541.3
Cloud Service and Data Discovery

SANS

THE MOST TRUSTED SOURCE FOR INFORMATION SECURITY TRAINING, CERTIFICATION, AND RESEARCH | sans.org

SEC541.3 | Cloud Security Attacker Techniques, Monitoring, and Threat Detection

SANS

Cloud Service and Data Discovery

Welcome to SEC541.3: Cloud Service and Data Discovery.

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

Cloud Service and Data Discovery

1. **Capital One Attack**
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Capital One Attack

A look at how attackers are pivoting from application to cloud environments

In this section, we will be looking at a quite famous cloud hack: the one against Capital One. This use case is enlightening because it involved a large company with a significant amount of its business in the commercial cloud, and also because the alleged attacker was arrested, making the story more complete than most we see.

Before we start, we should note that Capital One is one of the most security conscious users of cloud-based environments. An early adopter for the financial sector and has driven much of the security posture we see today in commercial clouds. Even one misstep can lead to vulnerability and a successful attack.

References:
https://www.justice.gov/usao-wdwa/press-release/file/1188626/download
https://arstechnica.com/information-technology/2019/07/feds-former-cloud-worker-hacks-into-capital-one-and-takes-data-for-106-million-people/

Capital One Attack

Case Study

March 2019

100 million credit card applications from 2015-2019 breached

Initial Intrusion: Exploit public facing application.

At Risk Infrastructure: S3 buckets containing sensitive information

SEC541

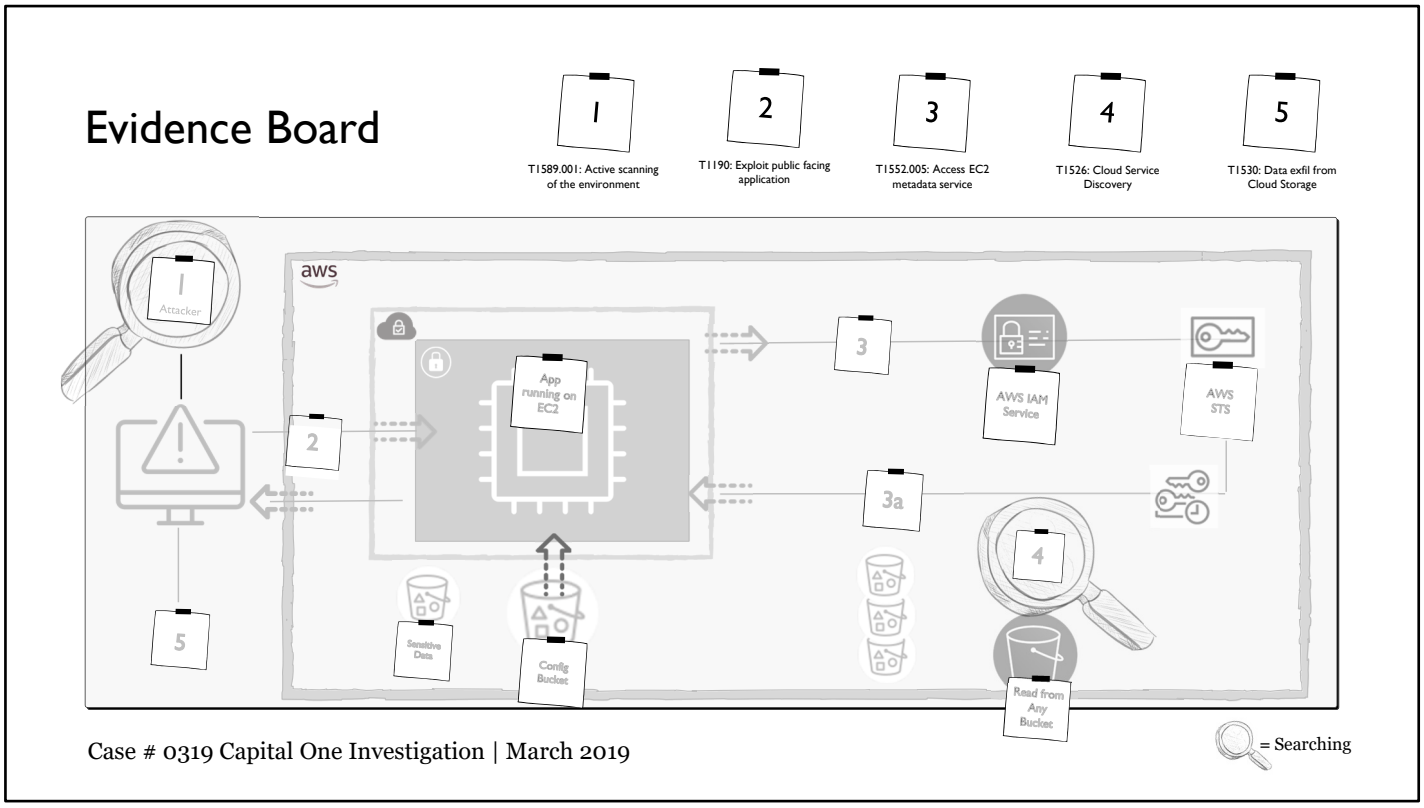
Capital One is largely known as a credit card company but also supports banking, auto loans, and savings accounts. Starting around 2013, Capital One has been migrating workloads into AWS, while building up security tooling and policies to support financial services in a commercial cloud. Recently, Capital One fully exited from eight on-premise data centers into AWS[1].

But even with years of experience and thousands of cloud focused security personnel, there can be mistakes. The attacker took advantage of a vulnerability on an externally facing service to gain access to an instance which was apparently over provisioned. It could read a large number of S3 buckets in the account, including one with over 100 million credit card applications dating from 2015-2019.

Over the years, there have been multiple[2] stories[3] of S3 buckets[4] left open to the world that have usually made the news. It used to be easy to accidently leave a bucket open. In response, AWS has added additional checks with a number "are you sure?" kind of prompts. One of the common trade-offs in security controls is data is meant to be used, so it must be available to whoever needs it. The hard part is ensuring **only** those who need it have access to it.

However, this is not an "S3 bucket left open to the world" kind of case study. The buckets were secured without public access, as they should be. This is a much more nuanced use case.

[1] https://aws.amazon.com/solutions/case-studies/capital-one/
[2] https://www.computerweekly.com/news/252476870/Exposed-AWS-buckets-again-implicated-in-multiple-data-leaks
[3] https://securityboulevard.com/2021/03/another-s3-bucket-leads-to-breach-of-50k-patient-records/
[4] https://cisomag.eccouncil.org/misconfigured-aws-s3-bucket/

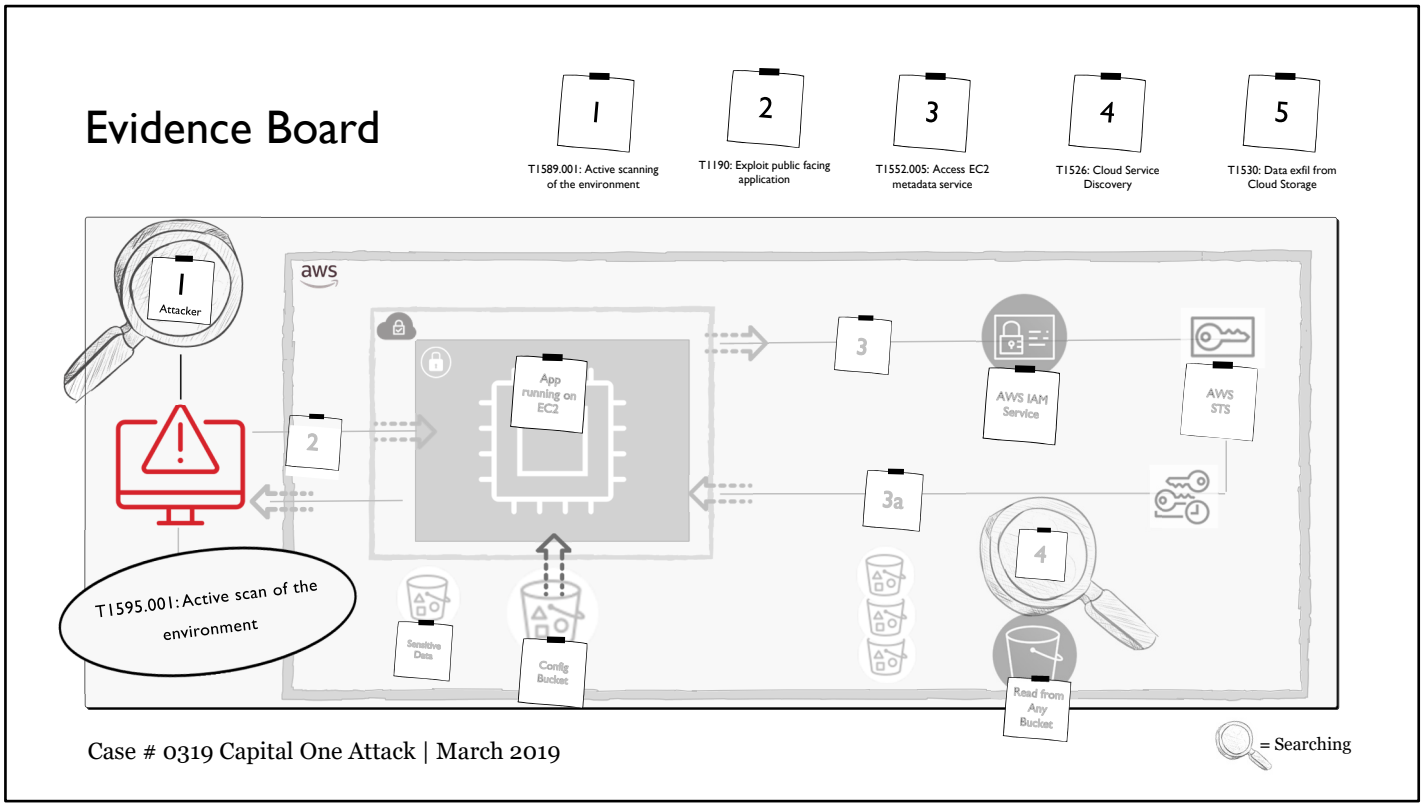


Here is our evidence board for the Capital One investigation. As before, we have diagramed the main resources in play, and the steps the attacker took to circumvent security and perform the attack.

These stories tend to have some common themes in terms of attacks. You will see the old cloud service discovery and exfil from cloud services. This section's investigation looks at a couple of key items.

The initial attack vector was apparently through an external facing application. There is some evidence that a web application firewall may have been in front of the application but was either bypassed or was configured only for logging and not blocking. The attacker used that access to investigate the environment. We know some of how this happened because the attacker discussed it on some forums, as a brag about the attack. Otherwise, it is likely they would never have been detected.

Let's walk through the attack and see the MITRE Attack techniques involved.



We start with the attacker actively scanning AWS, likely from its IP addresses. This is a MITRE ATT&CK Technique called *Active Scanning: Scanning IP Blocks*[1]. They likely scanned IP addresses to identify what is live, and what would have applications accessible. The AWS[2] and Azure[3] IP ranges are published, allowing an attacker to focus on their cloud victim of choice.

While scanning, the attacker likely discovered an external facing application, and then began *Active Scanning: Vulnerability Scanning*[4], that led them to the application to target.

The attacker was once employed by AWS, which may have given them some knowledge about how some customers set up their environments. There has been no indication, however, they had inside knowledge to this specific environment. Insider threat is a cause for concern in many organizations. Other times, a 3rd party user takes advantage of a trusted relationship[5] to gain initial access with valid accounts[6]. There are some references regarding insider threats listed at the bottom of this page.

There is evidence to suggest the attacker was either able to bypass a web application firewall (WAF), or the WAF was not properly configured for blocking. Getting through that WAF, the attacker was able to interrogate the application and discover an application vulnerability.

We must consider a couple of things in our public cloud environments. Our on-premise infrastructure probably relied on a handful of data centers with large detection/monitoring services monitoring traffic going in and out across a gateway. Much of the RSA vendor floor is filled with applications and hardware to monitor and react to that data crossing the gateway. Inside that data center, there is network separation, such as by Office. That hierarchy of separation is stable, set in stone, and cannot be regularly changed without the network team throwing away some weekends.

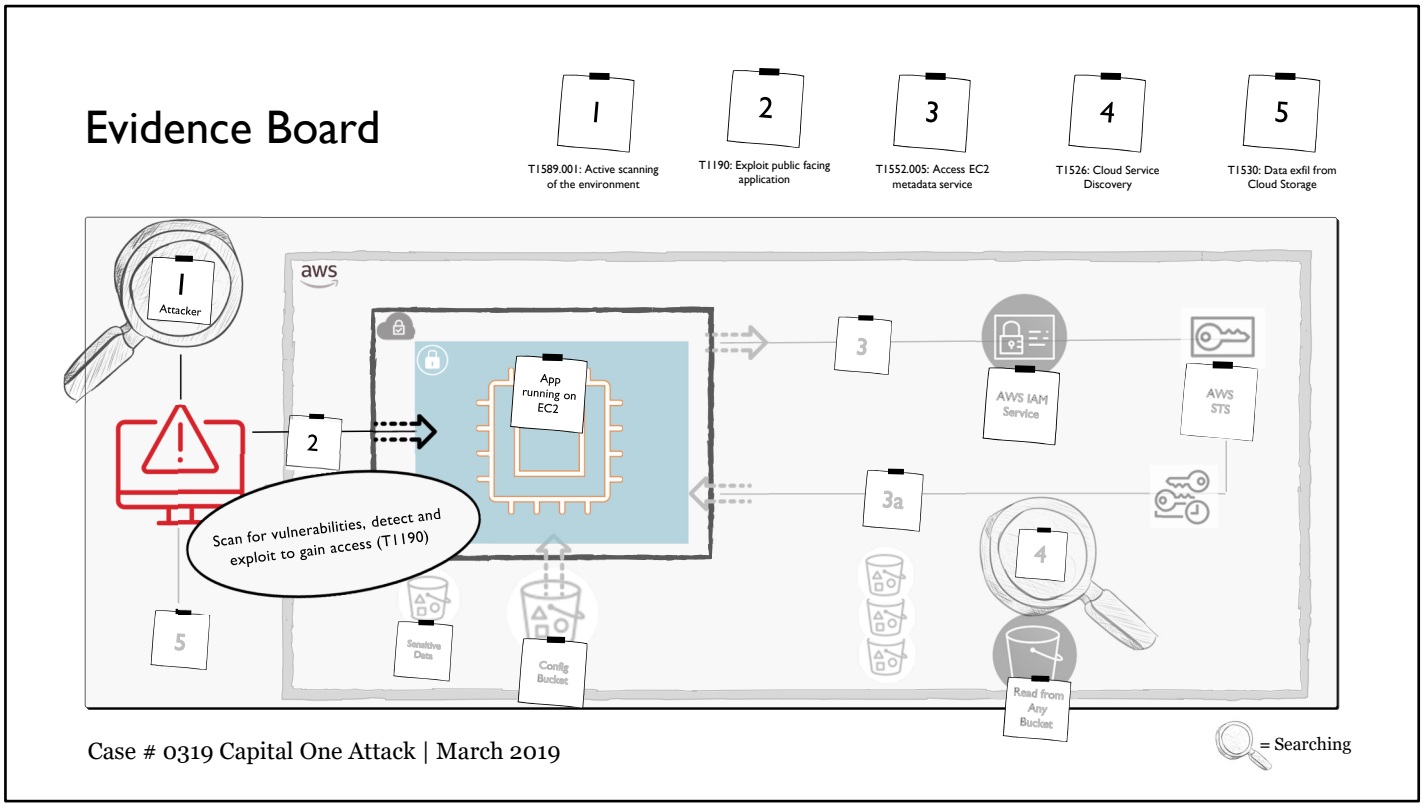
But our commercial cloud environments can be different. We can use AWS accounts or Azure Subscription to separate by teams, or products, or security risk. These accounts and subscriptions are elastic themselves, growing and shrinking as needed. That account/subscription level separation gives security teams a new way to isolate workloads better than we ever could before.

Although this could increase complexity to manage multiple accounts/subscriptions, allowing customer facing services to be completely separated from backoff workloads could reduce the blast radius IF an attacker finds a vulnerable web application on the internet. Attacking that application could give them access to what is in the account, which could be only what is needed to run that particular application.

Our external facing services should be considered high risk. When we start building our threat detection plan, we have to understand that those external facing accounts are at increased risk, but do they also contain the most valuable data as well?

[1] https://attack.mitre.org/techniques/T1595/001
[2] https://docs.aws.amazon.com/general/latest/gr/aws-ip-ranges.html
[3] https://www.microsoft.com/en-us/download/details.aspx?id=56519
[4] https://attack.mitre.org/techniques/T1595/002
[5] https://attack.mitre.org/techniques/T1199
[6] https://attack.mitre.org/techniques/T1078/

References:
https://www.sans.org/blog/decoding-insider-threat
https://www.sans.org/blog/create-an-action-plan-for-insider-threat
https://www.sans.org/white-papers/how-to-build-detection-response-strategy-for-insider-threats



As we mentioned, after scanning the IP ranges, the attacker then evaluates the application to look for an exploitable vulnerability[1]. Tools such as Nmap[2], Nikto[3] and others[4] , could have been used to help them identify potential vulnerabilities in the web application and allowed them to *Exploit a Public-Facing Application*[5].

Investigations show that the attacker was able to use a Server-Side Request Forgery (SSRF) [6] to extract information from the EC2 that the app was running.

We spoke in an earlier class section that one source of information for top threats for the web is the Open Web Application Security Project[7], especially the OWASP Top Ten[8]. As bad as the Capital One attack was, the initial exploitation vector of SSRF is not in the top 10. The attack centers around convincing the app, or the server it is running on, to initiate a request to a URL on the attacker's behalf. Where you, sitting out there on the internet in your Kali box, may be blocked from accessing a system, the app itself may have permissions to pull data. Internal databases, FTP servers, or other HTTP based REST APIs are all potentials.

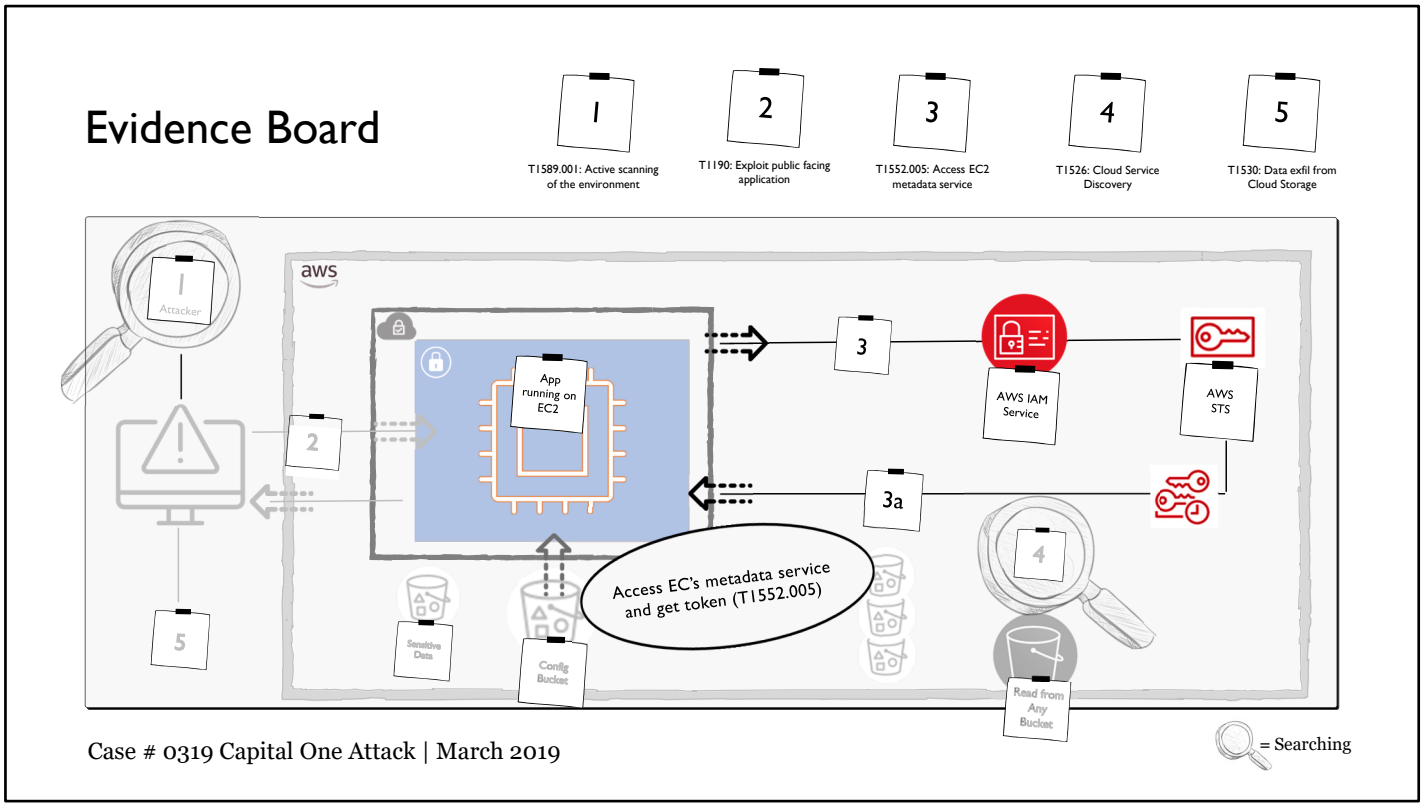
Let's say a web server has a form, where you enter a URL, and it grabs that page for you and displays it. Maybe the web developer intended the user to pick from a list of internal webservers, but the attacker changes the URL it hits. The attacker redirects to an internal webserver that has granted access to the vulnerable web application. It looks like a company server doing its job, but it's really the attacker gaining access they are not supposed to.

In this particular case, the attacker asked the application to return something even more valuable—its own secret key and token information provided by the AWS management environment—by querying the metadata service.

[1] https://attack.mitre.org/techniques/T1595/002
[2] https://nmap.org
[3] https://cirt.net/nikto2
[4] https://owasp.org/www-community/Vulnerability_Scanning_Tools
[5] https://attack.mitre.org/techniques/T1190
[6] https://owasp.org/www-community/attacks/Server_Side_Request_Forgery

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

References:
https://cwe.mitre.org/data/definitions/918.html
https://blog.appsecco.com/an-ssrf-privileged-aws-keys-and-the-capital-one-breach-4c3c2cded3af



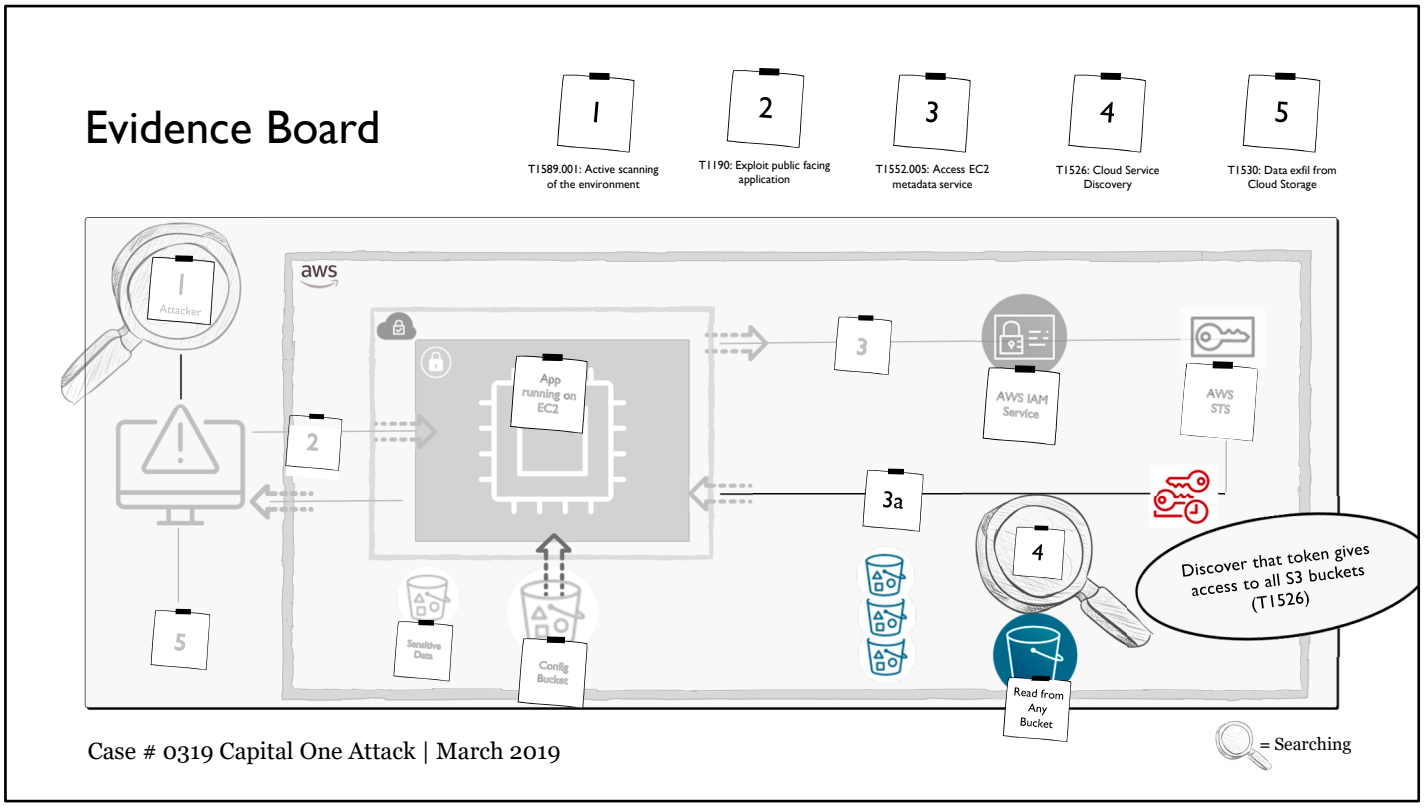
The attacker was able to use the SSRF to gain access to the EC2's metadata service. We will be diving deep into the metadata service in this section of the class, but in the way an EC2, container, lambda function, or application asks the cloud provider information about itself—IP address, instance ID, and its own access tokens. Using the SSRF to grab credential information is the *Unsecured Credentials: Cloud Instance Metadata API* attack[1]. AWS[2] and Azure[3] have different metadata that could have been leveraged by an attacker, but only AWS would have been vulnerable to the SSRF.

This metadata service is available by querying the IP address 169.254.169.254, which acts as a REST interface. This is not a normal IP address that is routable to some web server in AWS. The cloud providers are taking advantage of the virtualized networking and responding directly to the requestor about itself. An EC2 cannot query information about other EC2s—it is built into the underlining AWS software to only respond with your information.

The metadata service interacts with the AWS IAM service to create a temporary access token for that instance which can be used to make calls for resources through the AWS API.

Armed with the token and key information, the attacker could make calls to the AWS API from their attacker machine, pretending to be that particular EC2.

[1] https://attack.mitre.org/techniques/T1552/005
[2] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html
[3] https://docs.microsoft.com/en-us/azure/virtual-machines/windows/instance-metadata-service?tabs=windows



Using the SSRF vulnerability to graph IAM tokens, the attacker can send those tokens to the AWS API. The API, in turn, sees the tokens as legitimate and grants authorization to perform operations as if it was the EC2. Whatever privileges were granted to that EC2 are now available to the attacker.

Does the attacker need to run those commands from a shell on the EC2? No. Maintaining persistance[1] on a victim's computer is not needed. The attacker could copy/paste those tokens to their own computer running at home and perform those attacks. Think about it, if you install the AWS CLI[2] to run commands from your work computer, you are loading it with long-term access credentials. The CLI is hiding the generation and retrieval of the secret access token, or you may be using a Single Sign On (SSO)[3] service that generates the secret token and you copy/paste environment variables.

Grabbing the temporary token and key information, the attacker can now start conducting their own investigation through a cloud service discovery[4].

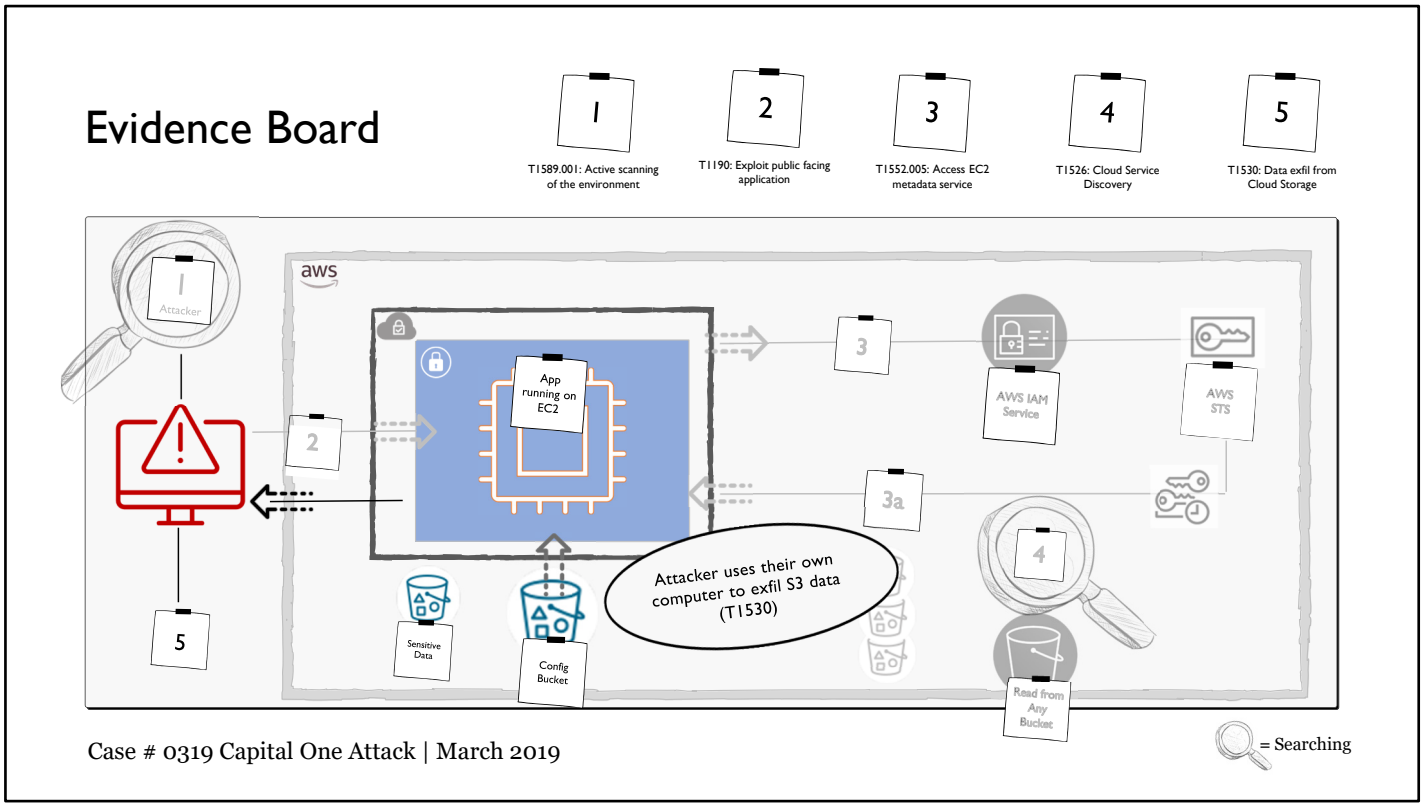
We have seen this cloud service discovery a number of times already. Without it, the attacker really does not know what they have access to. The old days of network sniffing (T1040)[5] is not needed. It is much easier just to ask AWS or Azure what they are allowed to do.

From the cloud service discovery, this time conducted from the attacker's own machine, they discovered that they had read access to some of the S3 buckets that are in this account. Next, the attacker will perform a *Cloud Infrastructure Discovery*[6], which is a bit nuanced from "Cloud Service Discovery". The Cloud Service Discovery answered the question "I have read access to S3 buckets". A Cloud Infrastructure Discovery will ask the question "How many S3 buckets are available to me, and what is in them?".

Right after a Cloud Service Discovery, the attacker will do a Cloud Infrastructure Discovery (T1580), which determines which infrastructure is available from these services

[1] https://attack.mitre.org/tactics/TA0003
[2] https://aws.amazon.com/cli
[3] https://aws.amazon.com/blogs/security/introducing-aws-single-sign-on

[4] https://attack.mitre.org/techniques/T1526
[5] https://attack.mitre.org/techniques/T1040
[6] https://attack.mitre.org/techniques/T1580



For this class, we will assume that the attacker had access to all the S3 buckets in the account with the credentials from the EC2. There are many reasons why an application may need to read data from an S3 bucket. One common problem is configurations. If you work in an elastic environment, virtual machines spinning up and down all the time, you may want the application to grab a fresh configuration file, rather than having that file backed into the AMI. When the EC2 spins up for the first time, a script could go to the S3 bucket and grab a particular S3 object. That EC2 would need read rights to that S3 bucket.

When building new tools, and testing them out, developers tend to create roles that are over provisioned, and thus violate the rule of *least privilege*[2]. As security professionals, we always want our compute application to be granted ONLY the permissions it absolutely needs. However, when building a new application, the S3 bucket might change, or privileges will need to change, or a mistake was just made. This happens and is probably happening in your environment somewhere right now.

When hunting, or looking for vulnerabilities that could spell problems, looking for over-provisioned resources is a good idea, but a bit hard to execute. Without knowing exactly what the application does, how can you tell what is over provisioned? Scott Piper[2] released a tool called CloudTracker through Duo Security[4] that will analyze CloudTrail logs, looking for what resources and actions are taken, and compare that with the IAM role assigned to the calling resource. More resources applied than used? It would create an alert. Since CloudTracker, AWS has released an update to IAM Access Analyzer[5] which will also help identify over-privileged resources. An account or workflow, with internet connections, and is over privileged might be a good place to start investigating for suspicious activities.

