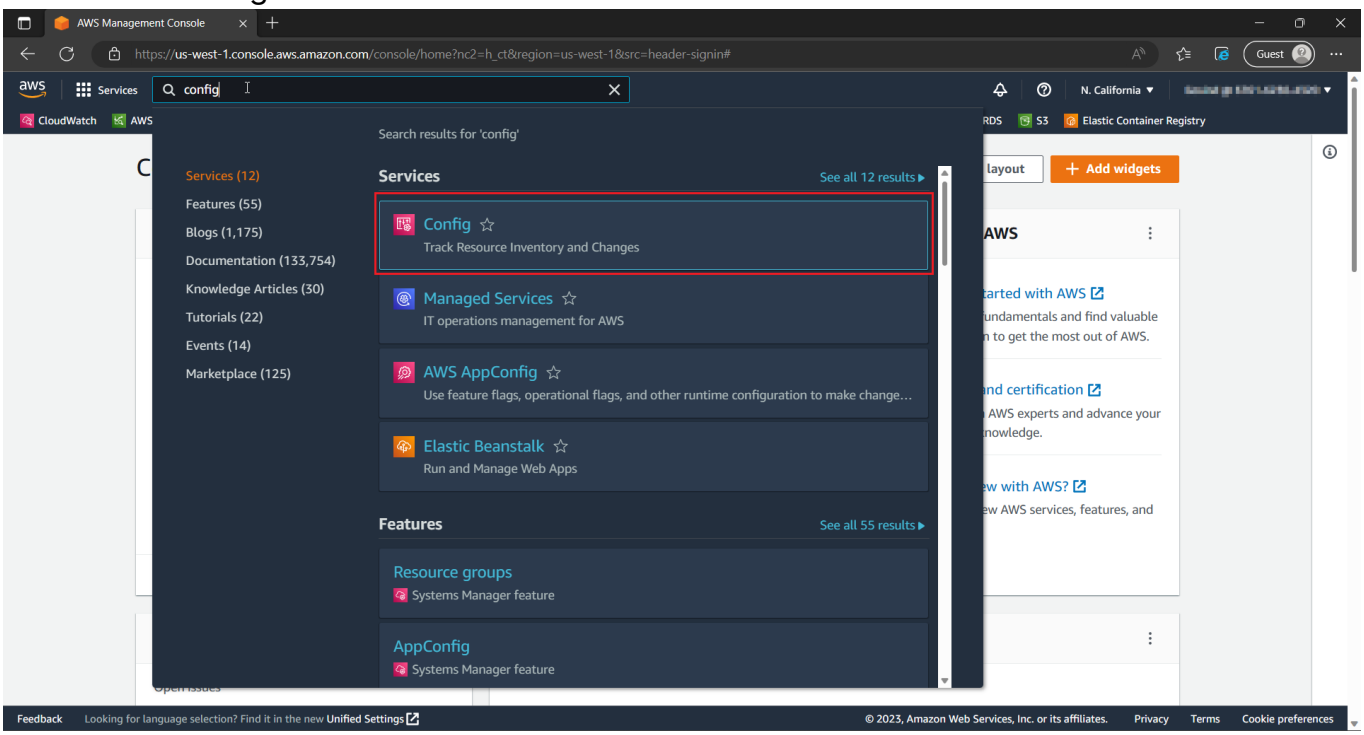
AWS PLAYBOOK

Detect changes in configurations

Search for Config



Click on Get Started and check The following options
"Record all resources supported in this region"
"Include global resources (e.g AWS IAM resources)"
"Use an existing AWS Config service-linked role"
Then Leave the rest as default

On the second page "Rules" page
Search for iam-policy-no-statements-with-admin-access underneath AWS Managed Rules and then select the rule
Click Next
Leave all other options as default and the create that rule
