Course: Cloud and Network Security
Name: Neville Ngothe Iregi
Student No.: CS-CNS10-25054
Date: Wednesday, 10th December 2025

Week 12 Assignment 2: Flaws AWS

Introduction

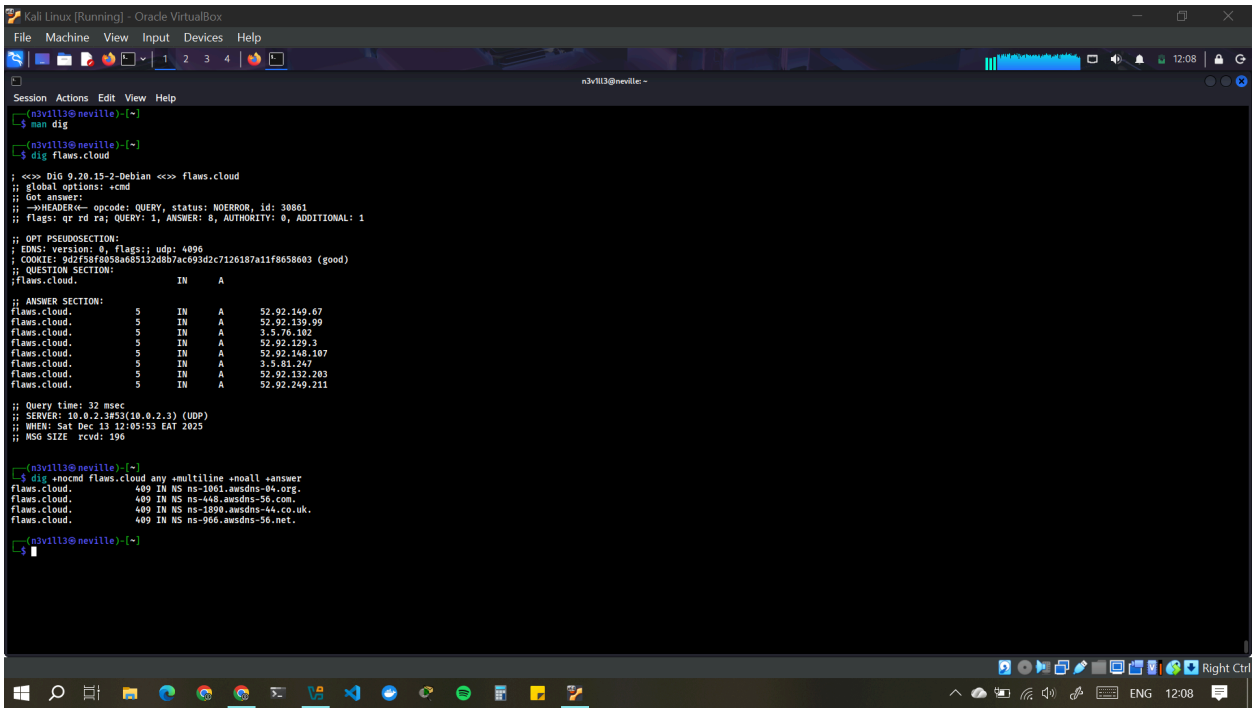
Through a series of levels, I learned about common mistakes and gotchas when using Amazon Web Services (AWS). There are no SQL injection, XSS, buffer overflows, or many of the other vulnerabilities you might have seen before. As much as possible, these are AWS specific issues.

Scope: Everything is run out of a single AWS account, and all challenges are sub-domains of flaws.cloud.