Microsoft also has tips and tricks[6] for implementing least privilege in Azure.

Back to our use case, the application needs access to the configuration bucket, but has accidently been granted read access to all buckets in that account. For this use case, there was a particular bucket that has sensitive data in it, that was exposed to the attacker using the credentials stolen from the metadata service. The attacker, from their own home machine (with network obfuscation hops that we will ignore for the moment), they were able to ask the AWS API to return all the data to their local machine, *collecting data from a cloud storage object*[7]. How hard would that be to do?

aws s3 sync s3://secretbucket .

That is it. That will copy everything from that bucket to the local system. Normal network monitoring would not catch this exfil like a Data Loss Prevention service might because you, as the security team for this account, do not have access to the S3 bucket network traffic. There are no network flow logs. Also, it takes extra steps to turn on CloudTrail to monitor data access to S3 buckets and Lambda invocations. The data overhead is significant.

However, for an S3 bucket with years of credit card information, it might be a good candidate.

The investigation indicates someone tipped off Capital One that an attacker was accessing sensitive information. The attacker herself claimed in chat groups to be downloading significant amounts of data across multiple S3 buckets. It is not known if the attacker would have been detected by Capital One's team.

In this section, we will talk through new ways to detect these attacks, focusing on some AWS specific tools and techniques.

[1] https://csrc.nist.gov/glossary/term/least_privilege
[2] https://twitter.com/0xdabbad00
[3] https://duo.com/blog/introducing-cloudtracker-an-aws-cloudtrail-log-analyzer
[4] https://duo.com
[5] https://aws.amazon.com/blogs/security/iam-access-analyzer-makes-it-easier-to-implement-least-privilege-permissions-by-generating-iam-policies-based-on-access-activity
[6] https://docs.microsoft.com/en-us/azure/active-directory/develop/secure-least-privileged-access

Reference:
https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege

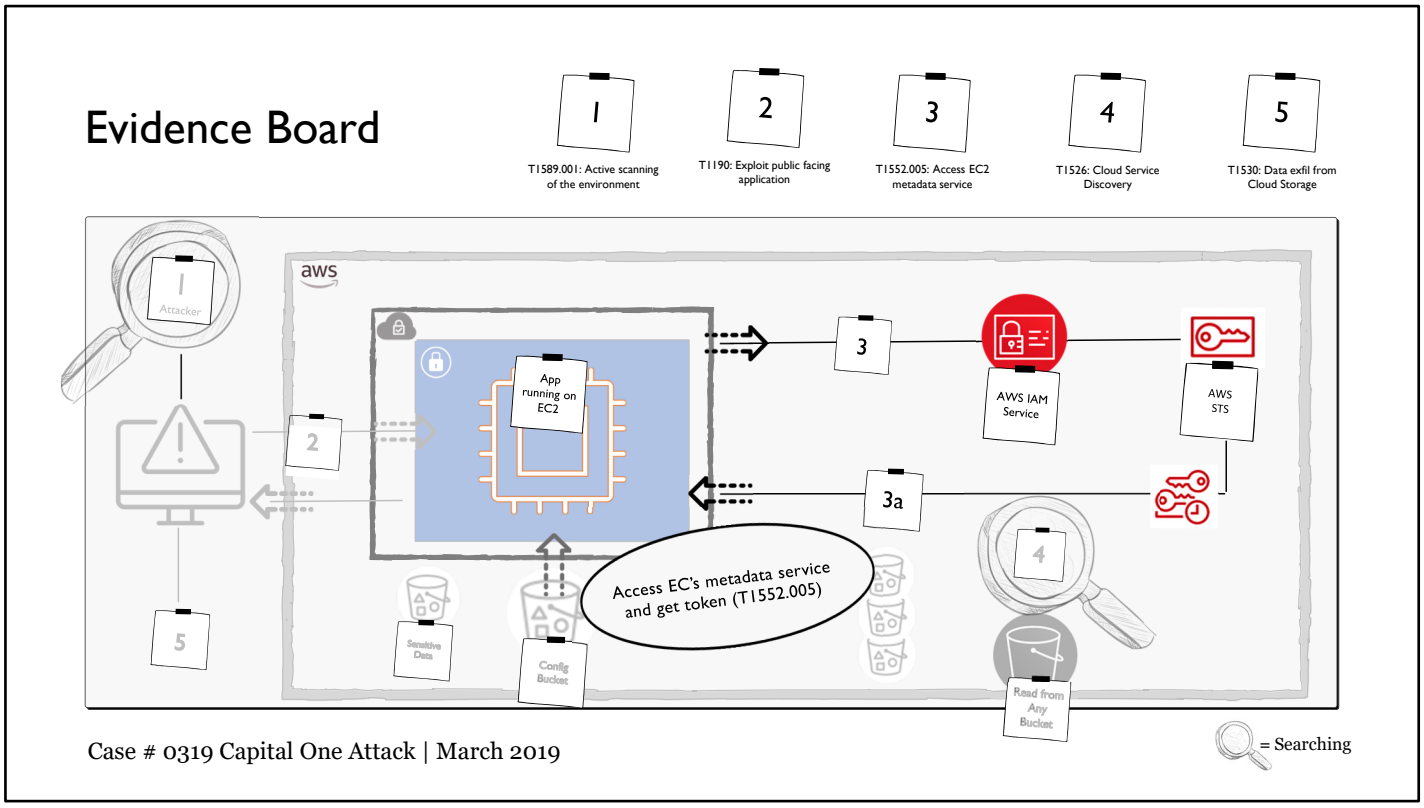
Course Roadmap

- 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

Cloud Service and Data Discovery

1. Capital One Attack
2. **Metadata Service and GuardDuty**
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.



Let's focus in on how the attacker was able to get access to the S3 buckets, jumping off from the EC2. The server-side request forgery allowed the attacker to forge a request from the server itself. This vulnerability usually comes from some web-based application that the attacker has gained access to.

We should note, that it's entirely possible that this application was supposed to be inaccessible from the internet. These services could have been part of an administration service. A misconfigured firewall, one that was not blocking but instead reporting, was likely involved in the workflow. (The reporting states there were IAM roles with the name "WAF_", so we are guessing a bit here).

Using this SSRF, the attacker is able to use the EC2 to get information it should have, the IAM token. In our above picture, the attacker makes the SSRF call and does something like:

"http://example.com/testurl.php?url=http://169.254.169.254"

The application running on that system then performs a GET to http://169.254.169.254, which is the metadata service. The metadata service they asks the AWS IAM service for a new token. That token is then returned in the results of the GET call.

In this use case, the attacker was performing SSRF through an application to get the token. We are going to try and figure out how we would detect this happening. However, the SSRF is not the most important part. The most important part is that whoever has control over a resource, be it a VM, container, or function, that compute resource has all the rights and privileges that were provisioned to it. Given admin access, that EC2 now owns the entire account. We need to be able to detect when someone is subverting the inherit trust relationship between resources and the cloud API.

References:
https://portswigger.net/web-security/ssrf
https://cobalt.io/blog/a-pentesters-guide-to-server-side-request-forgery-ssrf
https://www.csoonline.com/article/3635894/ssrf-attacks-explained-and-how-to-defend-against-them.html
https://attack.mitre.org/techniques/T1552/005/

AWS Metadata Service (1)

Instance metadata is information about your VM instance that a program can use to understand about itself. It includes:

- Host: AMI, storage, hostname, ID, etc.
- Network: public and private IP address, MAC, net cards
- Launch: Region, AZ, subnet
- Security: Security groups, IAM roles
- User data: Scripts that EC2 runs when first starts up
- Scheduled events: Any event such as a reboot or stop that is schedule for a specific time or window

There is an incredible amount of information that an application can gather about how it was deployed. In a new world of elasticity, automation, and immutable architecture, we need our applications to be able to operate on a system that might have changing placements, addresses, and startup information.

The metadata service is used by the configuration of an EC2 upon startup, by AWS SDK to communicate with the AWS management API, and for any metadata aware services.

For instance, an application running on an EC2 may need to know its own hostname for logging. The application can query the metadata service directly to get that information, or it could use AWS SDKs to perform the interrogation itself. This is not an exhaustive list, but some of the more common metadata we might be interested in.

Host Information:
When an EC2 launches, information about the instance is provided as metadata. This data may change when a system is stopped and restarted, while others are valid throughout the life of the EC2.

- Amazon Machine Image (AMI) is the binary image from which this EC2 was launched. Every EC2 starts from one of these images. In our labs, we have been starting up our instances from AMIs provided to us through a shared account.
- AMI Launch Index is really interesting. When an EC2 is launched, you can specify that you want to launch more than one at the same time. AWS tracks that these are launched together, as a group. When you perform “describe-instances" they come in a “Reservation” array, gathering groups of EC2s that were reserved, or launched, at the same time. The AMI Launch Index tracks which number this instance is in the launch group, starting at 0. So, if you are the 3rd launched, it will be “2”.
- Block device mapping data has information about the data stores such as the root device, the EBS volumes, and non-NVMe (non-nitro) based storages: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/nvme-ebs-volumes.html.
- Instance ID is important for loggers. The CloudWatch logger from Section 2 sends data to a log group of “/watson/” with the cloud watch log stream of the instance ID. The CloudWatch agent uses the metadata service to retrieve that Instance ID.

- Instance Type is a cool one. Maybe an application should run differently on an t2.micro than a c5.xlarge. Allowing applications to self-configure is a great way to improve automation in an elastic environment.

Network:
Applications can use the metadata services to query how the EC2 is configured.
- Hostnames for the public and private network addresses.
- Public and private IP addresses. Public IP addresses tend to change if a system is stopped and then started again. Any service that needs to know its own IP address can query the IP address from the metadata service.
- MAC addresses. You can request the eth0 mac address, and also iterate through any other virtual network device MAC addresses that might be attached to the system. You can retrieve each public IP, private IP, ipv6 info, subnet ID, hostname, etc., for each MAC assigned to the system.

Launch Information:
Does the application need to know **where** it was launched? A system can discover exactly where in AWS it was launched.
- AWS partition determines if you are launched in the standard commercial AWS regions most people use, or maybe a separate region such as China or GovCloud. Applications may need to operate differently if they are deployed in a more restrictive partition that does act a bit differently.
- Region and availability zone may be helpful for logging purposes.

Security:
Some security-based information is provided by the metadata service, and this is the information that has gotten some organizations in trouble.
- Identity and access management is the centerpiece for how AWS manages what resources/people are allowed to do in an account. The metadata service provides information about the IAM Instance Profile, and when it was last updated. We will talk more about this throughout this section.
- A resource can only communicate if a security group has been applied to that instance and it grants communications. The security group and IDs are available.

User Data:
When an EC2 starts up, it can run what is called “user data”, which is usually a bash script or a cloud-init script that has startup information. Maybe you have a single Launch Template for all your NGINX servers, and the user data has the code to build out the actual webpage. The user data is provided through the metadata service.

Scheduled Events:
If an EC2 has a scheduled reboot or stop, then that information can be pulled from the metadata service.

References:
https://cloudinit.readthedocs.io/en/latest/
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/monitoring-instances-status-check_sched.html
Full detail of metadata categories: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-categories.htm

AWS Metadata Service (2)

The metadata service is accessed by making an HTTP call to a special IP address: 169.254.169.254

This is not a real web server. The call never leaves the EC2 through the network. The call hits the AWS magic hypervisor network, retrieving the metadata.

```
$ curl http://169.254.169.254
```

From an EC2, you can use cURL to hit the system to retrieve metadata.

From our EC2, we can use the cURL[1] command line tool to interrogate the metadata service. The cURL command line tool will transfer data with a URL, including FTP, HTTP, POP3, SMB, and a bunch of other protocols. We will query the metadata service at the IP address 169.254.169.254, using an HTTP application protocol, and the metadata service will return the data.

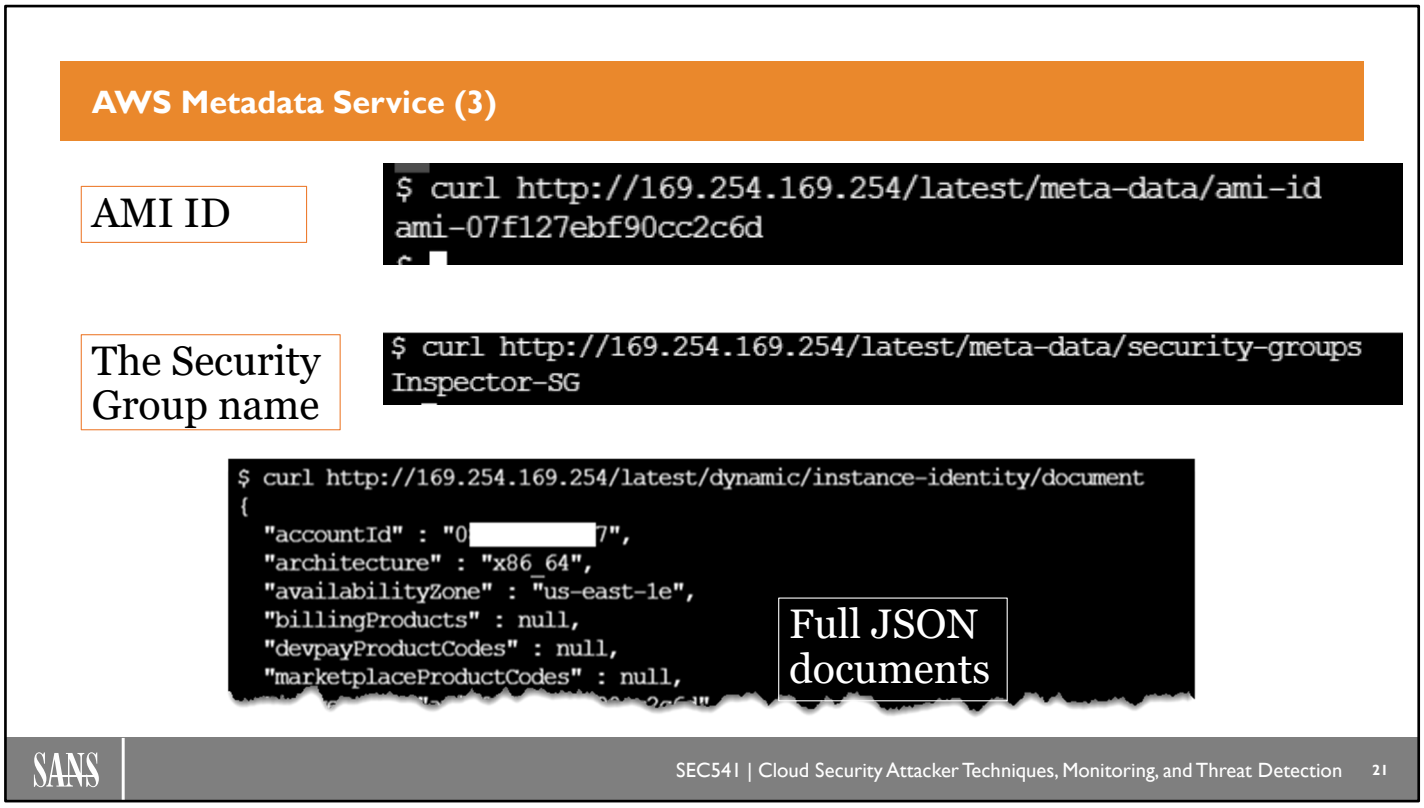
From an EC2, we can curl the metadata service on IP address 169.254.169.254[2]. It is a REST[3] interface[4], where the URL specifies which data we want to retrieve.

This IP is not routable—it never really leaves in a traditional networking sense. It is not traversing the VPC network, using the EC2's elastic network interface[5]. The AWS magical hypervisor service is responding to the IP address, providing the EC2 with information about itself. The metadata service is only returning the data about the service which made the call.

Think about the purpose of the metadata service. When a new virtual machine is deployed, its likely part of a larger fleet of instances. Applications, configurations, or startup scripts will need to know about the operational environment it is in. It needs the temporary access key that the AWS CLI, boto3, and the SDK will need to interact with AWS management console and other AWS services.

A metadata service for ECS[6] exists but is a bit different. Lambda[7] also has a metadata service, but most of the variables are injected into the runtime as environment variables.

[1] https://github.com/curl/curl
[2] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-retrieval.html
[3] https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
[4] https://docs.microsoft.com/en-us/azure/architecture/best-practices/api-design
[5] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-eni.html
[6] https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-metadata-endpoint.html
[7] https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars.html#configuration-envvars-runtime



What could we get from the EC2 metadata service? From the screen, we can see that the AMI ID and the name of the security groups are just two pieces of data. There is a long list of data[1], with more being added all the time. On September 30th, 2021, a new metadata value was added to return the EC2 tags. You have to explicitly set a parameter at launch time to allow the EC2 to return the tags[2]. This is an interesting problem. Let's assume you want to set customized parameters for a script running on that EC2 to use. You could hard code them into the script, but that is clunky. You could set them as an environment variable in the user data[3]. Or, you could have the script query the parameter store. For teams that build workflows around tags, they have to awkwardly allow the EC2 query *describe-instances* to return tag data. AWS now allows the EC2 to get tag data from its own metadata service. Bu what about all those organizations that were tagging the EC2 with sensitive information, such as the name of the person who launched the instance? You may not want a webserver to have that information in the metadata service. Thus, the instance has to be launched with the parameter to allow the access. This push and pull between automation and providing automated service with too much information is how attackers are finding new attack vectors. SANS class *SEC540: Cloud Security and DevSecOps Automation* talks more about these types of problems[4].

There are two basic kinds of responses, as we see above. We can request a specific piece of information, such as */latest/meta-data/ami-id*. That will return just the AMI ID and is usually how scripts are built these days. An older approach, that still works, is to get a JSON document with multiple pieces of information, such as the *latest/dynamic/instance-identity/document*, which will return account ID, architecture, availability zones, etc.

The AWS SDKs, such as Boto3 which powers the AWS CLI, perform these metadata calls in support of CLI commands[5]. Developers who are building AWS code against the SDKs may never have to interface directly with the metadata service, as it is wrapped up into the software development kits. More complex automation scripts, especially if you are running cloud unaware applications, will likely see developers interacting with these services. The author typically sees this when configuring a log forwarder or setting the hostname to match some company standard.

In the top examples, the results are endpoints, because there is no '/' after the returned value.

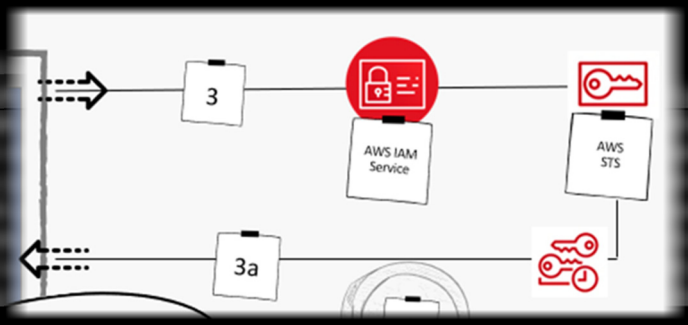
[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-categories.html
[2] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html#allow-access-to-tags-in-IMDS

[3] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html
[4] https://www.sans.org/cyber-security-courses/cloud-security-devsecops-automation/
[5] https://github.com/boto/botocore/blob/master/botocore/utils.py

AWS Metadata Service (4)

Retrieve the security credentials from inspector-role

```
$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/inspector-role
{
  "Code" : "Success",
  "LastUpdated" : "2021-05-22T16:45:42Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "                    ",
  "SecretAccessKey" : "
  "Token" : "
  "Expiration" : "2021-05-22T23:10:12Z"
}
$
```



How did our attacker gain access to all the S3 buckets? Well, the EC2 had a IAM Instance Policy[1] that was attached to a role[2] that had a policy granting read and list privileges on that S3 bucket.

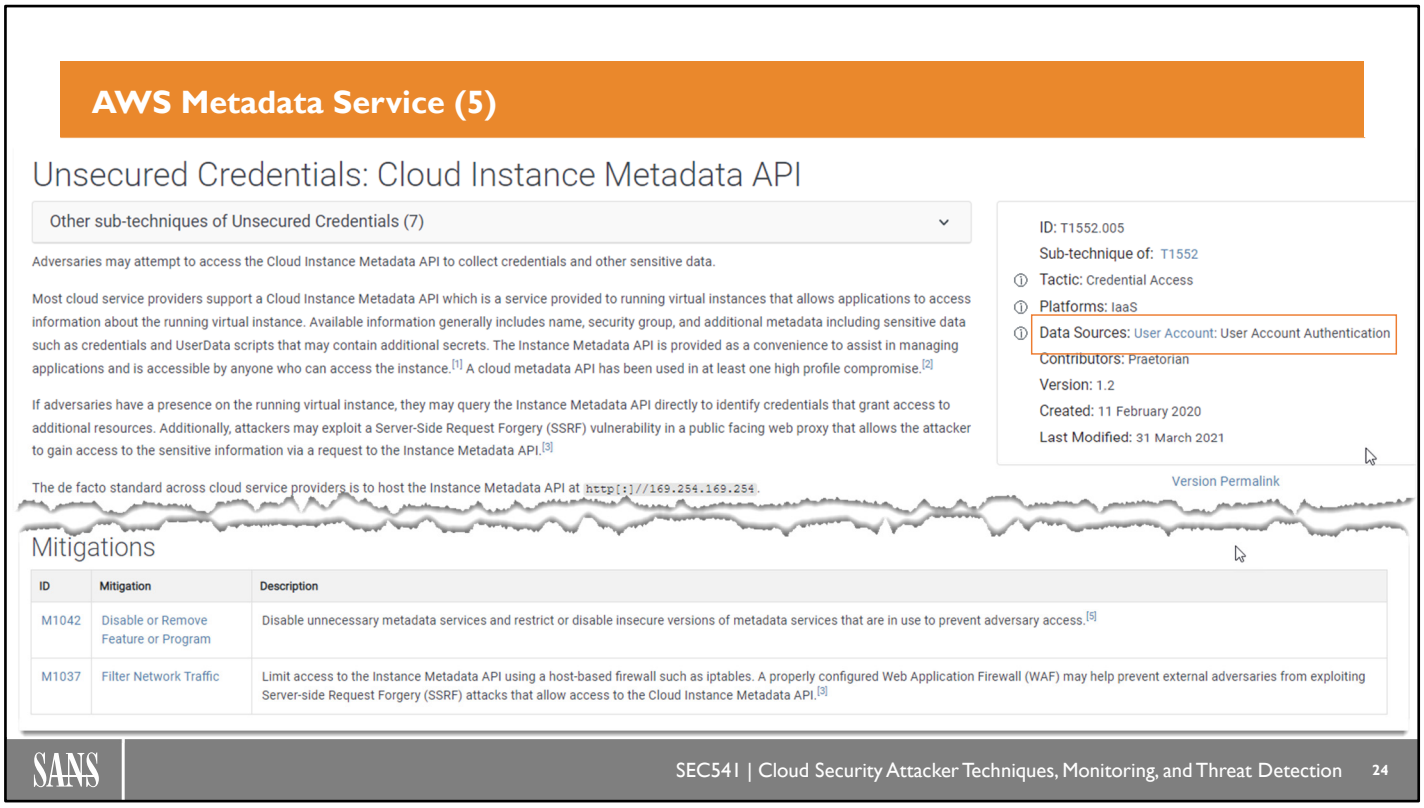
Reading the available reporting, the author is assuming that the victim EC2 was either running as a web application firewall (WAF) or was behind a vulnerable WAF. SSRF vulnerabilities are usually found in "internal" systems which perform download services, file reads, sharing, API demos, or preview of contents. Using the SSRF vulnerability, the attacker was able to have the EC2 perform a query on http://169.254.169.254/latest/meta-data/iam/security-credentials to retrieve the name of the instance profile.

For your inspector workstation, the instance profile is called *inspector-role*. So, we would return the key information with *http://169.254.169.254/latest/meta-data/iam/security-credentials/inspector-role.* By making this query, we are returned a JSON object with an `AccessKeyID`, `SecretAccessKey` and `Token`. These three pieces of information are all that is needed to configure an AWS CLI on any workstation, even sitting at home, with the credentials necessary to make calls to the AWS API service as if they were the inspector-role. They would simply need to update the AWS CLI's credentials file3 to use the retrieved secrets.

The attacker then can make a call with the AWS CLI, and the AWS API will actually think that the vulnerable server is the one making the calls. The IP address will be wrong, but the API credentials will be what was assigned to this VM.

Not only was the Capital One attacker able to copy out these creds, but until the expiration, they could run *the* `aws s3 sync`[4] command on their own workstation and copy out the data from the S3 buckets.

References:
[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-policies-for-amazon-ec2.html
[2] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html
[3] https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-files.html
[4] https://docs.aws.amazon.com/cli/latest/reference/s3/sync.html



This is some information about the MITRE ATT&CK in question, T1552.005[1].

On the right-hand side, the MITRE page states that the Data Sources can be User Account Authentication, and that is true. No need to escalate to root privileges. This means that no matter what user the application is running as, the SSRF will have access to the metadata service.

On the technique page, we can see a mitigation for M1037: Filter Network Traffic. One suggestion for blocking the attack on the metadata service is to use IP tables to block access to 169.254.169.254 for every user except for root, thus filtering network traffic.

```
iptables -A OUTPUT -m owner ! --uid-owner root -d 169.254.169.254 -j DROP
```

This was good advice until AWS made an update to the metadata service, which we will discuss later in this section.

Remember, when the curl to 169.254.169.254 happens, this call does not leave the EC2 through the VPC network but is provided directly to the instance from the virtualization service. There will be no VPC flow logs of this for us to use. Network detection of grabbing the security credentials is therefore out of the question. So, how do you detect this when the initial security credential result is fraudulent?

Our host-based agents could monitor for calls to IP address 169.254.169.254? This will cause a log of logs, and the EC2's are likely performing these operations quite regularly. Lots of noise.

You could detect CloudTrail logs that have an originating IP address from outside your VPC for an IAM Role provided EC2 role. This would not be difficult to do, but you would need to manage what IP ranges to compare, and every CloudTrail log would need to be evaluated, by sending it to CloudWatch and a metric to be created.

We have another solution to detect this in GuardDuty, but let's take a look at Azure's metadata service first.

References:
[1] https://attack.mitre.org/techniques/T1552/005
[2] https://attack.mitre.org/mitigations/M1037

Azure Metadata Service (6)

Azure has a metadata service that is similar in goals to AWS

```
$ curl -H Metadata:true --noproxy "*" "http://169.254.169.254/metadata/instance?api-version=2020-09-01"
```

-H Metadata:true
Creates a header in the GET HTTP command unique to a metadata call

--noproxy
Does not allow this command to go through a proxy

Azure has its own metadata service that has the same intention as AWS, but, with a different protocol. The API calls are different and there are longer JSON forms, but the big difference is how the metadata service is accessed.

We will use cURL again, but it will not be as simple as with AWS. We need to run cURL with an added HTTP header, and we have to explicity send the HTTP command so that proxes cannot be used[1].

If we run "curl --help", we will see a list of options. We will look at two of them.

`-H Metdata:true`
This creates a custom header and sends it as part of the HTTP GET request. The value of the header must be *`Metadata:true`* to signify in the request that yes, we know this is for metadata and nothing else. Without the header, the Azure metadata service will drop the query.

`--noproxy`
This stops the ability for a proxy to be used, from anywhere, to return the data—ensuring that the execution is happening right on the VM itself.

These two requirements for curling the metadata service in Azure add a few layers of security we do not have in AWS's metadata service.

First, the SSRF attack works by the application making an HTTP call, on your behalf, to another endpoint than the one expected. However, it is unlikely that the vulnerable application would allow for the HTTP call to be manipulated, allowing the attacker to add the Metadata:true header. Maybe the vulnerable application is actually a reverse proxy that would be sending HTTP calls on your behalf? The --noproxy keeps that proxy from being able to forward the curl command onto the metadata service on the attacker's behalf. This is great—how come Amazon didn't think of that? (Hint coming in a few pages)

Reference:
[1] https://docs.microsoft.com/en-us/azure/virtual-machines/linux/instance-metadata-service?tabs=linux#instance-metadata

Azure Managed Identities

- Similar to roles in AWS, can assign **Role-Based Access Control (RBAC)** to cloud resources
- Two types of managed identities
  - **User-assigned**: Identity is created by an Azure user and can be assigned to one or more resources
  - **System-assigned**: Enabled on cloud resource and its lifecycle is tied to that of the resource
- Can be acquired through Azure's metadata service

```
$ curl 'http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-0
2-01&resource=https%3A%2F%2Fmanagement.azure.com%2F' -H Metadata:true -s
{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6Im5PbzNaRHJPRFhFSzF
qS1doWHNsSFJfS1hFZyIsImtpZCI6Im5PbzNaRHJPRFhFSzFqS1doWHNsSFJfS1hFZyJ9.eyJhdWQiOi
```


Azure includes an option called managed identities which allows cloud resources, like Azure Virtual Machines, to access other cloud resources without the need of embedding credentials. Does this sound familiar? It should as AWS has a very similar option—AWS IAM Roles. These Azure Managed Identities come in two forms: user-assigned and system-assigned.

User-assigned managed identities simply means that the Azure customer will first create the managed identity user in the Azure Managed Identities service. This new identity can then be assigned to one or more cloud resources, and it is not automatically deleted when the assigned cloud resource is terminated. This means that it is the user's responsibility for lifecycle management of this identity. This is how AWS IAM roles work.

System-assigned, on the other hand, allows the Azure customer to allow Azure generate and assign a new identity during a cloud resource deployment (or even after the resource has been provisioned). The lifecycle is also handled by Azure in that, when the cloud resource in which this identity is assigned is deleted, so is the identity.

When the cloud resource is in need of credentials, it can ask Azure, via the local metadata service, for a bearer token which has a default lifetime of twenty-four hours. Above, you can see just how easy it would be for an attacker to acquire this token which will give them the same rights as the managed identity—ranging from very low-level to Owner permissions. This does require command-line access so attacks similar to Capital One where remote file inclusion was the Server-Side Request Forgery (SSRF) method of choice would fail since that special `Metadata:true` header would likely not be included in the forged request.

AWS Metadata Service (7)

In response to Capital One, AWS's new IMDSv2 is similar to Azure

```
$ TOKEN=$(curl -sX PUT "http://169.254.169.254/latest/api/token" -H "X-aws-ec2-metadata-token-ttl-seconds: 21600")
$
$ curl -sH "X-aws-ec2-metadata-token: $TOKEN" -v http://169.254.169.254/latest/meta-data/
```

- Use PUT request to initiate, returns TOKEN
- Include TOKEN in all GET requests
- PUT request rejected if "X-Forward-For" header

```
index
ami-manifest-path
block-device-mapping/
events/
hostname
iam/
identity-credentials/
instance-action
instance-id
instance-life-cycle
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
reservation-id
```


In November of 2019, AWS announced the Instance Metadata Service Version 2 (IMDSv2)[1], which adds a layer of security on metadata service calls. With an instance running IMDSv2, (as opposed to the SSRF enabled IMDSv1), the cURL command as an extra step designed to ensure that an SSRF attack is unsuccessful.

First, the cURL must use an HTTP PUT[2] command to `/latest/api/token` to return a TOKEN value. Applications who are vulnerable to SSRF are usually performing a GET[3], the requirement for a PUT will plug up that hole.

The PUT must include a custom header `X-aws-ec2-metadata-token-ttl-seconds` value, to set a time to live value. It can be up to six hours, or it can be significantly shorter. It's possible to request a token with every single call, but there is an upper limit to hitting the metadata service, so it depends on the number of calls.

The PUT command returns a token, which we stored in the variable TOKEN. Now, it's time to make a query for "meta-data". We must include the TOKEN in a new custom header for a GET command with the key *`X-aws-ec2-metadata-token`* and the value is the token.

Now, every request is protected by session authentication that is also resistant to SSRF. It also protects against a WAF that is openly forwarding headers, as they typically do not support PUT operations.

In secure programming style, a program will make a request of a TOKEN and store that data in memory for the session, perform its calls, then throw away the TOKEN, to be requested for the next round of requests.

[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
[2] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/PUT
[3] https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods/GET

Reference:
https://aws.amazon.com/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/

AWS Metadata Service (8)

It is advisable to transition your EC2 services to IMDSv2

- A CloudWatch metric "MetadataNoToken" tracks number of calls to IMDSv1
- IMDSv2 and IMDSv1 is on by default. You can turn off IMDSv1 at Launch.
- IAM condition ec2:MetadataHttpTokens
- API call ModifyInstanceMetadataOptions to change HttpTokens to "required"

What about detecting use of EC2 IMDSv1 tokens from a non-AWS environment? Time to talk about GuardDuty.

You should not be using IMDSv1, but you probably are somewhere. Or you are using AWS marketplace VM's that only work on IMDSv1 because they have not taken the time to upgrade to IMDSv2. Knowing where this is the case would be helpful. We can also enforce IMDSv2[1].

- CloudWatch has a metric of "MetadataNoToken" that tracks calls to the IMDSv1 service on EC2s through CloudWatch agent.
- When an EC2 is started, there is an option for making IMDSv2 optional or required. User IAM roles and service control policies can enforce that HttpTokens is required. This would also require changing any Terraform, CloudFormation, or other infrastructure building scripts.
- You can change running EC2s to require HttpTokens through the ec2 API call `ModifyInstanceMetadataOptions`[2].

Your applications need to be tested to ensure they work properly with the IMDSv2. If you are using newer Boto3, AWS CLI, or another SDK, ensure they have been updated and should work appropriately.

