can only be one value. We will need additional tools to do good analysis, filtering, and searching.

Note: In our command line shown above, we also used the --*query* and --*output* directives to create the table. You will see how to do this in the lab.

Reference:
https://awscli.amazonaws.com/v2/documentation/api/latest/reference/cloudtrail/lookup-events.html

Azure Resource Provider Operations

Used to define a role to be assigned to a requesting resource

- At the time of this writing, there are **9,528** permissions available

List of available operations can be retrieved two ways:

- **Portal**: when creating custom role, select "Download all permissions"
- **CLI: `az provider operation list`**

```
Microsoft.DBforMariaDB/servers/queryTexts/action
Microsoft.DBforMariaDB/servers/queryTexts/action
Microsoft.DBforMariaDB/servers/privateEndpointConnectionsApproval/action
Microsoft.DBforMariaDB/servers/read
Microsoft.DBforMariaDB/servers/write
Microsoft.DBforMariaDB/servers/delete
Microsoft.DBforMariaDB/servers/restart/action
```


Similar to AWS's actions, Azure provides users with resource provider operations. These permissions are used to determine which requests can be allowed or denied on a particular resource by a requestor. There are numerous available in Azure and can be retrieved a few different ways:

- Azure Portal: When attempting to create a custom role, there is an option to "Download all permissions". This download contains a CSV file with every available permission.
- Azure CLI: Run the `az provider operation list` command.

The example above shows how all Microsoft Compute-related permissions can be acquired using a little command-line kung fu:

```
az provider operation list | \
  jq -r '.[].resourceTypes[].operations[].name'
```

The command will retrieve all permissions, filter the results by the name field, and then filter out only the permissions that contain "Microsoft.Compute".

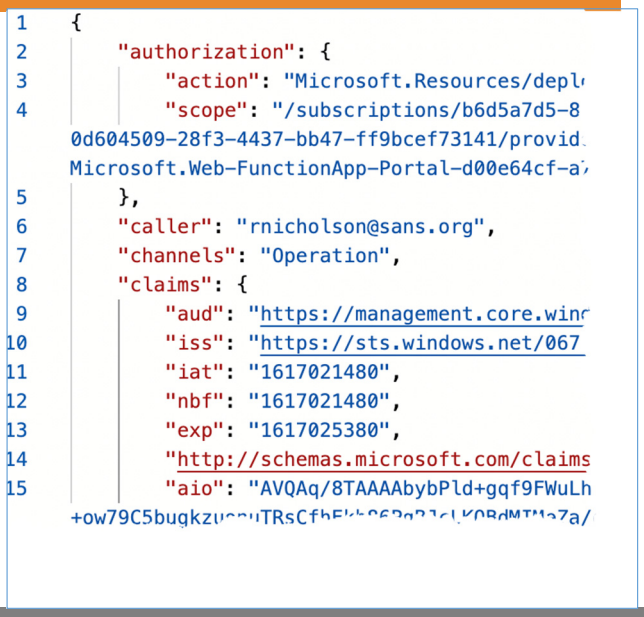
Azure Activity Log

Provides visibility into API activity within the Azure subscription

- Only provides **create**, **update**, and **delete** activity!

Can be accessed via multiple methods:

- Azure Portal
- Azure CLI
- PowerShell cmdlet's



Azure's answer to collecting API calls is the Azure Activity Log service. Here, any user account or cloud service that makes a change to the environment will be logged. This means that read-level API calls will not be captured here—only actions that create, update/modify, or delete a cloud resource.

Similar to AWS, there are multiple methods to review this data, first of which is using a graphical user interface within the Azure Portal. (https://portal.azure.com/#blade/Microsoft_Azure_ActivityLog/ActivityLogBlade).

There are two other approaches as well: using the Azure Command Line Interface (CLI) tool or some installable PowerShell cmdlets. Both of the command-line tools (Azure CLI or PowerShell cmdlets) require authentication to an Azure user account or service principal (i.e., service account) with appropriate read-level access to the Activity Log service.

Azure Activity Log—Portal (1)

Presented initially by operation, initiator, and few other fields

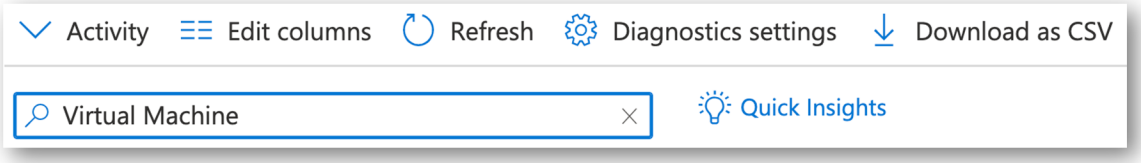
- Groups events by correlation ID
- Can drill in to see complete, JSON-formatted event data

When reviewing event data in the Azure Portal, there are a few nuances to understand in Azure that make this service quite different than AWS's CloudTrail offering. First, events can be grouped together and share the same `correlation ID` field. This field's value is shared among the events that belong to the same uber operation. In other words, related events are merged into one collapsible group within the portal.

For example, when deleting an Azure Resource Group, which Azure defines as "a container that holds related resources for an Azure solution", there will be many delete operations happening as several cloud resources are removed, yet it was one major action a user performed to cause all of these actions to be conducted. These many actions would share the same correlation ID.

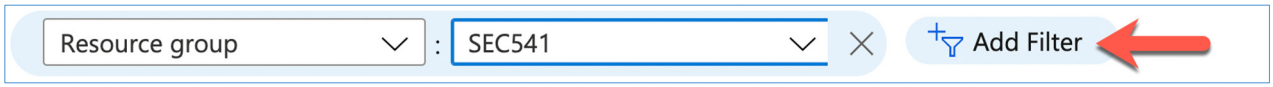
Azure Activity Log—Portal (2)

Very easy to filter using the search box



Filtering by time is quite easy as well using Azure-provided ranges (default is **last six hours**) or custom ranges

Many other filtering options by clicking on the **Add Filter** button

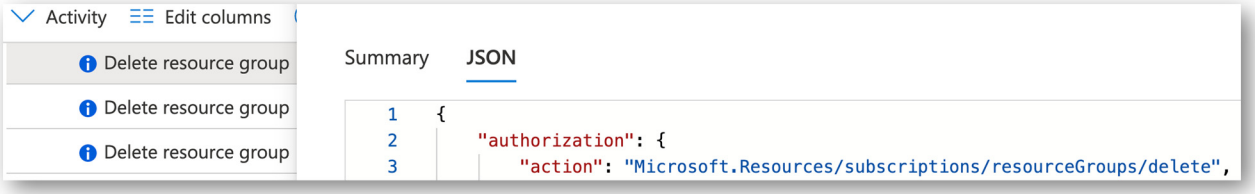


In a very busy Azure account, it may be advantageous (and certainly more efficient) to filter the events, and the Portal approach gives plenty of opportunities to do just that:

- Free-form text search: This allows you to search all of the events and event fields for a known piece of plain text.
- Time filtering: You can narrow your results down to a particular time frame. There are several built-in time frames to utilized that look back a certain amount of time (one hour, six hours, twenty-four hours, one week, two weeks, or one month). You also have the flexibility to create a custom time frame using a start date/time and end date/time.
- Filter by cloud resource specifics: More commonly searched fields, such as Resource group, Resource, Resource type, Operation, Event initiated by, and Event category can all be specified to narrow down the results.
- Event severity: Only display events based upon the severity determined by Azure (e.g., Critical, Error, Warning, Informational).

Azure Activity Log—Portal (3)

Complete, JSON-formatted data is **several dozen** fields per-event



Most relevant fields include:

- **authorization.action**
- **caller**
- **claims.ipaddr**
- **claims.name**
- **correlationId**
- **eventTimestamp**
- **resourceId**
- **status.value**

When viewing the data in the Activity Log's home page, it will only display limited data about the event. You can choose the event fields by clicking on the "Edit columns" button, but, at most, you will see the following in this view:

- Operation name (on by default and cannot be disabled)
- Status
- Event category
- Time
- Timestamp
- Subscription
- Event initiated by
- Resource group
- Resource type
- Resource

To view much more data, you can elect to dig deeper by clicking on the event and then clicking on the JSON tab. Here, you will find several dozen fields. Not all of this data may be useful, but what you see above will fit the needs of many investigations.

Azure Activity Log in Practice (1)

A sensitive system (LogServer) has suddenly (within the last hour) **gone missing** from our Azure account, and we must determine who is behind these shenanigans!

As a hint, this is what the malicious user ran:

az vm delete --name LogServer --resource-group SEC541

- Does Azure Activity Log give us the needed information?
- How would we query it?

The hypothesis in this scenario is that we can detect an attacker or rogue internal user that would like to cover their tracks by deleting a virtual machine named LogServer. If we are lucky, even if this server is lost, there may still be some artifacts in the Azure Activity Log service.

We need to determine if the Azure Activity Log service can provide the information we are requesting. If so, how do we query it so we can prune away any "business as usual" commands?

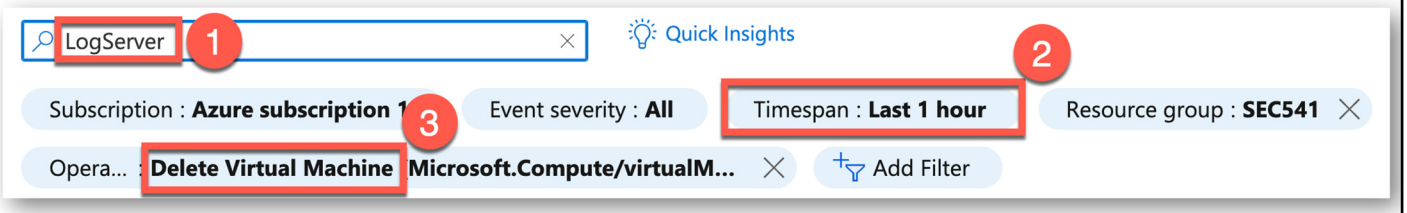
As was mentioned, when beginning a threat hunt or investigation, it's usually a good idea to simulate or create the types of logs we are interested in generating. It does not have to be a detailed replication of the attack, but enough so that any log filtering syntax can be worked out.

A command that can be issued in this instance is `az vm delete --name LogServer --resource-group SEC541` to generate the needed log event.

Azure Activity Log in Practice (2)

There is already quite a bit that can be used to filter down the Azure Activity Log data:

1. Cloud resource: LogServer virtual machine
2. Time frame: Last hour
3. Action taken: Suspected deletion of VM



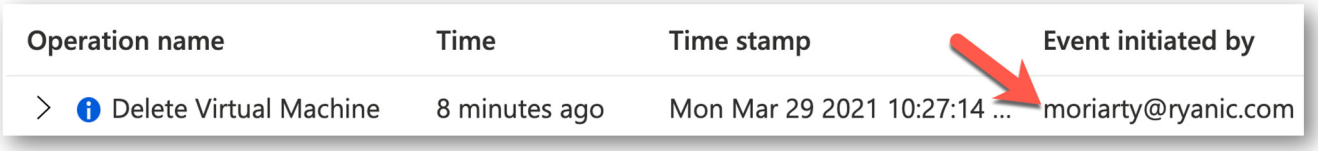
Based upon the scenario, what, exactly, should be searched for to narrow down the results? First, there is the cloud resource in question—the LogServer virtual machine. It is also known that the server went missing within the last hour, so there is a timeframe to filter on. Finally, it is assumed that the action taken was deletion of the VM.

Now off to Azure's Activity Log. Does this service give the ability to filter on these things? To look for the cloud resource by name, one option is to just do a free-form search for the VM name (LogServer). To address the time frame, using a built-in option of "Last 1 hour" will significantly narrow down the results. And finally, the Operation that is of concern is Microsoft.Compute/virtualMachines/delete. Of course, these actions can be memorized, on a cheat sheet, or looked up every time you would like to filter by Operation, but Azure gives a more friendly name to help find the correction action that occurred in Azure.

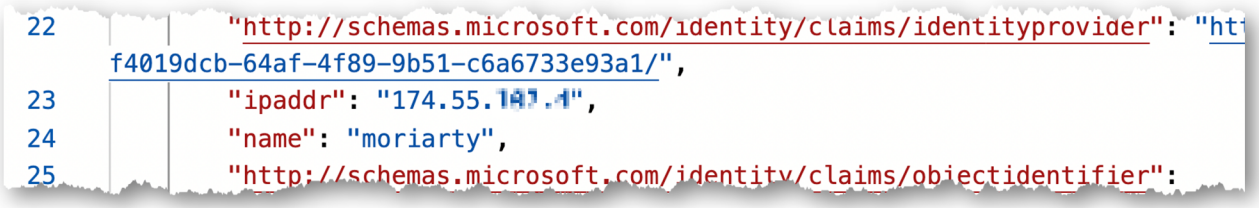
Armed with this filter, what is found?

Azure Activity Log in Practice (3)

So, who was behind this?



And where did they conduct this action from?



It appears that there is a match! In fact, eight minutes before this search was performed was when this VM was deleted.

Also, since one of the columns displayed is "Event initiated by," it can be determined which user account or cloud resource made this deletion request. The username moriarty@ryanic.com was the account behind this nefarious activity!

To acquire even more information surrounding this event, clicking on the event in question and the JSON tab on the following page will provide additional items such as the public IP address that the attacker was communicating with or through (if a proxied service).

Azure Activity Log—CLI

az monitor activity-log list --offset 1h
--query "[?contains(@.authorization.action,
'virtualMachines/delete')].[caller,claims.ipaddr]"

```
$ az monitor activity-log list --offset 1h --query "[?contains(@.authorization.a
ction, 'virtualMachines/delete')].[caller,claims.ipaddr]"
[
  [
    "moriarty@ryanic.com",
    "174.55.[redacted]"
  ],
  [
    "moriarty@ryanic.com",
    "174.55.[redacted]"
  ]
]
```


To sift through large datasets or execute additional commands or scripts that may take these results as input, it may be much more efficient or useful to perform a search using the Azure CLI tools. Looking at the following command, you will see some of the same filters that were used when filtering using the Portal:

```
az monitor activity-log list --offset 1h -query
  "[?contains(@.authorization.action,
  'virtualMachines/delete')].[caller,claims.ipaddr]"
```

To look back in time, the `--offset` flag can be used with a number followed by either `h` for hours or `d` for days. The `--query` flag is quite complex and uses JMESPath as a query language to filter JSON results. At a high level, this query is looking for any events where the action field (nested within the authorization field) contains the string "virtualMachines/delete". From there, the query will output both the caller field (Azure user) and public IP address of the request.

Again, you can see the exact results - moriarty@ryanic.com and the requesting IP address.

Azure Activity Log—PowerShell

Much more complex and less native options, but with the "power" of PowerShell, makes for a very flexible approach

$output = Get-AzLog -StartTime 2021-03-29T10:20:00 -WarningAction silentlyContinue | where { ($_.Authorization.Action -eq "Microsoft.Compute/virtualMachines/delete") -and ($_.ResourceId.Contains("LogServer"))}

$output | % {$_.caller ,$_.claims.Content.ipaddr -join ': '}

```
PS /> $output = Get-AzLog -StartTime 2021-03-29T10:20:00 -WarningAction s
ilentlyContinue | where { ($_.Authorization.Action -eq "Microsoft.Compute
/virtualMachines/delete") -and ($_.ResourceId.Contains("LogServer"))}
PS /> $output | % {$_.caller ,$_.claims.Content.ipaddr -join ': '}
moriarty@ryanic.com: 174.55.[redacted]
moriarty@ryanic.com: 174.55.[redacted]
```


Last, but not least, there are PowerShell cmdlets that can be installed on a system to parse this event data. In this case, the command line flags and queries that were showcased in the CLI tool approach are not natively supported in the cmdlet, but since PowerShell has an extensive command-set, any data output from the Azure cmdlets can be further parsed as shown here:

```
$output = Get-AzLog -StartTime 2021-03-29T10:20:00 -WarningAction
  silentlyContinue | where { ($_.Authorization.Action -eq
  "Microsoft.Compute/virtualMachines/delete") -and
  ($_.ResourceId.Contains("LogServer"))}

$output | % {$_.caller ,$_.claims.Content.ipaddr -join ': '}
```

There are two commands here. The first is doing most of the filtering:

- Pull down the Azure Activity Log data to the local system and filter for all logs that happened since March 29, 2021, at 10:20:00 (approximately one hour ago)
- Sift through those results and match any that have an action equal to "Microsoft.Compute/virtualMachines/delete" as well as affect the LogServer VM.

There is still more filtering to be done, but only to present the data of interest. The second command is used to only display the username and public IP address, which are separated by a colon and a space.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. **EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail**
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Lab 1.2 | Detecting Cloud Service Discovery Attack with CloudTrail

Exercise Duration: 45 Minutes

Objectives

In this lab, we will investigate the CloudTrail service and look for Cloud Service Discovery attacks

- Research the MITRE ATT&CK technique
- Perform discovery attack with Pacu
- Explore AWS CloudTrail service
- Initiate AWS API calls and use CloudTrail to detect it
- Create a custom trail

Welcome to the first investigation lab of the class. In this lab, we will be performing some log analysis of the CloudTrail service in AWS.

You will do some research on the MITRE ATT&CK matrix, looking at different types of discovery attacks you may want to investigate when you get to your home office.

You will then use an open-source tool called Pacu to perform some pen testing enumerations in the AWS environment. We will use it to perform a Cloud Service Discovery attack that we are not authorized to do.

After running Pacu, we will analyze the CloudTrail logs through the command line.

Last, we will create our own trail in CloudTrail to send data to an S3 bucket of our choice.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. **Parsing JSON**
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Parsing JSON (1)

$ aws cloudtrail lookup-events --lookup-attributes
AttributeKey=EventName, AttributeValue=ListBuckets

Here is a detailed view of a ListBuckets command. The valid attribute values from "lookup-events" is easily visible as JSON. These generic values are the same for all cloud trail events, irrespective of the resource type[1].

Besides what is in "lookup-attributes", other values in the command line that can be used to select down the returned cloud trail events are start-time, end-time, max-items. This command also leverages the CLI pagination commands[2].