Level 1

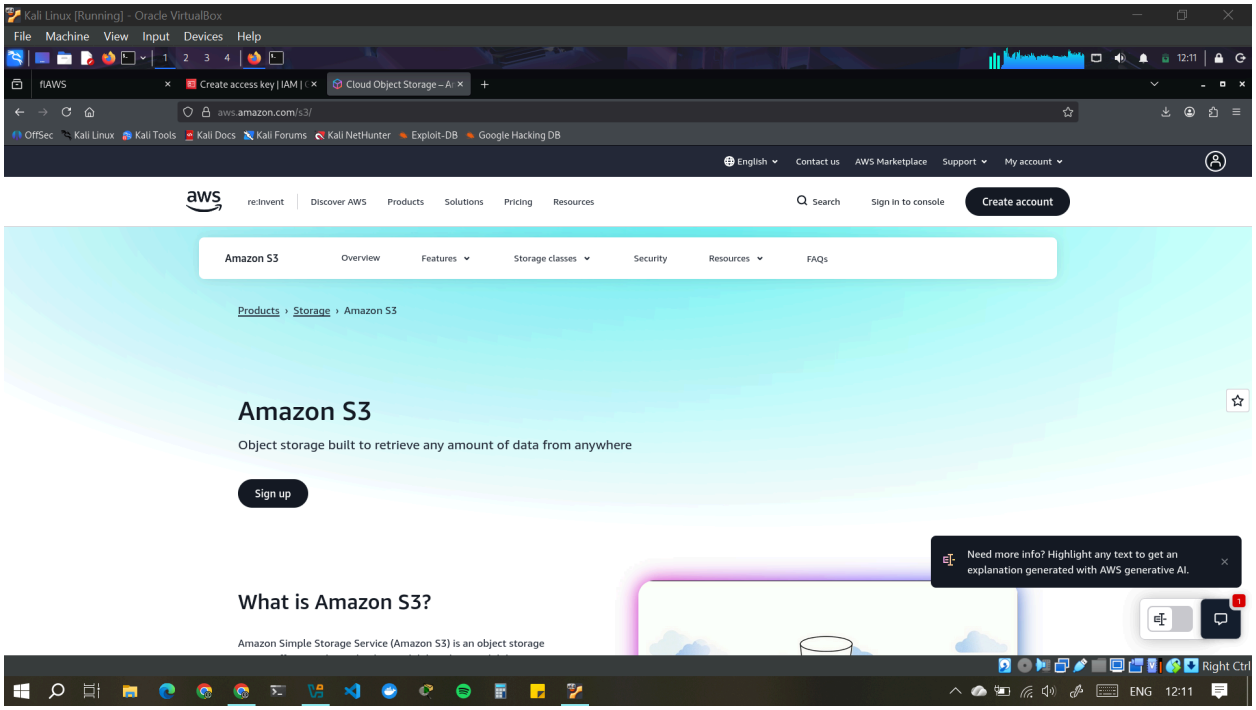
The site flaws.cloud is hosted as an S3 bucket. This is a great way to host a static site, similar to hosting one via github pages. Some interesting facts about S3 hosting: When hosting a site as an S3 bucket, the bucket name (flaws.cloud) must match the domain name (flaws.cloud). Also, S3 buckets are a global name space, meaning two people cannot have buckets with the same name. The result of this is you could create a bucket named apple.com and Apple would never be able to host their main site via S3 hosting.

1. I tried to determine the site is hosted as an S3 bucket by running a DNS lookup on the domain flaws.cloud (command: `dig flaws.cloud`)

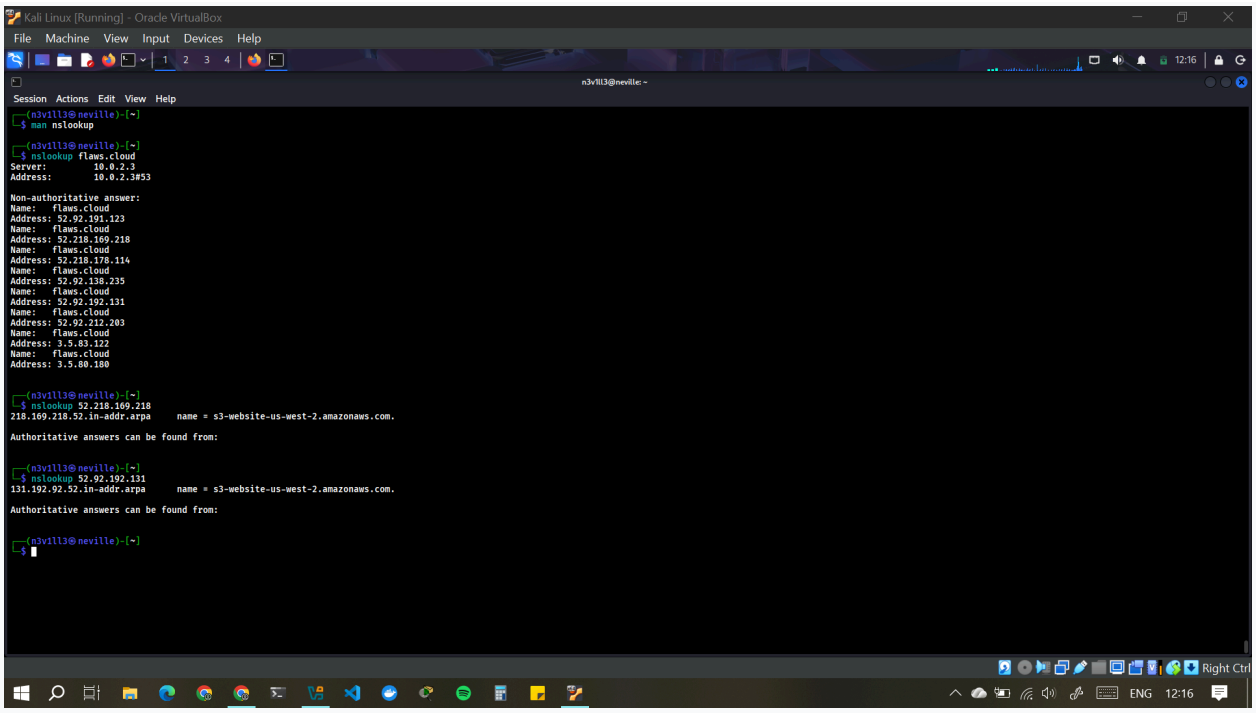


2. Visiting one of the IP addresses in your browser will direct you to https://aws.amazon.com/s3/

So you know flaws.cloud is hosted as an S3 bucket.