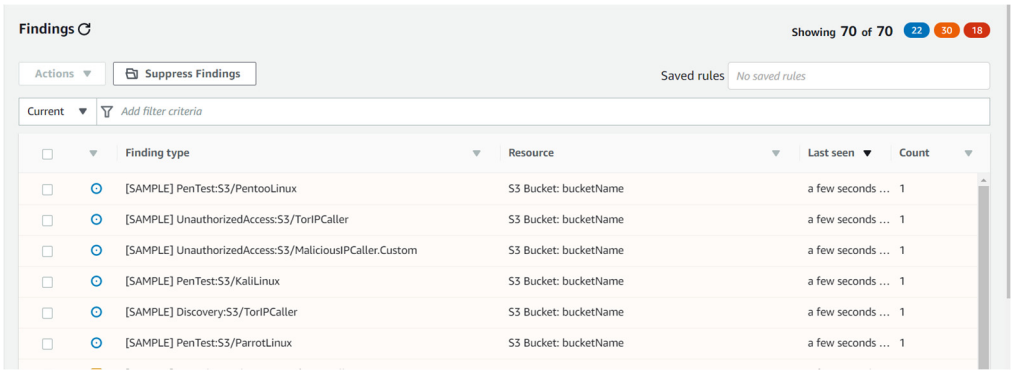
What about detecting when a user uses the EC2's IAM credentials from outside of AWS? We said that might be possible for us to build the tooling to detect it, but why not have AWS tell us. Let's talk about GuardDuty.

[1] https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-service.html
[2] https://docs.aws.amazon.com/AWSEC2/latest/APIReference/API_ModifyInstanceMetadataOptions.html

GuardDuty (1)

GuardDuty is a threat detection service that continuously monitors for malicious activity and unauthorized behavior to protect your AWS accounts, workloads, and data stored in Amazon S3

Released November 2018



GuardDuty is a service dedicated to detecting potential threats by analyzing the logs we have talked about up to this point. It is 100% focused on detecting suspicious behavior and providing alerts in a single platform. Like any full-service threat detection service, GuardDuty detects the threats that AWS decides to detect in the way it decides to detect them. New detection rules can be created and deployed to your environment whenever AWS wants. But unlike some other company products, AWS does a good job of telling you what they detect, so you can determine what they don't.

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

GuardDuty (2)

GuardDuty analyzes your environment and uses threat detection to alert on threat "findings." The kinds of threats are:

EC2 Findings
- Backdoors/Persistence
- Crypto Minors/Impact
- Recon Activity/Discovery
- Trojan Horse
- Unauthorized Access/Priv Escalation

IAM Findings
- Pentest Distro
- Anomalous Behavior
- Recon/Discovery
- Stealth/Defense Evasion
- Unauthorized Access/Priv Escalation

S3 Findings
- Discovery/Enumeration
- Exfiltration
- Object Impact
- Pen Test Tools
- Policy Changes
- Stealth
- Unauthorized Access

Kubernetes Findings
- Credential Access
- Defense Evasion
- Discovery
- Execution
- Impact
- Persistence
- Policy

These are most of the findings that GuardDuty detects. Loosely broken into three kinds of threats that are detected[1]. AWS does not map findings or threats to MITRE ATT&CK, like we do in this class. Some individuals have performed the mapping themselves, but they are not up to date[2].

Below is a description of the different types of techniques detected in GuardDuty. We want to call out at least two of them.

IAM Findings – Unauthorized access will detect an EC2's credentials being used from a different account, or outside the AWS environment. This would have detected the Capital One attack.

The Kubernetes Findings were recently introduced. A description is below but note that GuardDuty findings and new data sources are added regularly. When this finding type was added, everyone using GuardDuty was able to start using it right away, but it would increase cost. We will talk about that in two slides. One key thing to think about is that with GuardDuty's new Kubernetes findings, the Tesla story may not have happened. Of course, Tesla would have to use EKS, GuardDuty would need to be monitored, and AWS would have had to deliver the capability a few years ago. One thing to think about is that using the cloud providers' services, you get a set of easy-to-use security tools, that are likely not as sophisticated as you would like, but they are extraordinarily scalable across all your teams.

EC2 Findings[2]

EC2 findings are focused on what is happening on or to an EC2. Most of the detections use network logs to detect known bad behavior, or activity against known bad actors.
- Backdoor activities such as communicating with a known command and control server, could be mapped to the ATT&CK Persistence[4] tactic.
- Crypto Miners are detecting communication with known Bitcoin related IP addresses. They could be categorized as ATT&CK Impact[5] tactic.
- Recon activity appears to be port scans, or unprotected ports that are being probed. This would be categorized as the ATT&CK Discovery[6] tactic.

- Trojan Horse activities are when the EC2 is performing domain name queries that are known trojan horses. This is another persistence tactic.
- Unauthorized Access are activities where the EC2 is attempting to additional access. The EC2 initiated, or is the victim of, an SSH or RDP brute-force attack. This category of finding also detects that it is attempting a DNS lookup that resolves 169.254.169.254. This could be an attacker attempting to return the EC2's secret token by a DNS rebinding[7].

IAM Findings[8]

IAM Findings are specific to IAM entities and access keys and always have a resource type of AccessKeys. These are focused on techniques we might try and detect in CloudTrail.

- Pentest Distro detects and EC2 running Kali Linux[8], Parrot OS[9], or Pentoo Linux[10] and are making API Calls. This seems like a very narrow cast detection. GuardDuty documentation does not describe HOW it determines that the VM is determined to be running these OS. It is likely based on an AMI ID. It is likely a limited use case.
- There are a number of techniques detected that are deemed "IAMUser/AnomalousBehavior". They are categorized using ATT&CK tactic names such as credential access, defense evasion, discovery, exfiltration, impact, initial access and persistence. They are described as "informs you that an anomalous API Request was observed in your account[11]." What does "anomalous" mean to AWS? AWS does say that it uses GuardDuty's machine learning detection model. It is likely looking for outliers in how certain API calls are used in your environment. It is difficult to determine, however, exactly what is being detected. If this alert shows up, you should just go ahead and remediate the compromised AWS credential[11].
- Recon in IAM Findings detects API calls happening from known malicious IP addresses, or an API was involved from a TOR exit node. AWS API calls happening outside the environment in scary ways. Another ATT&CH Discovery tactic, likely.
- Stealth activities are detecting CloudTrail logging being disabled, or Password policies being changed. This would map to ATT&CK Defense Evasion tactic[12].
- UnauthorizedAccess alerts in IAM Findings that are detecting our Metadata service attack for this Section. Credentials were created for one EC2, but are being used in another account, from a non-AWS IP address, or multiple console logins for the same user from different geographical locations around the same time. The Capital One use case, where the attacker performed the S3 sync from their own machine would be detected here.

S3 Findings[13]

S3 findings are specific to S3 resource type of S3 bucket.

- Discovery is looking at known bad IP addresses performing list bucket operations.
- Exfiltration[14] is detecting known bad IP addresses performing get object type operations.
- Object Impact is known bad IP addresses performing a Put Object. Adding an object to the bucket. These three seem like they would work together. They are bad known IP addresses interacting with buckets.
- PenTest tools are looking for S3 API operations invoked from Kali, Parrot, or Pentoo Linux system.
- Policy changes are interesting. They are likely associated with ATT&CK exfiltration tactics. An IAM entity has changed the policy associated with an S3 bucket so that it is no longer blocked from being public.
- Stealth findings detect S3 access logging being disabled. In a production environment, logging may be turned on, but it is rarely turned off. Seeing this happen is a red flag.
- Unauthorized Access is detecting known bad IP addresses performing Pub Bucket operations.

Kubernetes Findings[15]

Kubernetes findings are the newest group of findings. They are detecting potential techniques against Amazon's managed Elastic Kubernetes Clusters[16]. It does seem that the newer GuardDuty detections follow the naming conventions of Mitre ATT&CK tactics better than earlier findings. Likely, that is on purpose.

- Credential Access, Defense Evasion, Impact, Persistence, and Discovery all deal with known bad IP addresses performing an action against EKS. The documentation is not clear what the specific activity is, like documented in other GuardDuty findings. We are left to try and figure it out ourselves, or just put our faith in GuardDuty.
- Execution detects a command executed in a pod with the *kube-system* namespace. This is uncommon in normal Kubernetes operation, and thus considered suspicious.

- Policy, similar to S3 finding's policy techniques, are looking for changes to the Kubernetes clusters security posture. Dashboards exposed, anonymous access granted, or a service account was granted admin privileges. These privilege escalation techniques make it easier to gain outside access to the Kubernetes cluster and all of its secrets.

[1] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-active.html
[2] https://github.com/amrandazz/attack-guardduty-navigator
[3] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-ec2.html
[4] https://attack.mitre.org/tactics/TA0003
[5] https://attack.mitre.org/tactics/TA0040
[6] https://attack.mitre.org/tactics/TA0007
[7] https://unit42.paloaltonetworks.com/dns-rebinding
[8] https://www.kali.org
[9] https://www.parrotsec.org
[10] https://www.pentoo.ch
[11] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_remediate.html#compromised-creds
[12] https://attack.mitre.org/tactics/TA0005
[13] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-s3.html
[14] https://attack.mitre.org/tactics/TA0010
[15] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-kubernetes.html
[16] https://aws.amazon.com/eks

GuardDuty Data Sources

You can configure which data sources that GuardDuty will pull from

- CloudTrail: $4.00 per 1 million events/month
- VPC Flow Logs: $0.15 up to $1.00 per GB
- DNS Logs: $0.15 up to $1.00 per GB
- S3 Data Events: $0.20 up to $0.80 per 1 million
- EKS Logs: $0.20 up to $1.60 per 1 million

You decide which data sources GuardDuty will use

GuardDuty leverages five data and logging sources: CloudTrail, VPC Flow Logs, DNS Logs, S3 Data Events, and EKS logs. When enabling GuardDuty, the user can determine which of these log types it will rely on.

One item to note: the user does not have to enable VPC Flow Logs or build a specific trail in CloudTrail for GuardDuty to get the information. GuardDuty is interacting with the underlining core services and extracting these logs themselves. Users cannot get to these logs themselves through GuardDuty either, only the events.

Pricing:
https://aws.amazon.com/guardduty/pricing/

GuardDuty Terms

Detector: Every finding is associated with a detector—a regional entity with a unique ID.

Data Source: Origin or location of a set of data. For instance, CloudTrail or DNS.

Findings: A *potential* security issue.

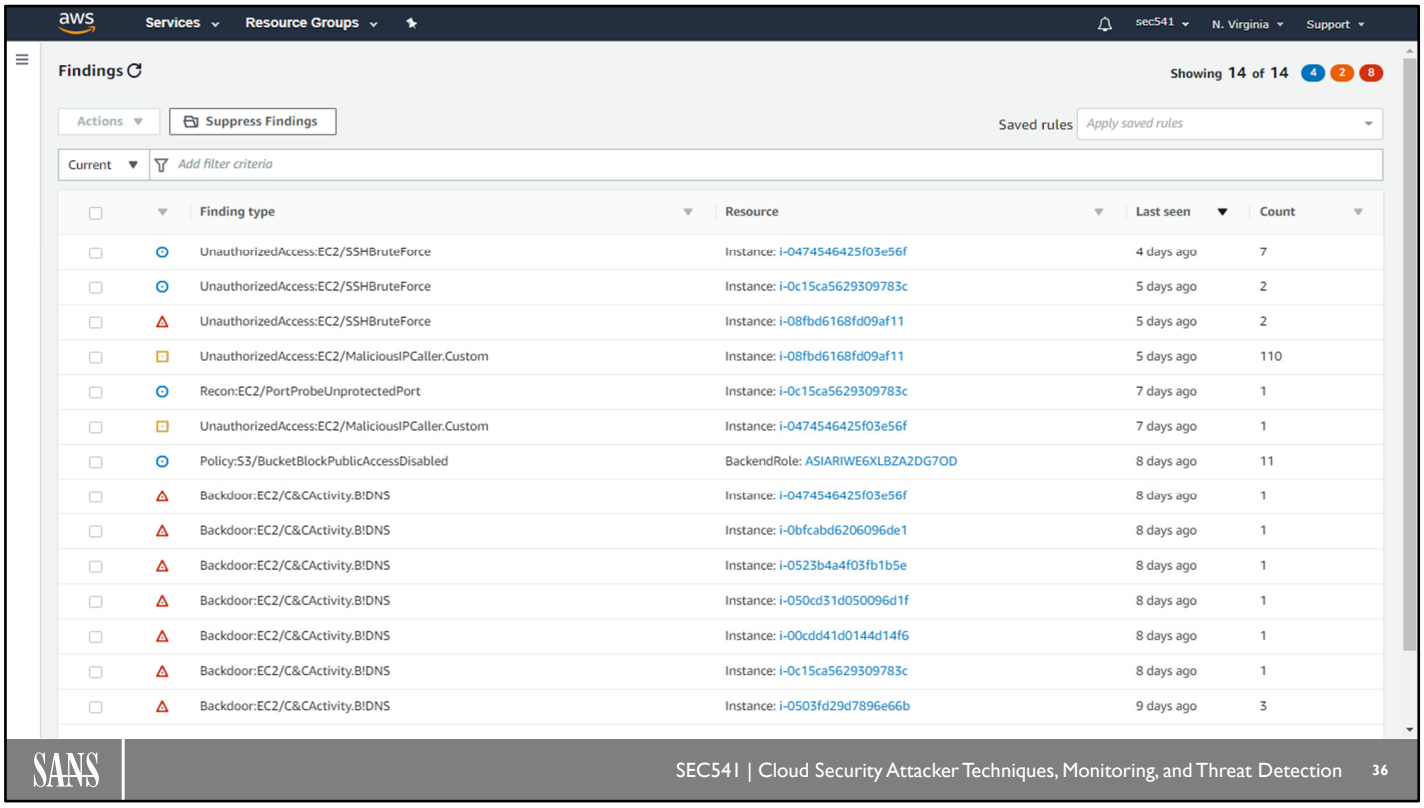
A Detector is the source of GuardDuty detections. As of now, there is a single, unique Detector in each account in each region.

A Data Source is one of the types of data that GuardDuty will use. The user can select which data sources to use, as a way to control costs.

A Finding is an event that AWS GuardDuty determines as a potential security issue.

Glossary of Terms:
https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_concepts.html

Findings:
https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_findings.html



On the GuardDuty main page, we see a list of events detected involving a particular resource, and how many times it has been observed.

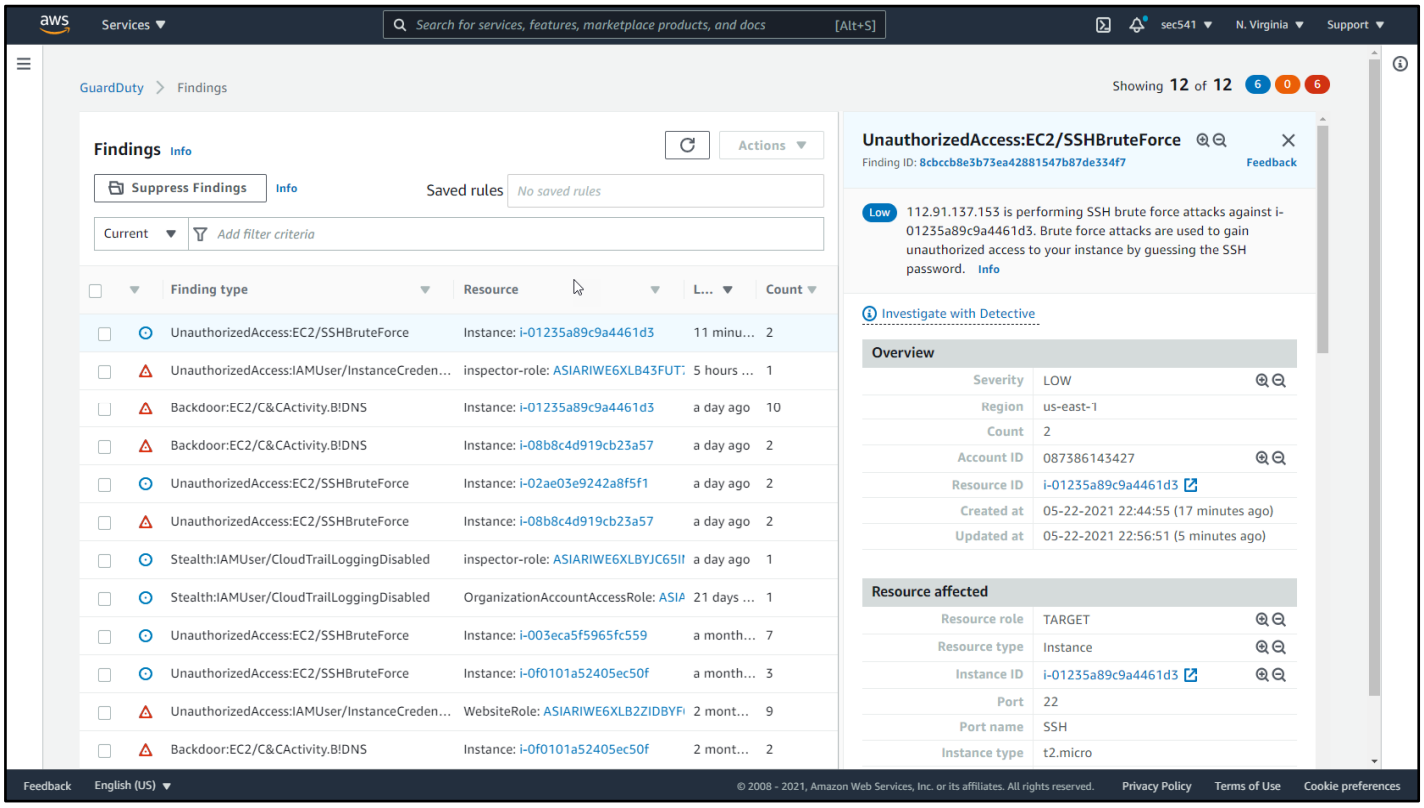
The filtering system can be used to look for findings with a certain property. Anything from Access Key ID used, to the Finding Type, to the network IP addressed documented in the findings. It's quite a long list.

One of the complaints about GuardDuty is how noisy it is. If you have 100 instances with SSH accessible from the internet, you likely will have 100 SSH brute force detections. It requires tuning based on what is acceptable in your environment. GuardDuty allows you to create a filter, such as "all SSH Brute force" attacks, and suppress those findings. Unlike the lame filtering in the CloudTrail UI, GuardDuty allows you to create filters and save them in the API. The filter is structured like a JSON object and is easy to use the *add filter criteria* box above. For instance, the filter to only display SSHBruteForce would look like:

```
 "type": {
    "Equals": ["UnauthorizedAccess:EC2/SSHBruteForce"]
}
```

The Boto3 documentation[1] has a good description of the filter structure in order to apply filters[2].

[1] https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/guardduty.html#GuardDuty.Client.create_filter
[2] https://docs.aws.amazon.com/guardduty/latest/ug/findings_suppression-rule.html



Selecting one of the findings will open up a pane on the right with a detail view, showing all the data associated with the finding. In the picture above, we are looking at an UnauthorizedAccess:EC2/SSHBruteForce finding. Maybe from our lab in section one, or possibly someone else just scanning. In this picture, we can see the severity level is LOW, this resource that was the victim of the attempted brute-force attack. Scrolling down in the pane we see other details collected about the perpetrator. Since this attack came from outside AWS, the only information we really have is the IP Address.

Notice how this detailed finding has a severity of LOW, but there are other SSHBruteForce findings that are HIGH? The next screen shows what that looks like.

GuardDuty Findings

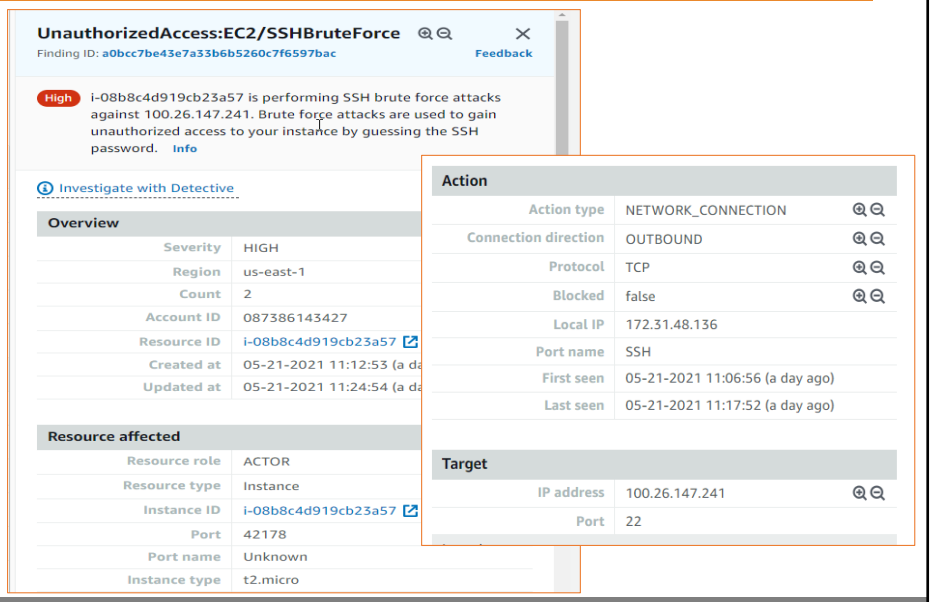
Basic finding information

The resource that is involved

Action is the activity type and source of the finding

TARGET or ACTOR: Info about the other party

Additional information about the finding



The GuardDuty's finding detail shows all the information about this finding that GuardDuty knows.

The basic information shows the severity, the total count of instances of this finding, the region and account ID, when it was created, and the last update.

The resources affected details which instance or service is involved in this finding.

The action describes how this finding was detected. For a CloudTrail based finding, it will include the service name, and the API call.

Target, or potentially actor, details anything known about the perpetrator or victim of the attack. Most of the time, it will be "actor", and may include the IP address that perpetrated the SSH Brute Force. However, in the event shown above, GuardDuty is detecting one of our EC2s that is attempting to do the brute force attack. So here it is showing who we are brute force attacking.

Additional Information is a catch-all for any other information about this finding, such as if it is archived.

Reference:
https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_findings-summary.html

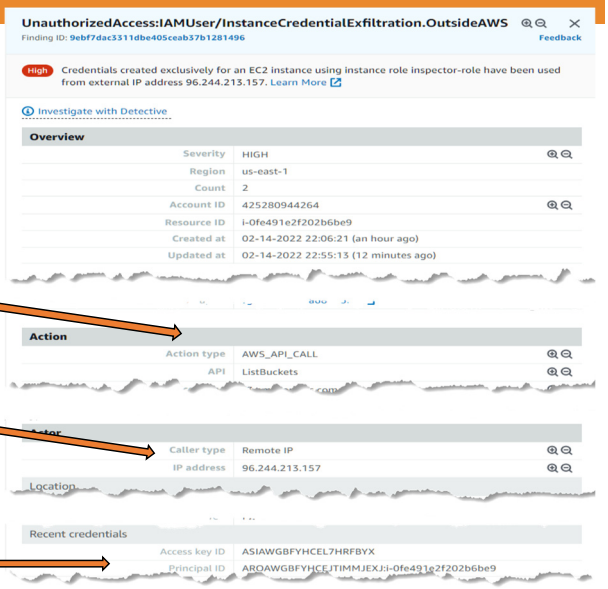
GuardDuty (3)

InstanceCredentialExfiltration.OutsideAWS means a role has been used outside of AWS

The API call that was made, tracked from CloudTrail

Where the originating IP was from

The Access Key ID is unique, and it can be tracked in CloudTrail



Remember that our Capital One attacker managed to extract the access key, secret key, and token from the EC2 and run it on their own machine, doing an AWS list-buckets, then starting to do a SYNC with every single S3 bucket in that particular account. Although a token can be used on a machine outside of AWS, a token for an EC2 IAM role should not.

GuardDuty now detects this in the UnauthorizedAccess:IAMUser/InstanceCredentialsExfiltration.OutsideAWS finding[1].

To combat this GuardDuty detection, an attacker could spin up their own AWS account and use the stolen credentials on their own EC2. So GuardDuty created the new "InstanceCredentialExfiltration.InsideAWS"[2].

What is great is there is a descriptive sentence at the top of the finding with everything you need to know about what happened.

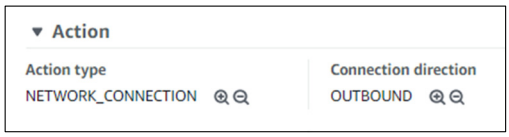
[1] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-iam.html#unauthorizedaccess-iam-instancecredentialexfiltrationoutsideaws
[2] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-iam.html#unauthorizedaccess-iam-instancecredentialexfiltrationinsideaws

GuardDuty Filtering

There could be a lot of findings. Filtering and Suppression can help.

- Saving a set of filters that focus your findings will make it easier to focus on a specific type of event. Resource, IP, Bucket, EC2 Instance, Username, etc.
- GuardDuty will generate a lot of findings; a filter can be turned into a suppression filter that will remove findings from your dashboard.

From the findings detail page, you can improve your filter just by clicking the icons



GuardDuty can be noisy, and the use of filtering can help make sense of everything. If you want to only look for events that meet a particular criteria, the "Filtering" tools can help[1]. This is especially important for "Threat Hunting", where we are looking to see if a particular threat attack has happened. "Let's look at all the SSH Brute Force and see if we can find a pattern?" The details of a finding have a plus and minus icon associated with some of the properties. Those can be used to refine the filter. Selecting the "plus" will show all matching findings. The minus will show everything but that matching pattern property.

Because GuardDuty can be noisy, and it may flag activity that is normal in your environment, or you may not wish to see reports, "suppression rule"[2] can be implemented. After creating a filter, it can be applied as a "suppression rule". Any finding that matches the filter will be "suppressed" or archived. Any future findings matching that suppression rule will also be archived..

GuardDuty can send findings to Security Hub, S3, or CloudWatch. If a finding is suppressed, it will not be sent along.

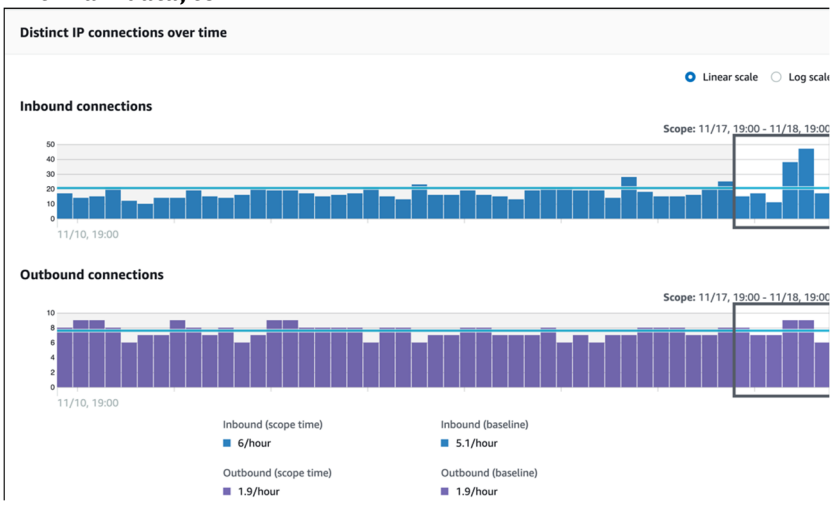
Regularly look at archived findings to see if your suppression rules may be overzealous.

[1] https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_filter-findings.html
[2] https://docs.aws.amazon.com/guardduty/latest/ug/findings_suppression-rule.html

Detective

AWS Detective allows security teams to evaluate events, coupled with raw data, to determine root cause

Released December 2019



AWS Detective[1] comes from the original Sqrrl team, purchased in early 2018[2]. AWS Detective bills itself as a managed threat hunting service, that generates visualizations of log data from logs and events from the AWS environment. Starting from security services such as GuardDuty, Security Hub, and Macie, we can pivot to Detective to conduct additional analysis with other, related AWS logs.

One of Detective's defining characteristics is the use of machine learning to detect "anomalies" in log data and event creation over time. Detective also uses graph theory and statistical analysis to bring value out of the data. Also, some Macie capabilities are being integrated more tightly into Detective.

Why only one slide on Detective? The authors have not seen many organizations using detective. Larger companies are hitting up against data quotas[3]. Most organizations are likely to be pulling data to a common SIEM, in which you can create histograms on your own. Detective will like grow in capability, but it is not the all-in-one hunting tool it hopes to be.

[1] https://aws.amazon.com/detective
[2] https://www.zdnet.com/article/aws-acquires-threat-detection-firm-sqrrl
[3] https://docs.aws.amazon.com/detective/latest/adminguide/regions-limitations.html

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. **EXERCISE: Metadata and GuardDuty**
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Lab 3.1 | Metadata and GuardDuty

Exercise Duration: 30 Minutes

Objectives

We will collect web server logs with CloudWatch and analyze them

- Look at the Sherlock Holmes Blog
- Research the SSRF attack
- Query the metadata service on the Inspector Workstation
- Attack Sherlock's Admin page
- GuardDuty investigation

In this lab, we are going to take a look at a blog webserver that Sherlock Holmes has setup. It contains some vulnerabilities we will use to perform an SSRF attack and query the metadata service.

We will also look at the GuardDuty service, see what it has been catching this week, and understand how to conduct an investigation with GuardDuty and the command line.

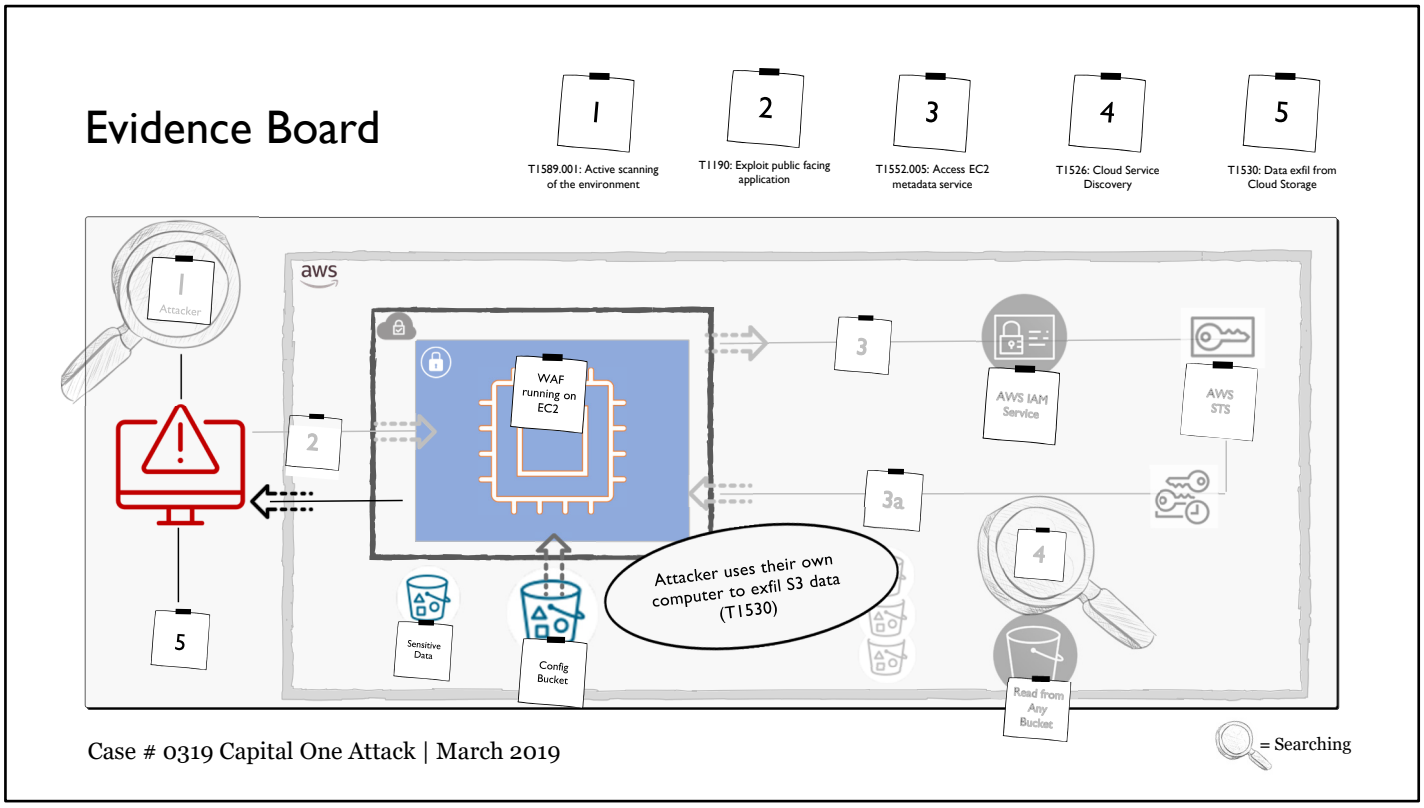
Course Roadmap

- 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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. **Cloud Inventory**
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.



When an organization gets an indication that something is amiss, maybe through a security tool alert, a customer reporting strange activity, or some partner with privileged knowledge notifies of a breach (FBI), it's time for the analysts to try and figure out what is at risk, where the vulnerabilities are, and the likelihood they were compromised.

The definition of "risk" is `Risk = Threat Probability * Vulnerability Impact.` When Capital One was notified that they may be the victim of a breach, their `threat probability` was high. To determine the risk, they must determine the vulnerability impact.