The --query parameter in the command line can also be used to filter which values are returned. The --query parameter accepts strings that are compliant with the JMESPath[3] specification.

[1] https://docs.aws.amazon.com/awscloudtrail/latest/userguide/cloudtrail-event-reference-record-contents.html?icmpid=docs_console_unmapped
[2] https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-pagination.html
[3] https://jmespath.org/

Parsing JSON (2)

What if we only want to show failed calls?

CloudTrailEvent is a string. AWS CLI --query cannot filter.

Let's look at the details of one of our events. The values we can use with *--lookup-attributes* are listed in JSON, and we could do some filtering in --query.

However, if we wanted to filter on a value that is part of the "CloudTrailEvent" returned parameter, the JSON values[1] must be a string, number, a JSON object, an array, a boolean, or null.

But the CloudTrailEvent is a string, not technically a JSON object. I know, it looks like JSON, but it is an escaped string. The quotes have been escaped with a backslash.

Investigating the string field shows that there is an "errorCode" of value "AccessDenied". We need a new tool to be able to convert this string to JSON, and then filter on AccessDenied.

One thing to note, this approach of threat analysis is just in time evaluation of the data. The logs are stored in their most raw format without any pre-parsing. That pre-parsing makes sense if it's data we know we want and will use regularly. We might send it to a SIEM of choice, which will normalize the data. But before we go through the hoops of preparing and storing that data, we may want to perform just in time analysis of the data in whatever format it is in. Easy if the data is in JSON, but more difficult if the data has been encoded like the CloudTrailEvent.

[1] https://www.w3schools.com/js/js_json_datatypes.asp

Parsing JSON (3)

jq is a lightweight and flexible command-line JSON processor that we can use to extract the json data

- Focus on making JSON easier to manipulate
- Like SED, can be piped through STDOUT
- Filter, slice, map, transform, and mangle JSON

We can pipe output of AWS commands to jq for further filtering

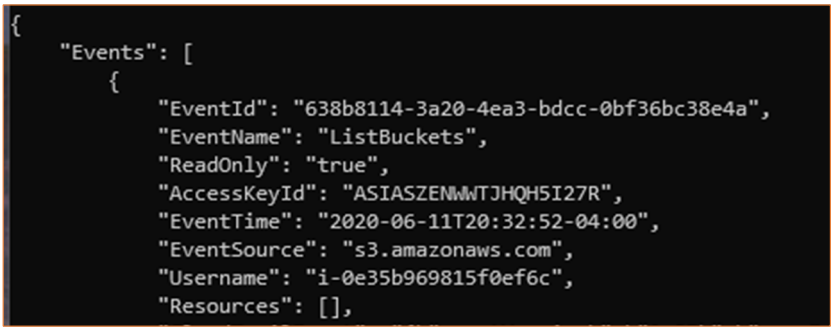
With these cloud services, we will regularly be dealing with the JSON data format. A big advantage is that every modern programming language has support for using the JSON format. Yet, we are not ready to write a program that automates the analytics. We are dealing directly with the CloudTrial logs in their native format. We can use the AWS command line tool to do some basic filtering with the *--query* directive, but we need something more robust. Especially when we have to deal with the CloudTrailEvent data, which is an encoded string.

We will use a tool called *jq,* similar to the "SED" tool, but for JSON data. You can filter, slice, map, and change the data with simple commands. The conversion of the data can transform the string escaped JSON data into structured JSON. You can control the look of output formatting, sort keys, and add, remove, or alter properties, among other things.

One of the big features is filtering an array of JSON data to output the values that meet some filter. That filter can be standard bool expressions, or leverage some built in functions such as searching for string at the beginning or end of a property value.

Reference:
https://stedolan.github.io/jq/

Parsing JSON (4)




```
$ aws cloudtrail lookup-events --lookup-attributes \
      AttributeKey=EventName, AttributeValue=ListBuckets \
      | jq '.Events[]|select(.Username | startswith("i-"))'
```


Here is a simple example where we are looking for all CloudTrail events that have an event name of "`ListBuckets`", but we are only interested in commands that were executed on an EC2. The "`Username`" would be the EC2 identity, such as "i-0e36b969815f0ef6c". We can use the "startswith()" function to find all values that start with "`i-`".

The pipe "|" will send the output of aws cloudtrail to the jq application.

The filter for jq is inside of single quotes.

Looking at the JSON data above, we can see the JSON formatted values start with an array of "Events" objects. From within each of those objects in the Events array, we want to select the ones where the username starts with "`i-`".

This diagram does not show jq investigating the CloudTrailEvent string escaped JSON object.

Parsing JSON (5)

```
$ aws cloudtrail lookup-events  --lookup-attributes
        AttributeKey=EventName,AttributeValue=ListBuckets
        | jq '.Events[]|select((.Username | startswith("i-")) and
         (.CloudTrailEvent|fromjson|select(.errorCode=="AccessDenied"))) '
```

The JSON key called "`CloudTrailEvent`" contains JSON data that has been converted to a string. We will leverage jq's built in function, "`fromjson`", to convert that string into a proper JSON object that jq can work with. The "`tojson`" function will convert JSON data into a parameterized string, which can be handy. The terms "`fromjson`" and "`tojson`" may seem to be the opposite of what they actually do, but that is how they are named (https://stedolan.github.io/jq/manual/#Builtinoperatorsandfunctions).

Piping the contents of "`CloudTrailEvent`" through "`fromjson`" can be sent to a jq function called "`select`", which will select only the rows of data that match a particular value. In our case, we are looking for the logs that have an "`errorCode`" value of "`AccessDenied`".

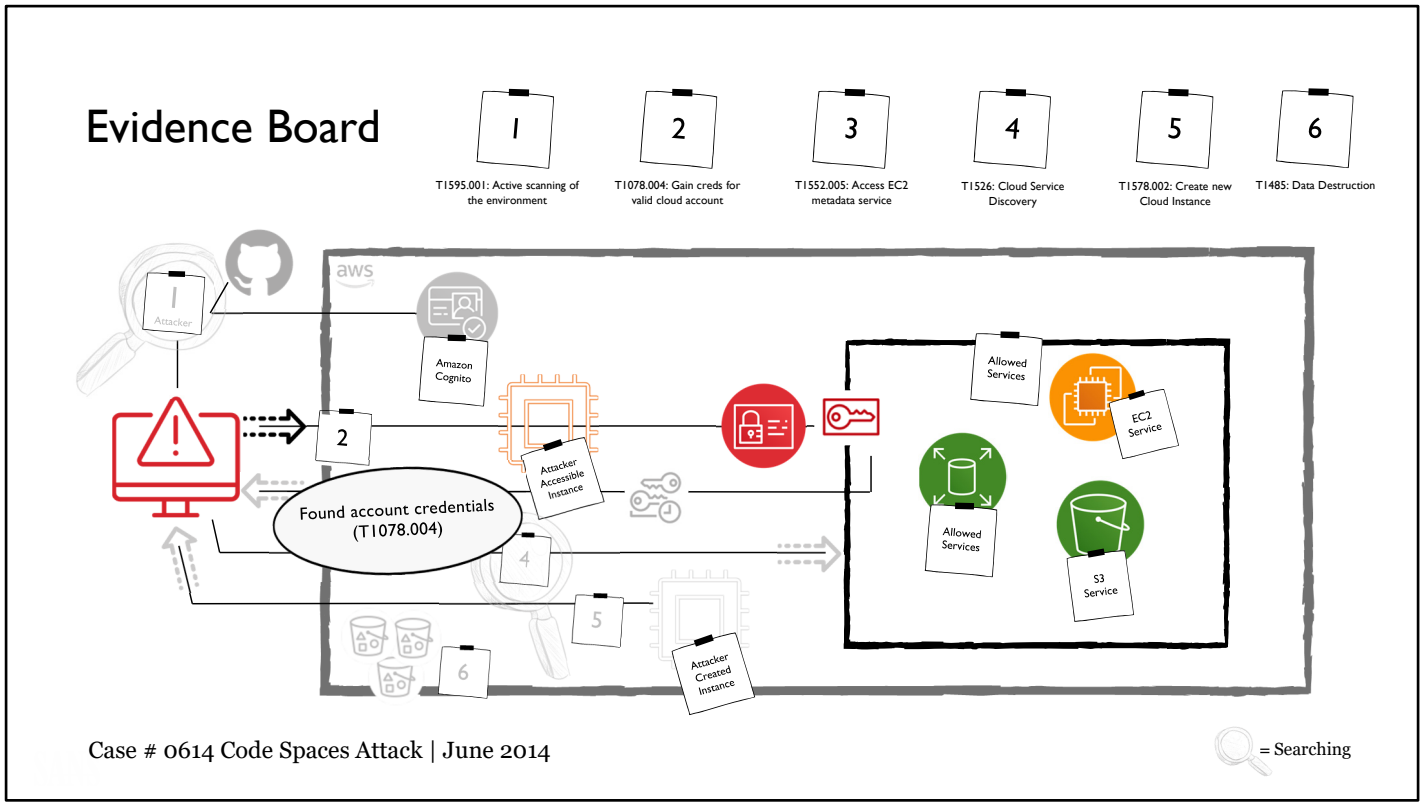
Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automate Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. **Cloud-Native Logging Services**
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.



It is not clear how the attacker gained the credentials. They may have been able to guess a username and password through brute force, or by making educated guesses from a set of common passwords[1].

One byproduct of automation, DevOps, and other "deployment by code" tools and capabilities is that access credentials are sometimes accidently checked into a repository. Ransomware and cryptocoin miners have bots that are constantly going through GitHub looking for security keys. One developer accidently published credentials to his GitHub account for 5 minutes. Once realizing his mistake, he pulled the keys. The next morning, he discovered 140 servers running, all mining Bitcoin for someone else[2].

This can happen if the wrong file gets pushed, or maybe it was a misconfiguration. Developers may use scripts to test code, and they include credential information. Or maybe they accidently grab the AWS or Azure CLI credential files? However it happens, security teams need to be on the lookout for it. GitHub and AWS will try and detect this happening, but you usually have to be running enterprise services[3].

Once the attacker found the account credentials, the AWS accepted them as a valid user with a given level of privileges.

We are going to need more logs than what CloudTrail provides. Let's look at Cloud Native Logging solutions in AWS and Azure.

[1] https://nordpass.com/most-common-passwords-list
[2] https://www.theregister.com/2015/01/06/dev_blunder_shows_github_crawling_with_keyslurping_bots
[3] https://www.dannyguo.com/blog/i-published-my-aws-secret-key-to-github

Cloud Native Logging

Cloud services generate a lot of logs, and not all are easy to use.

AWS and Azure have built services, tools, and capabilities to help customers gather, analyze, and even build automated response actions such as:

- Gather cloud native and application generated logs
- Dashboards to show metrics
- Create alerts and events

AWS and Azure services are different, both with pros and cons.

Of the five NIST cloud characteristics, the measured service characteristic is one that we will take advantage of the most when detecting threats in an environment. In order to automate elasticity in a cloud environment, logs and metrics at the right fidelity must be available to the cloud provider and the customer alike. In a Software as a Service cloud offering, the provider may be stingy on what logs the customer might receive, usually only providing user login activity, data access, and basic usage events. However, in an Infrastructure as a Service offering, where the customer is responsible for much of the security of the environment, they can build and automate workflows and builds from the ground up and require a lot more data.

AWS and Azure have services that allow you to collect log data from cloud native services, as well as from application created logs. These services provide dashboards that show trends, metrics, and can perform some level of querying. They also have alerting and eventing to support automations based on triggers from logs.

Much of the AWS and Azure log collection is focused on cloud operations: dashboarding CPU utilization percentages, used disk size, and application failures. We will use some of these logs to help tell a story of actors attacking, manipulating, or pivoting inside your environment.

CloudWatch Logs (1)

CloudWatch is AWS's log collection and analysis service. It is jammed packed with all kinds of features:

- Log collection and search
- Customizable, but limited, dashboards
- Automated event responses
- Event bus that ties to third-party services
- Docker container analysis from Container Insights
- Performance analysis through Contributor Insights
- Web URL testing from canaries

CloudWatch is not a replacement for a full featured SIEM. It does not have the slick monitoring displays, or easy to build alerts as you get with the more expensive options. However, it does have valuable features that organizations should be taking advantage of.

As with most of these kinds of services, it benefits from being so tightly connected to the AWS Services. CloudWatch dashboards might be limited in its visualizations, but dashboards can be built and deployed through the same CloudFormation stack or Terraform provisioning as the resources it will monitor.

Another big advantage is its ability to read in logs, identify a set of logs of interest, and then execute SNS, Lambda, ECS, or any number of triggered responses. CloudWatch can really be the center of event detection and automation.

- Collect and search across logs from AWS services, or from custom endpoints.
- Dashboards are limited and may not solve large scale monitoring. It is handy, however, to combine infrastructure deployment and a dashboard together in an Infrastructure as Code deployment.
- Event rules, metrics, and alerts can be used to monitor for a series of logs that match a filter and send alerts or automate an action. These logs could also be CloudTrail logs, which makes it easy to build custom reactions to API calls.
- The EventBridge is a service to connect CloudWatch events to other third-party services that may reside in different accounts. This is particularly helpful for security teams to pull logs back to a centralized location where they can be alerted.
- There are a number of "Insight" services in AWS that help customers understand how Docker, Lambda, and other services are operating by collecting, aggregating, and summarizing metrics and logs.
- Contributor Insights provides a view of the top contributors impacting system performance based on log analysis.
- Canaries and Synthetics help monitor web applications and detect broken links, page errors, and latencies in UI assets.

Although very operations focused, we can use the log analysis and response services to help detect security concerns as well as bad behaving applications.

CloudWatch Logs (2)

CloudWatch Log Collection:

- Log groups can have a set retention period between 1 day and 3,653 days. The default is "forever."
- Logs can be streamed to Lambda and AWS Elasticsearch cluster. However, this could be costly.
- Logs could be exported to S3 on a regular basis. Good option for storage.

Log groups are created for every Lambda, ECS container, and other services. They can get out of hand.

CloudWatch Logs are split into "log groups." Each log group should contain all the same format or type of log. All CloudTrail logs could be in a single CloudWatch log group. All NGINX connection logs would be in the "nginx" log group. Each log group has a "log stream." Each stream is typically a source of the log. In the "nginx" log group, there will be a stream for each EC2 that the data originated from.

In a highly elastic environments, log streams can spin up and spin down at a moment's notice. Analytics would look across all the log streams in a particular log group. However, diving into a particular log stream would allow you to see every log in order it was collected.

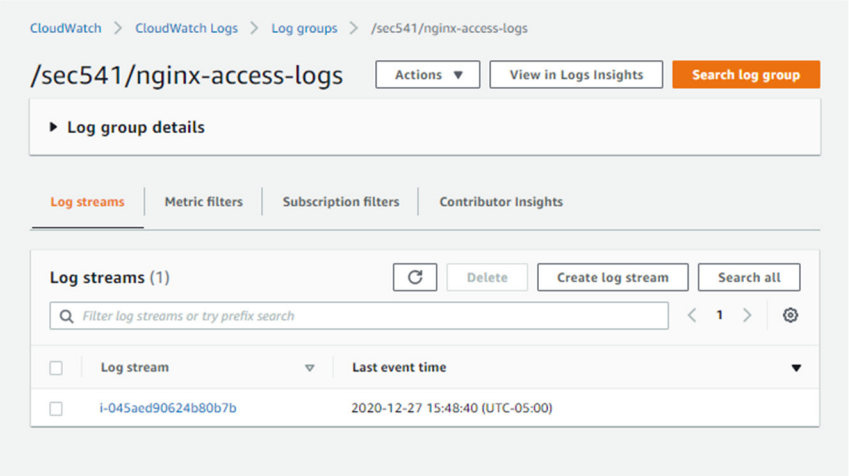
Log groups can be pre-created or will be created when a log shows up to be stored. However, pre-creating a log group has an advantage: the "retention period" can be set. If your organization's policy is to store NGINX servers from the marketing website for 30 days, then create the log group with a 30-day retention period. The retention period can *only be* one of the following days: 1, 3, 5, 7, 14, 30, 60, 90, 120, 150, 180, 365, 400, 545, 731, 1,827, or 3,653. AWS handles deleting the individual logs as soon as the retention period has expired.

Log groups without a retention period are defaulted to "forever." After operating an environment for a while, you may see a number of these from other AWS services such as ESC containers or Lambda functions. It's a good idea to go through and update retention periods on occasion. Logs can be streamed to Lambda or Elasticsearch, but that could be a lot of data. Understand import and data transfer costs: having a Lambda function wake up and perform an action on every single log in a log group might not be cost effective.

CloudWatch Logs (3)

A log group should contain logs of a particular type.

The log stream would be logs from a specific endpoint.



Log groups are a collection of log streams. Normally, the logs in a log group should come from the same kind of resource, or the same kind of log type, and should follow the same formatting., such as NGINX application logs from all the marketing web apps, or maybe all the reverse proxies across the organization. These could be in the same region or span across all the regions and be from multiple accounts.

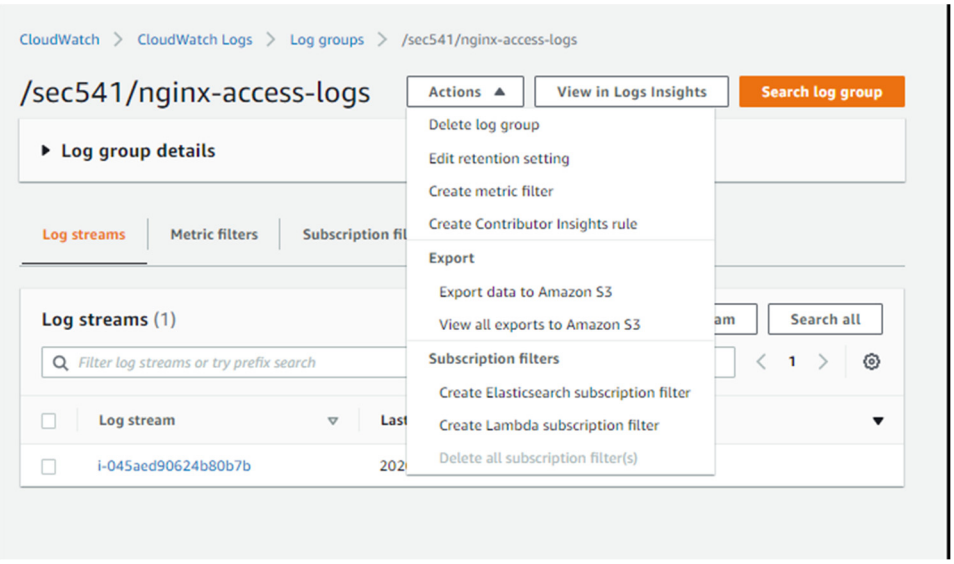
A log stream is a collection of logs, normally from the same exact endpoint.