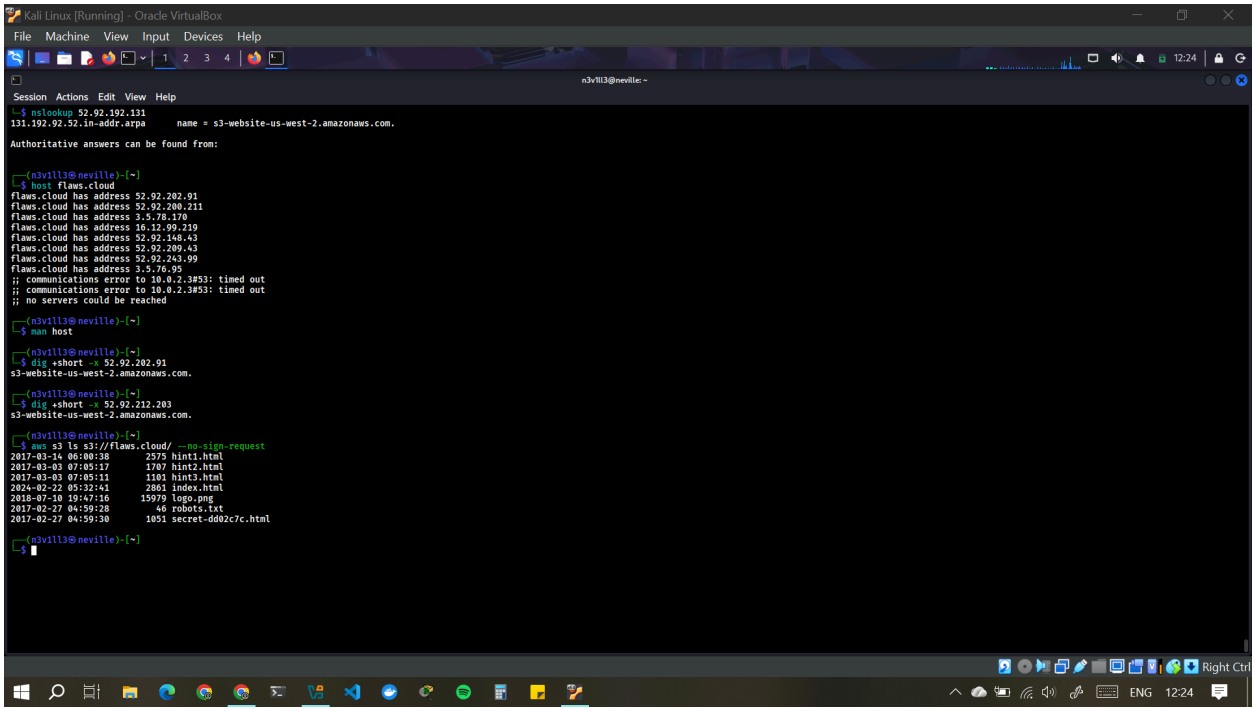


3. I then ran `nslookup flaws.cloud`



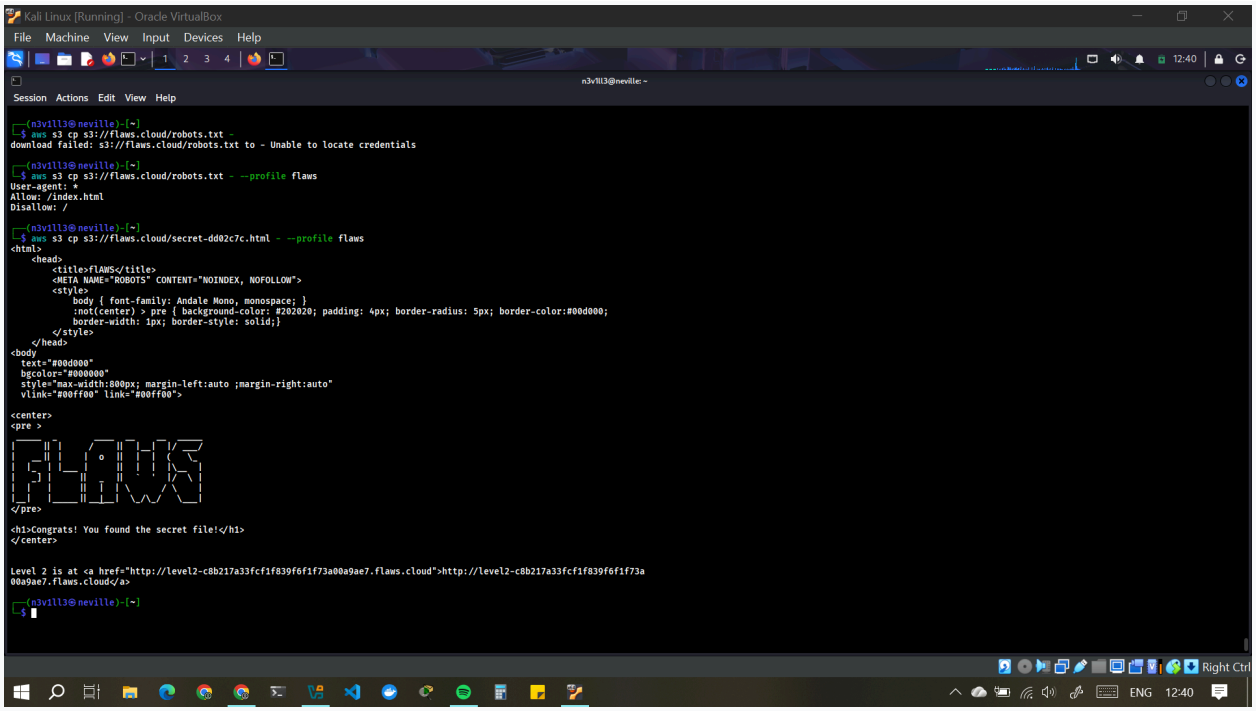
- So we know it's hosted in the AWS region us-west-2