This rule will monitor policies that have access to all services and all resources

Testing if the rule works (optional)

Go to IAM > Policies > Create Policy
Create a policy, select JSON and paste this

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect":"Allow",
            "Action":"*",
            "Resource":"*"
        }
    ]
}
```

Click Next and name the policy as FullAccessPolicy
Go to Config > Rules
So now you should see that newly created policy is non-compliant

Creating Remediation

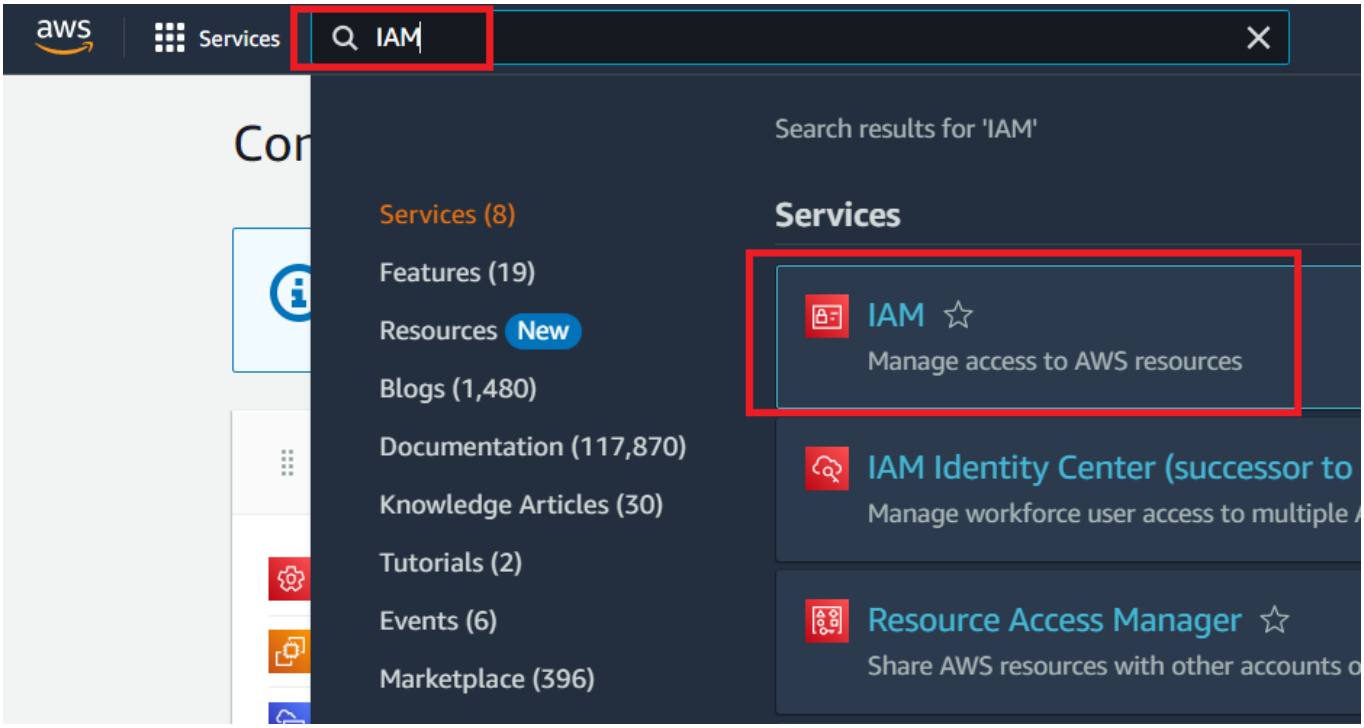
Go to Config > Rules Then select Manage Remediation under actions

On the next page check "Automatic Remediation" and set "Choose remediation action" to AWS-PublishSNSNotification (recomended)

AWS IAM Access Analyzer

This tool will identify resources shared with an external entity and unintended access to resources and data.

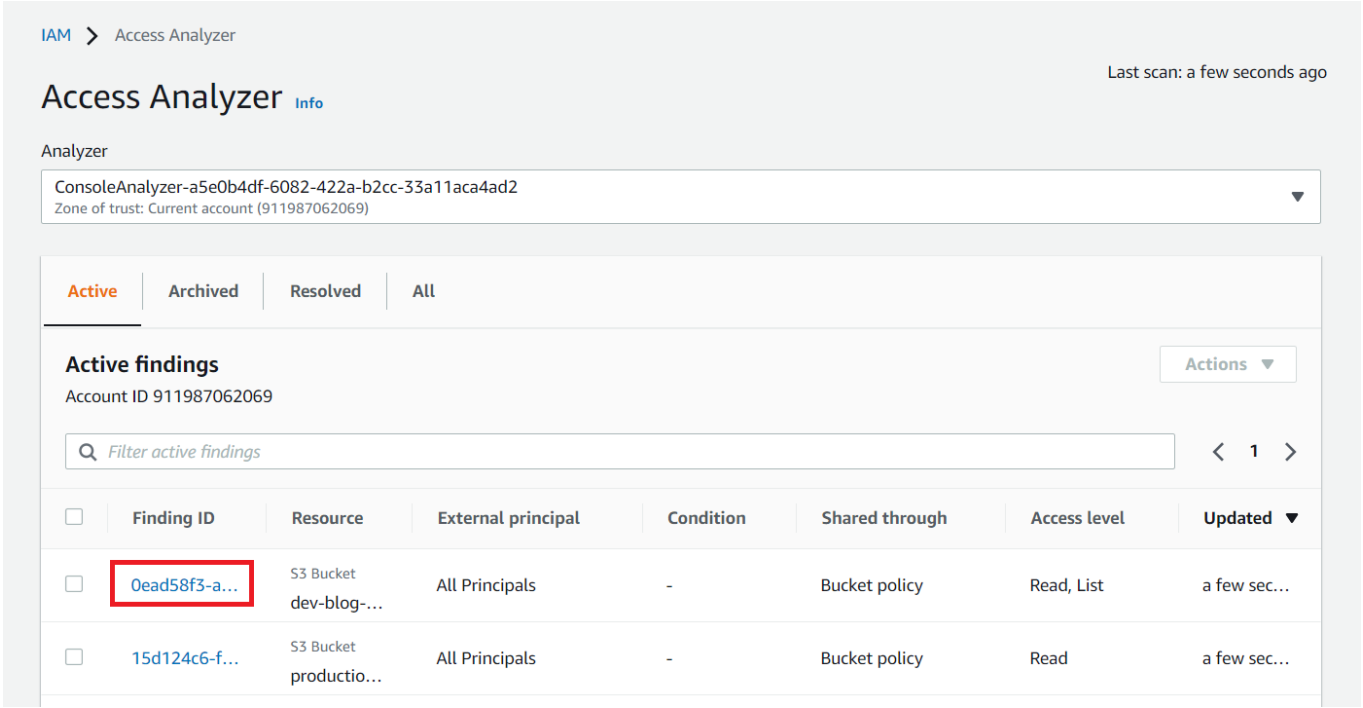
Search for "IAM"



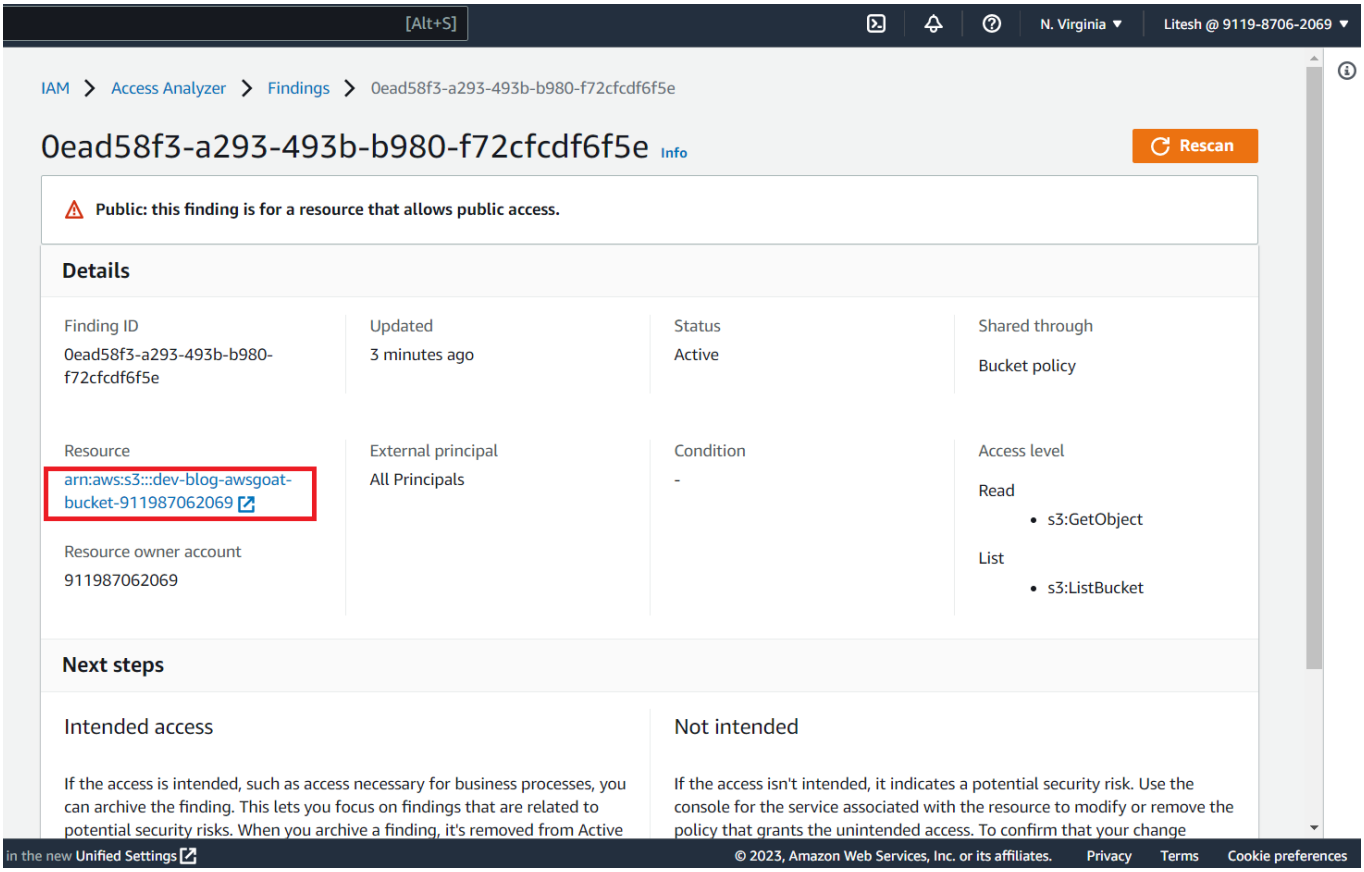
Select "Accesss analyzer" underneath access reports

Select "Create analyzer"