As an example, a fleet of NGINX servers are all deployed from the same AutoScaling group. There is a single log group for all these NGINX logs. Inside the log group, will be individual log streams—one for every EC2 instance. In a world where EC2 instances grow and shrink daily, the number of log streams will grow.

Reference:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html

CloudWatch Logs (4)

- Edit retention
- Create a metric
- Contributor Insight
- Export data
- Real time subscriptions



From this console, we can do a lot with the log group.

Delete log group: The log group and all logs will be deleted. However, if the services are still sending logs, it will be recreated. It's a good idea to ensure the logs are available in an S3 bucket in case of accidental deletion.

Edit retention settings: Change how long AWS will store your CloudWatch logs before deleting them automatically. Ensure that your organization's data retention policy is enforced here. No data retention policy? Good time as any to start one.

Create metric filter: CloudWatch metrics will monitor events *across* CloudWatch logs. A metric filter is really a way to count activities in logs: https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/MonitoringPolicyExamples.html

Create Contributor Insight rule: CloudWatch Contributor is used to identify activities that are unique, top N occurrences, total unique contributors, and usage. Find top talkers, most fails, or heaviest instance usage.

For instance, in VPC Flow Logs: Top Sources by Rejected TCP Connections. The Contributor Insight is looking for instances that stand out. But you can define what "stand out" means: https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContributorInsights.html

Export data to Amazon S3: One-time export of data, across a date range, to a particular S3 bucket. This is a good option if you are investigating, and you see something suspicious and you want to preserve the data. The ACL on the bucket must be created ahead of time. S3 buckets could be in the current account, or a different account, such as the Security Organization account.

View all exports to Amazon S3: Any time the data was exported to S3 buckets, you can see when those exports happened.

Subscriptions: Create real-time processing of log data with subscriptions. We can use subscriptions to get access to a real-time feed of log events form CloudWatch logs and have it delivered to other services such as Amazon Kinesis, Kinesis Data Firehose, Lambda, or other systems.

Logs are base64 encoded and gzip compressed[1].

[1] https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Subscriptions.html

CloudWatch Logs Insights (1)

CloudWatch Logs Insights enables search and analysis of CloudWatch logs:

- Includes query language specifically for log searching
- A "query" can be saved and re-run later
- Insights can "discover" log fields in some common log types
- Query up to twenty log groups with single request
- Works best if data is preformatted in JSON

A giant collection of CloudWatch logs can be difficult to search through or query. In the lab, we used *jq* to manipulate the JSON data that was returned to the CLI; however, you saw that it was a bit clunky.

AWS provides the CloudWatch Logs Insights tool as part of CloudWatch, a user interface that makes it easier to query across logs[1].

The CloudWatch Logs Insight Query syntax is a stripped-down query language to craft the queries. The query language lets you control what is displayed, filtering based on fields matching regular expressions or performing basic mathematical comparisons, performing aggregate statistics on the logs, and operating on data inside of unformatted strings[2].

CloudWatch Insights is able to parse logs that are VPC Flow Logs, Route 53 logs, Lambda logs, CloudTrail logs, and JSON formatted logs[3]. Other logs will require just in time parsing as part of the query syntax.

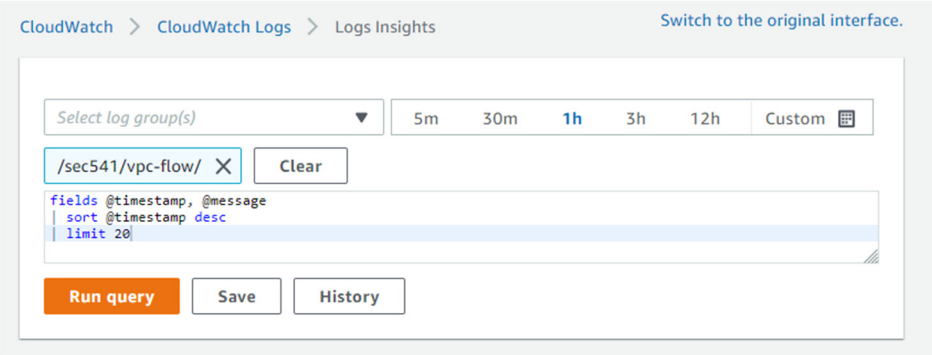
Query up to 20 log groups with a single request. This makes determining how you break down log groups and streams very important. Suppose that you are collecting NGINX web server logs across all your web servers. You could have all NGINX logs go into a log group called NGINX. Maybe you put the all your marketing web server logs in one log group and all the shopping cart logs in another. You can search just in the marketing logs, or you could search across both log groups.

For NGINX, we can send the data in a delimited string format, or in JSON. The JSON data will be significantly bigger and take up more space. However, the string format will require every query to include a parser directive so that the query understands how to evaluate the data. It is much easier to have all the logs in JSON formation ahead of time. Compare the data storage costs and make that determination. Honestly, data storage is fairly cheap compared to compute costs.

[1] https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/AnalyzingLogData.html
[2] https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax.html
[3] https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_AnalyzeLogData-discoverable-fields.html

CloudWatch Logs Insights (2)

Query language supports:
- Filtering with regex
- Sorting
- Basic statistics
- Limiting rows
- Parsing fields



The query language is specific to CloudWatch Logs Insights and does not resemble SQL at all. Querying and filtering on typed log messages, such as JSON, is a lot easier

The query language only has six main commands. Each set of commands is piped to the next command, similar to how we use jq. Here are the six main commands in the query language:

Display: Tells the query which fields to display as the results.

Fields: Retrieves the specific fields from log events to be passed to the next stage of commands. An action can also be performed on the fields and passed to the next step.

`fields concat(Operation, '-', StatusCode) as opStatus` <- Will concat the fields *Operation* and *StatusCode*, with a hyphen, and output it as opStatus

Filter: We usually do not want all the logs, but only those log events that meant some filter criteria. Filter will use an expression to determine if an individual log will be selected to be used.

filter (*duration* > 2000)

Stats: Calculate aggregate stats based on log field values. Some of the supported operators include `sum()`, `avg()`, `count()`, `min()` and `max()`.

stats avg (f1) by f2

Sort: Sort the log events in ascending or descending order. Great to use in conjunction with stats.

fields f1, f2 | sort f1 desc

Limit: Many times, we only want to see a small number of log results. Especially if we want to see top five. When developing queries, the course authors usually only return five, just to make sure the query works as expected.

Parse: A handful of log types are pre-parsed, and the specific fields in the message can be used. Or the log entry is JSON, and Insights can automatically discover the fields. However, the vast majority of third-party logs will have some file format that will be impossible for Insights to discover the field types. Therefore, the **parse** command must be used. It extracts data from the log field and creates specific fields that can be processed. Glob expressions and regex expressions can be used.

For the log line: 25 May 2019 10:24:39,474 [ERROR] {foo=2, bar=data} The error was: DataIntegrityException

Glob: parse @message "[*] * The error was: *" as level, config, exception

Regex: parse @message /\[(?<level>\S+)\]\s+(?<config>\{.*\})\s+ The error was: (?<exception>\S+)/

Note: There is always a debate about where to transform data for integration and usage. Transform the data at the data generation point or transform at the ingestion point. Telling a log generator (NGINX) to produce logs in JSON will take up significantly more space but will make ingestion and analysis super easy. It's a tradeoff you have to decide on.

References:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax.html#CWL_QuerySyntax-regex
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax-examples.html
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax.html

CloudWatch Logs Insights (3)

Insights will detect log properties for AWS log type

- VPC Flow logs
- CloudTrail logs
- Lambda
- Route53

It will also auto detect JSON data.

The parse command can be used to give data structure for the query.

The CloudWatch Logs Insights dashboard provides a number of tools to make it easy to make use of the results of a query.

The histogram gives a quick view of how the logs were spaced out in time. This could help identify anomalies or hot spots. Drag and drop across a section of the histogram to re-run the query across a different timeframe.

Export results: Export the table in either CSV downloadable file, CSV to clipboard, or Markdown. Great for including in a report.

Add to dashboard: Adds the results to a CloudWatch dashboard for visualization.

Visualization: If the query uses stats, then it's easy to create a visualization of the results. Similar to CloudWatch Dashboards, the visualization can be a pie chart, line chart, stacked area, or bar chat. This is a quick and easy way for the threat team to spin up some queries and graphs for their reports.

Reference:
AWS Discoverable Fields:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_AnalyzeLogData-discoverable-fields.html

CloudWatch Best Practices

CloudWatch logs best practices:

- There is no global retention period for log groups
- Watch out for infinite log group retentions
- You cannot query Insights across accounts
- AWS is implementing other "Insights" products to help you get meaning from the vast number of logs being collected, including Container and Contributor

If data retention is needed, make sure that you either:

1. Create every single log group through a CloudFormation template that has a retention period set. This is unlikely to get right every time. So, also do the second step.
2. Run a Lambda function when a CloudWatch log group is created, and if the log group does not have a retention period, set a standard amount. If you normally do step 1, then alert someone if a CloudWatch log group was created improperly.

Insights is only for querying in a single account. However, CloudWatch could send logs to your account. So, a security team could get a copy of all logs from across the organization.

The Container Insights and Contributor Insights are designed to help you make sense of the vast amount of data being collected.

Container Insights focuses on Amazon Elastic Container Service (ECS), AWS Fargate, and Amazon Elastic Kubernetes Services (Amazon EKS). The CloudWatch logs are collected, including metrics information. The insights extract important meaning from this data, such as restart failures.

Contributor Insights analyzes log data and creates time services data that displays information to help you find resources that are operating out of the norm of everyone else: hosts that are having CPU/memory issues, heavy network users, or URLs that are generating the most errors. Using this to tap down on systems or custom-made applications that are causing trouble or costing money is a great feature for system management. Strange-acting applications could also potentially lead to a threat operating on your environment, such as a Bitcoin miner using all the resources. This is a good example of an activity that can be good for saving money and improving security.

References:
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContainerInsights.html
https://docs.aws.amazon.com/AmazonCloudWatch/latest/monitoring/ContributorInsights.html

Azure Monitor

Azure's answer to all things monitoring:

- Operational troubleshooting using **Service Health**
- Collect data from cloud native and other resources
- Analyzing log data using **Log Analytics**
- Define alerting options
- Depends on agents and/or **diagnostic settings** for integration

Workbooks provide a customizable dashboard for analysis and reporting

Similar to AWS's CloudWatch service, Azure has a very similar service called Azure Monitor. This service has a lot going for it:

- Troubleshooting service health
- Collecting data from many types of resources, both in and outside the Azure environment
- Analyzing data once it is ingested into this service
- Aiding in automated alerts based upon user-provided queries which run at regular intervals

To begin incorporating Azure Monitor into your cloud and on-premise environment, integration efforts must be made as none of what you are about to discover is enabled by default! It really comes down to two methods:

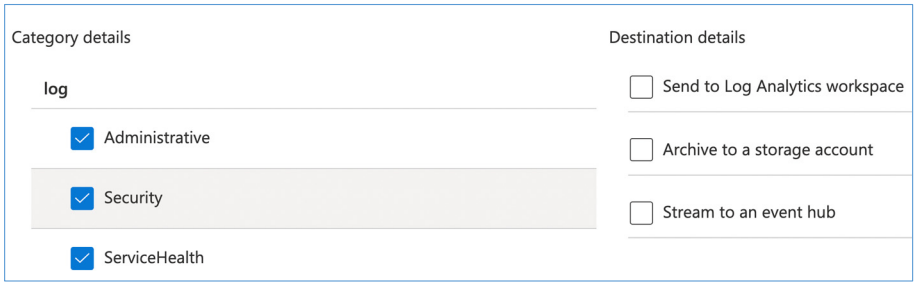
- Install an agent on a platform (e.g., Windows operating system, Linux operating system)
- Configure the Azure service's diagnostics settings

We will explore the latter at this time but stay tuned for the agent-based approach in an upcoming course module.

Diagnostic Settings

Diagnostic settings for each service and cloud resource control which data is collected and where that data is sent

- Categories can range from simple metrics to service interactions to security events
- Can output to:
  - Log Analytics workspace
  - Azure storage account
  - Stream to an event hub



When configuring diagnostic settings on a per resource basis, one must first define which categories of data to collect. This can range wildly from operational data (e.g., metrics, health checks) to some categories a security team may be more interested in (e.g., Security, FrontdoorWebApplicationFirewallLog).

There is very detailed documentation showcasing each category and the expected data within Microsoft's documentation of that service. For instance, see the following documentation for the Azure Front Door service's available metrics: https://docs.microsoft.com/en-us/azure/frontdoor/front-door-diagnostics.

Once it is determined what data categories to capture, a decision must be made as to where to send this data. There are three options: send the data to a Log Analytics workspace, archive the data in an Azure storage account, or stream the data to an Azure event hub. We will discuss that latter two options later in class, but we will now explore the Log Analytics service and what it has to offer.

Azure Log Analytics

Used to query available Azure event data

- Scopes can be used to make searches more efficient
  - Subscription-wide would be less efficient, but capture more data
  - Resource-specific would be very narrow, but more efficient

Queries written in **Kusto Query Language (KQL)**

- Common operators and functions include:
  - **where**
  - **extend**
  - **project**
  - **sort**
  - **count()**
  - **extractjson()**
  - **render**
  - **summarize**

Once the data is arriving at a Log Analytics workspace, there is a handy interface to sift through this data much more efficiently than what is possible by reviewing the data in the native cloud service (if it is even available for viewing in the cloud service in the first place). The data can be processed within a specific scope which must be defined by the user prior to issuing their first search.

To effectively filter and parse this data, one must understand the Kusto Query Language (KQL). There are many operators and functions that can be used to select, process, and present this data that an analyst may be trying to identify. Some of the more common options include:

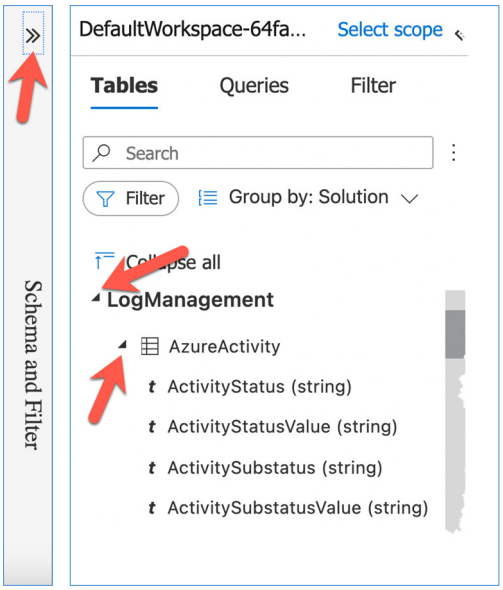
- **where**: Narrow down the search to specific fields
- **extend**: Add more fields based upon specific criteria or another function's output
- **project**: Determine which fields to display in the results
- **sort**: Sort the data once the results are found
- **count()**: Simply count the results within a summarization group
- **extractjson()**: Many fields may have nested JSON values, so this function will extract the user-requested nested JSON value from a particular field
- **render**: Used to create visualizations (e.g., bar chart, pie chart, table)
- **summarize**: Produces a table that aggregates the input table's content

The full list of options can be found here: https://docs.microsoft.com/en-us/azure/data-explorer/kql-quick-reference.

Tables

When data is sent to Log Analytics, it is placed in a **table**

- Tables can be found by expanding the **Schema and Filter** pane
- These table names are useful when narrowing down search results
- By expanding the table, you can explore the available fields to filter on and view



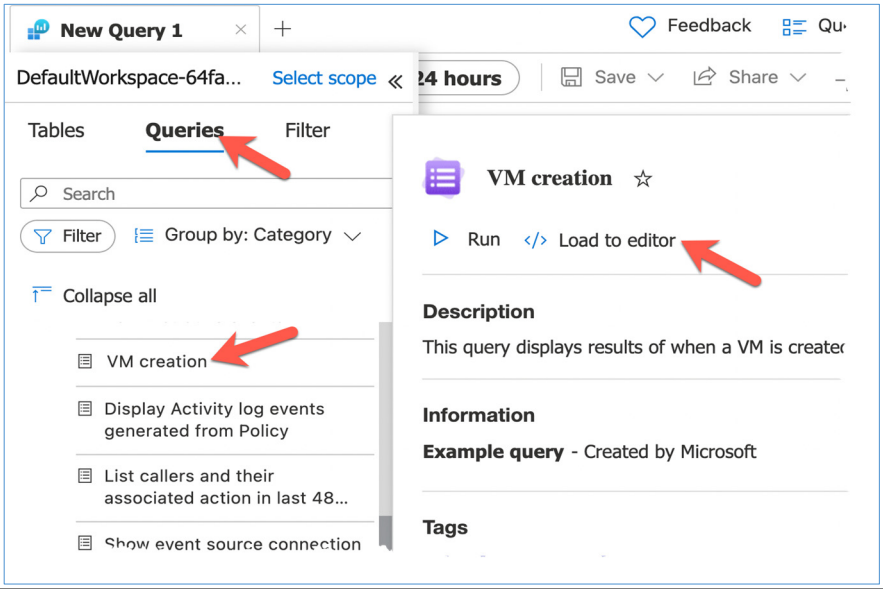
Once the diagnostic settings are configured to send the data to Log Analytics, the data is placed into a table. These tables are typically the first item called out in the query when analyzing data in the Log Analytics service. So, with that, it is important to understand which tables contain the data of interest.