4. I enumerated the bucket with : **aws s3 ls s3://flaws.cloud/ --no-sign-request** to view the files in the bucket



- Viewing the files was possible due to the permissions issues on this bucket

5. I viewed the secret-dd02c7c.html file using command: ***aws s3 cp s3://flaws.cloud/secret-dd02c7c.html - --profile flaws***



- At this point, the file indicated that level 2 was at the sub-domain **http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud**

Lesson learned

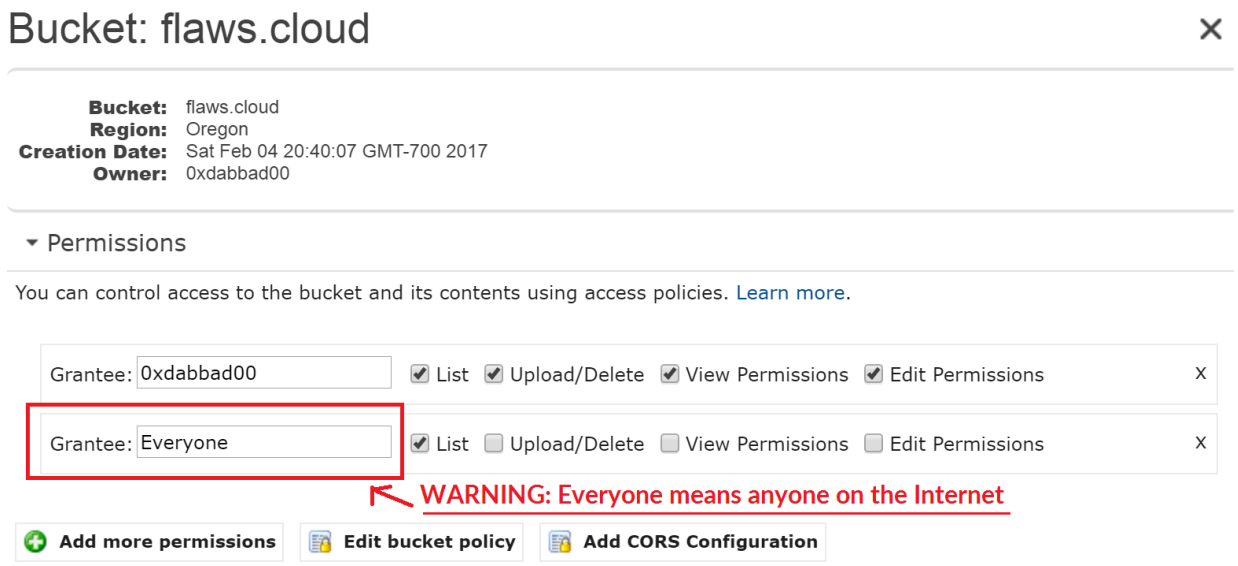
On AWS you can set up S3 buckets with all sorts of permissions and functionality including using them to host static files. A number of people accidentally open them up with permissions that are too loose. Just like how you shouldn't allow directory listings of web servers, you shouldn't allow bucket listings.

Examples of this problem

- Directory listing of S3 bucket of Legal Robot (link) and Shopify (link).
- Read and write permissions to S3 bucket for Shopify again (link) and Udemy (link). This challenge did not have read and write permissions, as that would destroy the challenge for other players, but it is a common problem.