Unless otherwise needed, leave the next page deafult and select "Create analyzer" again located on the bottom left of the page

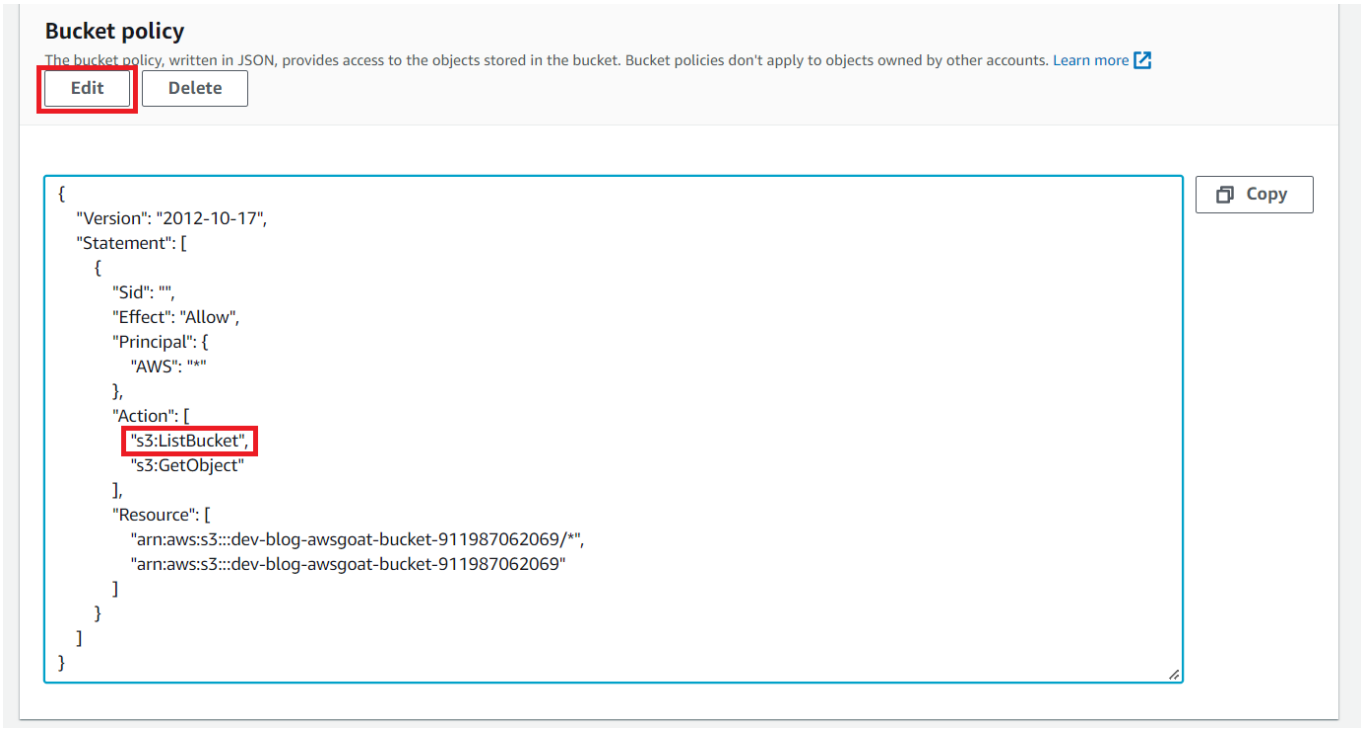
This should send you to the "Access Analyzer" page. Click on "Finding ID" of the resource "S3 Bucket". This will list details about the finding and then the finding shows different findings associtated with this resource.



Now youll be met with the following page. Open this resource in another tab and navigate to the permissions tab of the resource.