You can see the available tables and schema by expanding the Schema and Filter pane and drilling into the various categories and tables to see their structure.

Sample Queries

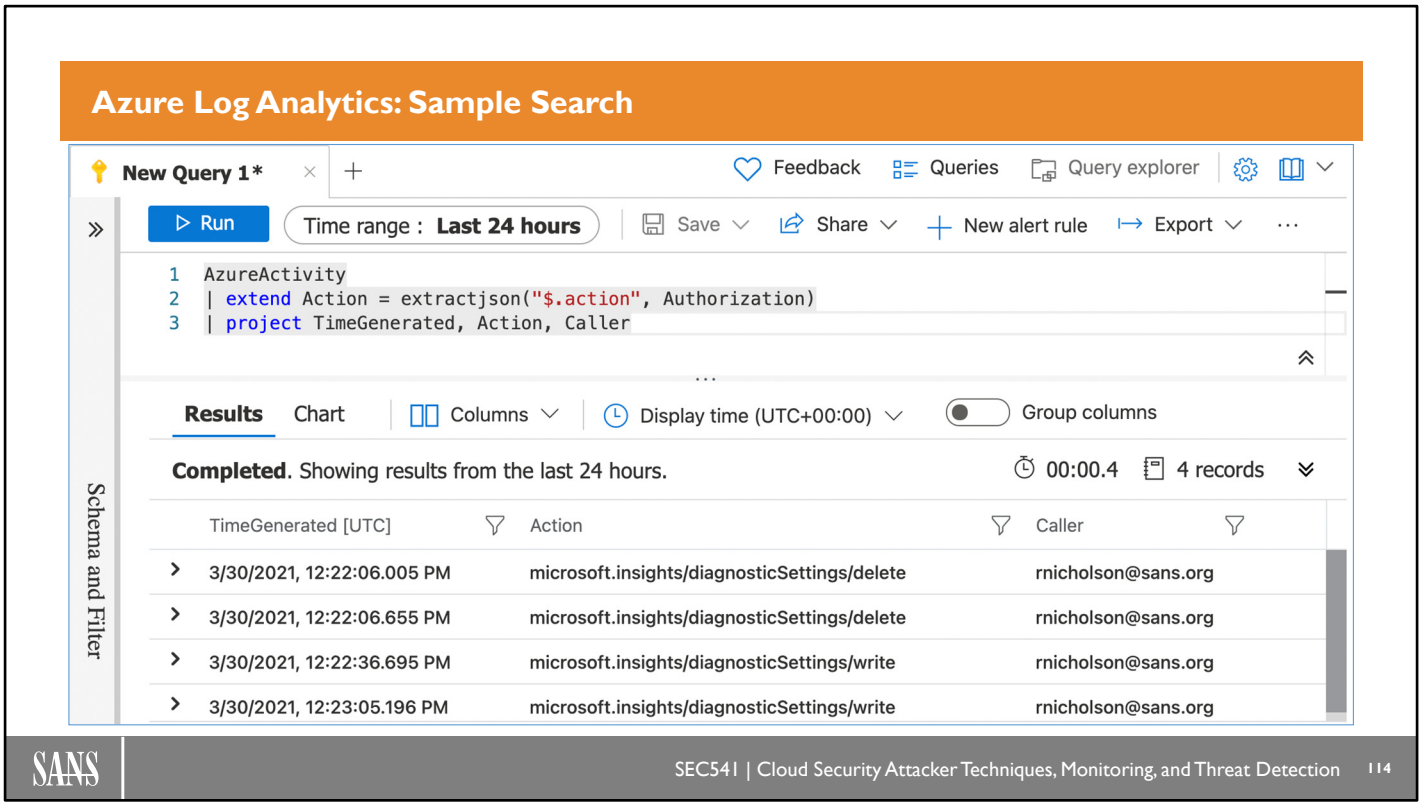
There are many pre-built queries which can be found in the left pane of the Log Analytics workspace

- Run "as-is"
- Use as templates and modifying as needed



Azure provides many commonly used queries that you can leverage by clicking on the Queries tab. Once a query is found, you have a few options. You can elect to just use it as written to immediately obtain results by clicking on the Run option, but the results may not be tailored to your liking.

If the query needs modified, you also have the option to load the query in the editor where you can make any necessary adjustments to the query.



If we were to look across all of the Azure Activity log for actions taken, who conducted those actions, and what time those actions took place, we could easily build a KQL query to display this within the Log Analytics workspace. We must first, however, understand the data we are working with. Acquiring the time (`TimeGenerated`) and username (`Caller`) is quite easy. However, the action is actually a nested JSON name/value pair within the `Authorization` field, so we must get a little creative.

Above is a complete search using some of the previously mentioned functions and operators. First, though, a table must be identified. In this case, since the Azure Activity log data is written to the `AzureActivity` table, that will be the first line in the query. Afterward, each operation will have a pipe character (|) separating them.

The challenge that was just brought up (extracting the `Action` from the `Authorization` field) is performed on the second line by using one operator **`(extend:`** to create a new field to later be displayed) and one function (`extractjson():` to extract the action field—the result will be set to the new `Action` field).

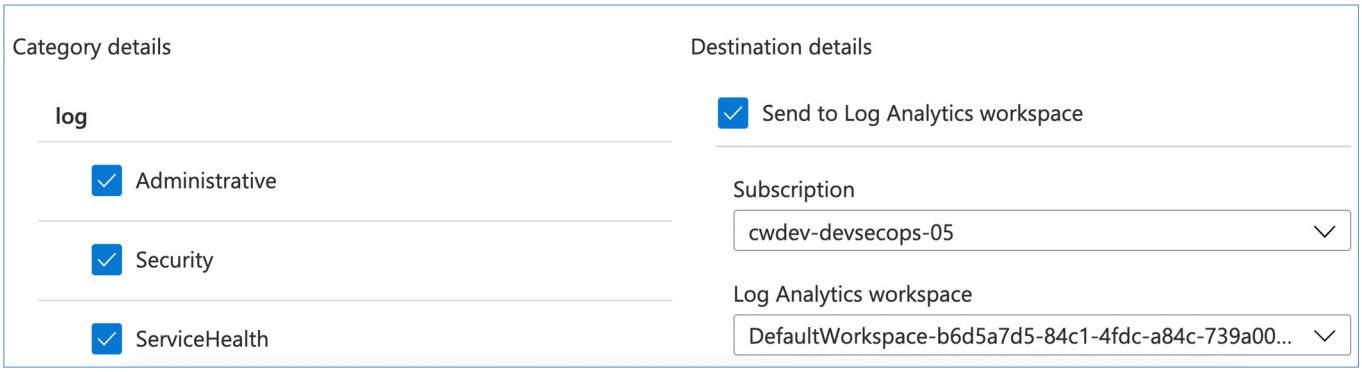
Finally, to present just the data of interest, **project** is used to filter out everything except the `TimeGenerated, Action`, and `Caller` fields.

It appears that the data is sorted in ascending order, so if that is not the most useful for this particular use case, **sort** may be a good option to add (which we will see in use shortly).

Revisiting Azure Activity Log

Searching is quite rigid in the Activity log service

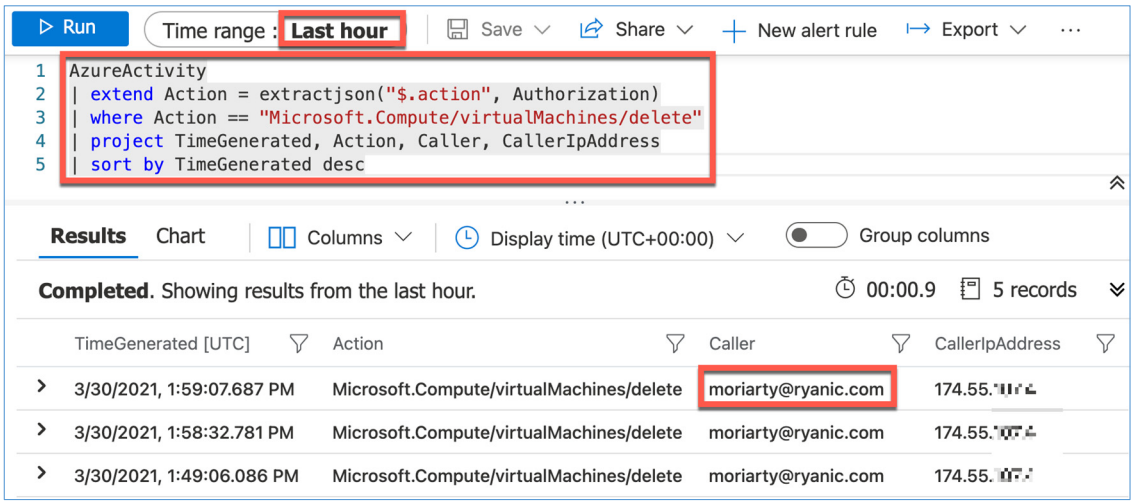
- It may be more advantageous to send data to a **Log Analytics workspace** to allow more granular searching



By navigating to the diagnostic settings of any Azure resource, you will find the options to select the category and where they are to be shipped to. In this example, we are going to revisit the previous issue where a rogue user in an environment deleted a VM. In this case, however, the Activity logs will also be sent to a Log Analytics workspace instead of relying solely on the Activity log service and its less flexible search options.

Conducting a KQL Query

Finding the deleted VM (again)



Although much more complex, you can see the necessary query to home in on the activity we are interested in. The query begins by choosing the appropriate table (`AzureActivity`) and then, on line 2, the `Action` is extracted from the `Authorization` field as we saw previously.

From there (line 3), the results are limited to only those actions that are equal to Microsoft.Compute/virtualMachines/delete—signifying the removal of a VM.

Lines 4 and 5 are used to first limit which fields are presented to the results pane (of course you can always expand a result to get the entire event details) and then to sort the data by a descending time frame (newest actions first).

With this, we again see that the `Caller`, moriarty@ryanic.com, is the one behind this activity, as well as the source IP address (`CallerIpAddress`) the activity was initiated from.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. **EXERCISE: Parsing Logs with jq**
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Lab 1.3 | Parsing Logs with jq

Exercise Duration: 20 Minutes

Objectives

In this lab, we will return to our CloudTrail logs, leveraging the power of jq to improve filtering, transformation, and outputs

- Researching jq
- Understanding JSON
- Querying the CloudTrail data with jq

In this lab, we will go back to CloudTrail, but this time we will use *jq* to further extract the data.

You will first look at *jq* and the capability it brings.

You will then look at the JSON data format itself and how it is structured. This is important to be able to use *jq*.

Last, you will perform more complex filtering and JSON manipulations with *jq*.

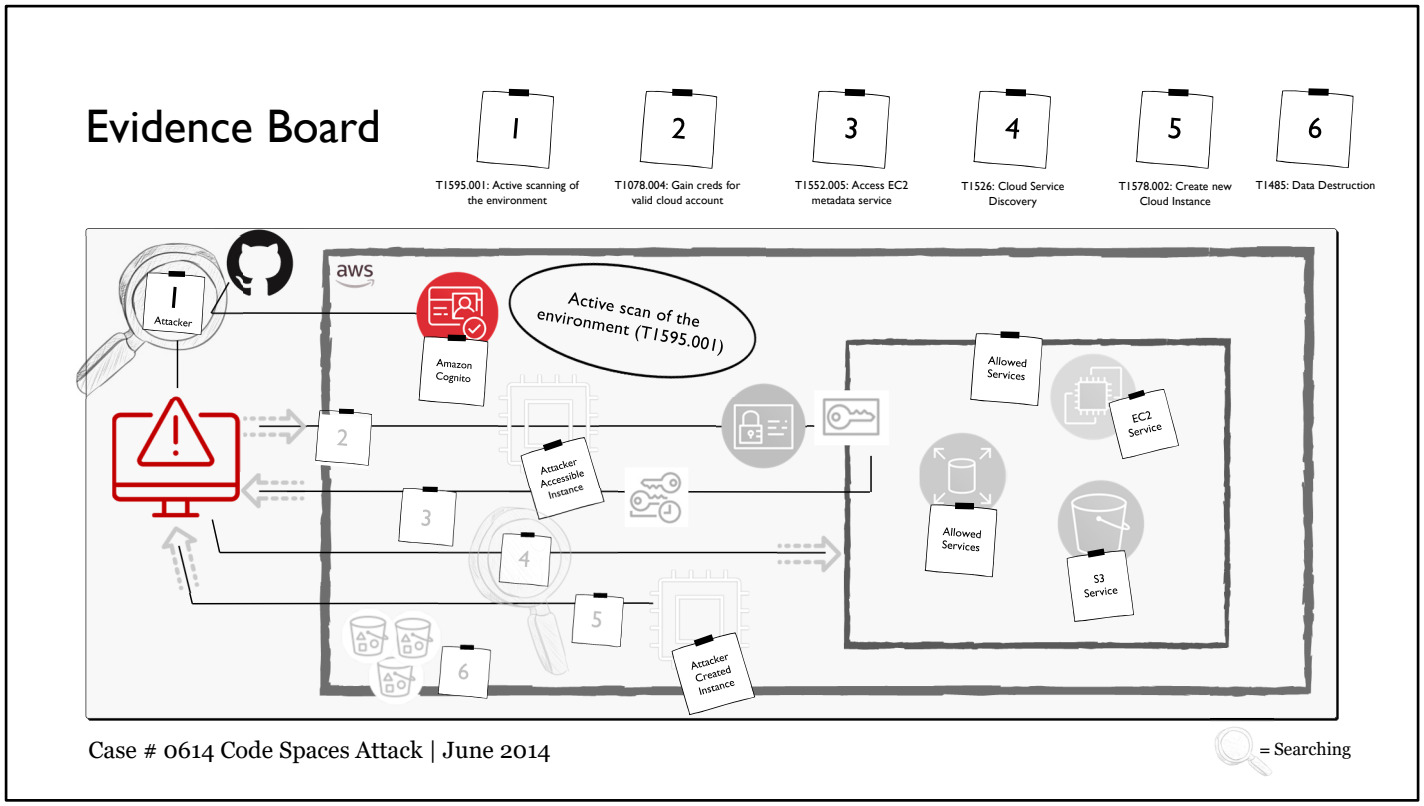
Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. **Network Flow Logging**
11. Capturing Raw Network Traffic
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.



Case # 0614 Code Spaces Attack | June 2014

As we talked about earlier, most initial attack vectors against a cloud environment happen in one of three ways:

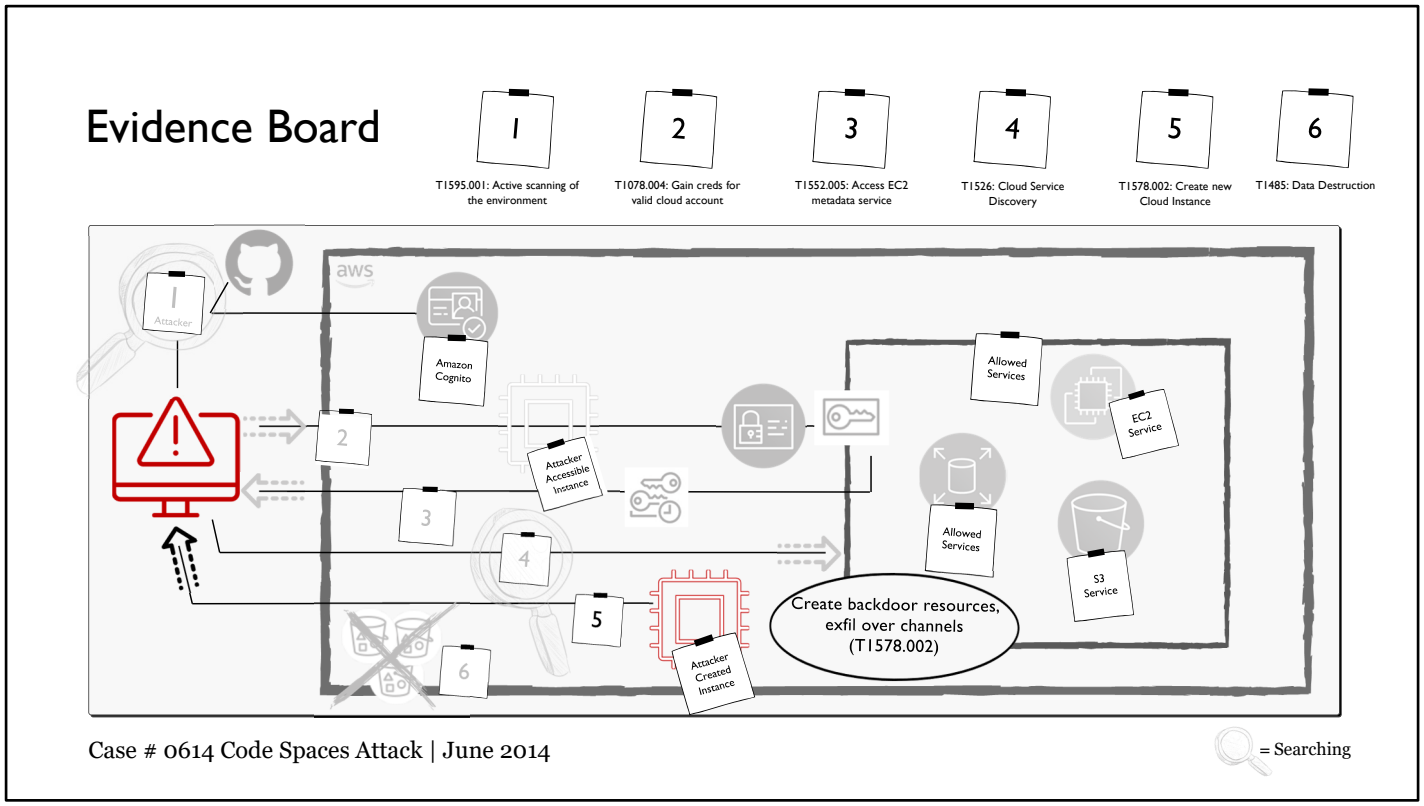
- Attacker found or discovered credentials through brute force, guessing, or creds found publicly
- Attacker found an unconfigured or default access vector
- Attacker found a vulnerability in an application or other customer-built service

There are other attack vectors that could be more effective, such as adding a backdoor to a shared resource or moving laterally from a trusted environment. But, in this case, the attacker discovered some credentials that gave them significant access to the AWS environment.