Avoiding the mistake

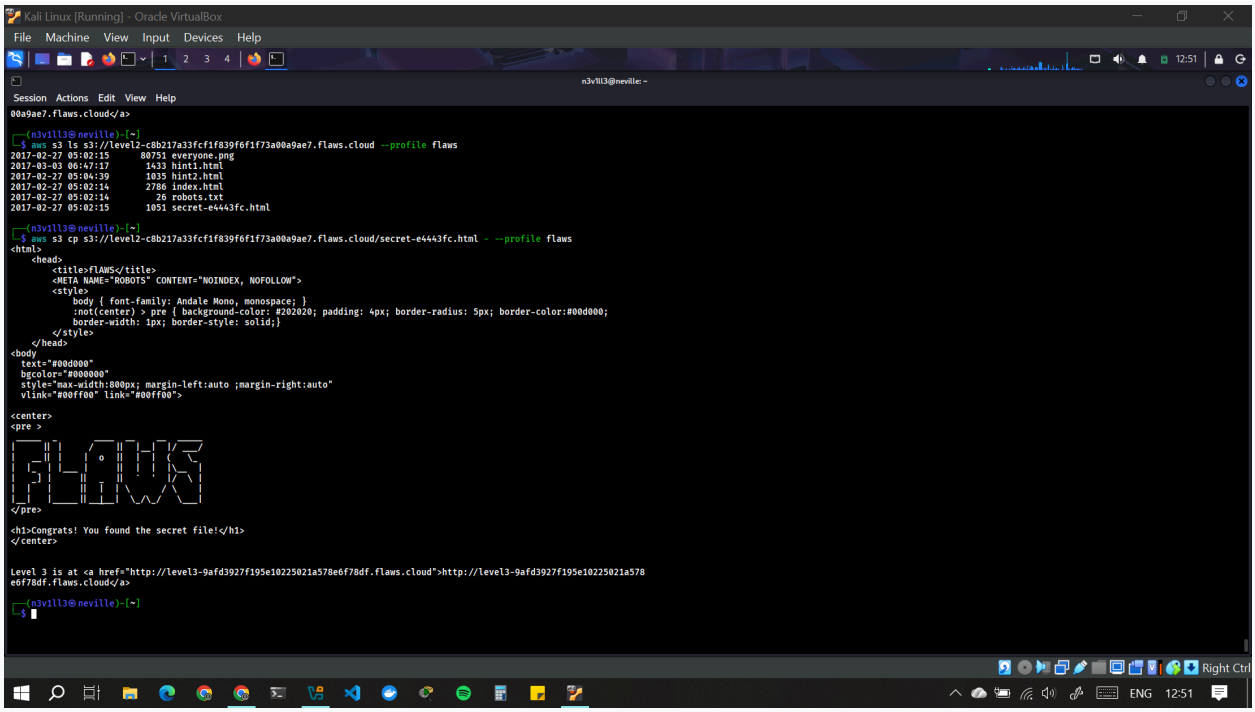
By default, S3 buckets are private and secure when they are created. To allow it to be accessed as a web page, I had turn on "Static Website Hosting" and changed the bucket policy to allow everyone "s3:GetObject" privileges, which is fine if you plan to publicly host the bucket as a web page. But then to introduce the flaw, I changed the permissions to add "Everyone" to have "List" permissions.



Everyone" means everyone on the Internet. You can also list the files simply by going to http://flaws.cloud.s3.amazonaws.com/ due to that List permission.

Level 2 (unauthorised authenticated access)

With my AWS profile already set up, I enumerated the next sub-domain from the 1st level using command: `aws s3 ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud --profile <yourprofile>`



- I found several files and tried to view the secret-e4443fc.html which gave me the sub-domain of the next level (http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud)

Lesson learned

Similar to opening permissions to "Everyone", people accidentally open permissions to "Any Authenticated AWS User". They might mistakenly think this will only be users of their account, when in fact it means anyone that has an AWS account.