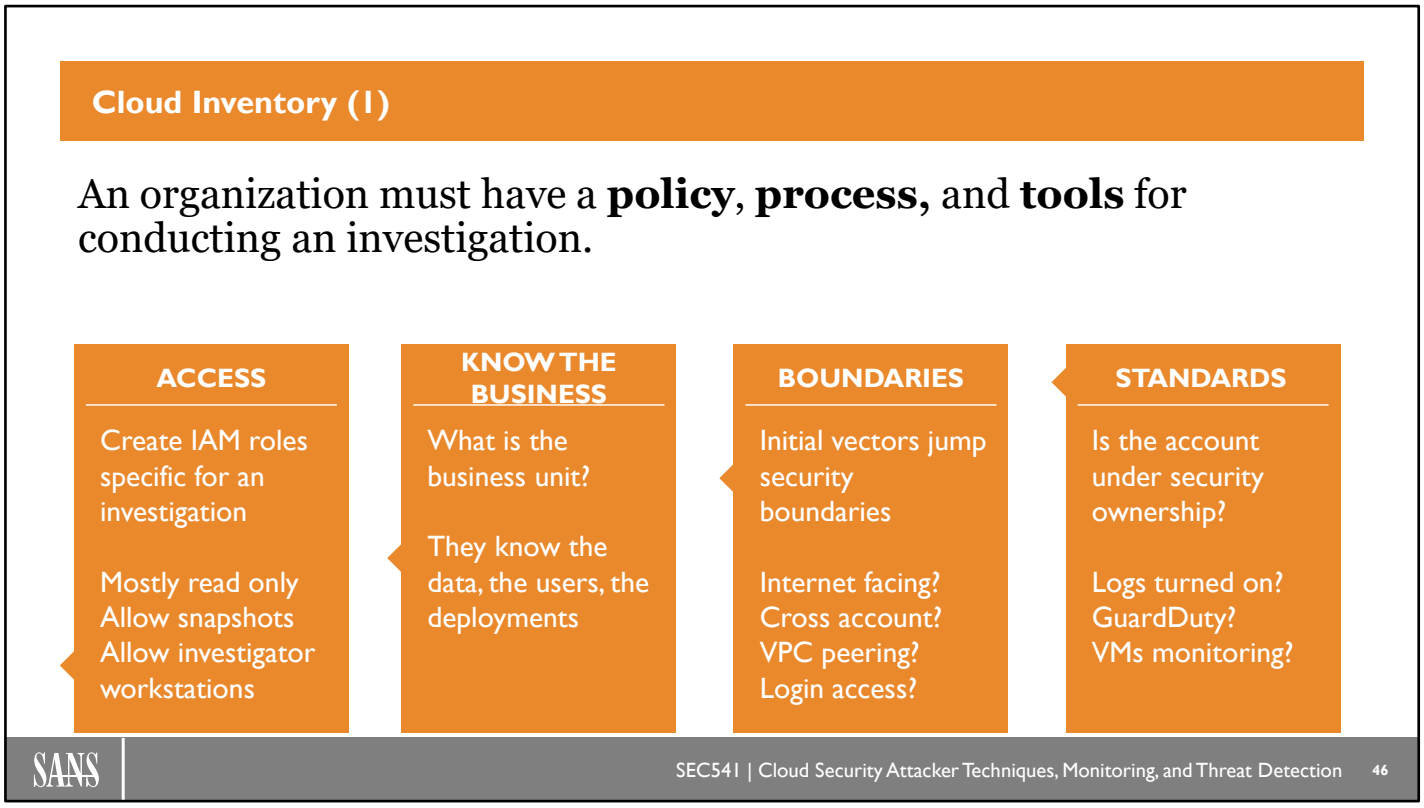
Is the attacker exfiltrating data from your environment? What data do you have that is at risk? Can the attacker actually get to the data? What other systems could they get access to?

Each organization runs their cloud operations differently. One of the benefits of a cloud environment is that we can bring the infrastructure operations and application development closer together. Some of you taking this class work in an area where the development teams are responsible for deploying and operating their applications, and the cloud security teams have various degrees of oversight. If there is a breach, you may not have a full mapping of the environment that is up to date and readily accessible.

In the Capital One case, it's likely when they were informed that they were losing data, so the S3 buckets would be the first resource to investigate. From Section 1, we talked about how CloudTrail does not, by default, track access to S3 buckets. Turning on CloudTrail data collection of S3 access is a lot of information and may not be deemed necessary, especially for less important data.

Credit cards? They are likely an increase in vulnerability impact and may have warranted special CloudTrails setup for S3 bucket capture. Knowing what logs you have, how resources are deployed, what "normal" looks like, and having the scripts and playbooks ready to go will mean the difference between cleaning up an attack, and allowing the attacker to get back in.

We do not know why an account with credit card information also deployed an internet facing web application with a misconfigured WAF. Capital One is a leader in cloud security, so this was likely a mistake, and mistakes will happen. They are happening right now, in your company. At this moment, a mistake is being made. Will you find it before the attacker?



The worst time to plan for an incident is middle of an incident investigation. Emotions are high, businesses are at risk, and it's Friday night and you realize you may have been breached. Just as the firefighter has their equipment on before they enter the building, you need to have the right equipment before the investigation.

Your investigators need the right accesses to the necessary environments to conduct the investigation. Organizations are increasingly splitting their AWS workloads into multiple accounts, or subscriptions, in Azure. The security team likely has one account, while the marketing team has another one. The security team, ready for an investigation, will need to assume a role that gives them read-only access to the resources. In addition to an investigation, if an investigator workstation needs to be stood up in the environment, snapshots need to be made, or evidence needs to be sent to an S3 bucket, then some write/create accesses will be needed. The SANS class FOR509: Enterprise Cloud Forensics and Incident Response[1] is a great class for forensics.

One reason for a separate security account is to separate the resources from the data. An investigator may need to know that the S3 bucket has credit card applications in it, but they do not need to read the data. In AWS[2] and Azure[3], a data key can be created and used to encrypt data in various data stores such as in AWS's S3, DynamoDB, or Parameter Store. Giving the investigator access to the resources, but not the keys, ensures that they cannot see sensitive data.

Larger organizations can have hundreds of accounts divided by business units. If it is Saturday afternoon and an alert says something fishy is going on in account 25523432, that does not tell the investigator very much. You will need to know what is in the account, who is the resource owner, and who is the data owner. The business owner may have to make the decision about remediating the attack once it is tracked down. But first, they may have to tell you where the important data is. Which S3 buckets has the credit card info vs. which buckets are keeping unimportant data. Knowing this ahead of time, or every tagging resources with data sensitivity levels, will significantly help with the risk assessment.

[1] https://www.sans.org/cyber-security-courses/enterprise-cloud-forensics-incident-response
[2] https://docs.aws.amazon.com/whitepapers/latest/efs-encrypted-file-systems/encryption-of-data-at-rest.html
[3] https://docs.microsoft.com/en-us/azure/security/fundamentals/encryption-atrest

One of the major benefits of a multi-account strategy in AWS[1] or Azure[2], is to create clear security boundaries around those accounts. For instance, the marketing team will have a lot of forward-facing web applications that may be of high risk (it's on the internet, it's risky!). But the data is meant to be shared, so it may not be as valuable. On the other hand, the account with credit card applications likely should be separate from high-risk resources, like internet facing systems. The previous paragraph recommended knowing the data sensitivity level—now we determine what is connected to that account. Who outside the business unit, security team, etc., may have access? Consultants, third party IT services? Are there trust relationships between VPCs? Are there S3 buckets or data forwarding services moving data out of the environment (into a higher risk account)? What is the internet access model? This helps determine the potential attack vector.

We see companies approach "security ownership" differently when moving into the cloud. Some organizations have defined rules for running production in the cloud, and they have established the right guardrails. Other organizations have some teams following best practices, and then others that have accounts for whatever business unit has a company credit card. Some security teams will work with the company accounting team to identify new cloud services popping up. If you are investigating an account that does not follow the standard security practices, then you will have to determine what logs are gathered, where they are going, how users access the environment, and a number of other "unknowns." If an organization is following your organization's best practices, then it becomes much easier, and your investigation can move along at a much more rapid pace.

This is not an exhaustive list, but SANS has a number of classes that deal with the process and program of incident responses. MGT520: Leading Cloud Security Design and Implementation[3] and SEC510: Public Cloud Security: AWS, Azure, and GCP[4] would be good ones.

References:
[1] https://docs.aws.amazon.com/whitepapers/latest/organizing-your-aws-environment/organizing-your-aws-environment.html
[2] https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/govern/resource-consistency/governance-multiple-teams
[3] https://www.sans.org/cyber-security-courses/leading-cloud-security-design-implementation
[4] https://www.sans.org/cyber-security-courses/public-cloud-security-aws-azure-gcp

Cloud Inventory (2)

We need to conduct an inventory of at-risk systems.

- Identify potential initial access—it can determine the attacker's blast radius (TA0001)
- Was the resource changed to facilitate evasion (TA0005) or persistence (TA0003) or escalation (TA0004)
- What would an enterprise discovery technique (TA0007) discover?

In the last section, we will discuss how to pull together multiple log types into a single analytic service to perform cross log analysis.

We need to start performing an investigation to determine what might have happened to the resource in question. Gathering an inventory of the systems in question may help to answer some questions to narrow down the risk exposure.

What was the initial access vector?
The initial vector may easily tell what was at risk, based on the IAM privileges they were able to gain. For Capital One, the initial vector was through an SSRF attack on a web app. If the web app is the likely culprit, then determine what roles are assigned to that web app. If user credentials were exposed in a GitHub project, then determining their accesses quickly is important, right after you invalidate their login. If an S3 bucket is exposed to anonymous, internet access, then it's likely only that data was exposed.

But what was in the S3 bucket? Stay tuned for the next section.

Was a resource changed?
A nightmare for an investigator is if user credentials allowed changes in the environment. In the Capital One instance, the attacker could only read S3 buckets, thus they did not make any changes. The Code Spaces and Tesla stories show what can happen when an attacker gains Update/Create/Delete privileges. Creating new resources can be a denial of wallet attack as with Tesla, or they can be used to create new backdoors as an Evasion[1] (TA0005) or Persistence[2] (TA0003) tactic.

The worst situation is an attacker that can use their accesses to Escalate (TA0004). An attacker gaining access to a system that allows modification of IAM roles/privileges can be detrimental to your business. If this is the case, assume the attacker owns everything.

[1] https://attack.mitre.org/tactics/TA0005/
[2] https://attack.mitre.org/tactics/TA0003/

It is important that we can easily build out a realistic look of the resources in question, and that likely will be a combination of web consoles and scripting.

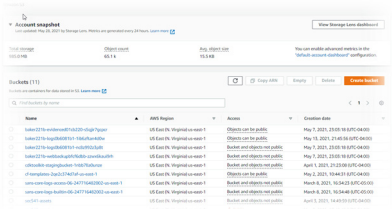
Research by Scott Piper into what resources in AWS are able to be exposed:
https://github.com/SummitRoute/aws_exposable_resources.

References:
Escalation: https://attack.mitre.org/tactics/TA0003/
Rhino Security: https://rhinosecuritylabs.com/aws/aws-privilege-escalation-methods-mitigation/

Cloud Inventory (3)

The Cloud Environments give us tools to see our resources through the GUI or the command line

Pros: The Cloud Services have nice GUIs that allow you to filter and sort objects, visualized related resources together, and click help make sense of the data.
Cons: You see the data the way the GUI was built for everyone, but not you specifically.



Pros: Command line and SDKs give you far greater control over data and the queries. It is the only way to go for automation.
Cons: The commands can be confusing, the data is vast, and it will take some time to learn how best to use it.

Investigators will need to become comfortable with scripting

Through the web console, AWS and Azure provide customers with a graphical user interface to visualize the environment. These environments can be extremely helpful to look up resources. The GUI also does a good job of educating the customer of the properties of a resource.

The GUIs tend to change over time as the commercial companies learn from the customers and understand their workflows. For AWS, there were times that GUIs would just change out from under you, and it could be confusing and difficult, especially for SANS course authors! AWS has gotten better at creating grace periods where customers can flip between the older and new console looks. The point is, the data behind the GUI likely never changed, just the wizard that AWS created to make things easy for you.

These wizards can make it very easy to create new resources, such as an EC2 with just handful of screens. But as your environment increases in size, security controls need to be implemented, and tools must be customized, then building/changing of cloud services may need to be automated.

This is the same with an investigation. Clicking through screens is really helpful when you are unsure of what you want to look for. But as you get better at investigations, you realize what data you need and what you do not, and you start wanting to automate data collection, then the software development kits, and AWS CLI becomes central.

The authors believe that efficiency in the cloud environments will happen when you are comfortable with the command line.

GUI
Pros: The cloud service providing the GUI can give some easy ways to filter and sort objects and their related properties. In AWS and Azure, we may see a virtual machine on the screen with its security groups listed. Those security groups are separate objects from the virtual machine. The GUI has grabbed the related objects and displayed them together, making it easier to see how those separate but related resources fit together.

Cons: You get the view and data that the company has put together, but it is not very customized. As you become more comfortable with the data, you will want more customized reporting. The GUI just does not support that as well.

Command Line:
Pros: The command line tools, and SDK's give you full control over the queries and how the data is returned. You will want to start pre-building your scripts to pivot across data that have multiple degrees of separation.
Cons: The SDK's can be confusing. The calls, and the data that is returned, is not always easy to understand or is even fully documented. It will take time to tinker and perfect.

Whatever approach you take, your investigators will need to be comfortable with some level of scripting. Some organizations try to bring all data back to their SIEM to use the tools they are most comfortable with. But at some point, you will need to reach out into the environments and interrogate them directly.

Command Line (1)

We have used the AWS command line this week, as a way to create resources and perform log analysis in CloudWatch, among other things.

They will use "describe" and "list" key words

```
cloud> aws ec2 describe-instances --output table
```

```
cloud> aws iam list-policies --output yaml
```

```
cloud> aws s3api list-buckets
```


A quick chat about the command line we have used in the labs this week. It is sometimes confusing why a list is used rather than a describe. There is often discrepancies between different AWS services that seems arbitrary. It is usually a byproduct of fast and parallel innovation, whereas the Azure services feel a bit more planned together.

The command line can be used to return text, JSON, a table, or now YAML format.

When creating scripts with the command line, you will likely switch between the output types. In our labs, we have made heavy use of "text" so that specific IDs can be saved into variables. JSON is easily passed to another script, and into jq. YAML and table are good options to allow humans to read, especially when you start to build up a set of scripts and tools to take inventory of a resource.

AWS has announced the "AWS Cloud Control API"[1], which attempts to standardize the create, read, update, delete and list of AWS and third-party resources. The AWS command line tool supports this new API, and it likely will make things a bit easier to code, by giving similar feel in the commands[2].

[1] https://aws.amazon.com/cloudcontrolapi
[2] https://docs.aws.amazon.com/cloudcontrolapi/latest/userguide/getting-started.html

Command Line (2)

AWS and AZ command line tools treat each resource separately. For instance, although the GUI shows everything, you may need multiple queries to go from EC2 to Security Group Ingress

```
1  SG=$(aws ec2 describe-instances --filters Name=private-ip-
2  address,Values=10.0.0.157 --query
3  Reservations[].Instances.SecurityGroups[].GroupId --output text)
4
5  aws ec2 describe-security-groups --filters Name=group-id,Values=$SG --query
6  'SecurityGroups[].IpPermissions[].[FromPort, ToPort, IpProtocol]' --output table
```

This was a two-step process. IAM roles are many more.

EC2 -> Instance Profile -> Role ->Policy

```
-------------------------
| DescribeSecurityGroups |
+------+--------+--------+
|  22  |  22    |  tcp   |
|  5000|  5000  |  tcp   |
|  -1  |  -1    |  icmp  |
+------+--------+--------+
```


In this example, we know that an EC2 of IP address 10.0.0.157 is acting funny. So, let's figure out what are the values for the *Security Group Ingress*.

We can query AWS to give us just the ID of the security groups attached to the EC2 with our known IP address of 10.0.0.157. Remember, this is querying live data for a live system. If you are investigating something that happened a week ago, and your VMs are elastic, then that system may no longer exist. You would have to rely on historical data in logs to retrieve this information. Later in this section, we will talk more about how to leverage AWS services to collect and store configuration information.

```
SG=$(aws ec2 describe-instances /
      --filters Name=private-ip-address,Values=10.0.0.157 /
      --query Reservations[].Instances.SecurityGroups[].GroupId /
      --output text)
```

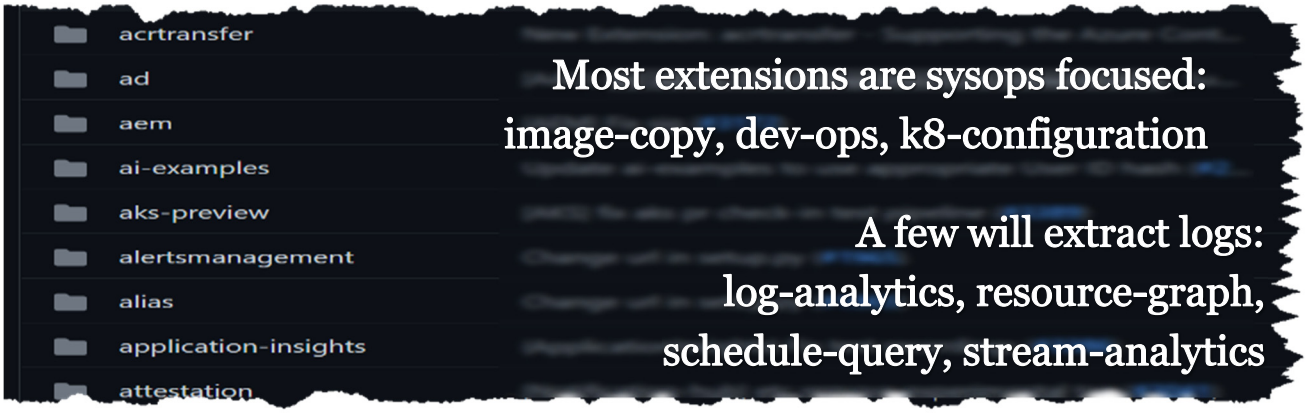
We have run "describe-instances" to return the security group ID of all resources with a private IP of 10.0.0.157 and assigned the IDs to the variable SG.

We can then use the SG variable to query the EC2 service using Describe Security Groups with a filter by group-id. This returns our security group information.

References:
https://docs.microsoft.com/en-us/cli/azure/
Describe Instances: https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/describe-instances.html
Describe Security Groups: https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/describe-security-groups.html

Command Line (3)

Azure's command line supports "extensions", or Python programs executed from the CLI, to help automate.



You can craft your own extensions to help automate tasks

In AWS, one can either create a script built off the CLI or write a program in one of the SDKs. Azure's command line tool allows you to create an "extension". These extensions, written in Python, are called from a command line, but are able to perform more complicated interactions with Azure. They give you a great way to build and share repeatable tasks but launch from a command line tool. The current set of Azure extensions tend to be heavily focused on system operations. A larger organization will likely want to build their own customized extensions that meet their needs.

SysOps:
https://github.com/Azure/azure-cli-extensions/tree/master/src/image-copy
https://github.com/Azure/azure-cli-extensions/tree/master/src/k8s-configuration

Log Analytics:
log-analytics extension: https://github.com/Azure/azure-cli-extensions/tree/master/src/log-analytics
resource-graph: https://github.com/Azure/azure-cli-extensions/tree/master/src/resource-graph
schedule-query: https://github.com/Azure/azure-cli-extensions/tree/master/src/scheduled-query
stream-analytics: https://github.com/Azure/azure-cli-extensions/tree/master/src/stream-analytics

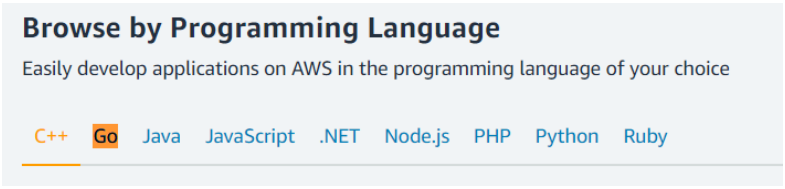
https://docs.microsoft.com/en-us/cli/azure/azure-cli-extensions-overview?view=azure-cli-latest
https://docs.microsoft.com/en-us/cli/azure/azure-cli-extensions-list?view=azure-cli-latest?WT.mc_id=docs-azuredevtips-azureappsdev

Building Azure CLI extensions:
https://microsoft.github.io/AzureTipsAndTricks/blog/tip200.html

Boto3 and SDK

AWS's solution for automation is through the SDKs, as discussed on Section 1 of this class. Boto3 is the Python based SDK, and the most popular to use.

The Go SDK is growing in popularity for automation infrastructure.



Your AWS automation will quickly outgrow the CLI+Bash

In AWS, writing complex automations in the CLI can quickly become frustrating. The author wrote an infrastructure build/teardown application for a project, and quickly wished they had used a full-fledged SDK. SDK's can handle the JSON objects easier, allow for rich inputs, better error checking, and more benefits from a programming language.

It becomes apparent that your system operations, infrastructure management, and threat analyst teams will need to grow their scripting expertise as the cloud environment becomes larger, more automated, and performant.

Most of the examples online tend to be in Boto3/Python, so that is a great place to start learning if you are not a coder. More examples are coming out in Go, especially if you are using Go for other automations. However, it is best to pick the language that your organization is most comfortable with. Don't have a preference? The author recommends Boto3 or Go, followed by JavaScript.

References:
Boto3: https://aws.amazon.com/sdk-for-python/
Go: https://aws.amazon.com/sdk-for-go/
C++: https://aws.amazon.com/sdk-for-cpp/
Java: https://aws.amazon.com/sdk-for-java/
JavaScript: https://aws.amazon.com/sdk-for-javascript/
.NET: https://aws.amazon.com/sdk-for-net/
PHP: https://aws.amazon.com/sdk-for-php/
Ruby: https://aws.amazon.com/sdk-for-ruby/

Tags

Tags are Key/Value pairs that are attached to a cloud resource

- The key and value can be up to you, depending on cloud service limitations.
- Adding investigation specific tags can help others collaborate, help build reports or automated tools, and can change how IAM roles behave
- Tags can be queried across resources, making it easier to perform an inventory of VM, users, IAM Roles, S3 buckets, etc.

Note: You may want to use non-specific tag names, so as to not tip off the attacker.

Tags is a way to add metadata to your AWS or Azure resource that is user controlled and can greatly help in an investigation. Tags themselves have no special magical property in and of themselves, but they can be used to change the way IAM policies work, how your automations use the resource, and how you bill sub teams.

A tag is a key/value pair you assign to a resource, with the value being optional. You define them both and they can be anything. Organizations use tags for different purposes, and they are completely up to you.

In AWS, a tag with the key of "Name" tends to be displayed in the AWS GUI as a resource's "Name", but there is nothing special about that particular tag key. It is simply a default in the GUI.

In our AWS deployments in class, CloudFormation has created three tags for the VMs:

- `aws:cloudformation:logic-id:` A unique identifier that the stack uses for that virtual machine.
- `aws:cloudformation:stack-name:` The name of the CloudFormation stack
- `aws:cloudformation:stack-id:` A unique ARN of the stack resource that created the virtual machine.

AWS is upgrading tagging throughout the resources so that tags can be applied when a resource is created, as opposed to after creation. This is important for automating code to launch with an object is created.

However, tags can help greatly when conducting an investigation, especially if multiple teams are involved. We know from previous labs that Sherlock VM has activity. When the investigation starts, we may assign some unique number to track this activity. Usually, it is the ID for the ticket you create in your ticketing system. We will name our investigation "ELEMENTRY-32", adding the tag with key "ELEMENTRY-32" to everything that needs to be investigated: the VM, the data stores, or S3 buckets that the VM has access to, even the IAM Role assigned to that VM.

Why are we talking about this? CLIs and SDKs allow you to query for all resources that contain a particular tag, rather than using multiple “Describe” commands in the CLI.

$ aws resourcegroupstaggingapi get-resources --tag-filters Key=aws:cloudformation:stack-name,Values=setup-lab --query ResourceTagMappingList[].ResourceARN

This will output all the resources that the CloudFormaton created in the setup-lab stack.

$ az group list --query “[?tags.SECURITY == ‘Quarantine’]” –o table

This will output all the resources that have been tagged with a key of SECURITY and a value of Quarantine

References:
AWS: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Using_Tags.html
Azure: https://docs.microsoft.com/en-us/azure/azure-resource-manager/management/tag-resources?tabs=json

AWS Config (1)

AWS Config was introduced in November of 2014, originally focusing on resource inventory and configuration history.

It added support for conformance and configuration rules, allowing customers to determine if resources are meeting a set requirement.

Set up AWS Config to take regular snapshots of **all resources** in the environment and deliver them to a security team control S3 bucket.

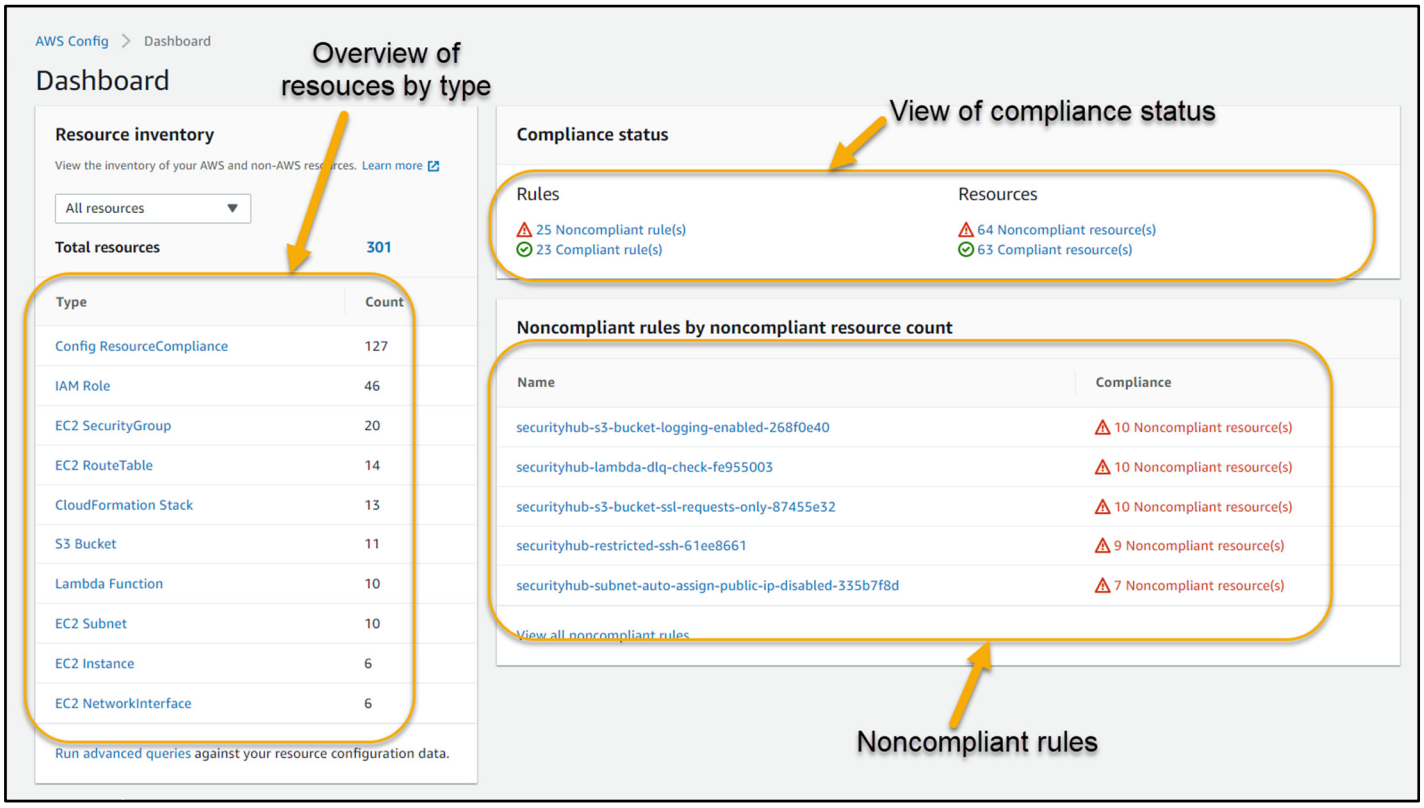
AWS Config was introduced in November of 2014 as an AWS resource inventory, configuration history, and configuration change notification service. AWS has since expanded AWS Config to support integration of compliance and conformance packs with configuration rules that lets you define exactly **how** resources should be configured and perform alerts or automated response actions when the rule is broken.

For investigating and gathering resource inventory, AWS Config is a fantastic service to show you the current, and historical configuration of particular resources.

Security teams collecting historical data about AWS environments are probably collecting CloudTrail, and likely VPC Flow Logs, but Config can be overlooked. AWS Config takes a snapshot of every monitored resource in a giant JSON file and stores it in an S3 bucket. The snapshot is configured to happen as frequently as every hour, up to twenty-four hours. Often, the receiving bucket is stored in the security team's account, and the individual development organizations do not see the data. However, we will show, like in CloudTrail, how to query the data directly.

AWS Config also can integrate with AWS System Manager to record configuration changes to software running on Amazon EC2 systems.

References:
https://docs.aws.amazon.com/config/latest/developerguide/deliver-snapshot-cli.html
https://aws.amazon.com/config/features/
https://docs.aws.amazon.com/config/latest/developerguide/config-concepts.html



The main dashboard of the AWS Config page shows its bias as a compliance service.

A “compliant rule” is a rule that specifies some configuration requirement for a service. For instance, “S3 buckets must have default encryption.” The rule will be compliant if all S3 buckets in the account have default encryption turned on.

AWS has a large number of standard rules already defined and available to be applied. The customer can also create their own custom rules using AWS Config and Lambda. A Lambda function performs the evaluation and returns the results in a standard format that AWS Config recognizes.

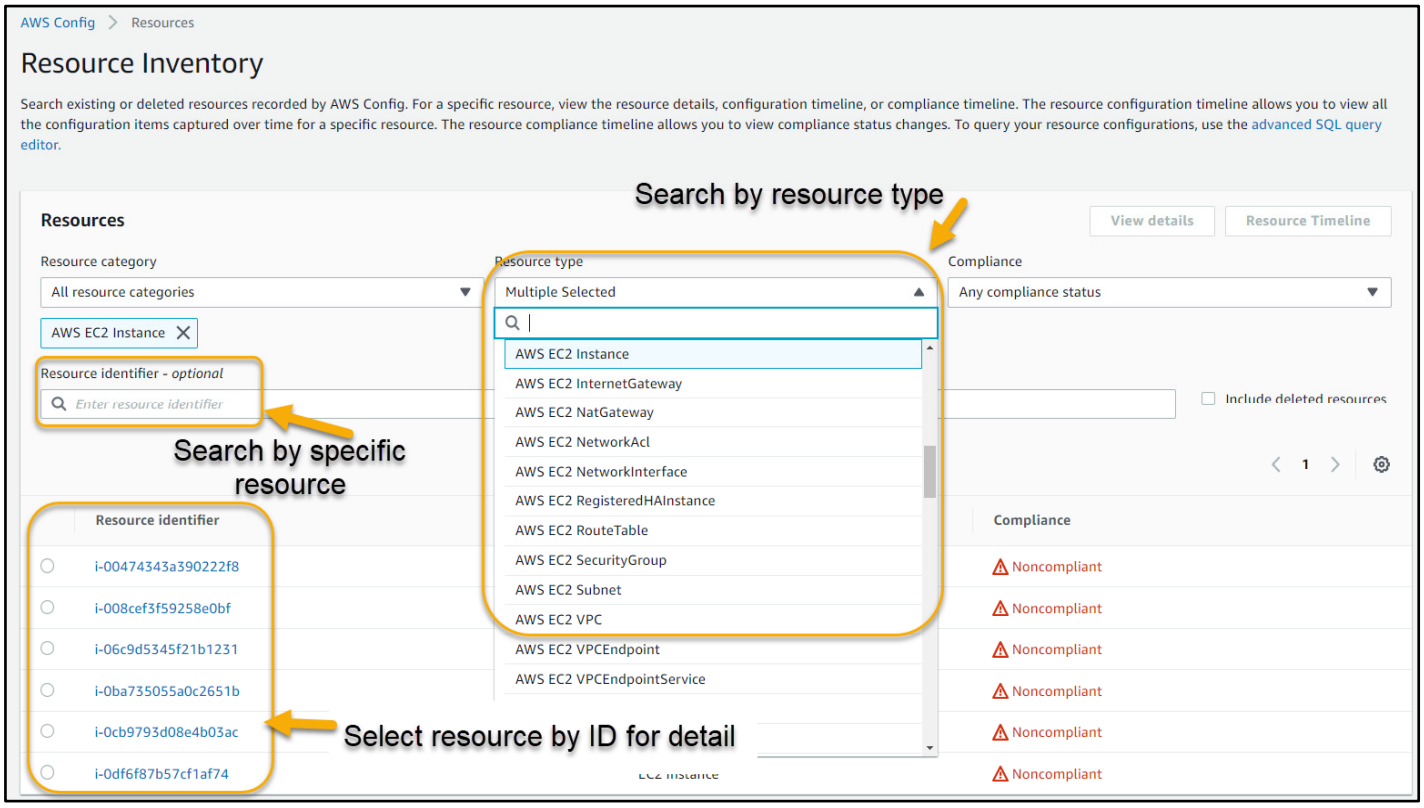
On the left-hand side, you get a snapshot of the resources being tracked in AWS Config. The compliance rules are running against the monitored resources.

As you can see from the snapshot, SecurityHub is starting to integrate with AWS Config by setting up rules that are part of SecurityHub’s security compliance checking.

There are limitations to AWS Config that some organizations feel are too constricting. It may not check all your resources exactly the way you want. When AWS Config rules were first introduced, they were limited, and larger organization just built their own tools. There is no real secret sauce that AWS Config provides that you could not put the time into making yourself. But what AWS Config does well is to interconnect to the rest of the AWS ecosystem. Non-compliant rules can fire off alerts or automated actions. SecurityHub shows a roll up of AWS Config and other services. Custom rules can be built and integrated with this visualization.

However, for this class, we are more interested in AWS Config inventory service rather than compliance.

Reference:
https://docs.aws.amazon.com/config/latest/developerguide/evaluate-config.html



Selecting the “Resources” option takes us to the Resource Inventory page. You turned on AWS Config in Section 1, and sometime later it started taking snapshots of resources in the environment to be able to tell a story.