There is a whole host of attackers just looking for a quick entry to an environment, ransom access, and then to make a quick buck (or Bitcoin).

Attackers are scanning your environment all day every day, looking for a way in. Automated bots diving into GitHub repos, active brute-force attacks against login dashboards, or maybe even attackers harvesting passwords that have leaked, and might be reused. Brute-force attacks against public infrastructure will happen constantly, alerting and investigating on every instance of a brute-force attack may be over-kill. But, knowing that if an attacker got in, you may want to see what traffic was hitting your environment around the time of entry. Not all alerts need immediate investigation.

References:
https://krebsonsecurity.com/2021/05/a-closer-look-at-the-darkside-ransomware-gang/
https://tldrsec.com/blog/lesser-known-aws-attacks/



In the case of Code Spaces, the attacker spun up some backdoored services, maybe a virtual machine. They likely added their own SSH access keys, giving them another access point in case the owners discovered and shut down the attacker's access point. That appears to be what happened when the attacker saw the company attempting to shut down their console accesses. Remember, with Identity Access Management, you can grant a compute service with all the privileges of an admin, if you have the right accesses. The account that the attackers gained access to likely had significant privileges granted to them.

We have talked about gathering API logs from AWS and Azure, and querying against application logs, but what about network traffic? Network traffic collection and analysis may help you detect an attack, but it may be a better tool for use in investigations. If you know the attacker is using a specific IP address, maybe through investigating the cloud API logs, then look at all network traffic to see where else the attacker has been. You may be able to investigate their access into the environment, or maybe a backdoor resource they are suddenly SSH'ing into.

Let's see how AWS and Azure help us with collecting network data.

Reference:
MITRE ATT&CK T1578.002: https://attack.mitre.org/techniques/T1578/002

Cloud Network Visibility

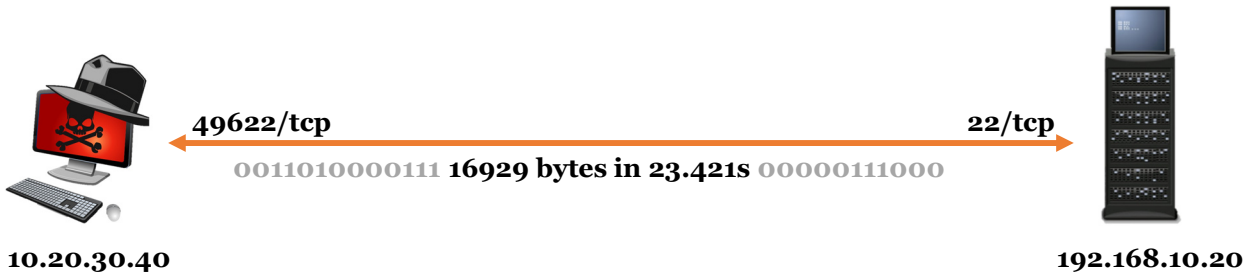
- Shifts from on-premise to cloud-hosted services likely inhibit network visibility
- Many vendors provide **flow logging**
  - Similar to NetFlow
  - Records metadata about network connections
- **Packet data** can also be provided by some vendors
  - When metadata is not enough, **raw packets** may be analyzed
  - Raw network data is collected and sent to a listening VM
  - Requires subscribing to yet another vendor service/capability

One major difference discovered early on when organizations were shifting their workloads to the cloud is losing access to network data. Cloud providers, without subscribing to additional services, remove the capability to observe the raw network traffic within the entire VPC as you can do on-premise by deploying taps or configuring switches to mirror traffic to a listening device.

Without recording the network traffic or deploying network security applications on every instance or VM within a VPC, network data required for proper incident analysis is lost.

Luckily, some cloud vendors provide the capability to learn more about the network traffic through metadata or, in some cases, give the customer visibility to the raw packets traversing the VPC.

Flow Logging



Sample Flow Record

src_ip: 10.20.30.40	**src_port**: 49622
dst_ip: 192.168.10.20	**dst_port**: 22
protocol: TCP	**in_bytes**: 3496
duration: 23.431s	**out_bytes**: 13433

The first technique that analysts may utilize to inspect network events is flow logging. Flows are events consisting of network metadata like:

- Source and destination IP addresses
- Source and destination ports
- Transport protocol
- Start time, end time, and duration of the communication
- Sent, received, and total bytes sent between source and destination

Armed with this data, analysts can determine many things by asking themselves the following questions:

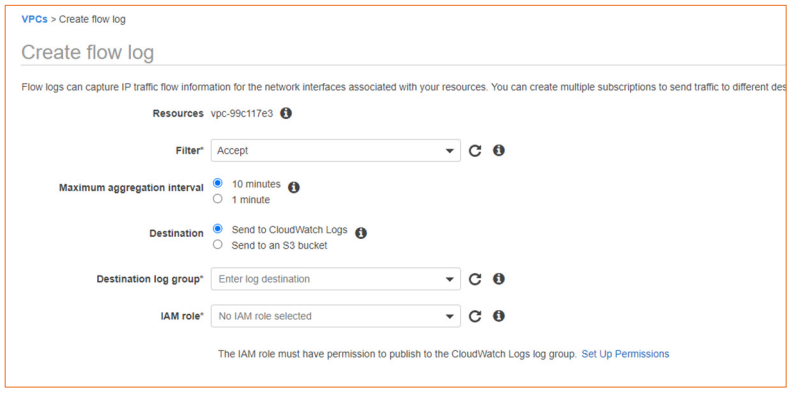
- Is this traffic normal in my environment?
- Is there a larger number of sessions than usual?
- Is there a much larger amount of data sent over these channels than usual?
- Did this connection last much longer than a typical connection over this channel?

Notice one thing in common with these questions: the analyst must know what normal looks like in their environment. It may take some time after enabling this functionality to discover anomalous network behavior.

VPC Flow Logs (1)

VPC Flow Logs is a feature that let's you capture information about IP traffic going to and from your network interfaces in your VPC

Released June 2015



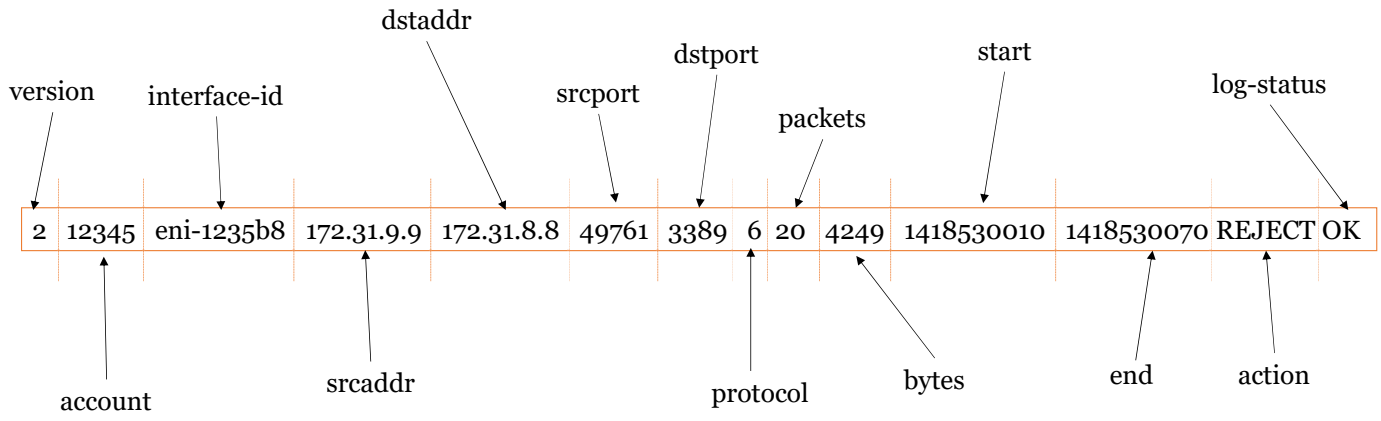
Every resource in your AWS account that has an IP address has a network interface inside your VPC. In an on-prem network, you likely would have deployed some type of network traffic collection system at a gateway switch, behind your DMZ, or other point of your environment.

AWS gives you access to some of the network traffic data through VPC Flow Logs. You create a flow log that directs the data to a bucket for further analysis.

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

VPC Flow Logs (2)

VPC Flow Logs shows metadata about network traffic.



A "flow" of network traffic is stored in a single log that provides metadata about the communication without the raw data itself. By default, a VPC Flow Log has thirteen fields.

`Version:` The VPC Flow Log version 2 is the default version. Versions 3 and 4 contain additional information about the flow that you might find helpful. This "version" number will likely be 2 unless you specified a custom log.

`Account:` The AWS account ID of the account that collected this data. This is most helpful when sending VPC Flow Log data back to a centralized security account from multiple accounts.

`Interface ID:` Each virtual network interface has a resource number that starts with "`eni-`". The Elastic Network Interface (eni-) could be used to query AWS and retrieve the EC2 or resource name, if that resource is still alive. In an elastic environment like AWS, that EC2 might be long gone, and it will be difficult to figure out what the heck this flow log is for.

`Source Addr and Destination Addr:` The IP addresses involved in the IP communication. Like the interface ID, the IP addresses might change regularly.

`Source Port and Destination Port:` The only way to know what *kind* of TCP/IP communication is happening is via the port numbers. SSH communication typically happens on port 22. Does it have to? No way. You could specify a completely different port. However, do not do this for legitimate deployments. It can cause a lot of confusion.

`Protocol:` Assigned Internet Protocol Number—6 for TCP, 1 for ICMP. You should see a lot of Protocol = 6.

`Packets:` This log entry shows a flow that took 20 packets. The total number of packets can be interesting if they are very large or very small. Basically, you could use statistical analysis (machine learning?) to identify communications that are out of the ordinary.

`Bytes:` The number of bytes transferred during the flow.

`Start/End:` The time, in Unix seconds, when the first packet was observed and when the last packet was received.

`Action:` If the traffics was permitted (ACCEPT) or not permitted (REJECT) by the security groups or network ACLs.

`Log-status:` The logging status of the flow log.

References:
https://www.ssh.com/ssh/port
http://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html#flow-log-records

VPC Flow Logs (3)

Default logs include "version 2" of the log format.

Version 3 includes:

- VPC ID, Subnet ID, Instance ID
- Tcp-flags (SYN, SYN-ACK, FIN, RST)
- Type (IPv4, IPv6, EFA)
- Also includes pkt-srcaddr and pkt-dstaddr, which is the IP address for the original src and dest of traffic

Version 4 includes: region, az-id, and "sublocation" information, which is useful for wavelength, outpost, or localzone

Version 5 includes: AWS services, flow direction, traffic direction options

By default, a flow log will contain version 2 data (as seen on a previous screen). However, updates to flow logs gave users the ability to put additional data into the VPC Flow Log. This additional data really helps when correlating data across logging environments or separating data when logs from multiple accounts and regions come back to a single security resource.

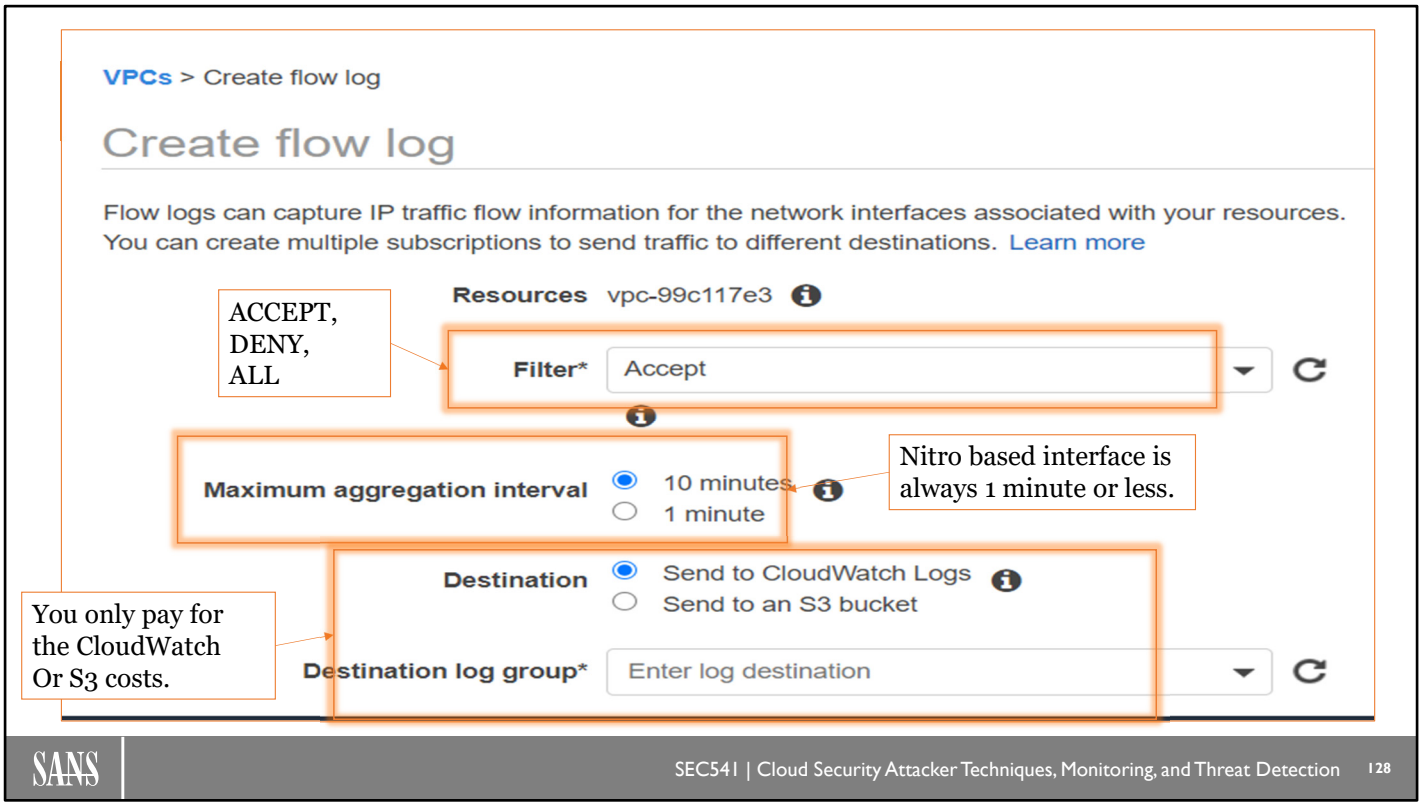
Version 3 includes additional data about the instance, subnet, and VPC about the flow. The instance ID could be specifically helpful if correlating data, such as CloudTrail logs.

Version 4 expands to regions, availability zones, and supports AWS's outpost, wavelength, and localzone deployments.

One thing to note: we are focusing on VPC Flow Logs for threat detection. However, VPC Flow Logs could be great tools to troubleshoot deployment and operations problems. Turning on a flow log specifically for a particular instance for troubleshooting would be a great way to dial in and focus on a problem system.

Keep in mind, VPC Flow Logs are traffic in one direction. The return packets are in a different flow log. Version 5 of VPC Flow Logs tries to help make sense of that direction.

Reference:
https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs.html#flow-log-records



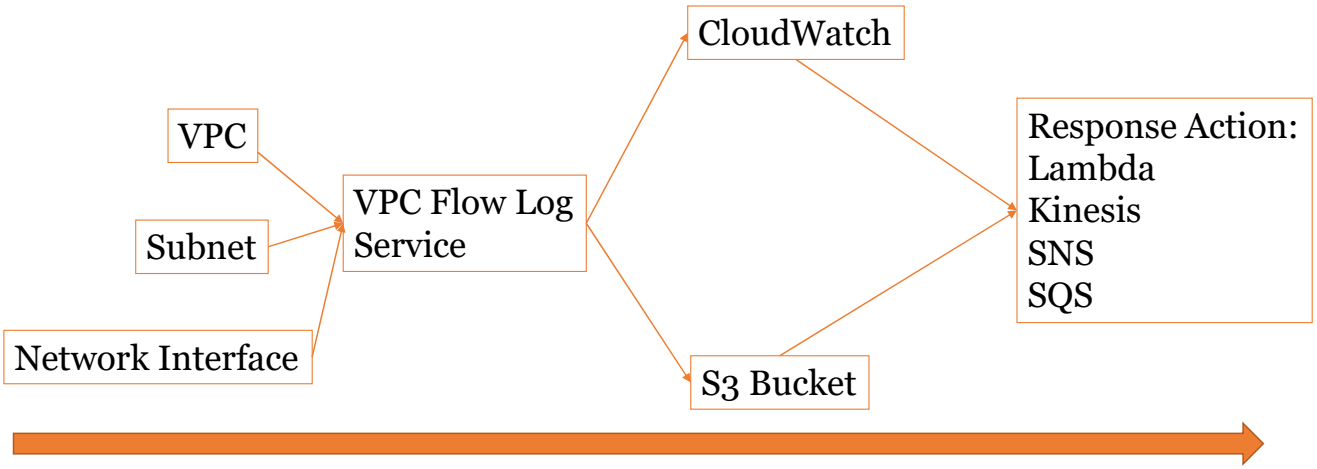
In order to gain access to the VPC Flow Logs, you have to set up a flow log. In CloudTrail, we could just go to the CloudTrail dashboard and see 90 days of logs. Not so with VPC Flow Logs.