Examples of this problem

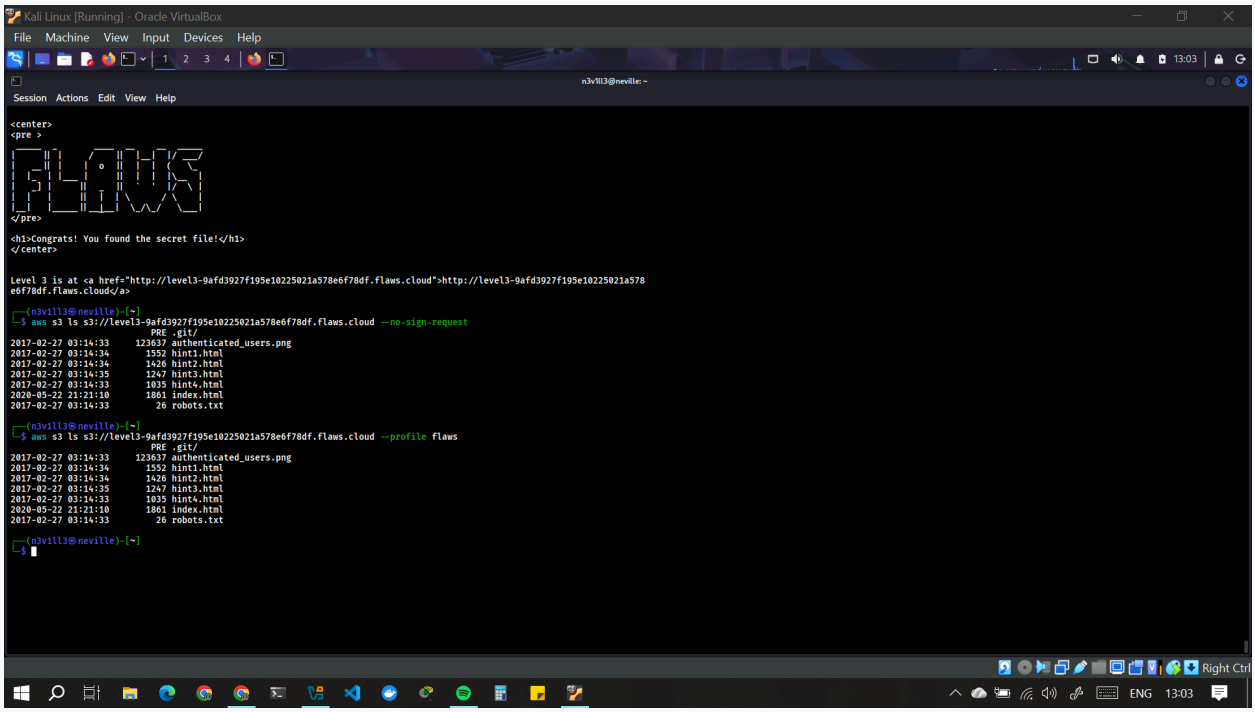
- Open permissions for authenticated AWS user on Shopify (link)

Avoiding the mistake

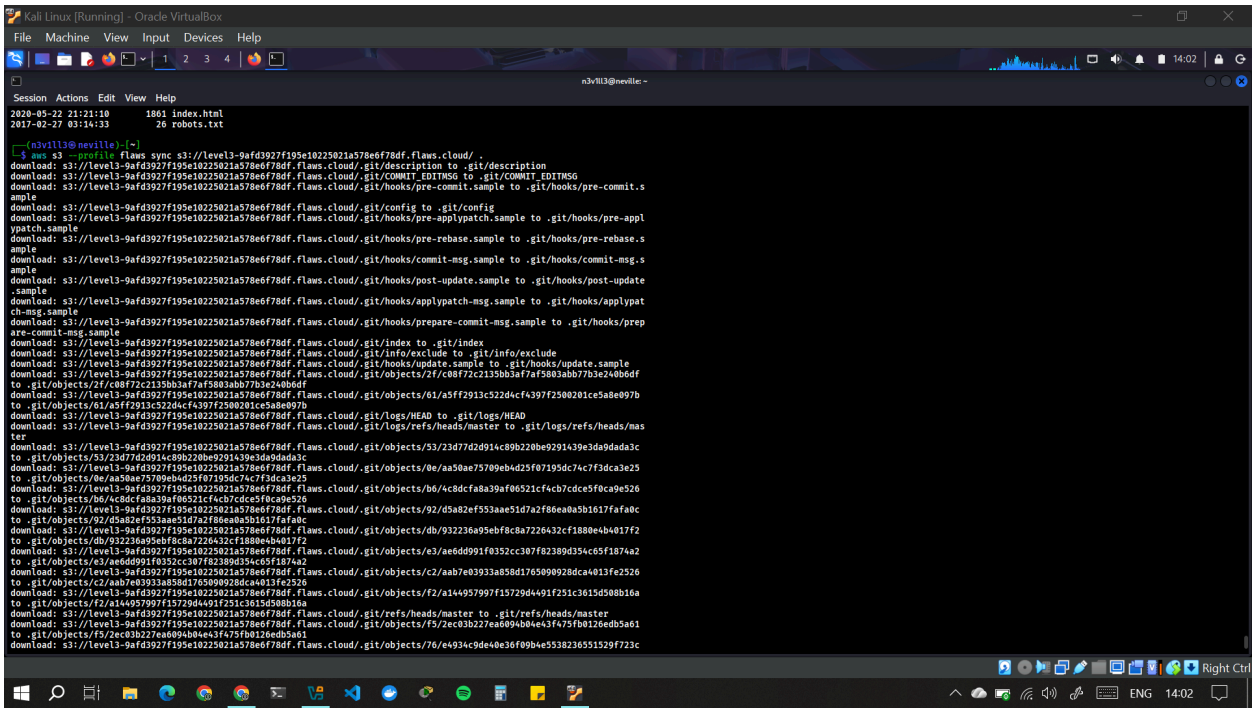
Only open permissions to specific AWS users.#

Level 3 (leaked credentials)