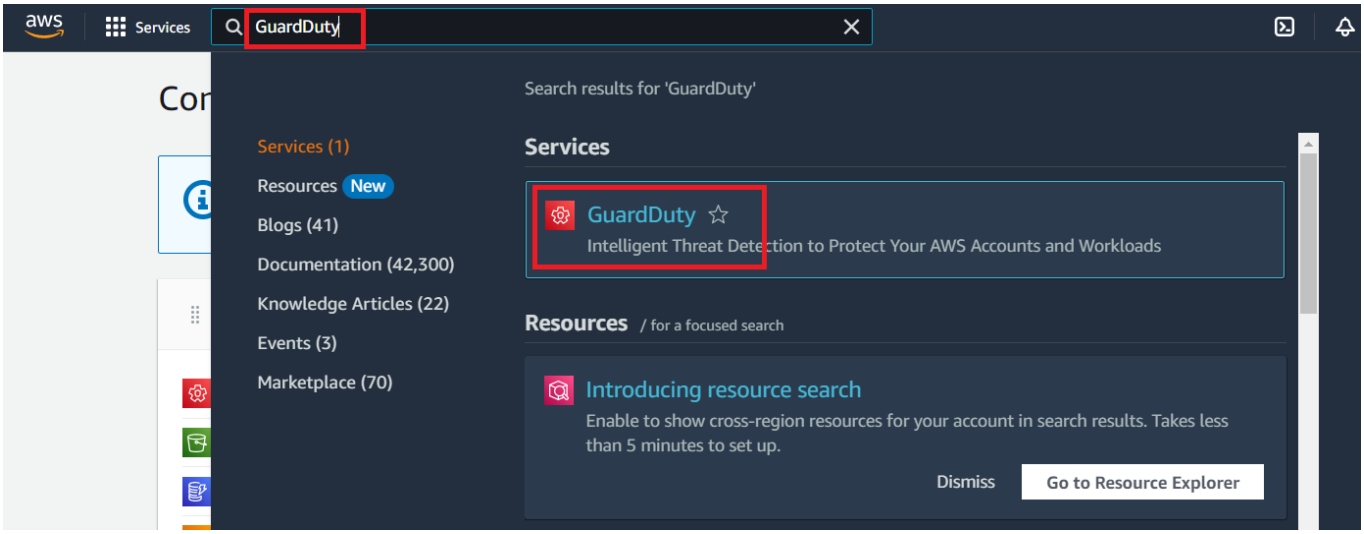
From here, click on "edit" and remove the "s3:ListBucket" action.



Then click on "save changes" on the bottom left.
Do this for all "FINDING IDs" that we can find in the earlier page when we opened the Access Analyzer page.

Amazon Guarduty

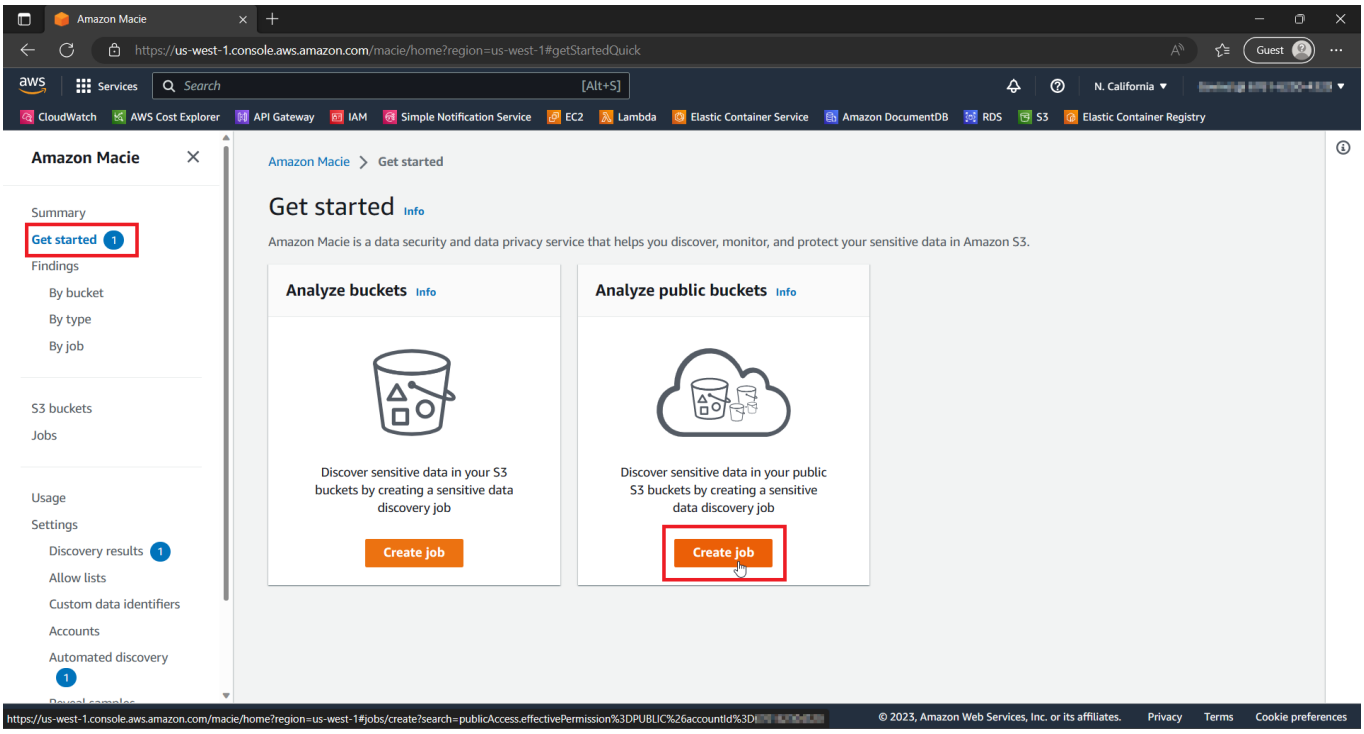
Search for "Guarduty" in the AWS console search bar. Amazon Guarduty is a threat detection service that will help us continuously monitor the AWS Service Accounts stored in Amazon S3. So it will be pretty important to get familiar with this and enable it.