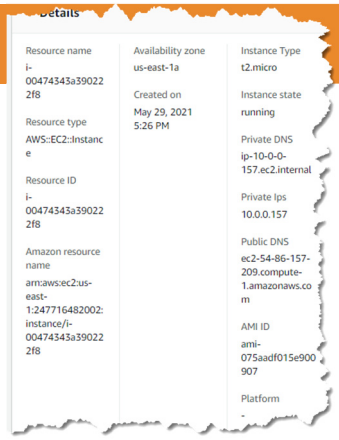
At the top, you can filter by resource type and compliance level. Or, if you know the resource identifier, you can enter it into the search box. That is the limit to the filtering for this page.

With every GUI in a cloud provider, the company has to figure out what the customer’s initial starting point will be. For compliance, you may want to see all the high priority compliance rules that have noncompliant resources, so you can start taking care of them. The other initial starting point may be “I need to figure out what is going on with a resource”. Maybe your GuardDuty detected unusual traffic. Maybe a metric you setup to monitor a CloudTrail event has fired. In these cases, you likely know the resources you want to investigate. In threat analysis, that is likely how we would use AWS Config.

Reference:
https://docs.aws.amazon.com/config/latest/developerguide/resource-config-reference.html

AWS Config (2)

Selecting an individual resource gives you full current configuration data, including the JSON object



The JSON snapshot is not the same as **aws ec2 describe-instances**

It has a more standard schema, and provides a "Relationships" JSON array

After selecting the individual resource to evaluate, you are presented with a details page with common properties like resource name, type, deployed zone, etc.

You are also provided with the same JSON data as the last snapshot taken of that resource. Using an EC2 as our example, the JSON includes a "configuration" property that has the same information as found when performing an aws ec2 describe-instances. There are some strange differences. In the describe-instances call, the property is "ImageId", while the AWS Config result is "imageId". When doing development, you will see these kinds of minor differences in resources that seem arbitrary.

However, the AWS Config also provides some nice additions in the snapshot. The "tags" property is different.

```
Calling describe-instances, the tags will be:
"Tags": [
  {
    "Key": "aws:cloudformation:logical-id",
    "Value": "SherlockBlog21D49B3A"
  },
  {
    "Key": "Name",
    "Value": "SherlocksBlog"
  }]

In the AWS Config, the tags will be
"tags": {
   "aws:cloudformation:logical-id": "SherlockBlog21D49B3A",
    "Name": "SherlocksBlog"
}
```

Why the differences? The author does not know. But be mindful of these changes between services.

Another addition found in AWS Config is a **`relationships`** property[1]. It pulls out the resources related to your target resource and provides them in an array for easy querying. For instance, the Security Group is a separate resource that is attached to the EC2. So, the relationship is:

```
"relationships": [
  {
    "resourceType": "AWS::EC2::SecurityGroup",
    "resourceId": "sg-0595b30667bf99f0d",
    "relationshipName": "Is associated with SecurityGroup"
  },
]
```

Keep in mind, not all resources are tracked in AWS Config—there will be gaps[2].

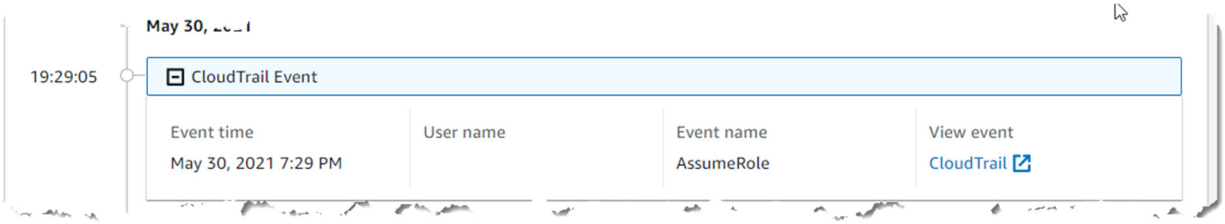
Note: During an investigation, you may be creating scripts to tease out a resource to go investigate. Your scripts could automatically generate the link to the AWS Config details page. Here is the standard URL to use to go right to a particular resources AWS Config page.

https://console.aws.amazon.com/config/home?region=<region>#/resources/details?resourceId=<ResourceID>

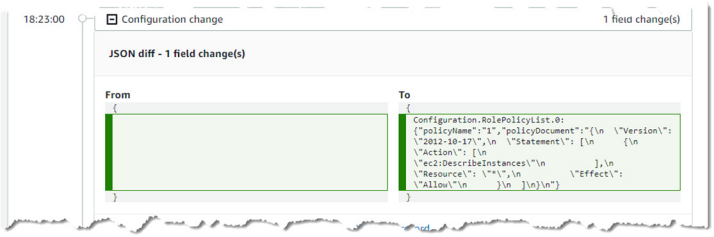
[1] https://docs.aws.amazon.com/config/latest/APIReference/API_Relationship.html
[2] https://docs.aws.amazon.com/config/latest/developerguide/resource-config-reference.html

AWS Config (3)

Config also provides a "Timeline" of events on those resources



CloudTrail events, compliance rules, and configuration change are gathered and provided in one easy place



A cool feature of AWS Config is a timeline of the changes in a resource, showing when a resource had a success or failed AWS Config compliance rule, a change in its configuration, or a CloudTrail event associated with that resource.

The CloudTrail events could get quite lengthy for some resources, and there is no way to filter the CloudTrail events on this GUI. Best to move over to Athena to do that properly.

However, you can easily see configuration changes on that resource. Or, better yet, you can see that a configuration change has happened, but not all resources are that easy to visualize. For the EC2, you see the differences in the snapshot of the instance, basically performing a "linux diff" on two JSON documents.

However, this diff can help you pinpoint *what* changed, and the CloudTrail can help you understand *how* it changed, but you will still need to figure out the *why* it changed.

AWS Config (4)

AWS Config is a sometimes-overlooked tool in your toolbox

- Provides GUI and command line access to resources and relationships
- Gives a timeline of changes, details of changes, and can be used to help tell the story of the resource
- As an investigator, the AWS Config service coupled with CLI and SDK can help you automate your data gathering about suspicious resources, but you will want to customize to fit your process

Elastic environments are ever-changing; adjust the investigation

AWS Config is a good tool in your toolbox to help you gather details about the resources in question. Automating report building and evidence gathering can be simplified using the AWS CLI or SDK.

The GUI's timeline page has been improved over time and will likely continue to do so in the future. At the moment, the GUI is not the greatest investigation tool, as it is hard to filter or search the events you want to look at based on some complex set of properties. Compliance driven workflows seem to be driving the innovation in Config. However, being able to pull a historical snapshot of changes over time could give an investigator insight into some problematic changes.

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. **EXERCISE: Cloud Inventory**
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Lab 3.2 | Cloud Inventory

Exercise Duration: 30 Minutes

Objectives

We will collect web server logs with CloudWatch and analyze them

- Understand how the use cases we are investigating this week may require us to investigate our inventory and potential resource changes
- Perform inventory queries with the AWS CLI
- Use the AWS Config service to see configuration history of a resource
- Use the AWS Config GUI to get an even better view of changes

In this lab, you will be using the CLI and AWS Config to integrate your environment.

You will look at how to build up AWS CLI commands that will work across multiple environments.

You will use the AWS Config through the command line to extract information about the resources.

You will then use the AWS GUI.

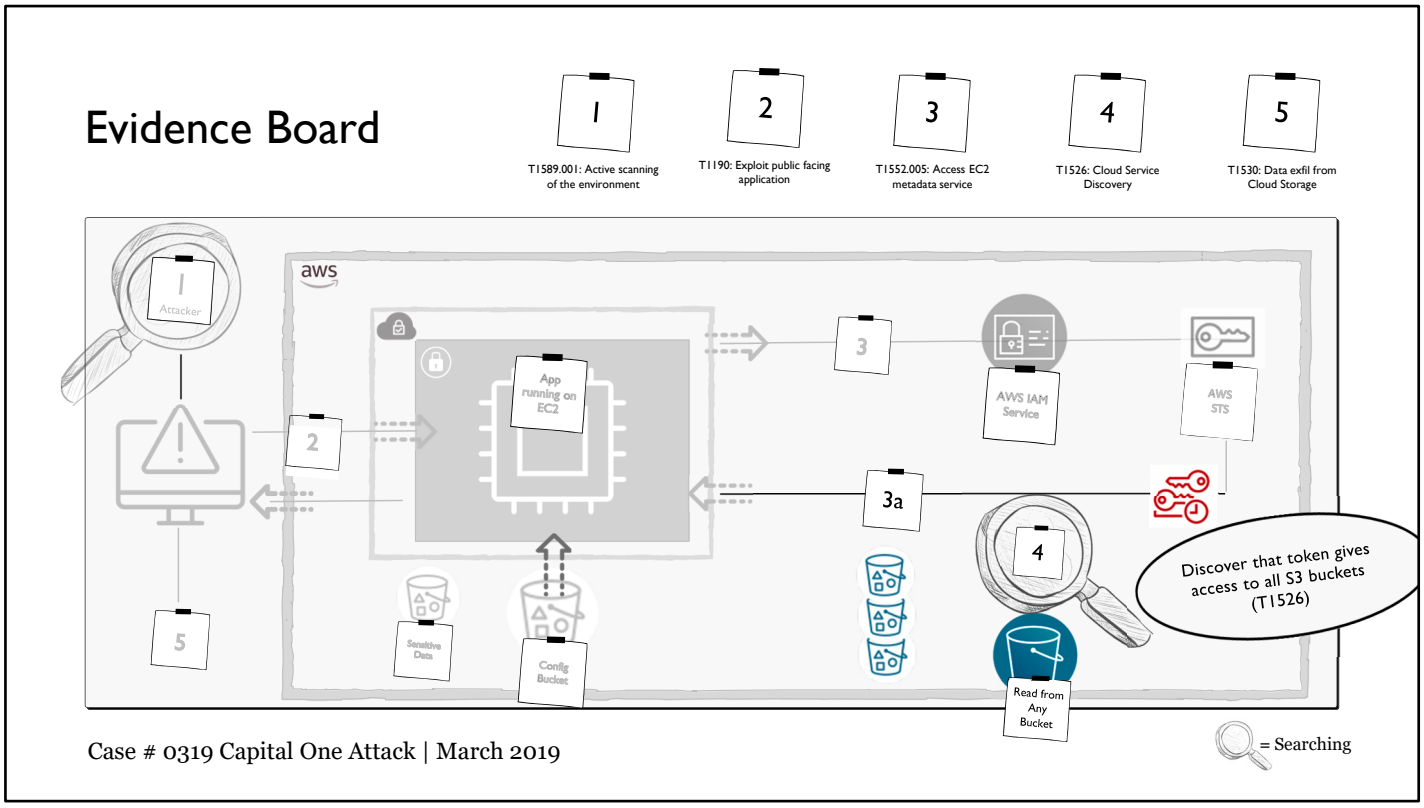
Course Roadmap

- 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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. **Data Discovery**
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.



After acquiring the temporary token and key information, the attacker can now start conducting their own investigation through a cloud service discovery (T1526)[1].

We see this cloud service discovery a number of times already. Without it, the attacker really does not know what they have access to, or even what is out there. It would be difficult to perform any kind of passive analysis of the environment, such as in network sniffing (T1040)[2].

It is much easier just to ask AWS or Azure what they are allowed to do.

From the cloud service discovery, this time conducted on the attacker's own machine, they discovered that they had read access to some, or all the S3 buckets that are owned by the account. This was a large violation of the "least privilege" characteristics that should be deployed in cloud environments[3]. Again, we are making assumptions here about what happened. When the author talks to an organization about "least privilege", they paint this picture. A virtual machine needs to pull configuration files from an S3 bucket. The app team does not know what the name of the bucket will be ahead of time, and they must get an IAM role created. Therefore, they make the IAM role be able to read from all buckets. That IAM role will now allow that VM to pull from the configuration bucket and all buckets in that account. Maybe that is what happened for Capital One.

Right after a cloud service discovery to determine what services are available to the hacker, they will likely do a cloud infrastructure discovery (T1580)[4] which determines what exact buckets they have access to and if any of them sound interesting.

Now, the company has been called and they know there is a breach. They likely know the IP address. They used AWS config or ran some command line tools to see how that instance is configured. They may realize that the instance has an IAM role that allows read access to the S3 buckets. Now the question is, was there any data that was sensitive? Remember our risk formula is Risk = Threat Probability * Vulnerability Impact. The threat probability is through the roof since they were called with proof of the attack, but what is the impact? What data could have been loss? Maybe the buckets are named automatically by terraform or CDK/Cloudformation like *thebucket123456*, without real knowledge of what is in them.

Or maybe we have not gotten notice of a breach and we want to do some preventative hunting. With hundreds of accounts managed my multiple teams all across the world, how do you know where sensitive data might be?

Or maybe we want to look for data that has been added to the environment. As with the case of Code Spaces, maybe a backdoor has been added to an application, or staged in a data repository. There have been attackers who have staged exploitation into data storage to be accessed by multiple resources. They have also been known to stage data ready to be exfilled, called the Data Staged technique[5].

Time to do some data hunting.

[1] https://attack.mitre.org/techniques/T1526
[2] https://attack.mitre.org/techniques/T1040
[3] https://docs.aws.amazon.com/IAM/latest/UserGuide/best-practices.html#grant-least-privilege
[4] https://attack.mitre.org/techniques/T1580
[5] https://attack.mitre.org/techniques/T1074

Data Hunting

- Indicators of Compromise (IoC) may be discovered in your cloud environment
  - Where else could they be?
  - You may need to hunt for them!
  - Cloud makes this slightly trickier than on-premise
- What will the data **look like**?
  - Format of your data must be considered
  - Understanding **regular expressions** can help tremendously!
  - Example: **`C:\Windows\System32\{16-printable-chars}.exe`**
    Regex: **`C:\Windows\System32\[0-9a-zA-Z]{16}.exe`**

Data hunting will be our attempt to discover potential Indicators of Compromise (IoC) elsewhere in the organization. To perform a successful search, we must know what the data looks like as it is stored on the systems or within the services. This simplest data to locate would in the form of simple strings (e.g., IP addresses, domain names, comments, or commands in malicious applications).

Some data types, however, get a little more complex and may require knowledge of regular expressions to successfully search for data. For instance, there may be a discovered attack technique which leverages a dropper which downloads a file to the `C:\Windows\System32` directory with an obfuscated name in the form of a 16-character, printable ASCII string which adds the .exe extension. With this knowledge, you could create a regular expression that could aid in the location of this malicious file.

You can see an example of this file path above which, at a high-level is explained as:
- `C:\Windows\System32`: The literal directory path to be searched
- `[0-9a-zA-Z]:` Square brackets indicate an OR operator for the characters or character ranges inside of it. In this case, it is specifying any number or mixed-case letters.
- `{16}:` Curly braces with a number indicate how many of the proceeding arguments to consider a match. In this case, exactly 16 letters or numbers in a row.
- `.exe:` The literal extension

Discovering Data on Volumes/Disks

- IoCs may end up on **cloud volumes or disks**
  - Must engineer a solution to search for this data
  - You may already have methods available to perform this search:
    - **Data Loss Prevention (DLP) suite**
    - **Configuration Management (CM) tools**
    - **Vulnerability Scanner**
- Once data is discovered, follow-up actions may include:
  - **Log:** Create an alert that the data was discovered
  - **Move:** Move data to approved location
  - **Quarantine:** Move data to a controlled folder/location for analysis
  - **Delete:** Remove the data

When using Infrastructure as a Service (IaaS), hunting for these IoCs in the underlying instance volume or virtual disk is a necessity. To perform this search, you can oftentimes leverage existing toolsets that are already deployed and can perform these tasks. Some examples that will be discussed in more detail are:

- Data Loss Prevention (DLP) suites look for behaviors that indicate that data is leaving the organization, either by inspecting the data itself, or seeing a change in the volume of data.
- Configuration Management (CM) tools evaluate how resources are configured and detect changes from what is approved. AWS Config is a good cloud specific example of this. Cloud Custodian is an open-source version, originally from Capital One[1].
- Vulnerability Scanners look at files and finds patterns that are known bad.

The examples above may even be, themselves, cloud-hosted. For example, to perform configuration management in AWS, your operations team could be leveraging AWS Systems Manager (SSM) or Config.

Once the data is found using one of these tools, it must be determined if that instance volume or virtual disk is permitted to process or store that data. If not, several options are available:

- Create an alert or log entry that the IoC was found and on which particular volume or disk
- Move the IoC to an approved storage location
- Move the IoC to a quarantine location in which the data can be reviewed by an analyst
- Simply remove the offending data

[1] https://cloudcustodian.io/

Extend Your DLP Solution

- DLP solutions provide:
  - **Protection** of sensitive data
  - **Automation** of safeguards to protect sensitive data
  - **Reporting** of violations
- Very robust policies can be created to detect IoCs as well
- Policies can include:
  - Wordlists
  - Regular expressions
  - Two or more terms within the same file, paragraph, or sentence
- Supports analysis of several file types

A technology that was built with protecting sensitive data in mind is Data Loss Prevention (DLP). You are likely requiring DLP agents on-premise which can also be installed on your computing resources where possible. These solutions can oftentimes meet all of the checkboxes regarding our data hunting and response efforts:

- Protect the data from unapproved clients
- Automate safeguards to protect against data theft or respond to policy violations
- Roll up reporting data to a central repository for quick review

If you have a DLP solution already deployed, it may be trivial to discover IoCs as well. Many vendors extend the capability to use commercial off-the-shelf policies to identify and respond to various forms of sensitive data as well as the ability to create your own policies for proprietary or custom data types. Again, it is important to understand what the IoC you are looking for will look like when creating these policies. The vendor will often allow you to define your own wordlists, regular expressions, or even complex scenarios such as one word within the vicinity of another.

Another huge advantage of using the DLP solution is that it is natively aware of various file types. For example, if you save a sensitive string in a Word document and attempt to perform a command-line-based search using `grep` in Bash or `Get-Content` with PowerShell, you will likely come up empty as Word encodes the data before writing it to disk. With the DLP solution, they can decode several different file types to determine if that IoC exists in that file and make automated decisions on what to do with the offending file.

Extend Your CM Tools

- **CM tools** are often used to:
  - Automate changes
  - Enforce configuration compliance
  - Control software versions
- Cloud vendors offer their own CM tools you could leverage
- These tools can also invoke **custom scripts**
  - Search for the IoC
  - If found, report and take action
    - Send alert to SIEM
    - Delete file
    - Move file to custom location for analysis

Your operations team is likely using configuration management (CM) tools to manage the various compute resources in your cloud environments. Typically, these tools are used to automate changes, enforce organizational policy and compliance, control software versions, and whatever else the operations team would like to perform at scale.

These CM tools can also be leveraged to search for IoCs. The tools often provide a means to launch customized commands to "hunt" for data using string matching or patterns. Simple recursive file content searches will oftentimes discover this data with ease, but keep in mind that the examples you will see in the next several slides are only capable of discovering plain text strings containing these IoCs—not encoded, compressed, or encrypted examples. In these cases, you will need to extend upon these capabilities to find data within these file types.

There are cloud-native CM tool services that you can leverage to discover sensitive data such as:

- AWS Systems Manager[1]
- AWS Config[2]
- Azure Run Command[3]
- GCP `gcloud compute ssh`[4]

[1] https://aws.amazon.com/systems-manager
[2] https://aws.amazon.com/config
[3] https://docs.microsoft.com/en-us/azure/virtual-machines/windows/run-command
[4] https://cloud.google.com/sdk/gcloud/reference/compute/ssh

AWS Systems Manager

- **Run Command** feature can be used to launch customized commands to search for IoCs
- Documents define actions to take on SSM-managed instances
  - Create one or more to search the filesystem for IoCs
  - **runCommand** property can invoke shell commands
  - Not required as this script data can be supplied to the built-in **AWS-RunShellScript** document

The content of this document is as follows:

```
{
  "schemaVersion": "2.2",
  "description": "Discover Netcat in home directories",
  "mainSteps": [
    {
      "action": "aws:runShellScript",
      "name": "find_netcat",
      "inputs": {
        "runCommand": [
          "egrep -r 'nc -[nlvp]{1,}' /home"
        ]
      }
    }
  ]
}
```


The AWS System Manager is an agent that runs on your EC2s with a dedicated GUI to run operations on your EC2s. What can you run? Anything at all. The real power of System Manager is how the AWS service can collect some of the results and provide visualizations and invoke automations.

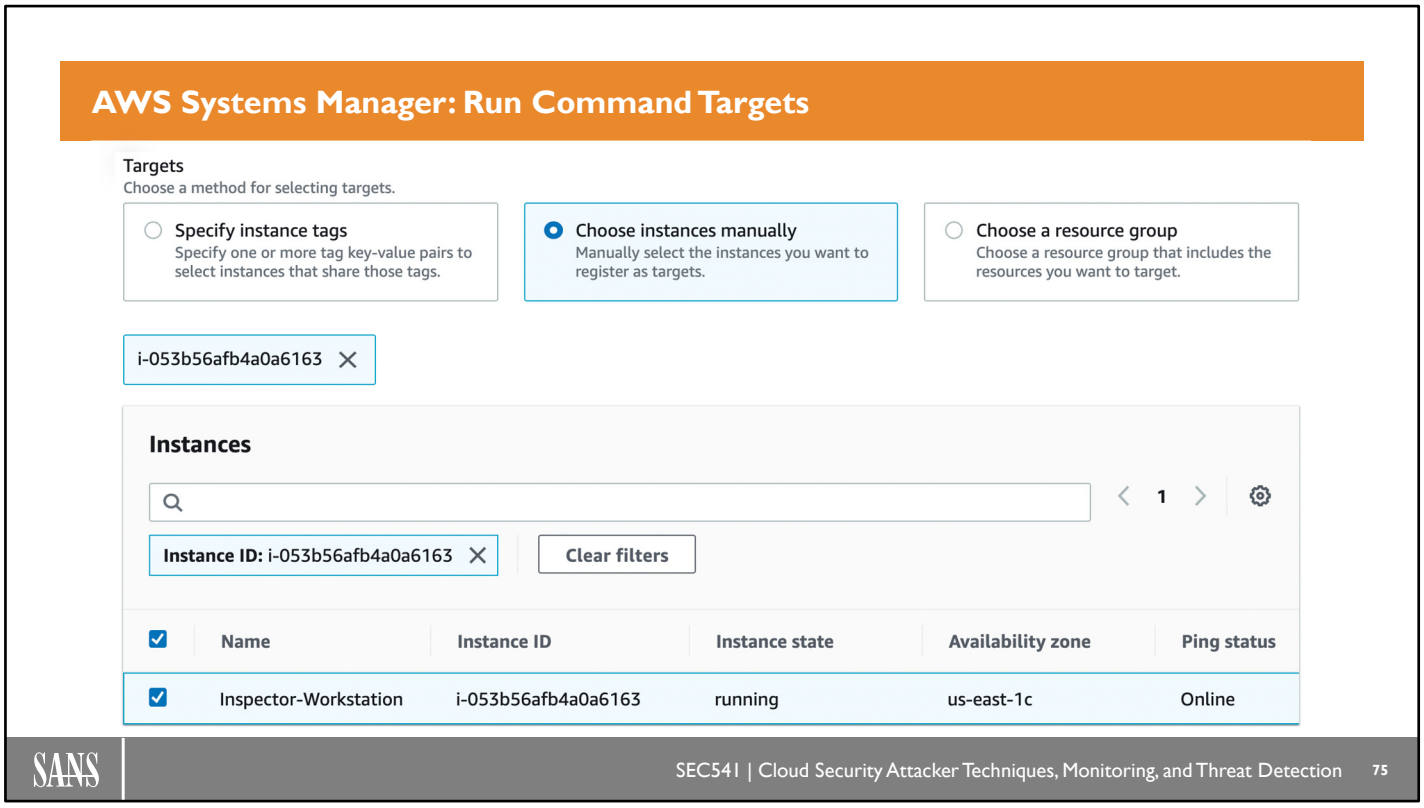
The AWS System Manager is also what is allowing us to jump into our Inspector Workstation with a command line without having to SSH or do RDP. It is tunneling our commands and the results through the System Manager.

System Manager was built to run commands across a fleet of systems. Most organizations use tags to separate the EC2s for System Manager commands. For instance, maybe you want to pull access logs from all your NGINX web servers on your marketing team (you didn't setup those CloudWatch log groups? For shame). The System Manager can run the commands on all systems with the tag of "System: NGINX" a tag of "Department: Marketing".

We can also use System Manager to look for files that contain sensitive information. To conduct the same operation at a regular interval, a System Manager Documents can be created which define any actions that you would like to take on your SSM-managed instances, such as invoking a shell command.

AWS System Manager does have a built-in SSM document called `AWS-RunShellScript`, but you may create your own custom documents just like the one shown above. This example document is rather simple in that the `runCommand` parameter instructs the instance to perform a recursive extended grep (`egrep`[1]) looking for any occurrences Netcat[2] executable (`nc`) with various command flags within the `/home` directory or any subdirectories (hence the `-r` argument). Netcat is not always malicious but is often leveraged by adversaries to create bind (e.g., listening) shells or reverse shells on compromised systems.

[1] https://ss64.com/bash/egrep.html
[2] https://nmap.org/ncat/



After the SSM document is created, the next step would be to identify the instances that will be targeted a few different ways. As shown above, you may select each instance within that region manually by selecting "Choose instances manually" and placing a check next to the instance(s) to target. There is a limit here, however, of up to fifty instances. If you have more instances than this and choose this method, you will have to conduct this run command more than once—targeting up to fifty systems each time.

A more efficient approach would be to choose the target instances by the tag they share. For example, if all the target systems you would like to query for the existence of an IoC belong to the development team and each development system has a "Department" tag with a value of "Development", you can use that as the basis for the target selection. This is done by selecting the "Specify instances tags" option and entering "Department" in the "Tag key" field and "Development" in the "Tag value" field. Click "Add" when finished.

This option may feel restrictive as tags would have to be created prior to the run command launch and that is mostly true. However, if you were to, for example, target all systems and if every instance has a "Name" tag assigned, you can fill out the "Tag key" field with "Name" and leave "Tag value" blank. This is because the value field is actually optional! You can target all systems quite easily—as long as they have a "Name" tag.

Finally, you can select an AWS Resource Group which, like the tags option, must be created prior to choosing a group of systems to query.

It's a good idea to come up with an enterprise-wide tagging strategy that is enforceable, so that you are able to evaluate your systems easily.

Resources:
https://www.cloudforecast.io/blog/aws-tagging-best-practices/
https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/ready/azure-best-practices/naming-and-tagging

AWS Systems Manager: Discovered IoC

Status	Detailed status	Response code
Success	Success	0
Step name	**Start time**	**Finish time**
find_netcat	Wed, 26 May 2021 12:46:22 GMT	Wed, 26 May 2021 12:46:23 GMT

Output

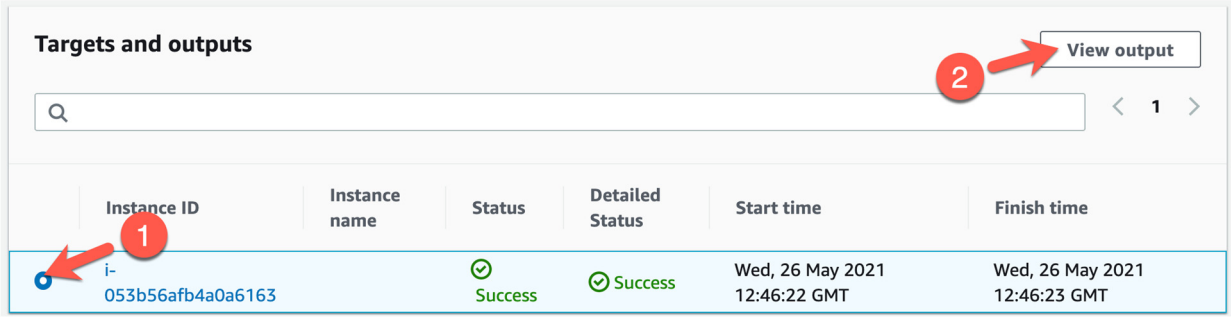
The command output displays a maximum of 48,000 characters. You can view the complete command output in either Amazon S3 or CloudWatch Logs, if you specify an S3 bucket or a logs group when you run the command.

Copy Download

```
/home/ec2-user/labs/pacu/pacu/modules/iam__privesc_scan/main.py
external_server = input('The EC2 instance will try to connect to
your server using a bash reverse shell. To listen for this, run the
command "nc -nlvp <an open port>" from your server where port <an
open port> is open to accept the connection. What is the IP and
port of your server (example: 127.0.0.1:80)? ')
/home/ec2-user/labs/pacu/pacu/modules/iam__privesc_scan/main.py
print('Start listening on your server now! Simple command to listen
on an open port: "nc -nlvp <port>".\n')
/home/ec2-user/49765:nc -lvp <port>
```

SANS SEC541 | Cloud Security Attacker Techniques, Monitoring, and Threat Detection 76

Once the command is run either via the AWS console or CLI tools, the output can be viewed within the AWS SSM service itself by selecting any of the instance targets and clicking on "View output" as shown below:



In the image above, you can see the results of the IoC search. It appears that the instance in question had three occurrences of the IoC: two in a file named `main.py` and another in a file named `49765`.

With these run commands, you may also choose to save the output to other AWS services such as AWS S3 or AWS CloudWatch—a location where you likely have other log data stored. Either of these may be a more efficient output location.

AWS SSM Run Commands via CLI

- Using the console is great, but oftentimes the command line is much more efficient—especially **at scale**
- Here is the same run command using the AWS CLI tools:

```
aws ssm send-command \
  --instance-ids "i-0123456789abcdef0" \
  --document-name "AWS-RunShellScript" \
  --comment "Netcat discovery" \
  --parameters "{\"commands\":[\"egrep -r 'nc -[nlvp]{1,}' /home\"]}"
```

```
    "CommandId": "227e1696-2c76-4b8d-a346-6aebd0c1f903",
    "DocumentName": "AWS-RunShellScript",
```


Utilizing AWS SSM via the web console is great for one-off searches involving limited systems and regions, but what if you have to search for the IoC across a large number of unrelated systems (with differing tags) or even multiple regions? You may want to enhance your capabilities by leveraging command line tools like the AWS CLI.

The example above shows a similar query that was made using the console approach, except this time the AWS-RunShellScript command document is leveraged. This document, instead of creating your own, can come in very handy when you are simply wanting to run a remote command using AWS SSM. The `aws ssm send-command`[1] command does require a few of the same options that were discussed—they just come in the form of command flags. To explain what you are seeing above:

- `--instance-ids`: List of instances to run the command against
- `--document-name`: The AWS SSM document to leverage
- `--comment`: What this run command is all about so that onlookers (or your future self) understand what you are trying to achieve
- `--parameters`: Either name=value or JSON-formatted parameters. This is where the command line instructions will be placed

Once the command launches, you will be able to retrieve the `CommandId`. This is very useful as you can run a follow-up command (once the command finishes running) to review the results and to reference the correct run command, you will need this `CommandId` value.

You can also take the automation even further. An SSM Document can be created by your favorite infrastructutre as code tool including Terraform2, CloudFormation3, or CDK4.

[1] https://docs.aws.amazon.com/cli/latest/reference/ssm/send-command.html
[2] https://registry.terraform.io/providers/hashicorp/aws/latest/docs/resources/ssm_document
[3] https://docs.aws.amazon.com/AWSCloudFormation/latest/UserGuide/aws-resource-ssm-document.html
[4] https://docs.aws.amazon.com/cdk/api/v1/docs/@aws-cdk_aws-ssm.CfnDocument.html

Viewing AWS SSM Results

Use the **CommandId** to review the output of the command

```
aws ssm list-command-invocations --command-id \
  227e1696-2c76-4b8d-a346-6aebd0c1f903 \
  --details --query \
  'CommandInvocations[].CommandPlugins[].Output' \
  --output text
```

```
/home/ec2-user/labs/pacu/pacu/modules/iam__privesc_scan/main.py:            exte
rnal_server = input('The EC2 instance will try to connect to your server using a
 bash reverse shell. To listen for this, run the command "nc -nlvp <an open port
>" from your server where port <an open port> is open to accept the connection.
What is the IP and port of your server (example: 127.0.0.1:80)? ')
/home/ec2-user/labs/pacu/pacu/modules/iam__privesc_scan/main.py:    print('Start
 listening on your server now! Simple command to listen on an open port: "nc -nl
vp <port>".\n')
/home/ec2-user/49765: nc -lvp <port>
```


The previous page ran the command but will not return the results of the command. It only returns the results of the fact that you created the command and it's queued up to be run by the SSM agent when they grab it. To retrieve the output, you will have to run the list-command-invocations.

The companion command to run to retrieve the run command results is `aws ssm list-command-invocations`. Just like the initial `aws ssm send-command` command, you will need to supply some flag values to retrieve the results:

- `--command-id`: The `CommandId` value that was output when running `aws ssm send-command`
- `--details`: Specify that more verbose output is needed to gather the command output
- `--query`: Not necessarily required, but is useful to extract *just* the command output from the very noisy results
- `--output`: Also not necessary, but outputs in plaintext instead of JSON for better readability

As expected, the same results were acquired on the command line as it was using the AWS console.