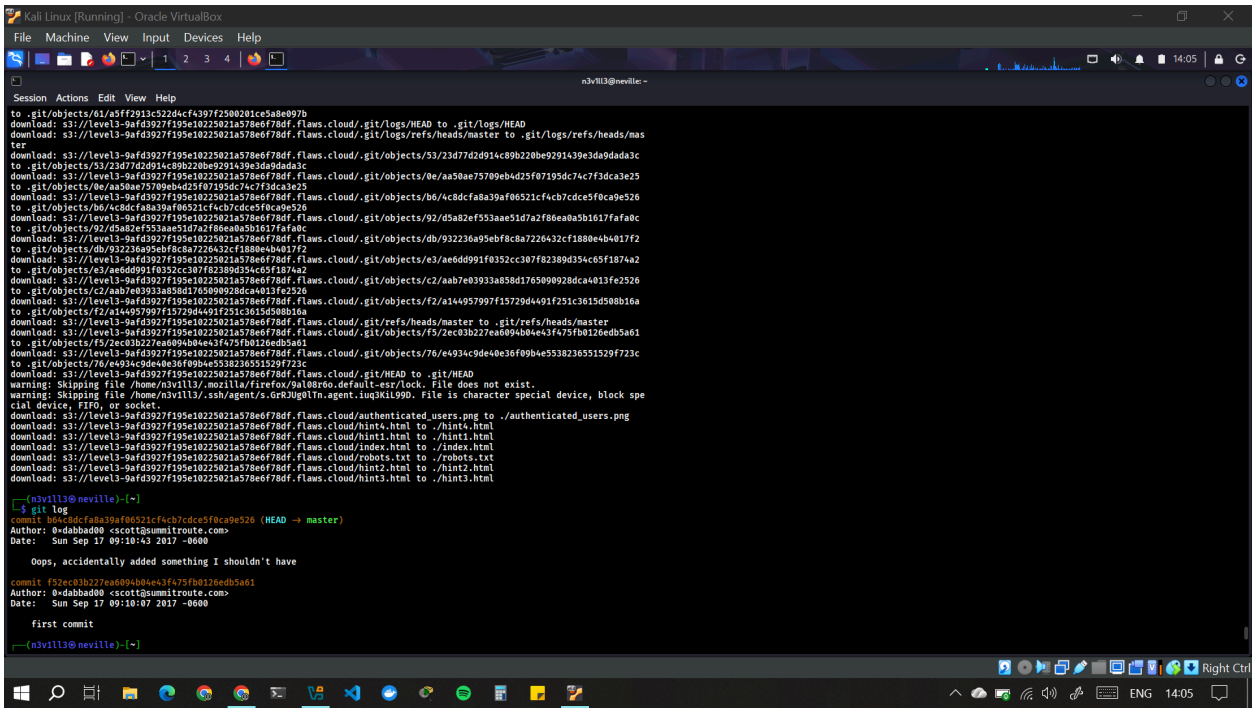
1. I enumerated the bucket using command: `aws s3  ls s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/ --profile <yourprofile>`
    - However, I noticed that I could do enumeration without authentication, using the **- -no-sign-request**



2. This S3 bucket has a .git file. There are probably interesting things in it. Download this whole S3 bucket using: ***aws s3 --profile <yourprofile> sync s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/ .***
    - The aws s3 sync command is a powerful utility within the AWS Command Line Interface (CLI) used to synchronize files between a local directory and an S3 bucket, or between two S3 buckets.