A flow log only has a view of parameters to deal with:

- The “filter” can let pre-filter data based on the “action” field. Data not blocked by security groups or network ACL would be "ACCEPT." Data that was dropped by security groups or network ACL would be “DENY.”
- Destination lets you determine if your flow logs are going to be gathered up and dropped into an S3 bucket or CloudWatch log. Sending to S3 buckets can allow for longer term storage. Sending to CloudWatch can allow you to create response actions. Maybe both?
- Maximum aggregation interval is the period of time during which a particular flow is captured, and a log record is created. Flow logs do not appear in your S3 bucket or CloudWatch in real time. An “aggregation interval” of 10 minutes means that data will be gathered up to 10 minutes before shipping off to CloudWatch or S3. Then, it can take additional time for the services to publish—up 5 minutes for CloudWatch logs and up to 10 minutes for S3.

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

VPC Flow Logs (4)



This is just a diagram to show how you could chain together AWS services for VPC Flow Logs.

Although the AWS Web Console's VPC Flow Log wizard will create a VPC Flow Log for an entire VPC, it is possible to select one or more ENIs or even a whole subnet to collect from. It makes sense that you might want to have your private and public subnet data in separate flow logs. Or maybe you create ENI specific flow logs during an investigation of an EC2. Anything is possible.

The VPC Flow Log service really just ships flow logs, based on some basic filtering, to *either* an S3 bucket or a CloudWatch log. Not both. If you needed both, you could create two VPC Flow Logs.

CloudWatch and S3 can perform events that might launch a Lambda function, make an SNS notification, or other services in an event driven architecture.

Remember, after the aggregation period, CloudWatch logs could take up to five minutes to publish and S3 Logs could take up to ten minutes to publish.

VPC Flow Logs (5)

VPC Flow Logs are sent to either CloudWatch or S3 Buckets

S3 Bucket:
- Objects are compressed in gzip
- Each gzip file contains all flow logs collected from the aggregate interval period
- Hard to analyze, easy to archive

CloudWatch:
- Each log flows into CloudWatch individually
- CloudWatch provides filtering and display tools
- CloudWatch is great for building metrics/rules, not so great for archiving

VPC Flow Logs in S3 will be compressed in gzip format. Each object file contains all the flow logs collected from the aggregate interval period. This is great for storing data to support compliance. But it makes it difficult to analyze across the gzipped files. We can always use Lambda to open each file, look for a particular event, and then react to it. However, this is just a clunky way to go about it. As we will see in the next section, the AWS Athena service allows this S3 analysis with ad hoc queries.

VPC Flow Logs to CloudWatch logs can be sent to AWS CloudWatch. We will talk about CloudWatch in the next section, but it is a service log collection across multiple services. CloudWatch also supports an "event rule" that can be run on those logs. For real time detections, with known and scriptable rules, CloudWatch is a good service.

You could also send logs to S3, and then pass them to CloudWatch or vice versa. AWS provides lots of ways to move logging through the environment. You will need to take into account data storage, computational costs, and network transfer fees. Another consideration is how close to "real-time" your analytics must be. As a general statement, real time costs more than delayed.

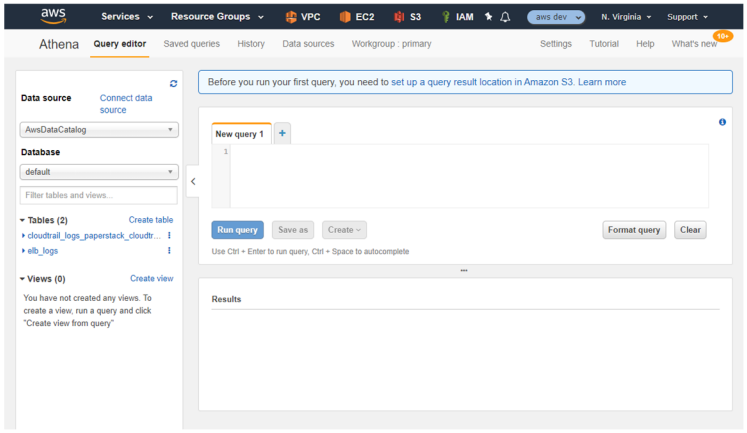
Reference:
https://docs.aws.amazon.com/vpc/latest/userguide/flow-logs-s3.htm

Athena (1)

Athena is an interactive query service that let's you do SQL against data in S3.

Released November 2016

- **Serverless, pay per query, could be costly**



Amazon Athena is an interactive query service that makes it easier to use standard SQL to analyze data across an S3 bucket. You pay per query and the S3 storage, but it is considered serverless.

Athena is built on top of S3 and AWS Glue Data Catalog but automates the data cataloging. If you know the format of the data, and understand the SQL language, you can start running queries quickly.

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

Athena (2)

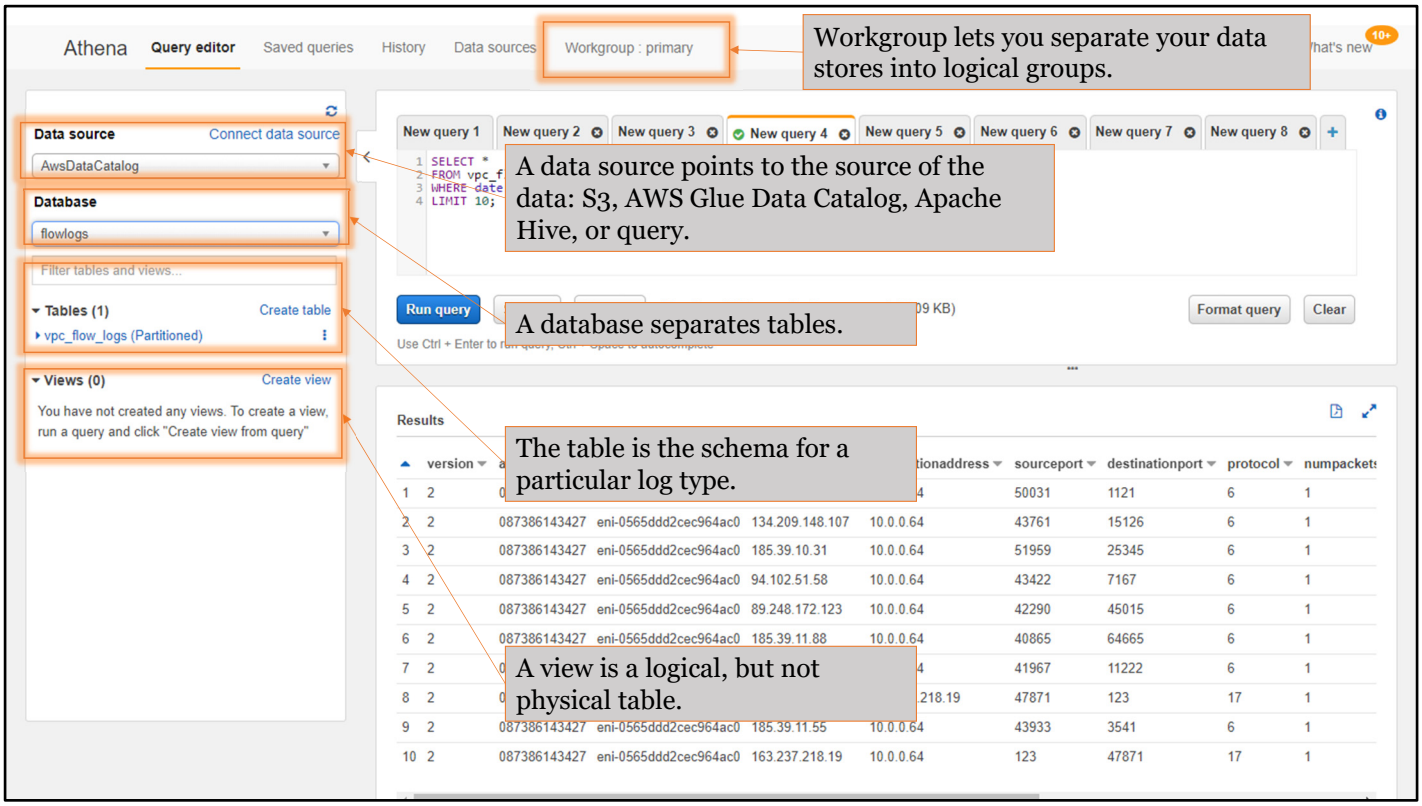
Athena makes it easier to run ad hoc queries against well formatted data in S3 or other data sources.

- The data should be in an ORC, JSON, CSV, or Parquet
- A "Table Schema" creates an Athena "table" that describes the schema for the target data
- Run a query against this table, and the data will be output as a CSV file in another S3 table
- You pay $5 for each TB of data scanned. So, be careful about how much data is in your table. We can partition the data to help bound the queries.

Athena can investigate inside those gzip compressed files we created in S3 that make up VPC Flow Log files. The files need to be of ORC, JSON, CSV or Parquet format so that Athena is able to interpret them properly. VPC Flow logs are space delimited CSV file format. Most logs generated from the applications in our environment could be pulled into Athena for querying. As with most query services, the real power is to analyze across different, but related logs to get a full picture.

Athena does not know the format of a VPC Flow Log, so a "Table Schema" will need to be created. That Athena schema is used to create a virtualized "SQL Table," from which you can run SQL queries.

You are paying about $5 for each TB of data scanned. That is $5 per TB per query. This could really add up. Athena is good for ad hoc query of data in discovery, but likely an expensive option for all your analytic needs.



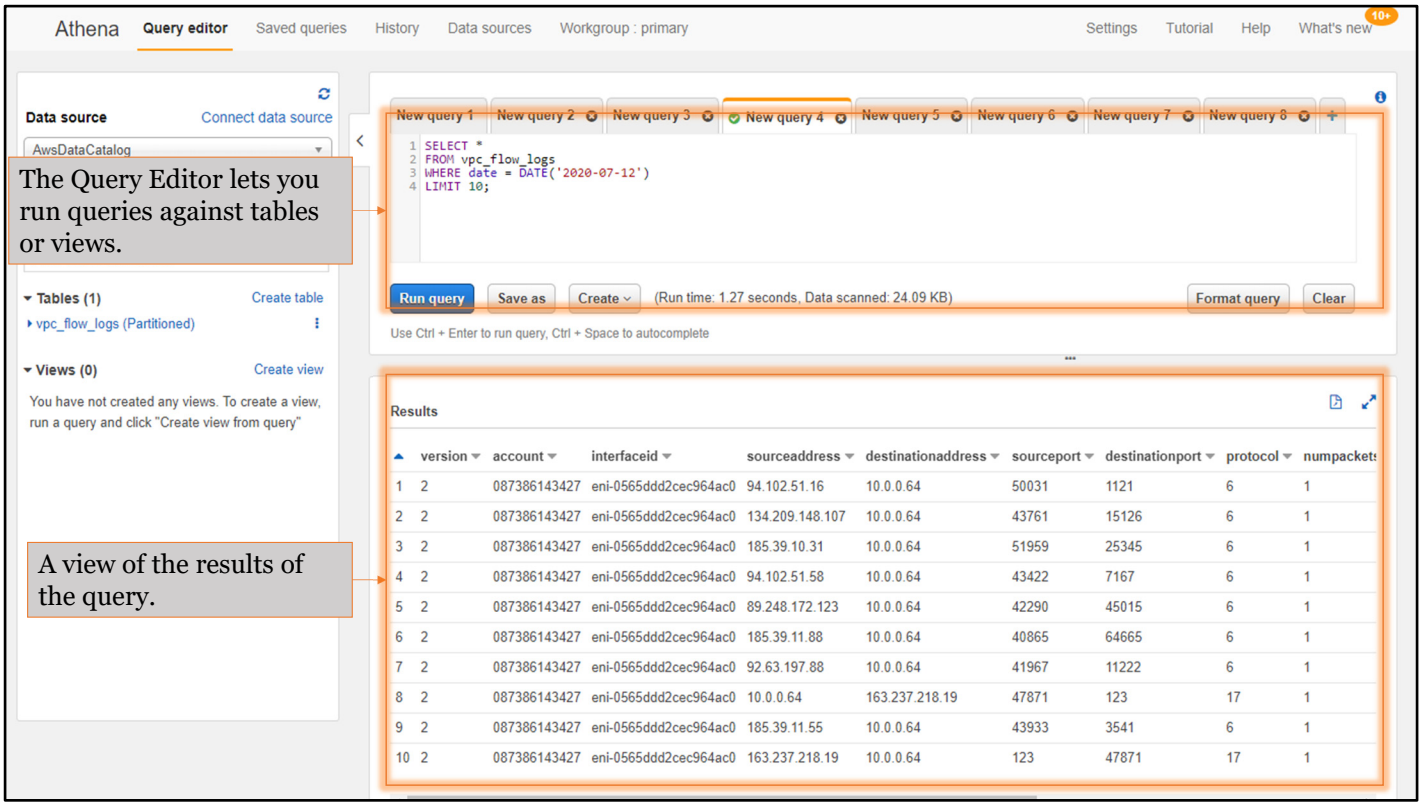
A workgroup is used to separate workloads, identify where the query results will go, and help with cost control. Workloads can be used to categorize by purpose, owner, or environment and allow teams to query the same data store differently. Tags are assigned to a workgroup, allowing organization to setup IAM policies with conditions based on those tags. AWS uses a "requester pays buckets" that lets you run operations in your environment but charge another account. So, if you are on the security team and you are asked to run a particular expensive query by the marketing team, you could have Amazon charge the marketing team for the storage of the data. The workgroup doesn't change the data sources, databases, or table schemas.

A data source is made up of the location of the data, and the metadata catalog that will contain the schema for the data. By default, the AwsDataCatalog data source will query S3. Other data locations could be CloudWatch, DocumentDB, Redshift, DynamoDB, or other databases.

Once the location of the data is identified in the data source, and the work group is set up, then we can describe the data schema. Tables and databases are containers for the metadata definitions that create the schema for the data source. A table describes the schema of the source data. A database is a collection of tables. You can create tables and databases using HiveQL data definition language (DDL): CREATE TABLE, CREATE DATABASE, DROP TABLE. It is also possible to create a table based on a query.

A view is a logical table that is based on an Athena table. A view is best used to reliably query a table for a subset, combined from multiple tables, or an altered format of a table. The view is easier to query, hiding some of the complexity of a particularly involved query. A view is best used when you want to separate the underlying structure of a query from the code that is calling the data. For instance, we could create a view specifically for SSH traffic that queries VPC Flow Logs and only returns ones that show SSH traffic. The view could be called "SSH_view." If the code (or us) wanted to query the SSH traffic, the command would be SELECT.

References:
https://docs.aws.amazon.com/AmazonS3/latest/dev/RequesterPaysBuckets.html
https://aws.amazon.com/about-aws/whats-new/2019/02/athena_workgroups/
https://docs.aws.amazon.com/athena/latest/ug/work-with-data-stores.html
https://docs.aws.amazon.com/athena/latest/ug/understanding-tables-databases-and-the-data-catalog.html
https://docs.aws.amazon.com/athena/latest/ug/views.html
https://docs.aws.amazon.com/athena/latest/ug/understanding-tables-databases-and-the-data-catalog.html



The Query Editor lets you type your SQL Query with some source highlighting. Athena supports a subset of the Data Definition Language and Data Manipulation Language statements, functions, operators, and data types based on HiveQL and Presto.

When a query is run, the results appear below in the Results pane. These results are just a visualization of the CSV data, stored in S3 as a result of the query, and displayed for easy viewing. There is some basic column ordering available.

Reference:
https://docs.aws.amazon.com/athena/latest/ug/ddl-sql-reference.html

Athena (3)

A table is a schema that describes the data that can be queried.

```
CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs (
    version int,
    account string,
    interfaceid string,
    sourceaddress string,
    destinationaddress string,
    sourceport int,
    destinationport int,
    protocol int,
    numpackets int,
    numbytes bigint,
    starttime int,
    endtime int,
    action string,
    logstatus string
) PARTITIONED BY (
        `date` date
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' LOCATION
's3://s3_bucket/AWSLogs/087386143427/vpcflowlogs/us-east-1/' TBLPROPERTIES
("skip.header.line.count"="1");
```


This is the query we would run to create the table `vpc_flow_logs`. Each column from the VPC Flow Logs is identified here: `version, account, interfaceid, sourceaddress, destinationaddress, sourceport, destinationport, protocol, numpackets, numbytes, starttime, endtime, action, and logstatus.`

The table is partitioned by the "date" field. New partitions will have to be created every day.

The "ROW FORMAT DELIMITED FIELDS TERMINATED BY" tells Athena how to read the VPC Flow Logs.

Athena (4)

Partitioning data lets you restrict the amount of data you scan when running a query.

- Partitions improve performance and reduces cost
- Partitioning can be multi-level, allowing you to query against different partitions of the same data
- Partition the data by any key

```
ALTER TABLE vpc_flow_logs
ADD PARTITION (`date`='2020-07-12')
location
's3://s3_bucket/AWSLogs/123456789012/vpcflowlogs/us-east-
1/2020/07/12';
```


Partitioning data is almost a must for properly Athena usage. Otherwise, a year worth of data might accidently be queried at once really costing you some money. It's always best to query for exactly the data you need, but it's not always apparent what that data might be.

Partitioning by date is one of the most common for security-based logs. Multi-level partitioning schemes could allow logs to be partitioned by year, month, date, and hour. Only query by the year partition if you really mean it.

Partitioning requires a SQL command against a table. If partitioning by date, then it's necessary to run the partition command hourly.

CloudTrail Partitioner is an open-source tool that will help set up a data partitioning job.

References:
https://docs.aws.amazon.com/athena/latest/ug/partitions.html
https://duo.com/blog/introducing-cloudtrail-partitioner

Athena (5)

```
SELECT *
FROM vpc_flow_logs
WHERE date = Date('2020-07-12')
LIMIT 10
```

In this query, we are returning all columns from vpc_flow_logs that were generated on 2020-07-12.

SQL statements typically follow a pattern. Those patterns can be more complex as you start to get more comfortable with the data. It can be helpful to limit returns so that you don't accidently create a crazy expensive and time-consuming query or get the wrong data.

`SELECT *` = SELECT defines which columns to return. In this particular case, we are returning all columns in this table.

`FROM vpc_flow_logs` = Identifies which table to query this data from.