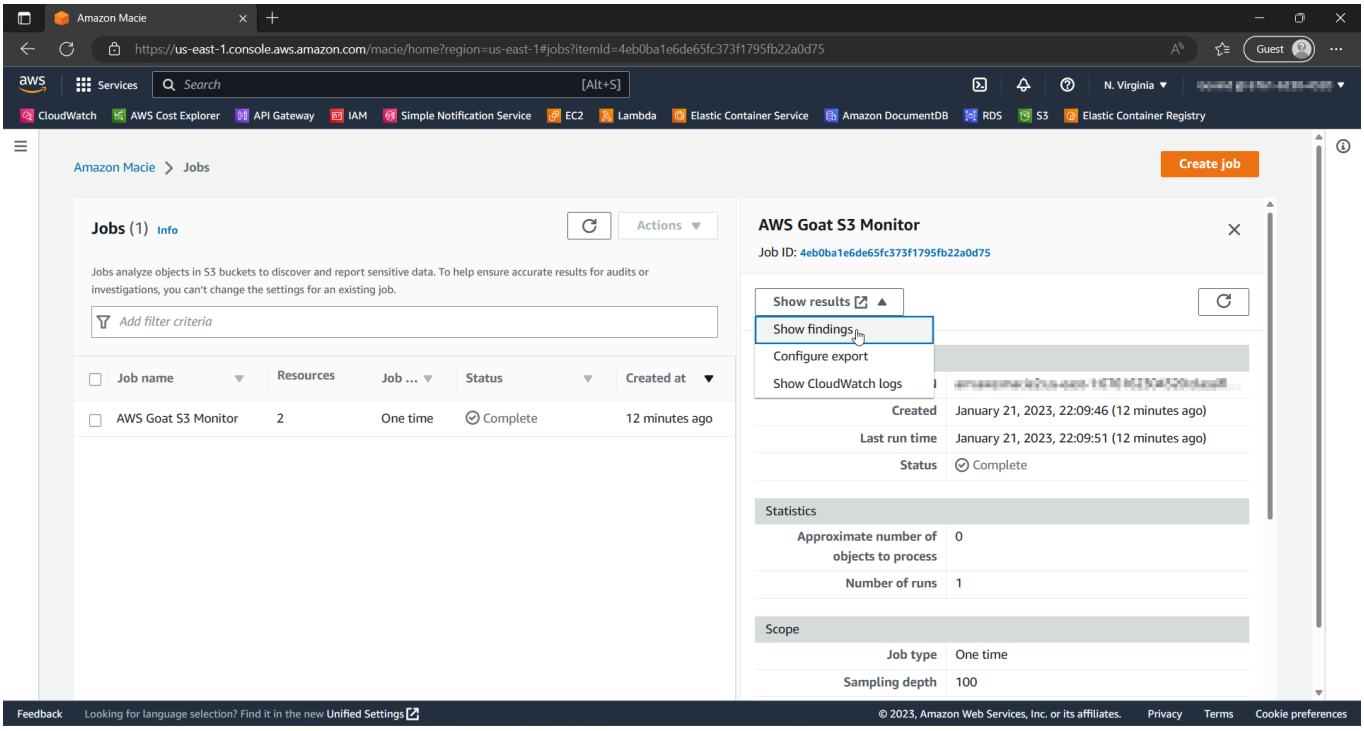
Anyways, From there click on "Get Started" if it isnt already enabled.
And from there click on "Enable GuardDuty". (its a paid service but we can activate a 30 day free trial lol)
It will take some time for GuardDuty to discover any findings. so check back on it from time to time. Once they appear, click on the individual findings and details associated with them. Depending on what the findings tell you, You will have to go to the affected resource and fix the underlying issue.

Amazon Macie

Search "Macie" into the amazon search bar and select "Enable Macie" located in the bottom right.
Once this is selected, you will be brought to the "Get Started" page. This is going to enable us to create jobs under "Analyze public buckets".