Azure Run Commands

- **Run Commands** can be leveraged to search the local file system for IoCs via custom scripts
- Looking for that same IoC in Azure

```
az vm run-command invoke -g SEC541 -n webserver \
  --command-id "RunShellScript" --scripts \
  "egrep -r 'nc -[nlvp]{1,}' /home" | \
  jq -r '.value[].message'
```

```
Enable succeeded:
[stdout]
/home/nicholsr/no-malware-here nc -lvp <port>

[stderr]
```


Azure has a very similar command line option to run remote commands on systems—Azure Run Commands. This is part of the Azure CLI tools and is a function of the `az vm` command set. Fewer options are required for the `az vm run-command invoke` command than the AWS SSM variant, such as:

- `--resource-group or -g`: The resource group in which the Azure VM resides
- `--name or -n`: The name of the Azure VM
- `--command-id`: The Azure command identifier
- `--scripts`: The user-provided data to send to the Azure command

You are not just restricted to `RunShellScript` as your `command-id` of choice. To view a complete list, you can run this Azure CLI command:

```
az vm run-command list -l <region> | jq -r '.[].id'
```

At the time of this writing (February 2022), here is a list of which commands are available in `eastus`:

```
DisableNLA              EnableWindowsUpdate     RemoveRunCommandLinuxExtension
DisableWindowsUpdate    ifconfig                RemoveRunCommandWindowsExtension
EnableAdminAccount      IPConfig                RunPowerShellScript
EnableEMS               RDPSettings             RunShellScript
EnableRemotePS          ResetRDPCert            SetRDPPort
```

Extend Your Vulnerability Scanner

- Many vulnerability scanners allow for **custom scripts** or **plugins**
  - Develop scripts or plugins to search for IoCs
  - Build plugins/scripts with time in mind as many vendors place a hard limit on how long a plugin/script can run
- ***Caveat***: This method only works with plaintext files!
  - Could push tools to endpoints to enable the script/plugin to process other file types

As your vulnerability scanner will likely already have administrative access to all endpoints, it may be advantageous to use this access to automate discovery of IoCs. Many vulnerability scanning vendors support some level of custom plugin or script development that you, as the administrator, can utilize to search for this potentially malicious data.

There may be some reverse-engineering or review of vendor manuals required to understand the vendor's scripting language, but it can be done. In fact, one of the course authors had a unique situation where the vulnerability scanner was leveraged to try and find sensitive data on systems that were unapproved for that classification of data. After reviewing some Nessus plugins to understand how the scanner logged into the system and executed local system commands, a custom script was developed to interrogate the scanned systems for the existence of sensitive data.

To the right you will see an example of that very script (with the classified data replaced with "Meeseeks" and "Portal Gun" (references to a certain animated show the course author enjoys).

This approach can be conducted to search for IoCs as well.

```
if(sshlib::get_support_level() >= sshlib::SSH_LIB_SUPPORTS_COMMANDS)
  enable_ssh_wrappers();
else disable_ssh_wrappers();

search_cmd = "egrep -r \([Mm][Ee3][Ee3][Ss\$][Ee3][Ee3][Kk][Ss\$]\|[Pp][Oo0][Rr][Tt][Aa@][
Ll1]\\s[Gg][Uu][Nn]\) /home 2>/dev/null";

ret = ssh_open_connection();

if(!ret) exit(0);

buf = ssh_cmd(cmd:search_cmd);

if(isnull(buf))
{
  ssh_close_connection();
  audit(AUDIT_NOT_INST, "Nothing to see here...");
} else {
  ssh_close_connection();

  report = 'Found the folowing files with proprietary infomation:\n';
  report += buf;

  security_report_v4(port:0, extra:report, severity:SECURITY_HOLE);
}
```

Discovering Data in Cloud Storage

- IoCs may find their way into **cloud storage solutions**
- Legacy search features will no longer work
  - Cannot run local commands on an instance/virtual machine
  - Cannot leverage DLP or vulnerability scanner (unless they can process cloud storage data)
  - Limited metadata regarding object's contents
- Must **interface** with the storage service with:
  - Another, purpose-built cloud service
  - File retrieval and analysis with custom tools (very costly)
- **Indexing** the data can add speed and efficiency to file analysis

To be fair, the solutions up to this point are either legacy techniques or revamped disk searching methods using cloud services, but what if the data is not stored on a virtual disk? Cloud storage solutions add a bit more complexity and effort regarding searching for and addressing policy violations for data of interest.

To find this data, you may need to turn to cloud-native, third-party, or custom-created tools. The costliest option being a process like:

- Retrieve data from cloud storage
- Write data to local disk
- Analyze data for IoCs
- If found, take action on file/object

In many of these cases, the data as it resides in cloud storage is not inherently "searchable" as most of what you can search on is not file or object contents, but metadata about a file or object. Indexes can be created to take your raw data and create searchable bits of text (in other words, even more metadata). If indexes are created from this data, you would now have something to search against.

NOTE: We have talked about how one of the top three initial access vectors is finding credentials accidently released on the internet. There is a service that is indexing openly available S3 buckets. I wonder if they have anything about Sherlock Holmes.

References:
https://buckets.grayhatwarfare.com/results/sherlock

AWS Macie

- Generates a near-real time **inventory** of S3 buckets
- As data is added to S3:
  - Provides best practice checks against bucket-level security and access controls
  - Conducts **sensitive data discovery** of S3 objects using machine learning and pattern matching
- Allows users to create their own **pattern-matching rulesets**
- Can view findings in Macie's console or automate actions with **CloudWatch Events**

The first cloud-native service to discuss is AWS Macie[1]. This solution first generates an inventory of the organization's S3 buckets. While doing this, several best practices are assessed and can provide you with deviations such as[2]:

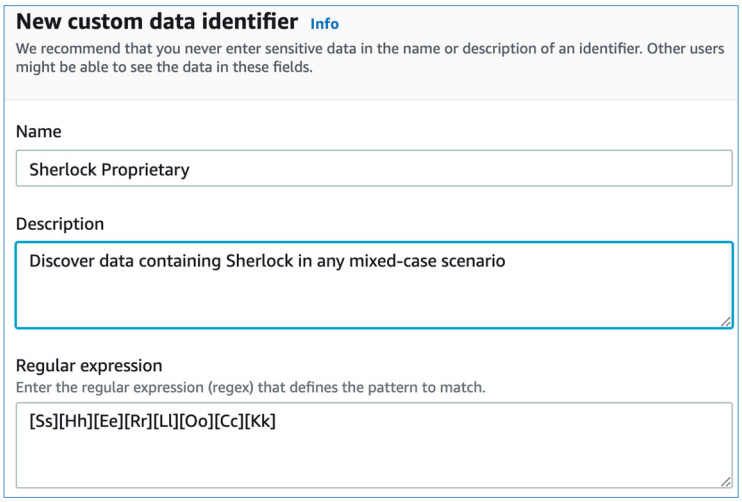
- Is the bucket publicly accessible[3]? This may be as intended, S3 buckets are wonderful static web servers and image servers. The author recommends that publicly accessible buckets not reside in the same account as sensitive buckets. It makes it harder for the hunters to organize the hay to find the needle.
- Unencrypted buckets really have no place in an AWS account, unless it is a public bucket. Even if it's not sensitive, at least use the standard AWS S3 bucket encryption key[4]. Sensitive data should be encrypted using a KMS key, where you have to specify who can decrypt the data.
- Buckets shared outside of organization[5]. Sharing among organizations and performing cross account access to certain S3 buckets makes sense in some cases. It can be difficult to see these relationship, as they are not a configuration setting such as "encryption on". Macie analyzing and displaying this cross-account access is a nice feature.
- S3 objects containing pre-defined sensitive data (e.g., PII)[5]

You can also extend Macie beyond just its default capabilities. Since it has visibility of AWS S3 bucket contents, over the next few pages, you will discover just how powerful this service can be regarding searching for sensitive data in places it should not reside (in an effort to identify data staging which is a precursor to data exfiltration) or IoCs.

[1] https://aws.amazon.com/macie
[2] https://docs.aws.amazon.com/macie/latest/user/findings-types.html
[2] https://aws.amazon.com/premiumsupport/knowledge-center/read-access-objects-s3-bucket
[3] https://docs.aws.amazon.com/AmazonS3/latest/userguide/default-bucket-encryption.html
[4] https://docs.aws.amazon.com/AmazonS3/latest/userguide/example-walkthroughs-managing-access-example2.html
[5] https://docs.aws.amazon.com/macie/latest/user/managed-data-identifiers.html

AWS Macie: Custom Data Identifiers

- Can instruct Macie to look for a particular **regex** within the S3 object
  - Example looks for an S3 object containing any variant of **Sherlock**
- Can be even more flexible:
  - Up to fifty **keywords** that, when combined with the regex and is within the configured **distance**, will trigger an event



The user can extend upon this service by creating their own jobs to search for their own proprietary or unique data which they would deem as sensitive. On top of this sensitive data, IoCs can be searched as well if they are in a supported file format. Supported file types in Macie as of this writing are[1]:

- Big data: Apache Avro object containers and Apache Parquet files (e.g., `.avro, .parquet`)
- Compressed files (e.g., `.tar, .gzip`)
- Documents: PDF and Microsoft Excel and Word documents (e.g., `.docx, .xlsx, .pdf`)
- Plain text files: Non-binary files (e.g., `.html, .js`)

Macie does not analyze data in images, audio or video. However, you could leverage AWS's transcription[2] services and store the transcript in a bucket.

In any case, the first step to do so is to create a custom data identifier. This can be in the form of a regular expression, or a regular expression combined with a list of keywords.

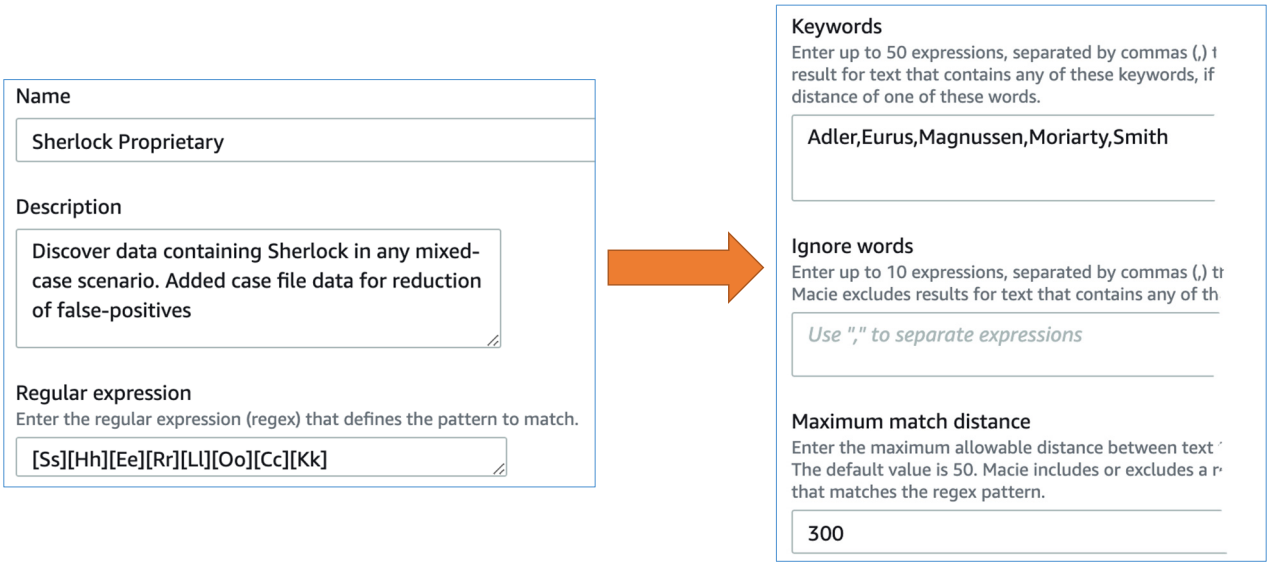
In the image above, the following regex is actually simple, looking for the word "Sherlock" with any combination of upper-and-lower case letters.

[Ss][Hh][Ee][Rr][Ll][Oo][Cc][Kk]

After the identifier is created, the user must create a job to be run to search for this data once or at a regular interval.

[1] https://docs.aws.amazon.com/macie/latest/user/discovery-supported-formats.html
[2] https://aws.amazon.com/transcribe/

AWS Macie: Combining Regular Expressions and Keywords



The regex on the last page is simply looking for the plaintext of Sherlock and various mixed-case forms of that word. That, however, may lead to many false positives. There is likely many files that contain Sherlock's name in our organization. If you would like to combine this regex-only approach with looking for this string within an object, you can see this in actions with the above AWS Macie custom data identifier.

Here, there are other terms (e.g., case file contents) that are deemed sensitive as well that, when combined with the regex of [Ss][Hh][Ee][Rr][Ll][Oo][Cc][Kk], could indicate someone is storing data in AWS S3 when they are not approved—either breaking policy or even worse, staging data to be exfiltrated!

You may be asking "Must I be worried about the many mixed case variants of the keywords?". The answer to that question is no, since AWS Macie views keywords as case-insensitive—so no worries there!

The course author tried this out and uploaded some case files containing many combinations of these sensitive terms. Did Macie find it?

AWS Macie Results

After a few minutes, Macie was able to discover these sensitive files that have no business being in S3 (let alone a bucket meant for web code backups)!

Severity	Finding type	Resources affected	Updated at
Medium	SensitiveData:S3Object/Multiple	baker221b-webbackupbfcf6dbb-1bknaimdo8i94/Magnussen.pdf	7 minutes ago
Medium	SensitiveData:S3Object/Multiple	baker221b-webbackupbfcf6dbb-1bknaimdo8i94/Eurus.pdf	7 minutes ago
Medium	SensitiveData:S3Object/Multiple	baker221b-webbackupbfcf6dbb-1bknaimdo8i94/Moriarty.pdf	7 minutes ago
Medium	SensitiveData:S3Object/Multiple	baker221b-webbackupbfcf6dbb-1bknaimdo8i94/Adler.pdf	7 minutes ago
Medium	SensitiveData:S3Object/Multiple	baker221b-webbackupbfcf6dbb-1bknaimdo8i94/Smith.pdf	8 minutes ago

It appears that Macie was able to find the data in question—even in an encoded document (i.e., not plaintext) like a PDF file. Given the titles of these documents, they look like true positives. The title, however, does not matter as Macie is analyzing data *inside* these files.

The severity for these findings *should* be high! Why are they scored as medium? Well, that is the default for AWS Macie regarding SensitiveData:S3Object/CustomIdentifier—they will always appear as medium[1]. You are able to adjust the severity levels, based on the number of occurrences of text that matches the criteria[2].

This is just something to be aware of until a time comes when you are able to adjust the severity for your custom data identifiers.

[1] https://docs.aws.amazon.com/macie/latest/user/findings-severity.html
[2] https://docs.aws.amazon.com/macie/latest/user/custom-data-identifiers.html#custom-data-identifiers-severity

Azure Cognitive Search

- After indexing Azure data stores, **Cognitive Search** can be used to discover sensitive data via custom searches
- Can search several formats (MS office files, PDF, image files, and plaintext files)
- Supports several data stores in Azure:
  - Azure Cosmos DB
  - Azure SQL Database
  - Azure Blob Storage
  - Microsoft SQL Server hosted in a VM

Once an index is created in the Azure Cognitive Search[1], custom searches can be created to discover data of various types in several data stores within the organization's Azure subscription. These sources include:

- Azure Cosmos DB, a fully managed NoSQL database[2].
- Azure SQL Database[3] is an awesome target for Azure Cognitive Search. Organizations leveraging a Microsoft ecosystem are probably running their SQL backed databases with SQL Server.
- Azure Blob Storage[4] is an object store, similar to S3.
- Microsoft SQL Server hosted in a virtual machine that you are controlling, as opposed to the Microsoft managed Azure SQL database.

A drawback with many solutions is that they only support discovering data in its raw, non-encoded form. For example, the Word document shown in its raw form below only consists of "Security488".


```
┌──[nicholsr@DESKTOP-LOQ61MV]─[~]
└─$ grep Security488 /tmp/sensitive.docx
┌──[✗]─[nicholsr@DESKTOP-LOQ61MV]─[~]
└─$
```

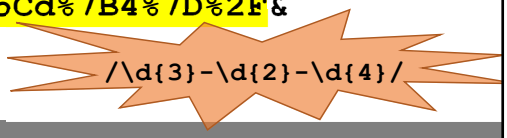
Looking at the raw bytes, it would be impossible to discover that string unless the tool could decode this Word document. A substantial advantage here is that Cognitive Search does not just have the ability to search within raw files but can search several common and Microsoft-proprietary files.

[1] https://docs.microsoft.com/en-us/azure/search/search-what-is-azure-search
[2] https://azure.microsoft.com/en-us/services/cosmos-db
[3] https://azure.microsoft.com/en-us/products/azure-sql/database
[4] https://azure.microsoft.com/en-us/services/storage/blobs

Azure Cognitive Search Syntax

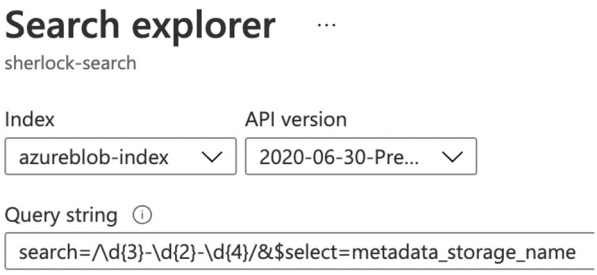
- Uses **Lucene query syntax** to search for data
  - Can form query using Search Explorer
  - Format:
    - `https://<search-service-name>/indexes/<index-name>/docs/? +`
    - `api-version=<api-version-date>& +`
    - `search=<URI-encoded Lucene search query> +`
    - `<additional-options>`
- Sample search (URI-encoded):

```
https://sherlock-search.search.windows.net/indexes/
azureblob-index/docs?api-version=2020-06-30-Preview
&search=%2F%5Cd%7B3%7D-%5Cd%7B2%7D-%5Cd%7B4%7D%2F&
%24select=metadata_storage_name
```



Once the data is indexed and ready to search, a Lucene-formatted query[1] can be leveraged to parse through the content in the configured backend datastores. Luckily, within the Azure Search services, there is a component that aids in the creation of these queries and will even convert the query string to URI-encoded as this search may be leveraged by other, authenticated Azure applications.

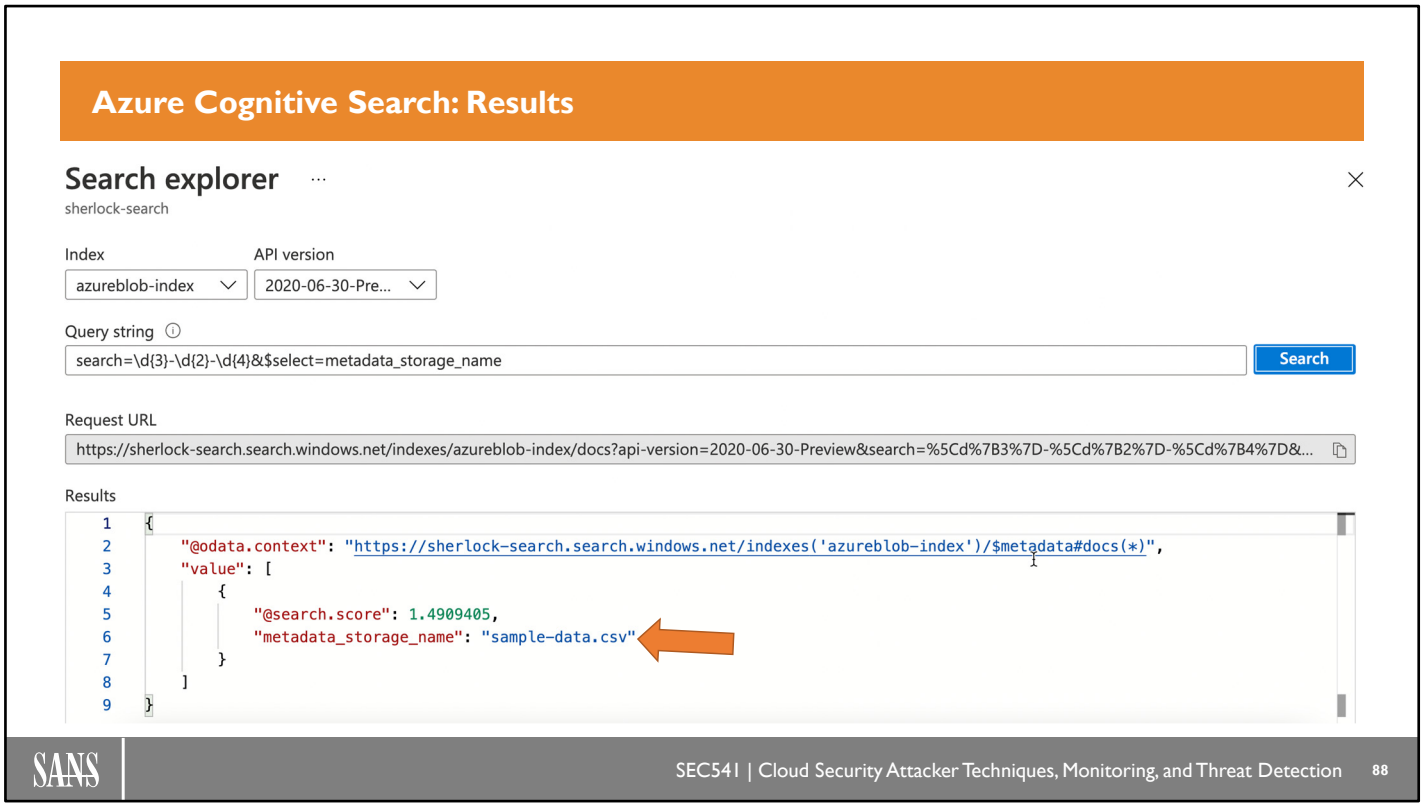
For example, see the search below where the course author is attempting to locate any United States Social Security Number (SSN) that may exist in a file that Azure Cognitive Services can index and parse:



Since this service supports regular expressions, the **`search`** criteria is `\d{3}-\d{2}-\d{4}` which should capture the most common form of an SSN `(###-##-####)`.

The other option included is `$select=metadata-storage_name` which simply means that when the data is located, output only the name of the file in which it was found (to avoid further exposure of the data).

[1] https://lucene.apache.org/core/2_9_4/queryparsersyntax.html



And here you can see the results of that search looking for sensitive SSN data. It is clear that there is a file that contains this information called `sample-data.csv` (as shown by the value in the `metadata_storage_name` field). But what is up with the `@search.score` value of `1.4909405`?

The **`@search.score`** metric is the output of a similarity ranking algorithm in Azure Cognitive Search. Simply put, this applies a relevancy score to the retrieved document to aid in ranking which retrieved result is more relevant to the search conducted. The higher the number, the more relevant. However, when searching for the existence of a string, the relevancy score of greater than 0 should be an indication that you have found a document with the search in question.

Build Your Own Data Discovery Tools

- Some cloud services do not lend themselves to easily be analyzed by other existing tools
  - **Build your own**!
  - Requires **scripting/programming**
  - More **regex** to account for more occurrences
- Example: Admins breaking policy in Lambda
  - Embedded **database credentials** found in Python code

```
import json
import pymssql
import time
import logging

db_server = 'servername'
db_user = 'db_user'
db_password = 'db_pass'
db_dbname = 'db_name'
db_tbl_log = 'log'
```

```
┌─[nicholsr@DESKTOP-LOQ61MV]─[~]
└─► $ if [[ $(curl -s $(aws lambda get-function --function-name databaseRetrieve |
        jq -r '. .Code.Location') | zcat | grep -PoH "[Pp]assword") ]]; then
        echo -e "\033[31m[!]\033[0m PASSWORD FOUND!"; fi
[!] PASSWORD FOUND!
```


Cloud providers are constantly coming up with new and inventive ways to aid in the discovery of sensitive data, IoCs, or even, in the case above, organizational policy deviations. There may be several circumstances in which you must create your own method to find and act upon this data. Traditionally, this meant being well versed in scripting or coding languages and, with cloud, it's not very different. Since the command line tools make this data easy to retrieve and transform, wrapping the tool output in another scripting language (with a little bit of regex) makes discovering this data relatively easy.

In the screenshots above, you will find that there is a Lambda function in AWS that, against all common security best practices, includes database credentials. Of course, you would want to use roles to allow Lambda to read and/or write to a database service[1], but here the developer, for whatever reason, embedded them in the Lambda script. Using a little command-line kung fu, you are able to retrieve the underlying Lambda code, decode it, and look for the sensitive data (in this case "password" with either a lowercase or uppercase "P").

[1] https://aws.amazon.com/secrets-manager

Database Sensitive Data Discovery

If a cloud-native service does not exist, you may need to manually discover sensitive data in a database

- Regex support may be limited
- May need to wrap SQL client in another scripting language

```
|  1 | Homer  | Simpson            | $50,000USD | 742 Evergreen Terrace, Springfield    |
123-45-6789 |
|  2 | Apu    | Nahasapeemapetilon | $65,000USD | 1810 Ward Road, Springfield           |
555-77-9999 |
|  3 | Barney | Gumble             | $35,000USD | 409 Wines Lane, Springfield           |
333-55-7777 |
```

```
student@demo-mgmt:~$ if [[ $(mysql -h 10.4.88.5 -u student@$DBSERVER -p  -e
  'select * from employee_info;' demo-db | grep -PoH "[0-9]{3}-[0-9]{2}-[0-9]{4}") ]];
  then echo -e "\033[31m[!]\033[0m SENSITIVE DATA FOUND!"; fi
Enter password:
[!] SENSITIVE DATA FOUND!
```


It was mentioned earlier that a service like Azure Cognitive Search can be fantastic for searching for data in a SQL database. What if your vendor of choice does not have such as solution for their database service? Again, you may need to create your own script or tool to do the heavy lifting for you.

In the example above, assume that a policy dictates that your company should not be storing US Social Security Numbers (SSN) in the "employee_info" database. To assess this requirement, some more command-line kung fu is needed to first grab all of the data in the database, piping the output to grep (which looks for the regex attributed to an SSN), and output a warning message if the unapproved data is found.

Since this data is not permitted, the next logical step would be to extend the script to remove the offending data.

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. **EXERCISE: Detecting Sensitive Data**
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Lab 3.3 | Detecting Sensitive Data

Exercise Duration: 30 Minutes

Objectives

We will explore AWS Macie to identify some very suspicious activity within our AWS S3 service

- Enable AWS Macie
- Perform data exfiltration using cloud storage
- Create custom data identifier to find proprietary data
- Discover unapproved cloud resource deployment and configuration
- Detect MITRE ATT&CK T1074.002: Remote Data Staging
- Find proprietary data amongst exfiltrated data

In this lab we will learn how to leverage AWS Macie to not only discover data deemed sensitive by AWS as it is stored within AWS S3 buckets, but also identify data that we deem very sensitive (e.g., proprietary data).

We will first turn on AWS Macie from the command line. Next, we will launch an attack that will move sensitive data into our cloud storage, which will simulate staging of collected data for exfiltration.

Next, we will use Macie to create custom identifiers, looking for proprietary data. Macie will use these identifiers to detect our files.

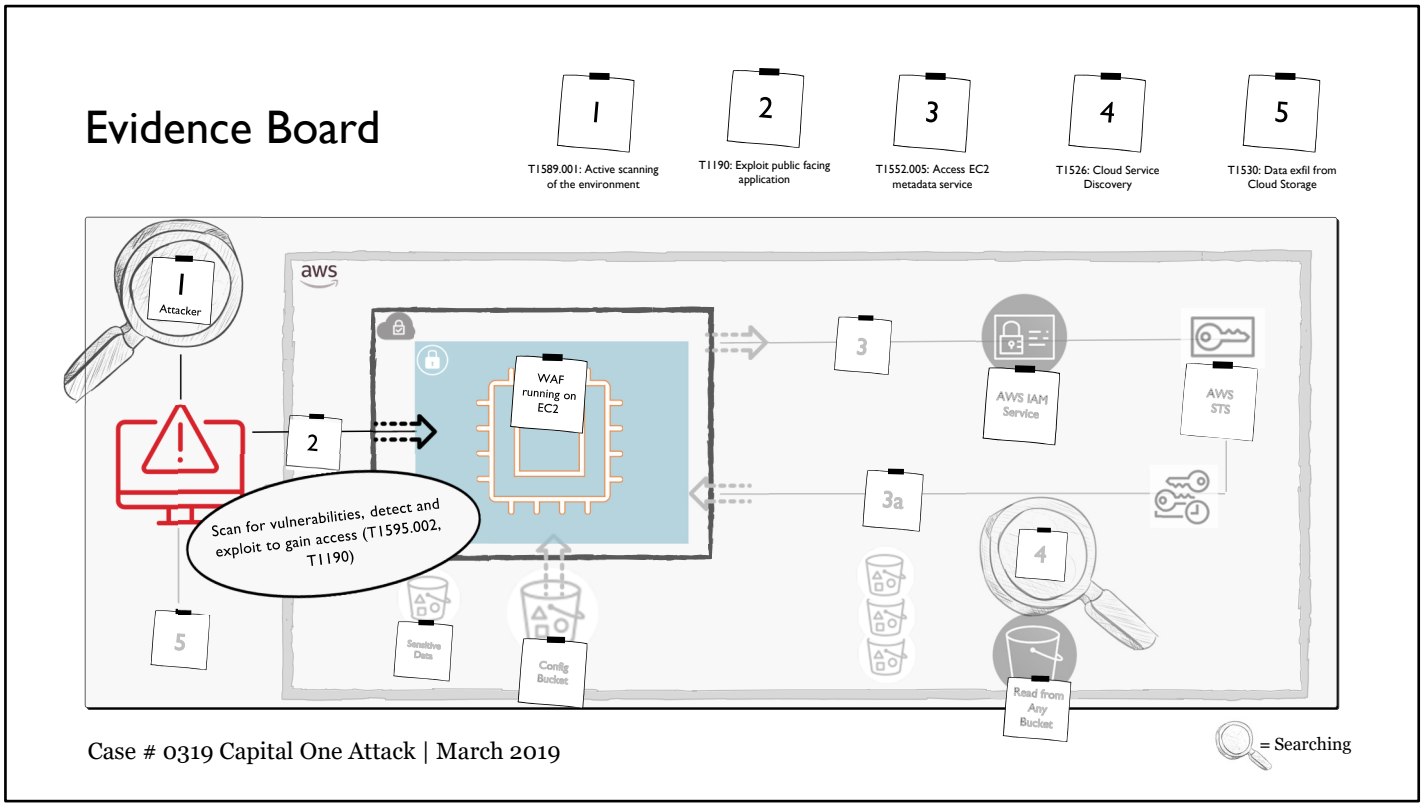
Course Roadmap

- 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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. **Vulnerability Analysis Services**
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.



When Capital One got the call that the attacker had access to S3 buckets, the team may not have known exactly how the attacker had gained access to the environment. We, however, know the attacker made use of a vulnerability they likely found through a scan[1], and leveraged the vulnerability to exploit the public-facing application[2]. How do we investigate and perform vulnerability analysis in a cloud environment? We will focus our attention on two different goals.

The first goal is to assess our environment for any vulnerabilities that were not identified by the vulnerability management team. This could obviously lead to a breach in and of itself if not addressed immediately, as the adversary may discover one or more vulnerabilities and exploit them to gain a foothold into our cloud environment. In our Capital One, use case, the vulnerability was in the application, not from a known vulnerable application or library.

The second goal will be to attempt to identify any new vulnerabilities presented during an adversarial campaign in the form of unexpected services. These new services may be attributed to data exfiltration or command and control channels left behind by the attacker. Of course, if these are found, it may be very late in the attack campaign, but, if we can react quickly enough, we may be able to limit the damage incurred during this breach.

[1] https://attack.mitre.org/techniques/T1595/002
[2] https://attack.mitre.org/techniques/T1190

Vulnerability Analysis

- Attackers scour our resources in search of vulnerabilities
- **Vulnerability analysis** is the approach to identify these issues *before* an attacker does
  - Of course, we must follow up our discovery with addressing the issue
  - We can also use this opportunity to find **unapproved modifications** to our environment by an attacker or operations staff
- The approach:
  - Automate or manually discover vulnerabilities
  - Patch, re-configure, or otherwise mitigate the vulnerability
  - Re-assess to ensure the vulnerability was properly addressed

As mentioned on the last page, this module will focus on leveraging some cloud-native as well as manual approaches to identify vulnerabilities in our cloud environments to get in front of any potential breaches.

Once we learn about the tools made available to you, in practice, you should follow the tried-and-true scan-patch-scan method. In other words, it is recommended to not only identify and address vulnerabilities that may be discovered, but to also verify that the fix actions did, in fact, address the underlying vulnerability. This can be performed in a few different ways such as a simple re-scan finding that the issue no longer shows up as a vulnerability or even a penetration test to see if someone acting as a real-world adversary can exploit this (assumed to be fixed) vulnerability.

We may falsely assume that because we deployed our systems using the latest and greatest security benchmarks and other security guidance that the system will remain in this secure state. This is far from the case as new vulnerabilities are being discovered quite often that would not have been addressed during the build of the system. Not only new vulnerabilities, but "configuration drift" may occur. This means that someone is altering the pristine, hardened system after the fact which may introduce new vulnerabilities into this system. That somebody might not necessarily be an attacker, but someone from your own operations team trying to "do the right thing" by making the system more performant, for example.

With this, the vulnerability analysis process needs to be a routine part of the organization's cybersecurity program.

Cloud Service Vulnerability Discovery

- Majority of the underlying cloud service infrastructure is managed by the vendor
- The service can still be used in an unsecure manner, for example:
  - Public cloud storage
  - Network controls lacking
  - Single-factor authentication with easy to guess passwords
- Luckily, many vendor benchmarks aid in addressing these issues and are included in many cloud vendor security solutions
  - AWS Best Practices
  - Azure Security Benchmark

Some may say, "but my application is hosted in the cloud—surely the vendor manages everything for me, right?". In some cases, the vendor certainly does manage quite a bit for the customer. However, the customer still has quite a bit of responsibility. AWS[1] and Azure2 have published their shared responsibility models for what they are responsible for, and what you as the customer are responsible for.

For example, when using the Infrastructure as a Service (IaaS) model, the vendor will manage the underlying hardware and data center elements, but the cloud provider does not enforce how the customer will deploy and configure their operating systems or applications, how they manage and configure their user accounts, how data is shared outside the organization, and so on. And how could they? If they did enforce these things, the customer's experience would be very rigid and would likely go against how many customers would like to operate their environments.