`WHERE date` = Date('2020-07-12'). A "where clause" is used to identify what subset of the data to return. In this case, only return the data from this particular date. This data is partitioned by day, in order to manage the queries.

`LIMIT 10` = It's usually best to return a subset or rows so as not to overwhelm the query and return. WHERE will limit based on some knowledgeable data, but LIMIT 10 just takes the first 10 rows and dumps the rest.

Athena (6)

```
SELECT sourceaddress, count(*) as total
FROM vpc_flow_logs
WHERE date = Date('2020-07-12') AND
(destinationport = 22)
GROUP BY DISTINCT sourceaddress
LIMIT 10
```


This SQL Statement is a bit more complex. Instead of selecting everything with SELECT(*), we are only going to get two columns. `Sourceaddress` is the IP address of the source of the particular VPC Flow Log entry. "`count(*)`" is a "select expression" that will count the number of rows in a group and just display the total. The "as total" will rename the column to be "total." What is a group? We have to look down a couple of lines to the "`GROUP BY`" clauses.

The "`GROUP BY`" will divide the output of the select statement into rows where "`sourceaddress`" matches. Therefore, we will get only one row for each distinct source address.

References:
https://docs.aws.amazon.com/athena/latest/ug/select.html
https://docs.aws.amazon.com/athena/latest/ug/presto-functions.html

Azure NSG Flow Logs

Captures similar data to AWS VPC Flow, such as:

- IPs, ports, and protocols used
- Timestamps of network communication
- Traffic decision (was it allowed or denied?)
- Flow state
- Amount of network traffic (bytes and number of packets)
- Geolocation of external systems

Configured in either the **Azure Network Watcher** or **Azure NSG** services

A very similar option to record network metadata in Azure is NSG Flow Logs. A big difference here is that, unlike AWS where you would enable their network flow feature within the VPC service, you enable it within either the Azure Network Watcher or Azure NSG service since the perspective is placed upon the Network Security Group, not the Virtual Network itself.

Once this feature is enabled, familiar data will begin to be collected, such as the usual network metadata (IP addresses, ports, and protocols that are communicating), timestamps of the start and end of flows, indicators of whether or not the NSG allowed or denied the traffic, the state of the traffic (did it just begin, is it a continuation, or did it end?), how much data was sent in either direction, and even the geolocation of the systems if Traffic Analytics is enabled (more on this later).

Azure NSG Flow Log Configuration

- Which **version**?
  - Version 1 does not include flow state whereas version 2 does
  - **Flow states** indicate when flows begin, continue, and end
- **Retention** options:
  - 0 (forever) or 1-365 days
- Where to store the data?
  - **Storage Account**
  - **Log Analytics Workspace**
    - Required when choosing to conduct **Traffic Analysis**
    - Determine how often to process this data (every 10 minutes or every hour)

There are few considerations when enabling NSG Flow Logs. First, you must decide which version to enable. There are two to choose from: version 1 and version 2. The advantage of version 2, and why the course authors would recommend this be enabled over version 1, is that flow states are included. In other words, was this flow in this record brand new, a continuation of an older flow, or is it the end of the flow?

Next is to decide how long to retain this data. Flow data adds up, so price can certainly play a factor here—especially in busy networks. Not only price, but how far back would you need to look? There is no answer that will suit all needs, but something that must be considered to both minimize cost but not lose valuable data too soon that may be useful during an incident.

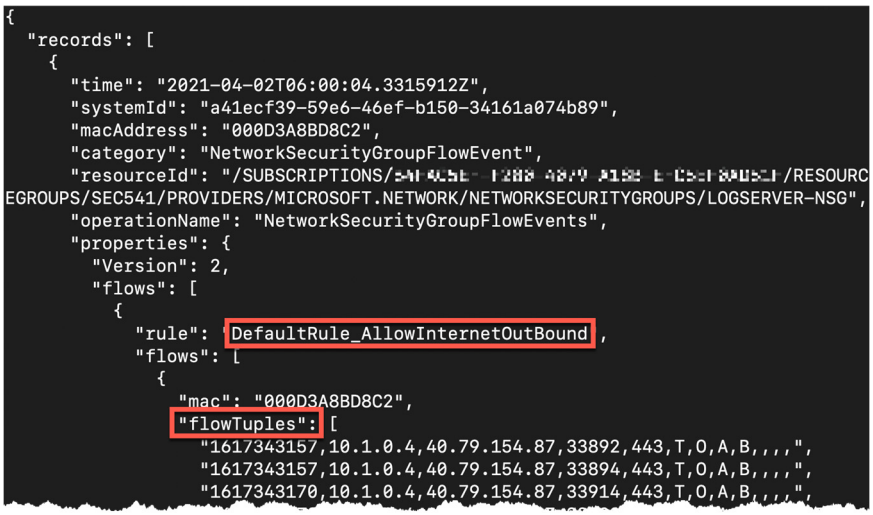
Finally, where is the data stored? This question is partially answered for you as you cannot enable NSG flow without storing it in a pre-created Azure Storage Account. Beyond this is the ability to enable what is called Traffic Analysis. This is a very powerful service which we will see very shortly, but when this is enabled, it must report to an Azure Log Analytics Workspace. Also, when Traffic Analysis is enabled, you can select how often the analytics are conducted—either every 10 minutes or every hour (the default behavior).

Flow Records in Azure Storage

Not easy to process and read

- This is even *after* **jq** made this much prettier to look at

Most of the valuable data is in the **rule** and **flowTuples** fields



Once more, the flow data will always be written to Azure Storage, so you can elect to retrieve the data yourself manually or have your analysis tools or scripts pull this data on a routine basis. There is a warning, however, that these logs are buried pretty deep within the Azure Storage account container. For instance, what you see above is a file located in one of the course author's Azure container locations of:

```
insights-logs-networksecuritygroupflowevent/resourceId=/SUBSCRIPTIONS/
  {subscription}/RESOURCEGROUPS/SEC541/PROVIDERS/MICROSOFT.NETWORK/
  NETWORKSECURITYGROUPS/LOGSERVER-NSG/y=2021/m=04/d=02/h=06/m=00/
  macAddress=000D3A8BD8C2/PT1H.json
```

And also, all flow log files will have the same PT1H.json filename!

From here, parsing this data is quite the challenge. The data starts out nicely as it is very clear what the fields and values are representing until the `flowTuples` section. It may appear as random numbers and letters, but this is actually where the bulk of the most crucial data resides, such as source and destination information, timestamps, traffic decisions, and much or what was just discussed. So how can you or any tools you are creating make sense of this?

Flow Tuples

Contain the bulk of the flow information in the following order:

- Timestamp (UNIX Epoch)
- Source IP
- Destination IP
- Source Port
- Destination Port
- Protocol (**T**CP, **U**DP)
- Traffic Flow (**I**nbound, **O**utbound)
- Traffic Decision (**A**llow, **D**eny)
- Flow State (**B**egin, **C**ontinue, **E**nd)
- Packets (Source→Destination)
- Bytes (Source→Destination)
- Packets (Destination→Source)
- Bytes (Destination→Source)

That random-looking `flowTuples` field does have a particular format that must be well understood, so above is exactly the flow information in the order it appears in each entry.

Take this `flowTuples` record, for example:

1617343173,10.1.0.4,40.79.154.87,33892,443,T,O,A,E,12,4075,11,4956

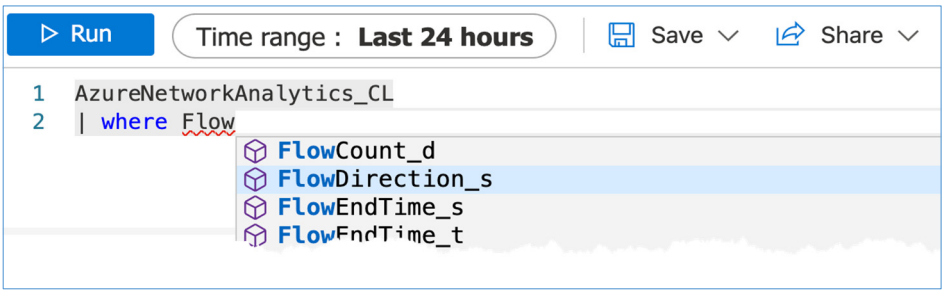
Here is the breakdown:

- Timestamp: 1617343173 (or Friday, April 2, 2021, 05:59:33 UTC)
- Source IP: 10.1.0.4
- Destination IP: 40.79.154.87
- Source Port: 33892
- Destination Port: 443
- Protocol: T (or TCP)
- Traffic Flow: O (or outbound)
- Traffic Decision: A (or allowed)
- Flow State: E (this is the end of this flow)
- Packets (Source→Destination): 12
- Bytes (Source→Destination): 4075
- Packets (Destination→Source): 11
- Bytes (Destination→Source): 4956

Azure Log Analytics to the Rescue

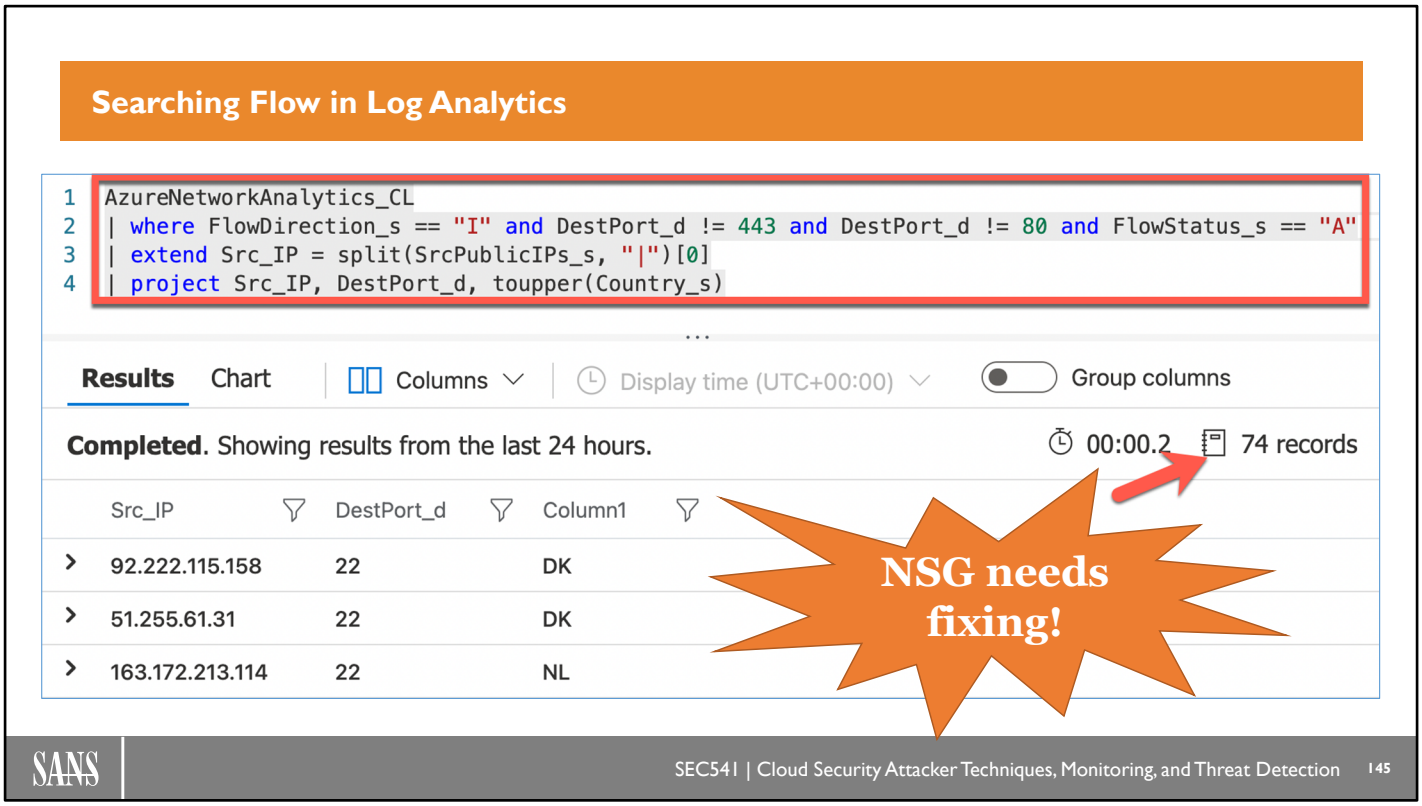
Azure Log Analytics can do all of this parsing for you and make the data much easier to search when **Traffic Analysis** is enabled

- Start a query with **AzureNetworkAnalytics_CL**
- Add a new line with a pipe (|) and start your operator/function
- When in doubt, use **auto-completion** or view the table schema



A lot of that heavy lifting can be done for you just by enabling the Traffic Analysis feature when configuring the NSG Flow Log. This allows you to leverage the Azure Log Analytics Workspace to conduct your searching by drafting a query based upon the newly analyzed fields. But what are those fields? Since Log Analytics provides auto-completion when generating your KQL query, you can see all of the available options and simply make the appropriate selection.

As was mentioned previously in another section, you can also take a look at the schema of the `AzureNetworkAnalytics_CL` table where this data ultimately resides.



Here is a rather simple example to show just how much more powerful and efficient searching this data in a service like Azure Log Analytics can be. This KQL query is looking for any inbound flows `(FlowDirection_s == "I")` where the destination ports are not what we have approved `(DestPort_d != 443 and DestPort_d != 80)` and yet, the traffic decision is that it is allowed `(FlowStatus_s == "A")`.

You may be wondering what the third line is doing. It is simply taking the entry in the `SrcPublicIPs_s` field (e.g., `92.222.115.158|1|0|0|0|0|0`) and grabbing the first IP address before the first pipe character (|) to ensure easier readability later.

It appears that, in the last 24 hours, there are results! In fact, those results show that there is network activity over TCP port 22 which is likely Secure Shell (SSH). Of course, this does not scream compromise as more analysis is needed to see if these sources connect to the end system's management interface successfully, but it is certainly worth raising the issue of incorrect security group settings allowing this type of activity from happening in the first place.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. **Capturing Raw Network Traffic**
12. EXERCISE: Network Analysis with VPC Flow Logs

This page intentionally left blank.

Raw Network Traffic

Flow Logs may not be enough information. We need packets.

- Driven by companies needing/wanting full packet capture, AWS and Azure can collect and deliver network traffic packets
- They can be costly, so consider the right source of packets
  - High risk services or services under investigation?
  - Will the logs be actively reviewed?
  - What information will they provide? Encrypted network traffic is boring to read.
  - How will the data be stored, for how long, and who should access it? Network capture can contain sensitive customer data.
- **Note**: Network costs tend be a surprise. So, start small.

AWS VPC and Azure NSG Flow Logs can be captured, stored, and analyzed using in cloud provided services or sent to your favorite analysis systems. One reason they are so portable is because they leave out a majority of the network data. The headers and metadata can give you some indication of who is talking to who, but now what they are saying.

Azure and AWS are providing customers the ability to tap into the network data itself and make use of it.

AWS VPC Traffic Mirroring and Azure Network Watcher allow you to capture the full network packet, including the data, and direct it to a service that you own, so you may analyze the data. Many larger organizations have built teams around full packet network capture, leveraging the on-prem's large network gateway services to capture data off SPAN ports or through some data capturing service. These expensive and highly functional pieces of hardware were installed in strategic locations in the network to capture all network traveling across.

But the cloud networks can look different and feel different. Systems sitting in public subnets with internet gateway access can talk out through the internet without going through an intermediary device of any kind.

Companies began designing their environments so that all hosts were routing traffic through large VMs that were acting as software-based routers and network capture services. As you can imagine, this added significant complexity to a network that is supposed to be elastic and always changing. Those home-grown systems were usually the source of network problems for organizations.

Larger companies demanded more, so Azure, and eventually AWS, started offering the ability to tap off and extract Raw Network Traffic.

AWS introduced VPC Traffic Mirroring in June of 2019, and Azure introduced Network Watcher in February of 2017.

There are some differences between the offerings. One thing to keep in mind: they can be really expensive with any decent size environment. Full take capture of all your network communications and then have the logs just sit there “just in case” might be too much.

Focused capture on the most high-risk resources or those under investigation? That data might be worth the cost.

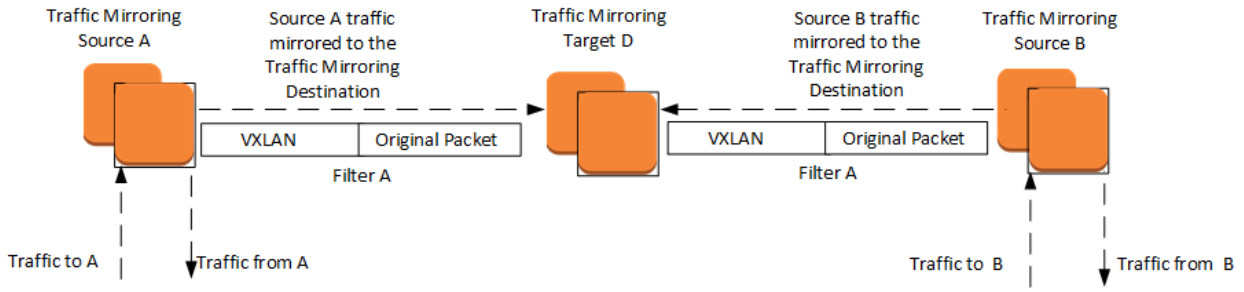
References:
https://aws.amazon.com/blogs/aws/new-vpc-traffic-mirroring/
https://azure.microsoft.com/en-us/blog/announcing-azure-network-watcher-network-performance-monitoring-and-diagnostics-service-for-azure/

AWS VPC Traffic Mirroring (1)

AWS VPC Traffic Mirroring drives traffic from a source ENI to a target ENI.

Packets in Virtual Extensible LAN (VXLAN) encapsulation.

1 or more Sources send data to an EC2 or NLB.



Traffic Mirroring copies inbound and outbound traffic from a network interface (ENI) attached to an Amazon EC2 instance (Source) and sends it another EC2 instance or Network Load Balancer that has a UDP listener (Target).

Until February 2021, only Nitro based EC2s could be a source. However, since February, select non-Nitro instances can be a source.