From here, you're going to select the buckets that are public, and select Next.
On the "Refine the Scope" page select "One-time job" located underneath the sensitive data discovery options. Then select Next.
In "Select managed data identifiers" choose "include" options and slecelt all dataypes with "CREDENTIALS" as Sensitive data catagory. Select Next. Select next on all the following screens until you are able to select the "Submit" button.
This is going to enable Amazon Macie to scan for vulnerabilities. Once it does. Select "Show Findings"



From here, you are able to see the different vulnerabilities that are present for each

resource.
Again, Depending on what the tool finds you'll have to remediate the affected resource.

Buckets

When enumerating Bucket Policies you'll need to ensure that they aren't misconfigured.

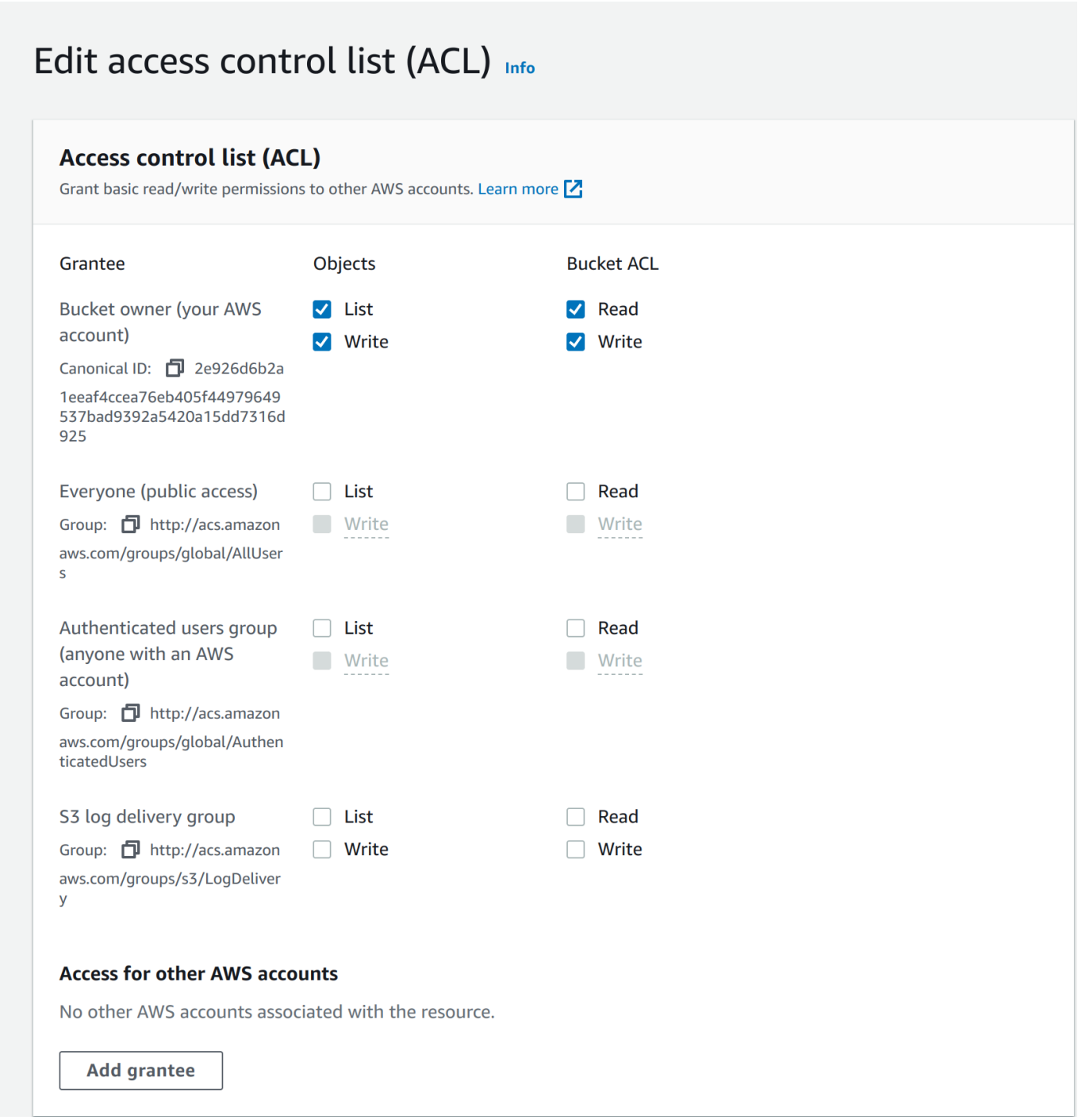
```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "s3:ListBucket",
                "s3:GetObject"
            ],
            "Resource": [
                "arn:aws:s3:::dev-blog-awsgoat-bucket-456011700550/*",
                "arn:aws:s3:::dev-blog-awsgoat-bucket-456011700550"
            ]
        }
    ]
}
```

Above is an example of an S3 Bucket that allows anyone to list tthe bucket.