If the customer has to manage everything "above the hardware level" in IaaS, where do they start? Luckily for the customer, there are several security best practices guides out there ranging from how to configure the cloud services, all the way down to how to harden the operating system, and everything in between. In fact, AWS[3] and Azure[4] offer security best practices guides to help their customer operate in a more secure manner.

[1] https://aws.amazon.com/compliance/shared-responsibility-model
[2] https://docs.microsoft.com/en-us/azure/security/fundamentals/shared-responsibility
[3] https://docs.aws.amazon.com/wellarchitected/latest/security-pillar/welcome.html
[4] https://docs.microsoft.com/en-us/azure/cloud-adoption-framework/secure

AWS Security Hub

- Security service which centralizes data from many other AWS security services, such as:
  - Amazon GuardDuty
  - Amazon Inspector
  - Amazon Macie
  - AWS Systems Manager
  - AWS Firewall Manager
  - And many others
- When enabled, can automate the configuration of other AWS services as well
  - For example, enabling AWS Best Practices **security standard**, will create AWS Config rules for analysis
  - Enabling security standards generate a security score

Vendors provide dedicated services to help customer assess their security posture. AWS Security Hub[1] is a service that brings in alerts from multiple AWS services and 3rd party services. It rolls up the appropriate data and presents the security findings to the customer in an easy-to-digest format.

At the time of this writing, Security Hub can receive findings from many of the AWS services that generate some kind of security or compliance alerts[2]. GuardDuty, Inspector, Macie, Systems Manager, Firewall Manager, Chatbot, Trusted Advisor, IAM Access Analyzer, AWS Health, Detective and Audit Manager can all generate and send the integrations.

Security Hub will also integrate with 3rd party services that AWS has picked to partner with[3]. Want to integrate with a tool not on the list? Or want to write your own? Security Hub will also allow completely custom integrations, since the event format is published[4].

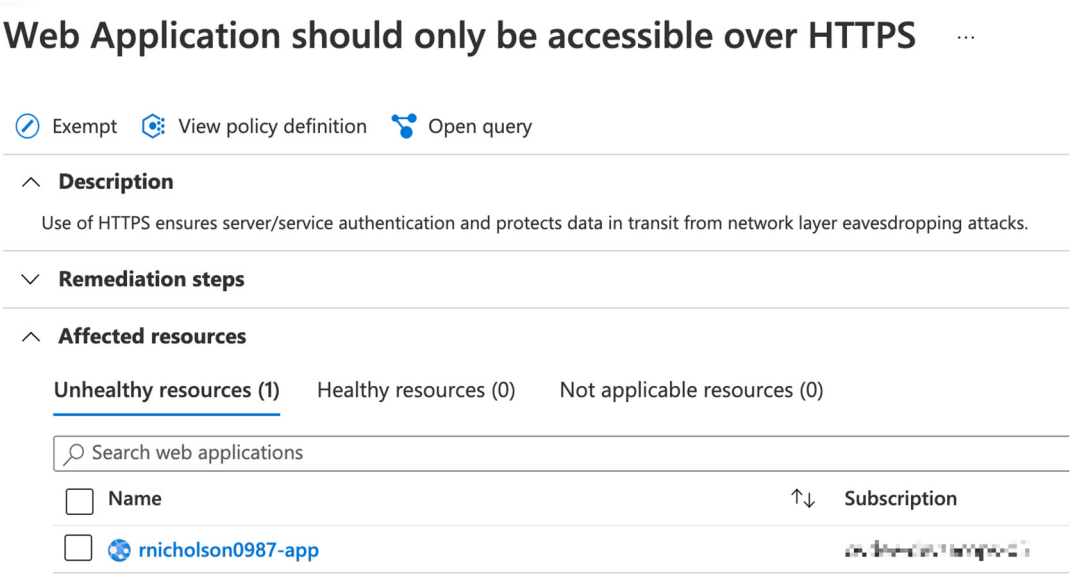
AWS Security Hub will even go as far as breaking down different compliance frameworks (which are selected by the user during set up) in the form of a security score. This score will show a rough percentage of how compliant the customer's cloud environment is with a certain framework. The supported frameworks, at the time of this writing, are AWS Foundational Security Best Practices[5], Payment Card Industry Data Security Standard (PCI DSS)[6], and the Center for Internet Security (CIS) AWS Foundations Benchmark[7].

Security Hub also will create event tickets to be evaluated. However, most organizations have a more sophisticated ticket system through Jira or some other service. One thing that Security Hub might be better for, is a single location to route all events from the services it collects from, and can route them to a central AWS Account, or a third-party service like Jira[8].

[1] https://aws.amazon.com/security-hub
[2] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-internal-providers.html
[3] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-partner-providers.html
[4] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-custom-providers.html
[5] https://docs.aws.amazon.com/securityhub/latest/userguide/securityhub-standards-fsbp.html
[6] https://www.pcisecuritystandards.org/pci_security

[7] https://www.cisecurity.org/benchmark/amazon_web_services
[8] https://community.atlassian.com/t5/Jira-Service-Management/AWS-Security-Hub-findings-in-Jira/qaq-p/1765422

Microsoft Defender for Cloud: Service Misconfiguration Example



Microsoft Defender for Cloud[1], is the Azure solution for posture management and protecting workloads. It will evaluate and look for service misconfigurations. An example of one of the service misconfiguration findings can be found above. As you can see, Microsoft Defender for Cloud breaks this down into three distinct sections. The Description section explains the finding and how the proper setting could enhance your security posture.

To close out the finding, Azure provides Remediation steps to the customer in the next section. Here you will find either manual or fully automated steps that Azure can implement on your behalf. Be cautious as sometimes, automated steps, while they will close the finding, could negatively impact the customer or even administrators' experience.

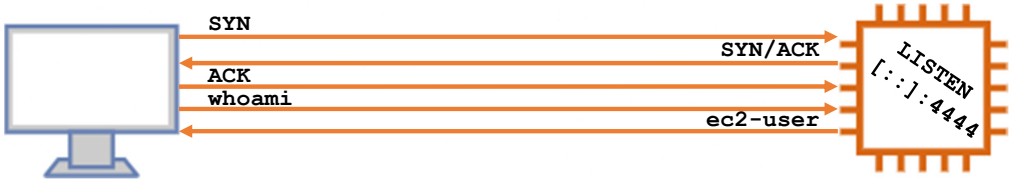
For example, one course author decided to allow Azure to close an issue with the SSH service being exposed to the internet and, when Azure closed the finding, removed all SSH access from the system. The vulnerability was gone, but so was administrative access until the course author added their IP address back into the Azure Network Security Group (NSG). Fully automated remediations can be a little dicey. You may only allow them for those configuration problems that are absolutely never supposed to happen, so you are comfortable with automating the remediation. But, if they should absolutely never happen, you probably built your deployment tooling to prevent it from happening in the first place. In Section 5, we will go further into automating response actions to security and configuration issues.

The final section highlights which systems have the issue. Those systems will appear under “Unhealthy resources”. Systems that do not have the issue or the issue does not apply will show up under the Healthy resources and Not applicable resources sections, respectively.

[1] https://azure.microsoft.com/en-us/services/defender-for-cloud/#overview

Network-Based Vulnerability Discovery

- Performing a network scan of systems can uncover:
  - Listening ports
  - Services that are running on those ports
  - Versions of those services
- Based on those results, can identify:
  - Outdated or vulnerable services
  - Abnormal listening applications (e.g., command and control channel)



Addressing the cloud services themselves are a great start, but there is more to defense in depth regarding cloud than just properly configuring cloud services—especially when it comes to IaaS. The next area will be assessing the customer's networking responsibilities. In most cases, the customer will be responsible in an IaaS deployment for Open Systems Interconnection (OSI) model layer 3 and above networking[1]. This includes which IP addresses, transport layer (layer 4) ports and protocols, and application layer (layer 7) protocols are permitted to and from their cloud resources.

Not only should this traffic be limited to approved systems—both inside and out—but the customer should be aware of any abnormal network connections that either are or could be established with their systems. Knowing normal network connections for the given environment would be very good to be aware of so that these abnormalities can be spotted.

Looking at the diagram above, you can see what may be deemed quite the suspicious connection. An outside system (on the left) is first establishing a Transmission Control Protocol (TCP) three-way handshake with a cloud resource over TCP port 4444. If this is not already suspicious (because an attack tool like Metasploit will use this TCP port by default in many cases), the fact that, if visibility of the network is possible, you can see that some shell commands and responses are traversing this connection.

A network vulnerability scan can be one way to identify these strange listeners or vulnerable, external-facing services on the virtual machine. Keep in mind that this is only part of the story as you would not be assessing the entire system—only what is responding to an outside resource.

[1] https://docs.microsoft.com/en-us/windows-hardware/drivers/network/windows-network-architecture-and-the-osi-model

Host-Based Vulnerability Discovery

- Network vulnerability scanning is only a small part of the picture.
  - No specifics to service configuration—only sees "from the outside"
  - No visibility of unexposed services at all unless an authenticated scan is conducted
- Host- or agent-based scanners provide much more complete data.
  - Unexposed application configurations and versioning
  - Operating system configurations and patch levels
- Cloud Service Providers (CSP) often provide their own agent which integrates with their own services to uncover security gaps.
  - Many third-party vulnerability management solution vendors have agent-based solutions of their own

To complete the three-part vulnerability assessment of a typical IaaS environment, the virtual machines running on the hosts need to be assessed as well. To acquire visibility of these systems, the network scan will not be enough. Oftentimes, vulnerability scan solutions that can acquire this level of assessment will use one of these strategies:

- Authenticated network scan: The network vulnerability scanner will log into the system and assess it over one or more sessions using Secure Shell (SSH), Windows Remote Management (WinRM), or other management connections supported by both the scanner and the host being scanned
- Agent-based: The scan solution will require an installed agent to run locally on the system and report its results back to the management server

Since there is a deeper level of access (oftentimes root/administrator-level), more potential vulnerabilities can be assessed like kernel patch levels, operating system and all application patches, and operating system and application configurations.

Many Cloud Service Providers (CSP) offer their own network and host-based vulnerability analysis solutions which we will cover over the next few pages.

AWS Inspector

- Can perform **network assessments** (no agent required) and **host assessments** (SSM agent required)
- Actively monitors EC2 instances
- Analyze Elastic Container Registry (ECR) images.
- Conducts checks against compliance standards, vulnerabilities, and best practices
- Results can be viewed in the service itself of forwarded to:
  - AWS CloudWatch
  - AWS Security Hub
  - AWS Simple Notification Service (SNS)

AWS Inspector[1] can perform both network-only and full on-host scans. AWS Inspector uses the System Manager agent to perform its on-host scans[2]. AWS Inspector service was rearchitected in late 2021 to be your one stop shop for vulnerability management across your compute workloads. Rather than run another agent, the AWS Inspector capabilities were rolled into the SSM agent. Since our systems in the labs already have SSM, starting inspector should be easy!

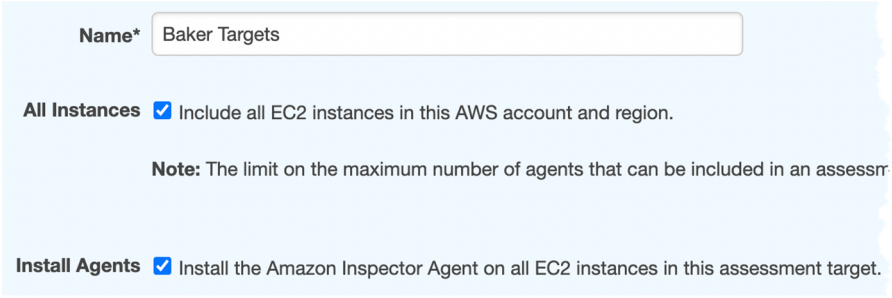
AWS Inspector can assess the systems against a multitude of standards and, once the assessment completes, there are several options to acquire the assessment data depending on the cloud customer's preference, such as sending the data to AWS CloudWatch, AWS Security Hub, and/or AWS Simple Notification Service (SNS).

Next, we will cover the required components to conduct an AWS Inspector assessment to identify gaps in the organization's vulnerability management program as well as unapproved applications or network services which may be vulnerable.

[1] https://aws.amazon.com/inspector
[2] https://aws.amazon.com/about-aws/whats-new/2021/11/amazon-inspector-continual-vulnerability-management

AWS Inspector: Assessment Targets (Classic Inspector)

- Determines which systems the assessment will run against
- Can be based on:
  - **All instances** in the region
  - Instances with one or more specified **tags**
- If the systems are **SSM-managed**, can choose to install the inspector agent automatically



This is only for Classic Inspector, which is still used throughout many enterprises.

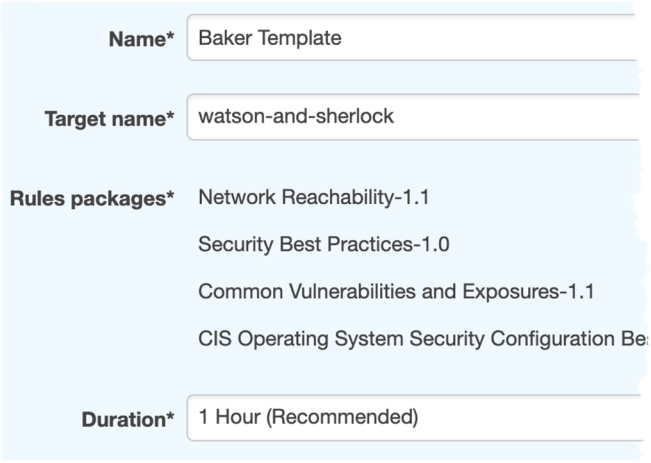
There are two main components to configure when preparing for an AWS Inspector Classic assessment run: the targets that will be scanned and the template dictating how the targets will be assessed. Configuring the targets is quite simple, but you do have a few different ways to determine which systems will undergo the assessment.

If your goal is to assess every instance in the region, you can check the box next to All Instances. The pre-winter 2021 version of Inspector, now called "Inspector Classic", has limitations on the number of instances that can be assessed. With the retooling the agent to use System Manager, there is no longer a limit.

If those limits require multiple assessment target configurations, or you would like to target a limited number of systems (i.e., not all of them, but a select number), you have the ability to add systems to the target configuration based on one or more tags that the instance(s) have applied to them. A tedious way to accomplish this would be to target each system by their `Name` tags, but a more efficient solution may be to add tags to each system that will be scanned and target by that single tag (e.g., `Inspector=BakerAssessment`).

AWS Inspector: Assessment Templates (Classic Inspector)

- Dictates the assessment specifics, such as:
  - Targets
  - Rules packages
    - Network Reachability
    - Security Best Practices
    - Common Vulnerabilities and Exposures (CVE)
    - CIS OS Security Benchmarks
  - Maximum duration
- Can also establish a schedule



We covered which systems could undergo an AWS Inspector Classic assessment run, but what about how the run will be conducted? This is where the second required component comes into play—the assessment template. An assessment template allows the customer to select the target group to be assessed, the rules packages which contain the assessment items to be checked, the maximum duration of the scan (15 minutes, 1 hour, 8 hours, 12 hours, or 24 hours), and even the AWS SNS topic to forward the results to. This template can be configured to run "on-demand" or on a set schedule.

Of the rules packages, there are up to four that can be chosen:

- Network Reachability: Displays which ports and services are exposed and where they can be accessed from over the network. This rules package does not require the agent package, but with the agent installed, can show the listening process information.
- Security Best Practices for Amazon Inspector: Assesses EC2 instance security posture (e.g., SSH configuration, password policies).
- Common Vulnerabilities and Exposures (CVE): Looks at installed software packages and determines if any are vulnerable to one or more CVEs.
- Center for Internet Security (CIS) Security Benchmarks: More rigorous EC2 instance security posture assessment that also covers more platforms than the Security Best Practices for Amazon Inspector.

AWS Inspector Version 2

AWS Inspector Version 2 fits in the same space as Inspector Classic, but its operation is remarkably different:

- No need for target groups or templates.
- Specify if scanning EC2 and/or ECR. Can be controlled at the organization level.
- Continual assessment, automated resource discovery, integration with EventBridge.
- Leverages a new API, so any automations have to be updated.

The AWS Inspector went through a major update in 2021. What was once an Inspector service specific agent was combined with the System Manager agent and rethought as a continuous analysis service. This is not an upgrade but providing the same business case in a completely new and streamlined way[1]. At the time of writing this class, there was not a log of information about Inspector 2 and how it was being used. The user guide is the best place to go read about Inspector[2].

Where the customer previously had to determine which EC2's to scan, and which set of rules to scan with, the Inspector just scans all eligible EC2's contionously. What is "continuous"? When a new system is deployed, when a new CVE is released, and when a new application is deployed.

Enabling AWS Inspector is now integrated with AWS organizations and can be turned on across all your AWS accounts. You specify if scanning for EC2, ECR, or both.

Continuous assessment is great, but there is little information about exactly what is being analyzed. The user guide states that the EC2's and ECR's will have "packages" analyzed for known CVEs, and network reachability. Inspector will evaluate instances, load balancers, subnets, security groups, and more to determine if your deployed environment is reachable and by what measure. However, there is no published data about what exactly is scanned, how up to date the CVE list is, and if there is private threat information that is evaluated. It is the author's recommendation to leverage those contracts to get more details about the Inspector scanning before ditching your favorite host scanner.

AWS also promotes integrations with 3rd party services[3], and integration with EventBridge and Security Hub for alert roll up, and maybe automated response actions.

The Inspector 2 is so different, a completely new API endpoint is needed[4], so any automated tooling that interacted with the AWS Inspector Classic will need to be re-written.

[1] https://aws.amazon.com/blogs/aws/improved-automated-vulnerability-management-for-cloud-workloads-with-a-new-amazon-inspector/
[2] https://docs.aws.amazon.com/inspector/latest/user/inspector-guide.pdf
[3] https://aws.amazon.com/inspector/partners/
[4] https://urldefense.com/v3/__https://docs.aws.amazon.com/cli/latest/reference/inspector2/index.html__;!!MlQdS1fu!GsmJB6HH0l_J6pJEBohDx0ndOSu4HW2ZyIKTSWx4Ie-JqK3YM2p8_euGc5xeYxH4xac$

AWS Inspector: Assessment Runs

As scans are performed, the results are displayed in the "Findings"

- Severity between Low and Critical. Focus on High and Criticals
- Clicking each line will open a Finding Details.

Severity	Title	Impacted ...	Type	Age	Status
Critical	CVE-2021-35942 - gli...	i-0cedcf6...	Package Vulnerability	43 minu...	Active
High	CVE-2022-0185 - kern...	i-0cedcf6...	Package Vulnerability	43 minu...	Active
High	CVE-2021-28691 - ker...	i-0cedcf6...	Package Vulnerability	43 minu...	Active
High	CVE-2021-4083 - kernel	i-0c20d87...	Package Vulnerability	42 minu...	Active
High	CVE-2021-3656 - kern...	i-0cedcf6...	Package Vulnerability	43 minu...	Active

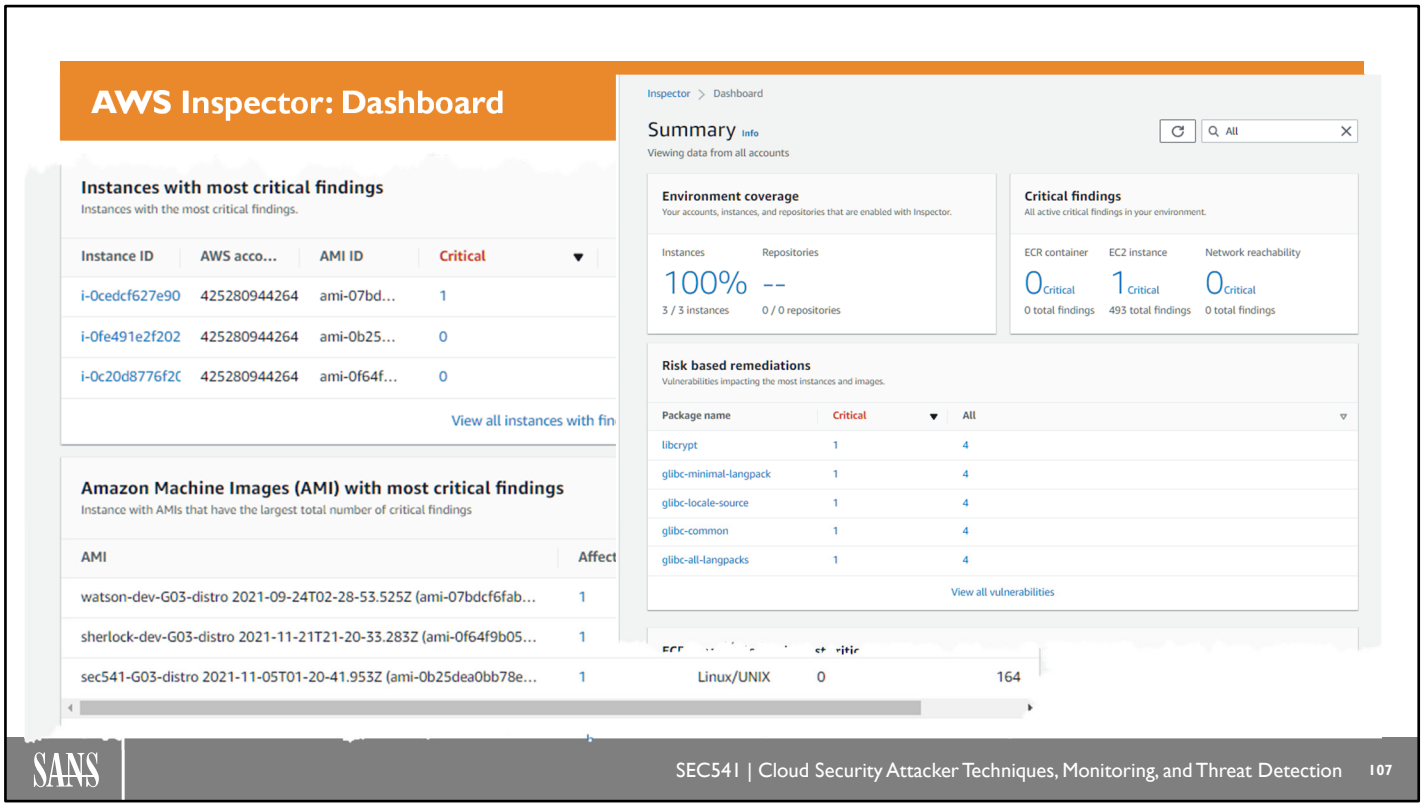
As Inspector starts to discover potential findings, they will be displayed on the "findings page"[1]. The findings page can be sorted by Instance type, by vulnerability, by container image, by container repository, or just all findings. The GUI also provides a filtering toolbar to limit by EC2, tag, severity, port, etc.

A filter can also be used as a suppression filter, to remove findings from the GUI that you have chosen to ignore.

Clicking a finding will provide detailed information about the finding, and also the resource that was scanned. It will provide the CVE, which package (software) that is vulnerable, the ID, type, and VPC information about the resource. Tags are also prominently displayed in the results, which can be helpful to try and track down the human owner, or the project team responsible for that instance.

Resources are broken down by severity of Critical, High, Medium and Low, Informational, and Untriaged.

[1] https://docs.aws.amazon.com/inspector/latest/user/findings-understanding.html



The dashboard will provide a big overview of the biggest risks, the most critical findings, the instances, AMI's, and ECR repositories that should be addressed first.

The dashboard also identifies the number of instances scanned and the number with critical findings.

Another nice features is that the Inspector will give a projected cost in the "Settings" section. So, first time users with a 15-day free trial, can also see what it will cost them after their trial runs out.

Discovering T1571: Non-Standard Port

- Once "normal" is known regarding expected listening ports, vulnerability analysis tools can help spot abnormal connections
- In this example, SherlocksBlog is listening on these TCP ports:
  - **22/tcp**: management traffic (SSH)
  - **5000/tcp**: web application traffic (HTTP)
- **AWS Inspector Network Reachability** assessment results:

On instance i-0d7756617d39aa96e, process 'ncat' is listening on TCP port 4444 which is reachable from the internet

On instance i-0d7756617d39aa96e, process 'python3' is listening on TCP port 5000 which is reachable from the in…

On instance i-0d7756617d39aa96e, process 'sshd' is listening on TCP port 22 which is associated with 'SSH' and i…

  - Notice anything out of sorts?

As mentioned, the main intention of the Vulnerability Analysis Services module is to learn how to discover two threats: adversarial activity via unapproved services and/or configurations as well as vulnerability management issues which lead to a more vulnerable environment. Let us first look at the adversarial activity by identifying MITRE ATT&CK T1571: Non-Standard Port[1].

To start, we must know normal so that the abnormal can be identified. If you have been paying close attention to your SherlocksBlog instance, you may have noticed that there are two TCP ports listening and reachable from the internet through your security group configuration: 22/tcp (where the SSH server service is listening) and 5000/tcp (where the custom, Python-based web application is listening). After conducting an AWS Inspector assessment run, you can see the Network Reachability results above after just a few minutes.

It appears that there is a new TCP port reachable from the internet and, since the AWS Inspector agent is installed on the instance, you can see that **`ncat`** is the process listening on that port. If this were adversarial activity, it appears that this could very well be a backdoor connection into this instance for persistence or a command and control channel!

[1] https://attack.mitre.org/techniques/T1571/

Discovering T1552.005: Access EC2 Metadata Service Vulnerability

- IMDS, if running, can accept one (or both) of these versions
  - Version 1: Leveraged during the Capital One attack, simply allows access to instance metadata over **`169.254.169.254`**
  - Version 2: Requires a token to be requested and then sent along with subsequent requests to **`169.254.169.254`**
- AWS Inspector *does not* have a check for IMDS version 1
  - Make your own check by looking for **`HttpTokens=optional`**

```
aws ec2 describe-instances | jq -r '.Reservations[].Instances[] |\
  select(.MetadataOptions.HttpTokens == "optional") .InstanceId'
```


And now let us look at the issue of a vulnerability left behind during our vulnerability management processes. Maybe it was due to the security team being unaware of this threat vector, but the Instance Metadata Service (IMDS) should not be configured in its default state—allowing both version 1 and version 2 queries. Version 1 is the dangerous IMDS implementation as this is what allowed the Capital One (and likely many other) AWS attacks where instance role credentials were stolen to be successful. This is because a few different web application attacks like remote file inclusion or command injection can trick the instance from querying the IMDS service directly, without providing any custom headers, and directly acquire the AWS access key ID, secret access key, and session token. This information will allow the attacker to pretend to be the AWS instance and make all of the same API calls that the instance is permitted to make.

IMDS version 2, on the other hand, forces the attacker to send a PUT request to the IMDS service to acquire a token and then pass that token with each subsequent request. This does not sound like much more work (and it really is not), but it does prevent certain attacks, like remote file inclusion targeting the IMDS service to acquire credentials, from being successful. This does not completely mitigate the attack but does force the attacker to use other means like command injection or establish a local present on the instance to be successful.

The issue here is that AWS Inspector does not currently have a method to assess for the IMDS version, so you will have to take matters into your own hands. This is not so hard to do, however, as an AWS CLI command and some filtering (as shown above) can output all of the instance IDs for a given region that is not forcing IMDS version 2 in its configuration. This same command should be repeated in each AWS region to ensure none are missed.

Microsoft Defender for Cloud: VM Vulnerability Assessment

Enabling the Vulnerability Management capabilities (powered by Qualys) creates two Microsoft Defender for Cloud findings:

- *Vulnerabilities in security configuration on your machines should be remediated*
  - Severity: Low
  - Results are found via Azure Log Analytics
- *Vulnerabilities in your virtual machines should be remediated*
  - Severity: Low
  - Results are found natively in Microsoft Defender for Cloud (as shown below)

ID	Security Check	Category	Severity
374891	Sudo Heap-based Buffer Overflow Vulnerability (Baron Samedit) (Generic)	Local	High
105936	OpenSSH Command Injection Vulnerability (Generic)	Security Policy	Medium

In the Azure ecosystem, an equivalent service to AWS Inspector would be a sub-component of Microsoft Defender for Cloud: the VM Vulnerability Assessment[1]. This assessment, if enabled on your Azure Virtual Machines (VM), is powered by Qualys to identify any common security misconfigurations and missing patches on your VMs.

It does take a few hours after enabling this feature to start seeing results, but after that time, you may see up to two Microsoft Defender for Cloud findings. That does not sound like a lot, but each of those findings can represent many more findings if you drill into them. In fact, both will show up as a low severity which can be quite misleading.

The first finding is *Vulnerabilities in security configuration on your machines should be remediated*. When drilling into this finding, you will see which operating system and application configurations are deemed to be in a vulnerable state. *Vulnerabilities in your virtual machines should be remediated*, on the other hand, identify any security patches that the VM is missing.

You can see an example of the missing patch results up above for a newly deployed Ubuntu 18.04 system in Azure.

[1] https://docs.microsoft.com/en-us/azure/defender-for-cloud/deploy-vulnerability-assessment-vm

Azure Log Analytics: Security Configuration Results

```
1  SecurityBaseline
2  | where Computer contains "logserver" or ResourceId contains "logserver"
3  | summarize arg_max(TimeGenerated, *) by BaselineRuleId, Computer
4  | where AnalyzeResult == 'Failed'
5  | summarize count() by BaselineRuleId, Description
```

Results | Chart | Columns | Display time (UTC+00:00) | Group columns

Completed. Showing results from the custom time range. 00:01.8 45 records

BaselineRuleId	Description	cour
4c066a3d-8eba-a21...	Disable the installation and use of file systems that are not required (freevxfs)	1
9967cbaf-44be-0dd...	Disable the installation and use of file systems that are not required (cramfs)	1
1895704d-a326-90c...	Ensure lockout for failed password attempts is configured.	1
b042fda5-55f6-17ba...	Ensure password creation requirements are configured.	1
eb6c608a-2316-41b...	Ensure default deny firewall policy	1
429123d4-7e7d-273...	Ensure core dumps are restricted.	1

Here is an example of drilling into *Vulnerabilities in security configuration on your machines should be remediated*. Notice that this looks a bit familiar and not like Microsoft Defender for Cloud. That is because these results are stored in Azure Log Analytics. The query is actually created for you but can easily be modified to drill into the results as you deem appropriate.

The results above are from the same Ubuntu 18.04 system so you can tell that the vulnerability management team missed a large number of security configurations when this system was deployed. In fact, no security recommendations were made as this system is a newly-deployed Ubuntu 18.04 system with a few pieces of software installed (i.e., no security hardening was conducted at all). That is why you see so many findings!

Additional Vulnerability Discovery Techniques

- What was discussed up to this point is far from a complete list of available options regarding vulnerability analysis in the cloud
  - **Infrastructure as a Service (IaaS)** was the main focus
    - Customer is responsible for most components that could become vulnerable
  - In fact, there may not be a cloud-native option
    - May need to extend your current solution or build you own
- Since **containers** are becoming increasingly more popular, let's take a look at options for this technology
  - Find the vulnerabilities (yet again) before they are deployed
  - Identify breakdown of any security processes within the organization

One technology that was not yet covered in this section that is very important to account for is container images. Just like instances and VMs, container images can have vulnerabilities "baked into them" that, if not addressed, could lead to a compromised container running in our infrastructure.

If you can address those by updating or reconfiguring the image, that should decrease your number of threat vectors, but you first must be able to identify those flaws. The same vulnerability scanning software that you use in your on-premise environment may still be leveraged here to some degree, but there are some cloud-native options at your disposal yet again from major cloud providers and third parties. These solutions will identify any configuration or patch-related issues resident in the container image itself.

Containers, by nature, tend to be deployed from a locally managed binary image in a repository. AWS and Azure have repositories in your accounts. AWS Inspector and Azure Defender will scan the images in your repositories, evaluating them for known vulnerable applications or libraries. This is more efficient than running some type of agent system on running containers, which will significantly increase the execution requirements. Besides, you aren't running strange containers you pulled from the internet, are you?

AWS ECR

- Container registry allowing the creation of public or private container image repositories
- Security scanning options to assess image on "push"
  - Specifically identifies **Common Vulnerabilities and Exposures (CVE)** present in container images
  - Provides direct links to the discovered CVE for more information

Name	Package	Severity
CVE-2017-16931	libxml2:2.9.4-r4	HIGH
CVE-2018-18312	perl:5.24.1-r2	HIGH
CVE-2018-6913	perl:5.24.1-r2	HIGH

The first solution to discuss when dealing with container image scanning is the AWS Elastic Container Registry (ECR). This is often thought of for its primary purpose: housing container images. These images can be public (similar to how many will share their images using DockerHub) or private for use within the AWS Elastic Container Service (ECS) or the AWS Elastic Kubernetes Service (EKS).

One security service that comes along with AWS ECR is, when a repository is created to store the images, a simple checkbox (Scan on push) enables security scanning on any image upload. This scan that is being conducted searches for any CVEs that may be present in the container image as reported by Clair—an open-source container image scanning software package. This is now integrated with AWS Inspector's newly updated service.

When the scan completes, you can view the results and their indicated severity (Critical, High, Medium, Low, Informational, Undefined). Above are a few results for the `httpd:2.4.25-alpine` container image. AWS maintains a list of the major operating systems that Inspector can scan in EC2 and ERC mode3.