The traffic source and target can be in the same VPC, or in a different VPC connected via intra-Region VPC peering or network transit gateway. It is a best practice to have a Security account established for your organization that is the recipient of VPC Flow Logs, CloudTrail, AWS Config, and now VPC Traffic Mirroring.

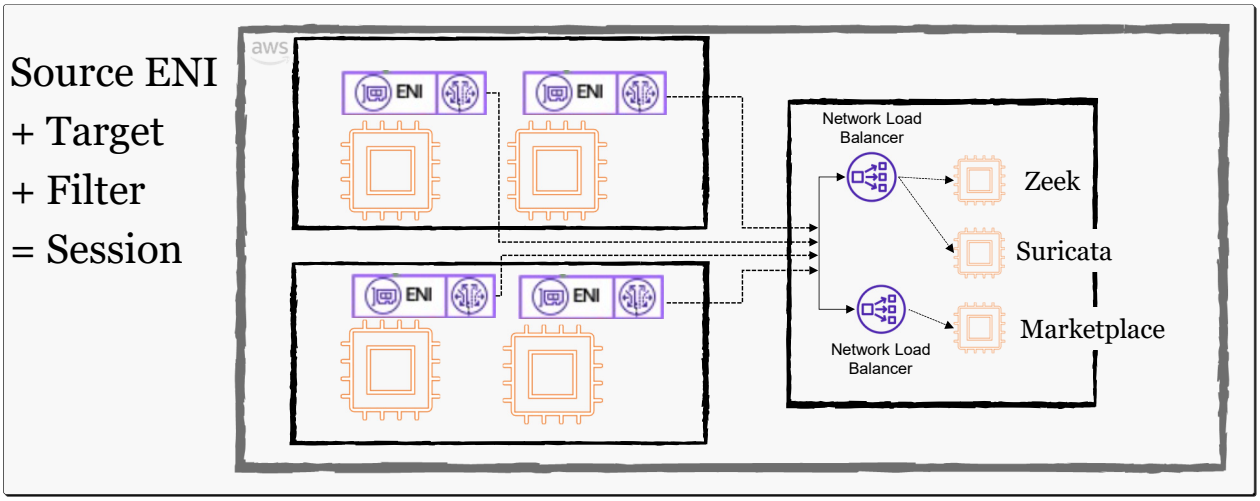
Network traffic sent is in the VXLAN format, a network virtualization technology that attempts to help address scalability problems with large cloud computing. It is VLAN-like encapsulation of OSI layer 2 withing layer 4 UDP datagram.

VXLAN format is documented by IETF in RFC 7348: https://datatracker.ietf.org/doc/html/rfc7348.

References:
https://aws.amazon.com/ec2/nitro/
https://aws.amazon.com/about-aws/whats-new/2021/02/amazon-vpc-traffic-mirroring-supported-select-non-nitro-instance-types/
https://en.wikipedia.org/wiki/Virtual_Extensible_LAN
https://docs.aws.amazon.com/vpc/latest/mirroring/what-is-traffic-mirroring.html
https://docs.aws.amazon.com/vpc/latest/mirroring/traffic-mirroring-how-it-works.html
Packet type: https://docs.aws.amazon.com/vpc/latest/mirroring/traffic-mirroring-packet-formats.html

AWS VPC Traffic Mirroring (2)

A Source is an ENI. Target is always EC2 or NLB.



A "session" is a combination of the Source ENI, Target ENI, and a filter applied. We will talk filters in the next section. Each source can handle up to three sessions.

Open-source resources such as Zeek or Suricata can be set up as the target instances. Always use a network load balancer and autoscaling group in front of your analytic systems. It is a best practice to *always* put a load balancer and EC2 in an autoscaling group even if you only have a single EC2 instance. It makes it easier to maintain those systems.

Remember from the previous page that all the traffic is encapsulated in a UDP packet, so you must configure Zeek, Suricata, or any monitoring service to convert and analyze the UDP packets.

The AWS Marketplace has a number of recommended services that are set up to analyze the VPC mirrored data. Companies such as ExtraHop, Palo Alto, Cisco, Gigamon, Symantec, Corelight, FireEye, and Fidelis have Marketplace products that are ready for installation and operation.

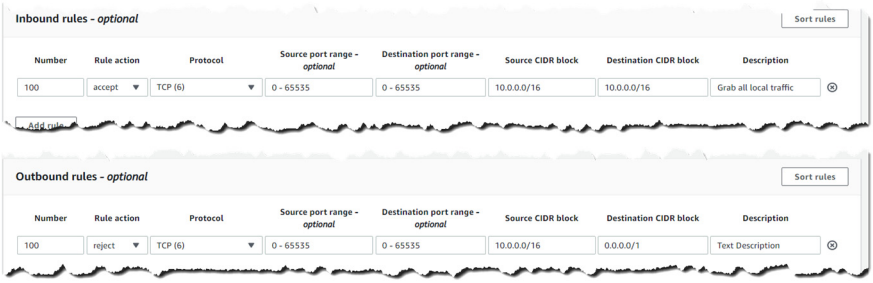
The source must be an individual ENI. If you want to tap all network traffic in a VPC, you will have to do the legwork to automate the creation of new sessions for every new EC2 that is created. This is different than a VPC Flow Log, where you can select a VPC or Subnet ID to apply the VPC Flow Log.

Reference:
https://docs.aws.amazon.com/vpc/latest/mirroring/tm-example-open-source.html

AWS VPC Traffic Mirroring (3)

Filter Options

- Direction
- Protocol
- Src/Dst Port
- Src/Dst CIDR




```
aws ec2 create-traffic-mirror-filter-rule \
    --description "TCP Rule" \
    --destination-cidr-block 0.0.0.0/0  \
    --protocol 6 \
    --rule-action accept \
    --rule-number 1 \
    --source-cidr-block 0.0.0.0/0 \
    --traffic-direction ingress \
    --traffic-mirror-filter-id tmf-04812ff784b25ae67
```


A filter is necessary to describe what data to capture from the Source. A Source + Filter + Target is called a session. It is possible to use filters to send HTTP traffic to one set of targets, while sending UDP to another target, all from the same source. You can have up to three sessions per source ENI.

A filter can have multiple rules. The GUI breaks this up into inbound and outbound rules, or "ingress" and "egress" from the CLI: https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/create-traffic-mirror-filter-rule.html.

There are eight filter options for a rule:

- Traffic direction is either **ingress** or **egress.**
- Rule Number is a unique number that gives the rule order. They are processed by ascending rule numbers.
- Rule action is either **accept** or **reject**. You can use this to control whether this rule is allowing or blocking traffic. Used with precedence, you can create a single filter for sending HTTP traffic to one target. Then, another filter is created that allows all traffic, but rejects HTTP, and is sent to another target.
- Destination port range is what port ranges are being evaluated in this rule.
- Source port range is the to and from ports to evaluate. Source and destination port ranges are a must for controlling what data you would like and tagging it for evaluation. For instance, a Source Port of 80 on a web server would grab HTTP traffic. Source and destination are optional and assume **any** port if not specified.
- Protocol is "UDP, TCP, or any of the IANA assigned protocols": https://www.iana.org/assignments/protocol-numbers/protocol-numbers.xhtml
- Destination cidr and source cdir blocks are the CIDR range of IP addresses for the traffic rule.

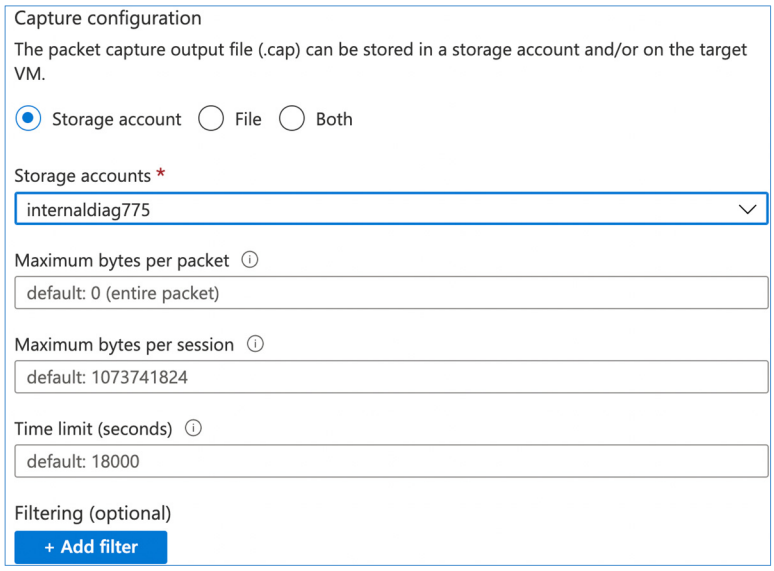
References:
https://docs.aws.amazon.com/vpc/latest/mirroring/traffic-mirroring-filters.html
https://github.com/aws-samples/aws-vpc-traffic-mirroring-source-automation
https://docs.aws.amazon.com/cli/latest/reference/ec2/create-traffic-mirror-filter.html

Azure Network Watcher: Packet Capture

Feature of the Network Watcher service

Captures traffic to/from a specified VM

Trigger capture via Web Portal, CLI, PowerShell, or REST API



According to the documentation, "Azure Network Watcher provides tools to monitor, diagnose, view metrics, and enable or disable logs for resources in an Azure virtual network. Network Watcher is designed to monitor and repair the network health of IaaS (Infrastructure as a Service) products which includes Virtual Machines, Virtual Networks, Application Gateways, Load balancers, etc."[1]

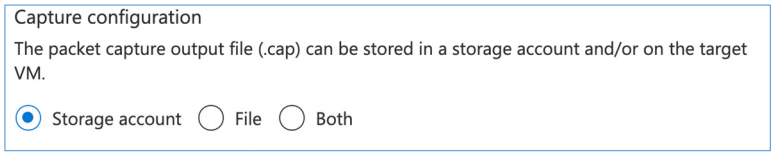
A feature of Network Watcher called "variable packet capture" captures packet sessions to and from a virtual machine. Using variable packet capture requires a virtual machine extension called "AzureNetworkWatcherExtension" but this eliminates the need to run a packet capture manually on the VM. Five-tuple filters (protocol, local IP address, remote IP address, local port, and remote port) limit the traffic to just what is desired. Since packet capture can be triggered through the portal, PowerShell, CLI, or REST API captures can be collected when certain security events occur.[2] The Azure packet capture feature is unique among the CSPs as the other cloud service providers rely on third-party tools or virtual appliances to perform the packet capture.

[1] https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-monitoring-overview
[2] https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-packet-capture-overview

Azure Network Watcher: Packet Capture Location

Three options exist to save packet capture:

- In an Azure Storage Account
- Inside a VM
- Both



The following outbound traffic must be allowed to start a capture:

- To the chosen storage account over 443/tcp (if configured)
- To 169.254.169.254 over 80/tcp
- To 168.63.129.16 over 8037/tcp

There are two options (in which both can be leveraged at the same time), to store the packet capture. You may elect to store the data in an Azure Storage Account (which must be in place first) and you may choose to save the file on the local virtual machine itself.

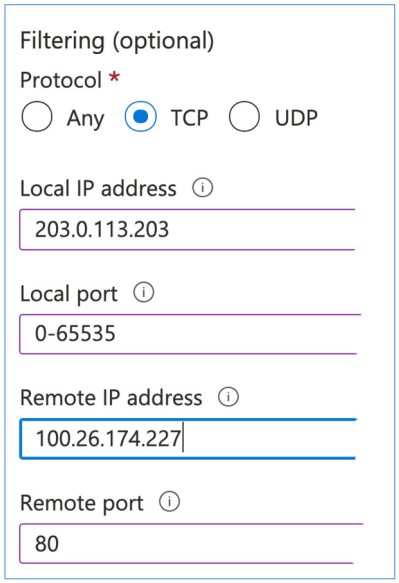
When creating a new packet capture, there are a few other pre-requisites that must be in place regarding outbound network traffic. The chosen VM must have unfettered access to the storage container over TCP port 443 (if one were specified to store the packet capture), outbound TCP port 80 access to 169.254.169.254, as well as outbound access to 168.63.129.16 over TCP port 8037.

Azure Network Watcher: Packet Capture Filters

Filters can be used to limit what is included in the capture file

- Protocol
  - TCP
  - UDP
  - Both
- Local/Remote IP address(es)
- Local/Remote port(s)

*Not limited to a single filter

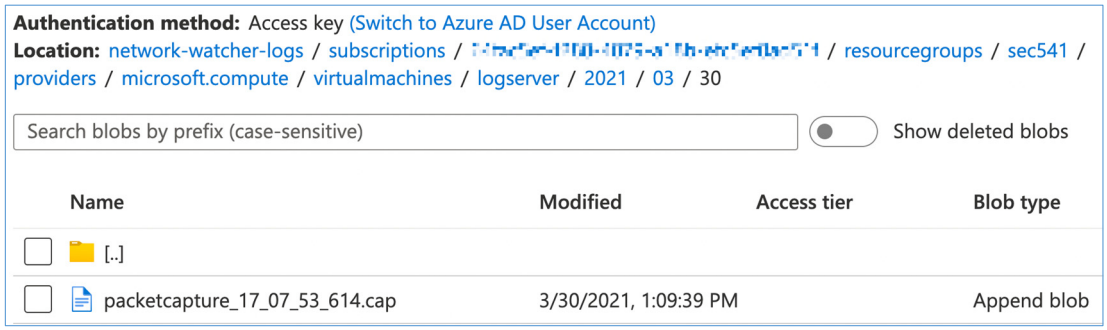


To add even more flexibility, you may choose to filter which traffic you are interested in to capture. Let's say you have some indication that there is malicious activity occurring on your virtual machine and you would like to quietly capture traffic destined for a known adversary over a known port. Sure, you could log into the system and initiate a `tcpdump` session or launch Wireshark on the system itself, but that would surely be recognizable by the adversary. With Azure Network Watcher's packet capture ability, you may establish a filter (or filters) to narrow down the traffic of interest and silently record this data to a storage container.

The filtering options can account for specific details, such as the protocol of interest (TCP, UDP, or any), the local (VM) IP address, the local port, the remote IP address, and remote port. In the example above, it appears that there is suspected malicious traffic destined to 100.26.174.227 over TCP port 80 from the local VM (203.0.113.203). To include all source ports, a range of 0-65535 can be entered).

Capture File: File Location

- Located in the VM's file path specified, or:
- Buried quite deep in the Azure Storage Container
  - **network-watcher-logs/subscriptions/{subscription}/ resourcegroups/{resource_group}/providers/microsoft. compute/virtualmachines/{vm_name}/{YY}/{MM}/{DD}/**



When a capture file is generated, it is stored in up to two different places:

- The file path on the VM that was specified (if File or Both was chosen in the capture configuration)
- Within the Azure Storage account that was specified (if Storage account or Both was chosen in the capture configuration)

You will have to do some spelunking within the Storage account's container to find the packet capture as it is buried several folders deep as shown above. On the next slide, we will take a look at the suspect traffic identified earlier to see just what this traffic was.

Capture File: Wireshark Follow TCP Stream

The capture files, once retrieved, can be moved to other network analysis tools you may be more familiar with, such as Wireshark (which is shown here). It appears that this traffic (through Wireshark's Follow TCP Stream feature) may, in fact, be malicious due to the strange User-Agent (totally-not-a-bot). You would expect something that would indicate an approved web browser or tool within your cloud environment.

This system could be part of a botnet based upon the overly obvious User Agent. Upon further investigation, it was found that this stream is occurring everyone fifteen seconds—another behavior synonymous with a botnet infection. In other words, the system appears to be "phoning home" to receive instructions on what to do next.

Course Roadmap

- **Section 1: Management Plane and Network Logging**
- Section 2: Compute and Cloud Services Logging
- Section 3: Cloud Service and Data Discovery
- Section 4: Microsoft Ecosystem
- Section 5: Automated Response Actions and CloudWars

Management Plane and Network Logging

1. Code Spaces Attack
2. Course Overview
3. EXERCISE: Deploy Section 1 Environment
4. MITRE ATT&CK and Definitions
5. API Logging
6. EXERCISE: Detecting Cloud Service Discovery Attack with CloudTrail
7. Parsing JSON
8. Cloud-Native Logging Services
9. EXERCISE: Parsing Logs with jq
10. Network Flow Logging
11. Capturing Raw Network Traffic
12. **EXERCISE: Network Analysis with VPC Flow Logs**

This page intentionally left blank.

Lab 1.4 | Network Analysis with VPC Flow Logs

Exercise Duration: 30 Minutes

Objectives

In this lab, we will investigate brute force attack techniques with VPC Flow Logs

- Investigate MITRE ATT&CK for brute force attacks
- Conduct a brute force attack with NMAP
- Query the VPC Flow Logs to detect SSH attempts
- Build Athena tables and query Athena for SSH attempts
- Create a customized VPC Flow Log with custom properties

In this lab, we will look at some MITRE ATT&CK techniques that are not specific to the cloud, but which require new tools to analyze. A set of brute force techniques consist of continuous attempts to guess a username/password against a credentialed service. You will look at the different types of brute force techniques and look at how to detect that one has occurred.

You will conduct an SSH brute force attack using NMAP and a provided NMAP script, which will show up in the VPC Flow Logs.

You will then learn how to build an Athena virtual table for VPC Flow Logs and run queries to discover failed SSH attempts.

Section 1 Wrap Up

End of Cloud Management plan and Network Logging

- Discussed Code Spaces attack
- Detecting ATT&CK T1526 with API Logging
- Parsing logs with jq
- Detecting ATT&CK T1499/T1078 with Cloud-Native Logging
- Detecting ATT&CK T1048/T10595 with Network Flow Logging

In the next section, we will look at more Compute and Application logs

This page intentionally left blank.

COURSE RESOURCES AND CONTACT INFORMATION

AUTHOR CONTACTS

Shaun McCullough
smccullough@sans.org

Ryan Nicholson
ryananicholson@gmail.com

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

CLOUD RESOURCES
sans.org/cloud-security
Twitter: @SANSCloudSec

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.