Underneath "Object Ownership" You are going to want to enable ACLs and Object Ownership to The "Object Writer".

Edit the ACLs so that the correct users and groups are able to access the resource while any unauthenticated ones aren't. You may have to create specific groups with access to

specific resources. This is dependent on CCDC injects.



AWS Lambda

What is AWS Lambda?
AWS Lambda is Amazon's way of running your code on their AWS servers. All you have to do is write the code and deploy it to your AWS Lambda instance, then AWS will handle running it on their servers. This is what AWS considers *serverless* since you don't have to worry about the infrastructure your code is running on.

The uploaded code will only ran if it's triggered. There are various trigger configs that you can add to run your code, such as API gateways which are likely going to be the most

common one used.

SnapStarts

Lambda functions can easily be backed up using the AWS **Snapstart** functionality which takes a snapshot of:

- memory
- device state
- CPU registers
- all initialization steps that were taken before the lambda function starts

To enable **Snapstart**:

1. Open `Functions` page of lambda console
2. Choose name of function to config snapshots for
3. Config > General Config
4. General config > Edit
5. Edit basic settings > Snapstart > Published Versions
6. Save
7. Publish function version

- Lambda initalizes your code, creates snapshot of the initialized execution env, & caches the snapshot for low-latency access

8. Invoke function

Lambda SnapStart Function States

There are different function states to SnapStarts, they include:

- `Pending` - Lambda initializing code & will take snapshot of the environment after initialization. Any invocations or API actions that operate on function will fail
- `Active` - Snapshot is created & you can invoke function. To use Snapstart, must invoke published function version (`$LATEST`), not unpublished version
  - Must publish it before using snapstart
- `Inactive` - Function version hasn't been invoked for 14 days. Lambda will delete snapshot of any function versions that are inactive.
  - Invoke the function version after 14 days, `SnapStartNotReadyException` err will be returned & a new snapshot initialized
  - Wait until function version reaches `Active` state & then invoke it again

- `Failed` - err encountered when running initialization code or creating snapshot

Uploading Lambda Code to AWS

To create lambda function w/console:

1. Functions page of lambda console
2. Create function
3. If you want to use a **blueprint**, click use a **blueprint**

- *blueprint* - provides sample code to do some minimal processing. Most will process events from specific event sources such as Amazon S3, Amazon DynamoDB or a custom app
- **Select Blueprint** > select one from here

4. Enter the function name
5. For **Execution Role**, choose **Create new role w/basic Lambda permissions**

- *Lambda* creates **execution** role that gives function permissions to upload logs to Amazon CloudWatch
- Lambda function assumes execution role when it's invoked & uses execution role to create creds for AWS SDK & to read data from data sources

To invoke the function from console:

1. Test tab
2. Test event action > create new event
3. Enter event name, choose Private for Event sharing setting
4. For Template, leave it to default

- Make sure you change the handler in AWS:
  - Code > runtime settings > edit > handler > set the handler to the name of the executable
- When compiling, use the following params:
  - `GOOS=linux GOARCH=amd64 CGO_ENABLED=0`
  - If you're coding in golang, zip the executable file, then upload to AWS console

1. Open functions page > choose function > Under function overview, choose Add trigger > API gateway > Create an API/Use existing API > Security choose Open > Add

Integrating API Access/Gateway To Invoke/Trigger Lambda Function

- Create **RESTful** API & a method to interact w/the API:
  - Lambda > functions > select your lambda function
  - add trigger > API gateway > create new API > REST API > Security > OPEN
  - Configuration > triggers > click the API link to get to the API gateway console
  - Actions > create resource > give it a name & path
  - Highlight the created resource > actions > create method
    - This will create a method that can be used to interact w/the created resource
  - Go to the created method to config it > set integration type to **Lambda Function** > specify the Lambda function name
    - When specifying the lambda function name, it should come up as suggested. This is how you can confirm whether or not the function name you entered exists
- Define **mapping template**:
  - Select the method (EX: GET) > Integration request > mapping template > select **When there are no templates defined (recommended)** > add mapping template > input **application/json**
- Deploy the API:
  - Actions > Deploy API
    - Create new deployment stage if one doesn't already exist
- Stages > expand until you see the method that you created for the associated function & URI path > click the method > you now have the invoke URL & can make a cURL request to the invoke URL to test to see if it's working
  curl -X POST https://f0s9dwqcc9.execute-api.us-east-1.amazonaws.com/default/GoblinoCode/gobin -d '{"key": "test"}'