[1] https://docs.aws.amazon.com/AmazonECR/latest/userguide/image-scanning-enhanced.html
[2] https://docs.aws.amazon.com/inspector/latest/user/supported.html#supported-os
[3] https://docs.aws.amazon.com/inspector/latest/user/supported.html#supported-os

Microsoft Defender for Cloud: Container Image Scanning

- Microsoft Defender for Cloud also scans container images for vulnerabilities
  - Natively powered by **Qualys**
  - Options are available to leverage **Aqua Security** or **Twistlock** via the **Azure Marketplace**
  - Can indicate CVEs **and** configuration issues within the image
- Will indicate category, severity, and whether a patch is the ultimate fix to address the vulnerability

Security Check	Category	Applies To	Severity	Patch Available
User(s) With Blank Password	Security Policy	1 of 1 Scanned Images	High	No
Linux Docker Image Hard-Coded Credential Vulnerability	Local	1 of 1 Scanned Images	High	Yes
Apache httpd Server ap_get_basic_auth_pw() Authentication Bypass Vulnerability	Web server	1 of 1 Scanned Images	Medium	Yes

Just like scanning Azure VMs, Microsoft Defender for Cloud allows one to identify vulnerabilities within a container image. This scanning is also powered by Qualys to identify CVEs and configuration issues within the image. All flaws are viewable in the Microsoft Defender for Cloud service and breaks the results down by the security check description, category of the finding, how many images are impacted, the severity of the flaw, and if the issue is fixable via a patch.

The same image that was scanned and displayed previously by AWS ECR (`httpd:2.4.25-alpine`) is shown here. Notice the biggest difference: vulnerabilities that are not CVEs (something that, as of the time of writing, AWS ECR does not assess).

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. **EXERCISE: Vulnerability Analysis**
10. Data Centralization Techniques
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Lab 3.4 | Vulnerability Analysis

Exercise Duration: 30 Minutes

Objectives

We will explore AWS SSM, AWS Inspector, and AWS ECR to identify vulnerabilities within our AWS environment

- Install Inspector on both WatsonsBlog and SherlocksBlog instances using AWS SSM
- Create Inspector template and launch vulnerability assessment
- Upload development team's candidate web server container image and assess with Clair
- Review AWS Inspector results

We will be using several AWS services to assess our current AWS environment as well as analyze a container image before it is placed into the production environment. This is all in an effort to get in front of these vulnerabilities before an attacker can leverage them to successfully infiltrate our cloud environment. We will ensure that Inspector is installed and available on WatsonsBlog and SherlocksBlog, then we will perform the analysis.

We will upload a container to the account for assessment.

When Inspector is done, we will analyze the results.

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. **Data Centralization Techniques**
11. EXERCISE: Data Centralization with Graylog

This page intentionally left blank.

Data Centralization

- Cloud service- and resource-generated log data at this point may be in several different locations:
  - API Activity Logs
  - Cloud-Native Log Collection Solution
  - Host Operating System and Application Logs
  - Cloud Storage
- Analysts and Incident Responders are likely to prefer the log data in a single location
  - More efficient to query one system
  - Requires knowledge of a single platform
  - Easier correlation of events

Up to this point in class, you have learned about the many services that can aid in the detection and analysis of adversarial activity occurring within your cloud environment. This is a great start but put yourself in the shoes of a cybersecurity analyst. Would it be convenient or even efficient to have to pivot between many different service consoles to analyze a complex attack? Not likely.

Take the Capital One attack, for instance. The attacker compromised a public-facing EC2 instance, stole instance role credentials, and used those credentials to steal data from an AWS S3 bucket. If logging is enabled on every affected resource, you would likely have:

- AWS EC2 instance's operating system and application log data (possibly forwarded to AWS CloudWatch)
- AWS CloudTrail logs capturing the instance role credential activity
- AWS S3 events in CloudWatch (if AWS CloudTrail is configured to generate this level of activity)

Some centralization can be had by sending as many events to as few unique locations as possible: like AWS CloudWatch and AWS S3 (to be searched via AWS Athena), but what happens if the services you are trying to monitor do not support forwarding to these locations? Or even further, what if you have services that you are leveraging outside of AWS? Would it not be nice to have all of that rich activity and event data in a single location?

This module will focus on centralizing your data to ease the burdens that you or your cybersecurity analysts may be dealing with when using disparate systems.

Security Information and Event Management (SIEM)

- Gather **security-related event data** from various sources
  - Host operating systems and applications
  - Cloud services
- Search services allow for quick and efficient analysis
  - Some solutions will **automatically correlate** data for the analyst
  - **Alerts** can be created if a predetermined set of actions occurred
- More advanced SIEMs can:
  - Generate **User and Entity Behavioral Analytics (UEBA)**
  - **Automate** response actions
- In other words, **not a single place to ignore your log data**!

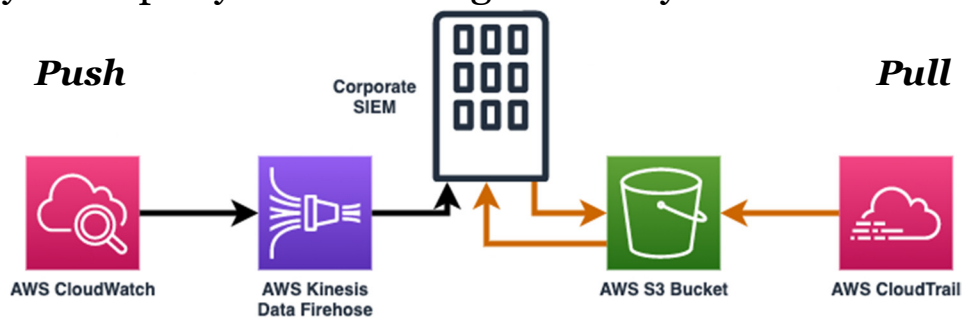
A very common solution to receive log data is a Security Information and Event Management (SIEM) platform. Not only can these solutions ingest and store data at very impressive speeds (often advertised with the metric of events per second), but these platforms also allow analysts to search and parse the data very quickly.

Furthermore, the SIEM is customizable by the analyst to, either using their own indicators or community rules, generate alerts based upon all of the correlating data that is discovered. This is typically performed by crafting custom searches and, if any hits or so many hits within a given timeframe are discovered, alert that something suspicious is happening with some context around this discovery.

More advanced SIEMs can even monitor user activity for oddities such as off-hours connections or an "impossible traveler" scenario where the user connects from one location and then another, distinct location where it will have been impossible to go from point A to point B using today's commercial travel in that short amount of time.

Log Data Export

- Log and event data stored in cloud storage or within a cloud-native logging service can oftentimes be exported
  - **Push**: Cloud vendor solution will send data to an external endpoint
  - **Pull**: Log aggregation solution will pull data from cloud environment
- Many third-party vendors integrate nicely with cloud services



A common approach to centralize your event data could be to either procure a new or reconfigure an exist log aggregation solution. Whether it is a full-fledged Security Information and Events Management (SIEM) platform or a simple log repository that has search capability, cloud vendors will often include techniques to transition this data from the cloud-native services to the log aggregation platform.

When the data is migrated outside of the cloud environment, it can happen in two different ways: push or pull. When the cloud service pushes data to the logging solution, it is typically configured, once an event is generated, to transfer the event details directly listening service on the logging solution. The example above shows a service that will be discussed shortly: AWS Kinesis. This service gives the ability to push data from multiple cloud services to an external source with relative ease. The hardest part, or course, is establishing the proper network configuration to allow the communication to the log aggregator and do so securely (e.g., over an encrypted channel). Typically, this technique will be the more performant of the two due to the data being streamed in real time to the log solution.

The other approach, as mentioned, is pull. With this technique, the logging platform will be configured to, on a regular basis, connect to the cloud service and download any data since the last request. There is much more involved for log aggregator as it must track which data is considered "new" since the last pull so that data is both not duplicated as well as not missed. The example above shows the corporate SIEM regularly checking into an AWS S3 bucket for any new data. If there is new data, it will be pulled into the SIEM. This method can also be not as performant as the data is not streamed in real-time.

Data Minimization

- It is easy to say, “log all the things”, but
  - Analysts could become overwhelmed
  - False positive counts could be high—hiding the real threats
  - Log storage costs will skyrocket in busy environments
- We must only collect the log sources and field/value pairs that are absolutely needed
  - Easier said than done!
  - Not all vendors allow you to select which fields will be logged
- Filtering the logs **in transit** may be the option to handle this

Let's face it: these cloud services can generate a lot of data, but not all of it may be useful to an analyst or even an operator within the cloud environment. For example, AWS CloudTrail will generate 45 unique field/value pairs when the `ListBuckets` API call is made. There are quite a few of these field/value pairs that are useful like many of the `userIdentity` sub-fields, `eventTime, eventName`, and many others, but there are a few that would have little to no value like `webIdFederationData` if your company is not using federation services.

Of course, this is an edge case, but there are many events with many fields that may never be leveraged by anyone on your staff and storage of this data can add up and negatively impact the analysis process and budget. The more data to search through, the more latent the search results will be, and this "useless" data may be a part of the scope of the search.

This additional storage will not only cost the company more funds to consume, but some SIEM vendor licenses charge by the amount of data ingested as well. If we could minimize this, yet still get the most valuable data, we can save the company some money which could be used to fund other security projects.

Data Enrichment

- Taking what used to be minimal data and add more context
  - Many SIEM providers support this capability in some way
  - To perform outside of the SIEM, the data would be enriched in transit
- Examples:
  - `src_ip = 1.1.1.1` could become
    - `geo_country = AU`
    - `geo_city = South Brisbane`
    - `geo_timezone = GMT +10`
  - `www.amazonaws.com` could become
    - `resolved_ip = 72.21.206.80`
    - `tld = .com`

On the flip side, there may be cases where we would want to increase the amount of data to add more context to the event. This is performed by a process known as enrichment: taking a piece of information and adding more context. For example, we could have an event with an IP address contained in it. The data, as it is, does not tell us which country this IP address was registered in. If we have the proper SIEM plugins or homegrown processes in place to acquire this data and add it to the log entry, we may be able to see things like the country, city, time zone, or anything else the enrichment process can give us related to this IP address.

With this new data that we have not seen before, more suspicious activities may stick out. Maybe the analyst finds out that there are countries communicating with our cloud systems that we do not conduct business with or maybe there are some very strange top-level domains (TLD) that we would not otherwise recognize as easily. Enrichment will be necessary for those services which provide very little insight.

Field Normalization

- Log sources may represent the same data in different ways
  - Field names may not be consistent, for example:
    - `{ "src_ip": "203.0.113.42" }`
    - `{ "sourceIPAddress": "203.0.113.42" }`
  - Value types may not be consistent
    - `{ "totalBytes": 541 }`
    - `{ "totalBytes": "541" }`
- A few vendors took initiative to normalize these fields and values
  - **Elastic Common Schema (ECS)**
  - **Splunk Common Information Model (CIM)**
- SIEM products or our own processes can **normalize** this data

As data is exported from various sources, even within the same vendor in some cases, you may find that fields representing the same data may contain wildly different names. For example, product A may have a field name for a source IP address notated as `src_ip`, but vendor B, representing the same source IP address information, may notate the field as `sourceIPAddress`. The same can be said for the value types. One vendor may show a numeric value as a string while another may represent the value as a number.

This can lead to quite a bit of confusion and inefficiency when analyzing an attack. If we can, it would be wise to standardize (i.e., normalize) these field names and value types to be consistent and expected by the analyst. There is no right answer to what these fields should be called, but it should be consistent across all of the organization's log data where possible. A few vendors, in an attempt to create standards across the industry, created their own schemas/models for log data and the vast array of potential fields and values which could be ingested by a SIEM or log aggregation platform. The folks at Elastic created the Elastic Common Schema (ECS)[1] and Splunk created the Splunk Common Information Model (CIM).[2]

Again, neither one is better than the other and you may find that you have your own ideas, but the point is—help you analysts by being consistent in your approach!

[1] https://www.elastic.co/guide/en/ecs/current/index.html
[2] https://docs.splunk.com/Documentation/CIM/4.19.0/User/Overview

Log Data in Transit Minimization and Enrichment Example

Using a tool like **Logstash**, logs can be parsed and manipulated prior to arrival at the SIEM

- Data Flow: AWS Kinesis Data Firehose → Logstash → Elasticsearch

```
filter {
  # Resolve Hostname
  dns {
    resolve => "host"
    action => "append"
  }
  # Remove version and type fields (not needed)
  drop {
    remove_field => ["version", "type"]
  }
}
```


A method to enrich or minimize data may not actually exist in the cloud services themselves or even in the log aggregation service, but you may be able to modify this log data in-transit after it exits the cloud service and before it arrives at the log aggregation system or SIEM. One of many products that may aid with this, and one of the course authors' favorite tools for this and many other purposes, is Logstash.[1]

Logstash, from Elastic, can act as a "machine in the middle" which, instead of conducting nefarious business, will ingest the log data, filter this data, and then output the reformatted or modified data to the log aggregation platform or SIEM. The example above, as simple as it is, both minimizes data (removing, what is deemed in this particular example, as "useless" fields like the version of the sending platform and the log type) and enriches the data (when a hostname is seen, resolve its IP address).

If we started with a scenario where AWS Kinesis was sending data directly to an Elasticsearch database, one such way to involve a tool like Logstash would be to reconfigure the AWS Kinesis Data Firehose configuration to, instead, direct the data to a customer-managed Logstash instance. From there, Logstash will filter the data accordingly and forward it to the Elasticsearch database.

[1] https://www.elastic.co/logstash

AWS EventBridge

- Watches for an event and forwards data to another service/endpoint based on a user-created rule
  - **Event**: Some change or activity in AWS environment
  - **Rule**: Processes the event and forwards to a downstream target
  - **Target**: AWS service or custom location receiving the rule's output
- Many useful options, including:
  - Watch for specific API events and send to custom HTTP endpoints
  - Trigger an AWS Lambda function if suspicious activity is identified
  - Automate response actions

The first AWS solution we will look at to push data from one cloud service to a custom location is AWS EventBridge[1]. This service has three major components for the cloud customer to configure. First, the event in question must be identified. This is configured by selecting the cloud service whose activity must be captured and which events are needed.

This may sound like a lot of work if you had to consider every single event that you want to forward, but it can be much easier than that as you can elect to forward all of the cloud service's events in one shot. However, you do have the flexibility to choose which events, like specific AWS CloudTrail API calls, to forward to a target.

Once the events are determined, rules will be used to take these events and forward them to an appropriate target. These targets can be other AWS cloud services or custom locations which can communicate via HTTP. Rules can also transform data into particular formats if the receiving system would prefer them in a format other than JSON or if certain data should be excluded.

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

AWS Kinesis

- **Data Stream**
  - Short-term (one day to one year) data storage service that can receive cloud events in near real-time
  - External services can then query and retrieve data
- **Data Firehose**
  - Can forward Data Stream or direct PUT requests to:
    - Other AWS services (e.g., Amazon S3, Amazon RedShift, Amazon Elasticsearch)
    - HTTP Endpoint
    - Third-party service provider
  - AWS Lambda can be involved to transform records, if needed
  - Data can also be converted from JSON to Apache Parquet or Apache ORC format, if needed

Similar in some ways to AWS EventBridge, AWS Kinesis[1] allows cloud customers to forward event data outside of the AWS ecosystem. In fact, AWS Kinesis supports both pull and push methods. In any case, the data must arrive at the AWS Kinesis service. This is done by configuring a Data Stream.

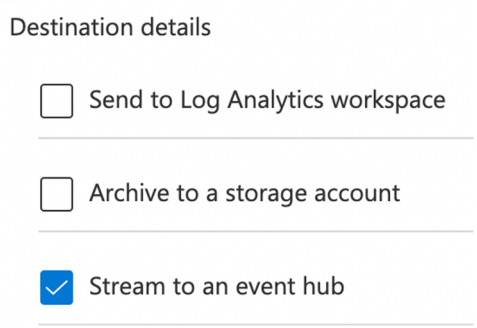
A Data Stream consists of a few sub-components: a producer and a consumer. The producer is the AWS service or any system that can send log data via an HTTP PUT request to the Data Stream for the short-term (one day to one year). From there, a consumer will connect to (via an HTTP GET) the Data Stream to retrieve the log data.

What was just discussed was the pull method (log solution will pull the data from the Data Stream), but how does push work? This is where the Data Firehose component will come in. With this, the Data Stream is still receiving data, but Data Firehose will take this streamed data and forward it to a destination of the cloud customer's choosing. This could, again, be another AWS cloud service or a custom HTTP endpoint of the log aggregation solution. Furthermore, AWS Kinesis Data Firehose allows the cloud customer to reformat the data if needed.

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

Azure Event Hubs

- Centrally collect log data at rapid speed (millions of events per second) from Azure cloud services
  - Once collected, can forward to long-term storage
  - When used with Azure Stream Analytics, can forward event data to external services
- Very simple to set up when configuring **Diagnostics Settings** for each Azure resource
  - If cost is not a problem and for failover, recommended to also send to a backup location (e.g., Azure Storage Account)



Azure's answer to moving data outside of the Azure ecosystem is via the Azure Event Hubs service[1]. Like AWS Kinesis, Azure Event Hubs act as a staging area for event data sent by the cloud customer's cloud services.

Configuration of the Azure services to send their event logs and metrics to Azure Event Hubs is quite simple. Through the Diagnostics Settings configuration within the Azure service, the customer can select which data to forward (as was previously discussed in class) and where the data should be forwarded to. One such option is to stream the data directly to the Azure Event Hub where the data can be pulled from or pushed to (using Azure Stream Analytics) the log aggregation platform.

[1] https://azure.microsoft.com/en-us/services/event-hubs/#overview

Security Orchestration, Automation, and Response (SOAR)

- Automates response actions
  - Identify what is believed to be malice
  - Communicate with security product to take action against malice
  - Example: If **brute force attack** from **203.0.113.42**, then **block** in **Network Access Control List (NACL)**
- Oftentimes, you can build your own SOAR-like functionality using serverless technologies
  - Example: Respond to non-EC2 instance making API calls using instance role credentials
  - How could we build this using cloud-native AWS services?

To differentiate themselves from SIEMs, Security Orchestration, Automation, and Response (SOAR) platforms perform more advanced actions like automating the identification of threats in the environment and responding appropriately to those threats. These responses to the suspicious activity could come in the form of disabling user accounts, placing firewall deny rules, and so much more. These actions, however, will require integration with other security products. For instance, if the SOAR platform wishes to disable a Windows Active Directory user due to suspicious activity, it will need to have a connection with proper access to do so.

In lieu of a full-fledged SOAR platform, we may have the opportunity to create similar techniques using cloud-native services. Since many cloud services can integrate with each other to support operations, we, as the security team, can use this to our advantage by creating a cloud service workflow to capture event data, determine a threat, and respond by modifying the cloud environment where necessary.

But what is a practical example of this? Let's assume we have a system outside of AWS using an instance role just like we saw in the Capital One attack. What techniques could we deploy to automate a response to this?

AWS Automated Response (1)

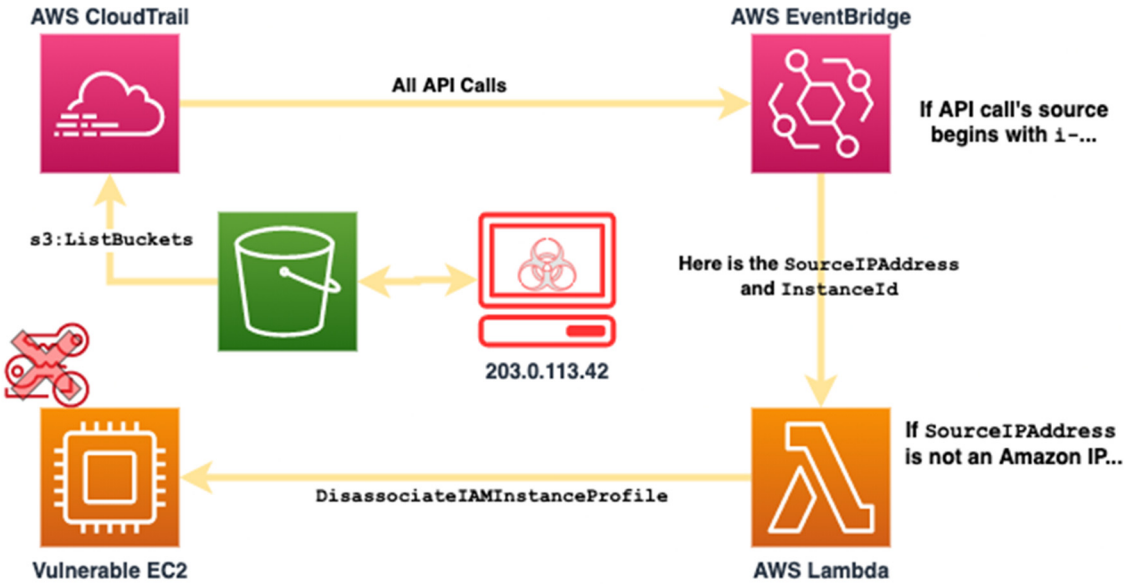
- **AWS CloudTrail API events** will have the data we need
  - But this service alone cannot respond to the threat
- Integrate AWS CloudTrail with **AWS EventBridge**
  - AWS EventBridge does not take action, but a **rule** can be created looking for API calls from instance roles
- Forward relevant calls to **AWS Lambda**
  - This service can make changes to the environment, if needed
  - The change could be to **revoke the instance role** so that it can no longer be used

Let's take a look at the real-world scenario from the previous slide and how we can take cloud-generated data and respond automatically. The initial source, in this example, is AWS CloudTrail API events that would contain instance role activity. As we know, AWS CloudTrail, itself, cannot respond to a potential attack—let alone determine if the instance role activity is from outside the AWS environment—but it can forward data to another service like AWS EventBridge. Why AWS EventBridge? Because this service, as we discussed, can forward data to other cloud services.

Just like AWS CloudTrail, AWS EventBridge cannot make any automated adjustments to protect the environment, but AWS Lambda can. AWS Lambda will be the destination in which AWS EventBridge forwards the AWS CloudTrail data.

From here, AWS Lambda can parse the log data, look for the requesting IP address, and see if that address belongs to any instances in the AWS account. If not, the activity can be assumed malicious and AWS Lambda can then put appropriate measures in place like revoking the instance role (which, in turn, will make the credentials unusable from here on out).

AWS Automated Response (2)



Above is a breakdown of what this automated process would entail. The attacker in this situation that stole instance role credentials performed some activity (**`s3:ListBuckets`**) in the AWS account which is captured by AWS CloudTrail.

Since AWS EventBridge is monitoring AWS CloudTrail API calls and forwarding them to AWS Lambda, AWS Lambda can do the analysis we discussed: compare the `SourceIPAddress` value to what is actively used in the AWS account. If no match, assume a compromise and remove the role from the instance with the `InstanceId` in the original event.

Example Investigation in Graylog

- This is *not* a Graylog 101 training course, *but* we will drive home why analysts prefer getting all of their data in one familiar place
- The Graylog server will have data ingested from the following cloud-based log sources:
  - AWS CloudTrail (forwarding to AWS CloudWatch)
    - Added AWS S3 data events to view bucket object activity
  - AWS CloudWatch (Apache web server access logs)
- With only three queries in a single platform, we will investigate a Capital One-style attack
- Let's see what was happening in this cloud environment!

To see a data centralization approach in action, we will take a look at Graylog[1]. This is just an example platform we will discuss, but the process will be very similar with other products allowing you to search all of your cloud data in one place.

The data that Graylog is ingesting is actually via AWS Kinesis Data Streams. The Data Streams are receiving data from AWS CloudTrail and AWS CloudWatch. It's it quite the complex setup though. First, AWS CloudTrail is configured to forward not only its default API activity to an AWS CloudWatch log group, but also the AWS S3 activity is captured and forwarded to the log group. From there, the AWS CloudWatch log group has what is called a subscription filter to send its event data, as it is received, to an AWS Kinesis Data Stream.

There is a second Data Stream that is receiving data from another CloudWatch group which is receiving data from an EC2 instance—a vulnerable web server. Finally, Graylog is configured to pull data on a regular basis from the Data Stream.

With all of this, we should be able to capture all of the key Capital One-style attack stages all in one place—the Graylog server. Let's see how one would analyze this attack using Graylog or a similar SIEM product that leverages Lucene search syntax.

[1] https://www.graylog.org

Example Investigation in Graylog: T1552.005: Cloud Instance Metadata API

- Search: **`log_source: apache AND "169.254.169.254"`**
  - Purpose: Identify credentials stolen from IMDS

2021-05-31 21:14:55.000 +00:00 aws-kinesis-raw-logs

```
203.0.113.42 - - [31/May/2021:21:14:55 +0000] "GET /geolocation.php?ipaddr=|curl%20http://169.254.169.254/latest/meta-data/iam/security-credentials HTTP/1.1" 200 12 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36"
```

2021-05-31 21:15:01.000 +00:00 aws-kinesis-raw-logs

```
203.0.113.42 - - [31/May/2021:21:15:01 +0000] "GET /geolocation.php?ipaddr=|curl%20http://169.254.169.254/latest/meta-data/iam/security-credentials/HRPortalRole HTTP/1.1" 200 1322 "-" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36"
```

- Looks like two command injection attacks were attempted:
  - First: Identify name of any attached instance role
  - Second: Acquire keys and token from **`HRPortalRole`** instance role

Before we begin analyzing this data, we must understand the available fields to search. The engineers of this SIEM took it upon themselves to work toward consistency and ease of use by creating a field called log_source which can be used to pivot between various log sources: `Apache` (EC2 instance's web server logs) and `CloudTrail`. We will use both of these to analyze the attack.

First, we will identify T1552.005: Cloud Instance Metadata API[1]. One very common approach an attacker may take is to try a Server-Side Request Forgery (SSRF) in the form of a Remote File Inclusion (RFI) attack. As we know from the metadata services section, an attacker may attempt to force the vulnerable server to navigate to and return the results from `http://169.254.169.254/latest/meta-data/iam/security-credentials` so that they know the name of the instance role. This will allow the subsequent attack in which the server will be forced to navigate to `http://169.254.169.254/latest/meta-data/iam/security-credentials/<role-name>`.

With that key information, a search to identify this data in Lucene syntax would look like this:

```
log_source: apache AND "169.254.169.254"
```

This will show any web server log that contains the text "`169.254.169.254`". Looks like we have two hits that look like the same attack outlined above!

[1] https://attack.mitre.org/techniques/T1552/005/

Example Investigation in Graylog: T1526: Cloud Service Discovery

- Search: `log_source: cloudtrail AND eventName: "ListBuckets" AND "ASIAXKBAMZT7KTCIFKK2"`
  - Purpose: Listing AWS S3 buckets using stolen credentials

timestamp	source	sourceIPAddress	eventName	userIdentity_accessKeyId
2021-05-31 21:28:47.176 +00:00	aws-kinesis-raw-logs	203.0.113.42	ListBuckets	ASIAXKBAMZT7KTCIFKK2


```
{"eventVersion":"1.08","userIdentity":{"type":"AssumedRole","principalId":"AROAXKBAMZT7AHJXZZHXV:i-0a1ca1c66d9313688","arn":"ar
n:aws:sts::012345678910:assumed-role/HRPortalRole/i-0a1ca1c66d9313688","accountId":"012345678910","accessKeyId":"ASIAXKBAMZT7KT
CIFKK2","sessionContext":{"sessionIssuer":{"type":"Role","principalId":"AROAXKBAMZT7AHJXZZHXV","arn":"arn:aws:iam::01234567891
0:role/HRPortalRole","accountId":"012345678910","userName":"HRPortalRole"},"webIdFederationData":{},"attributes":{"creationDat
```


- It appears that just a few minutes later, buckets were listed
  - But were bucket objects accessed?

If we are concerned that these stolen credentials may have been used to discover AWS S3 buckets, we could look at another log source: `cloudtrail`. One of the fields that comes along with AWS CloudTrail events is the `eventName` field. We have seen these before, so the API call we should be looking for here is `ListBuckets` as this will show an attacker which AWS S3 buckets are accessible using these stolen credentials.

There may be many occurrences of s3:ListBuckets across your AWS account and they likely are not all malicious. In fact, most will be benign, so how would we narrow down only those calls from the stolen credential? That easy! Just add the access key ID that is associated with the `HRPortalRole` as another filter to our search. The final search for this scenario would look something like this:

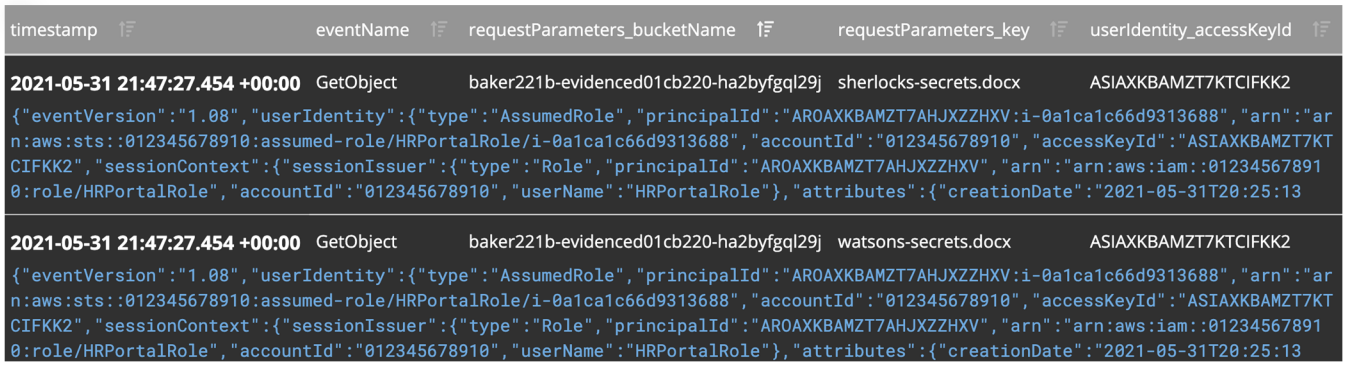
```
log_source: cloudtrail AND eventName: "ListBuckets" AND
  "ASIAXKBAMZT7KTCIFKK2"
```

It again appears that our suspicions are correct. But did any sensitive data find its way into the hands of the attacker?

Example Investigation in Graylog: T1530: Data from Cloud Storage

Search: `log_source: cloudtrail AND eventName: "GetObject" AND "ASIAXKBAMZT7KTCIFKK2"`

- Purpose: Discover downloading AWS S3 objects from sensitive bucket



timestamp	eventName	requestParameters_bucketName	requestParameters_key	userIdentity_accessKeyId
2021-05-31 21:47:27.454 +00:00	GetObject	baker221b-evidenced01cb220-ha2byfgql29j	sherlocks-secrets.docx	ASIAXKBAMZT7KTCIFKK2

{"eventVersion":"1.08","userIdentity":{"type":"AssumedRole","principalId":"AROAXKBAMZT7AHJXZZHXV:i-0a1ca1c66d9313688","arn":"arn:aws:sts::012345678910:assumed-role/HRPortalRole/i-0a1ca1c66d9313688","accountId":"012345678910","accessKeyId":"ASIAXKBAMZT7KTCIFKK2","sessionContext":{"sessionIssuer":{"type":"Role","principalId":"AROAXKBAMZT7AHJXZZHXV","arn":"arn:aws:iam::012345678910:role/HRPortalRole","accountId":"012345678910","userName":"HRPortalRole"},"attributes":{"creationDate":"2021-05-31T20:25:13

timestamp	eventName	requestParameters_bucketName	requestParameters_key	userIdentity_accessKeyId
2021-05-31 21:47:27.454 +00:00	GetObject	baker221b-evidenced01cb220-ha2byfgql29j	watsons-secrets.docx	ASIAXKBAMZT7KTCIFKK2

{"eventVersion":"1.08","userIdentity":{"type":"AssumedRole","principalId":"AROAXKBAMZT7AHJXZZHXV:i-0a1ca1c66d9313688","arn":"arn:aws:sts::012345678910:assumed-role/HRPortalRole/i-0a1ca1c66d9313688","accountId":"012345678910","accessKeyId":"ASIAXKBAMZT7KTCIFKK2","sessionContext":{"sessionIssuer":{"type":"Role","principalId":"AROAXKBAMZT7AHJXZZHXV","arn":"arn:aws:iam::012345678910:role/HRPortalRole","accountId":"012345678910","userName":"HRPortalRole"},"attributes":{"creationDate":"2021-05-31T20:25:13

To see if the sensitive data was stolen, we can again use the `cloudtrail` log source. This time, we will just need to change the `eventName` to `GetObject` as this would indicate that the stolen credential was attempting to download AWS S3 data. In this case, the search will look like this:

```
log_source: cloudtrail AND eventName: "GetObject" AND
  "ASIAXKBAMZT7KTCIFKK2"
```

Uh oh! Two files appear to have been accessed: `sherlocks-secrets.docx` and `watsons-secrets.docx`.

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

Cloud Service and Data Discovery

1. Capital One Attack
2. Metadata Service and GuardDuty
3. EXERCISE: Metadata and GuardDuty
4. Cloud Inventory
5. EXERCISE: Cloud Inventory
6. Data Discovery
7. EXERCISE: Detecting Sensitive Data
8. Vulnerability Analysis Services
9. EXERCISE: Vulnerability Analysis
10. Data Centralization Techniques
11. **EXERCISE: Data Centralization with Graylog**

This page intentionally left blank.

Lab 3.5 | Data Centralization with Graylog

Exercise Duration: 45 Minutes

Objectives

This lab will be a bit different in that we will be analyzing a previously-run attack in an unfamiliar environment (welcome to the day job of an employee of a Managed Security Service Provider)

- Deploy Graylog server to analyze a previous attack
- Walk back the attack using the provided data to determine which MITRE ATT&CK techniques were used to support the following tactics:
  - TA0040: Impact
  - TA0006: Credential Access
  - TA0003: Persistence
  - TA0001: Initial Access (x2)

In this lab, instead of analyzing data in your own AWS account, we will be assisting another organization analyze an attack. Luckily for us, they had deployed a Graylog server which was ingesting very relevant data during this attack. We will discover very quickly how much more convenient and efficient parsing all of this otherwise disparate data in a single tool.

Section 3 Wrap Up

Cloud Service and Data Discovery

- Discussed the infamous Capital One attack
- Detecting T1530 with Cloud Inventory
- Detected T1105 with Data Discovery
- Learned about Macie
- Detected T1190 with Vulnerability Analysis Services
- Improving detection with data centralization

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.