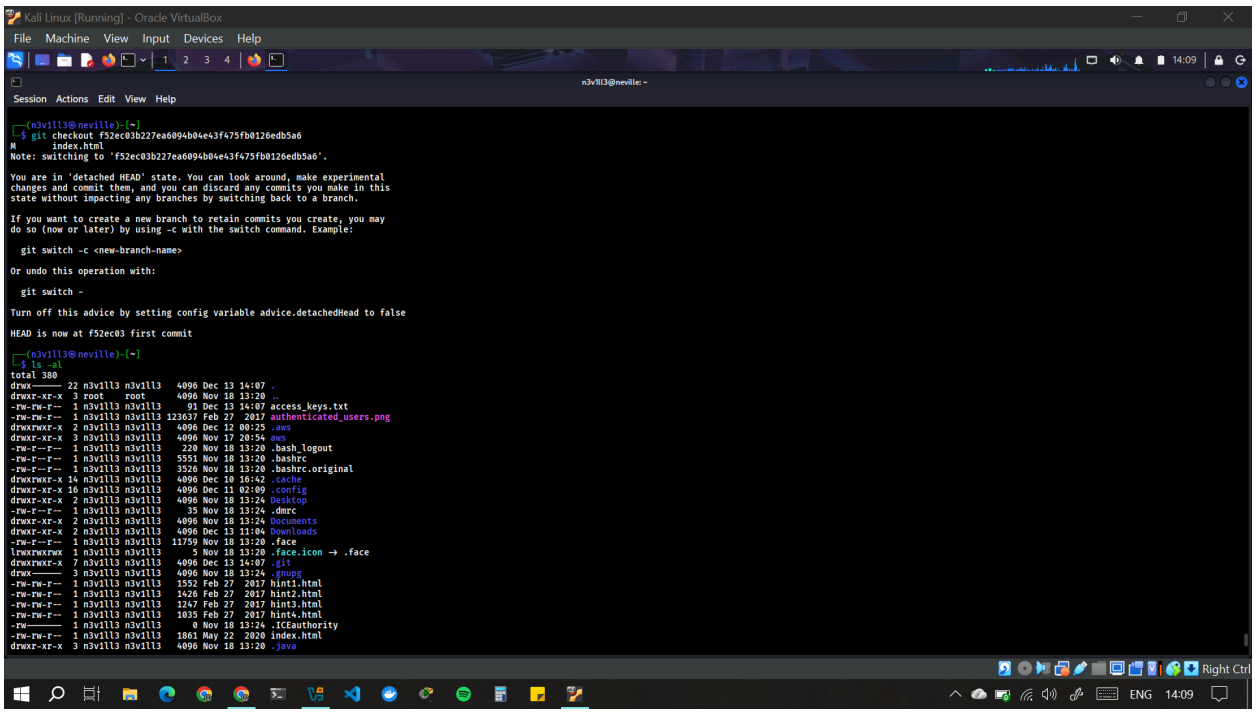
3. People often accidentally add secret things to git repos, and then try to remove them without revoking or rolling the secrets. You can look through the history of a git repo by running: ***git log***



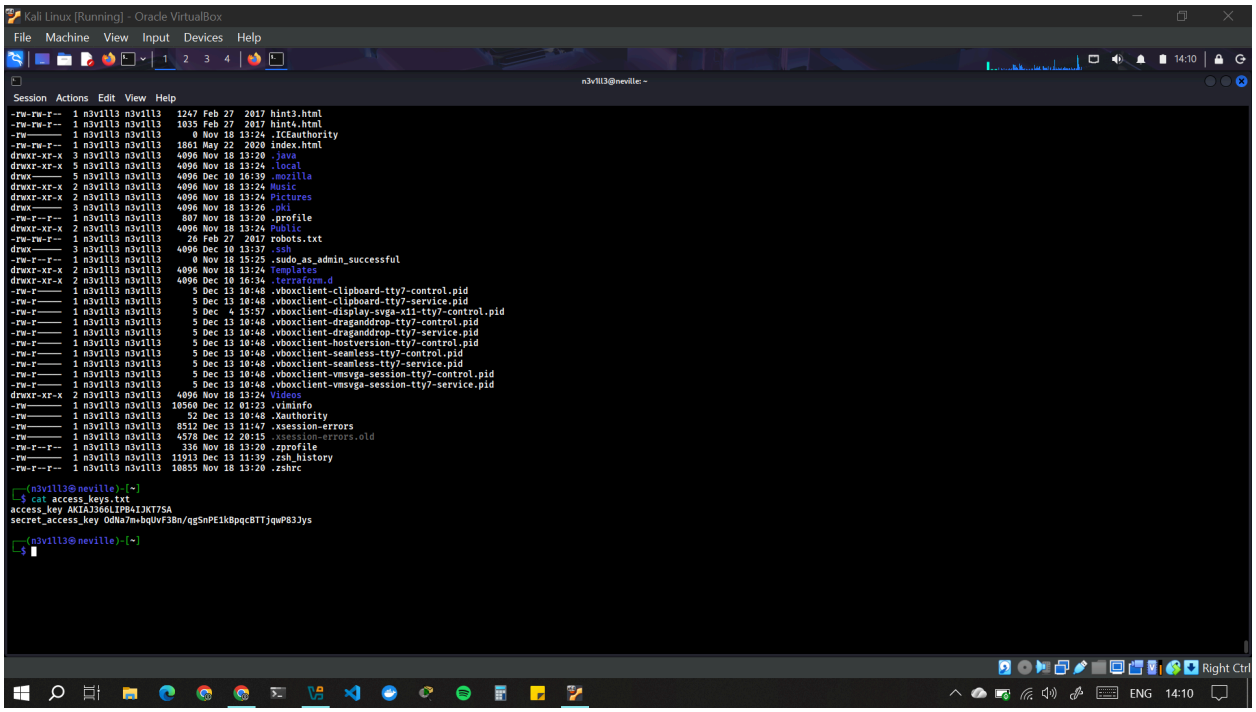
4. I then checked what the git repo looked like at the time of a commit by running:

   git checkout f52ec03b227ea6094b04e43f475fb0126edb5a6

   - where `f52ec03b227ea6094b04e43f475fb0126edb5a6` would be the hash for the first commit shown in `git log`.



5. On checking the repo again, a new access_keys.txt appears. I viewed it:

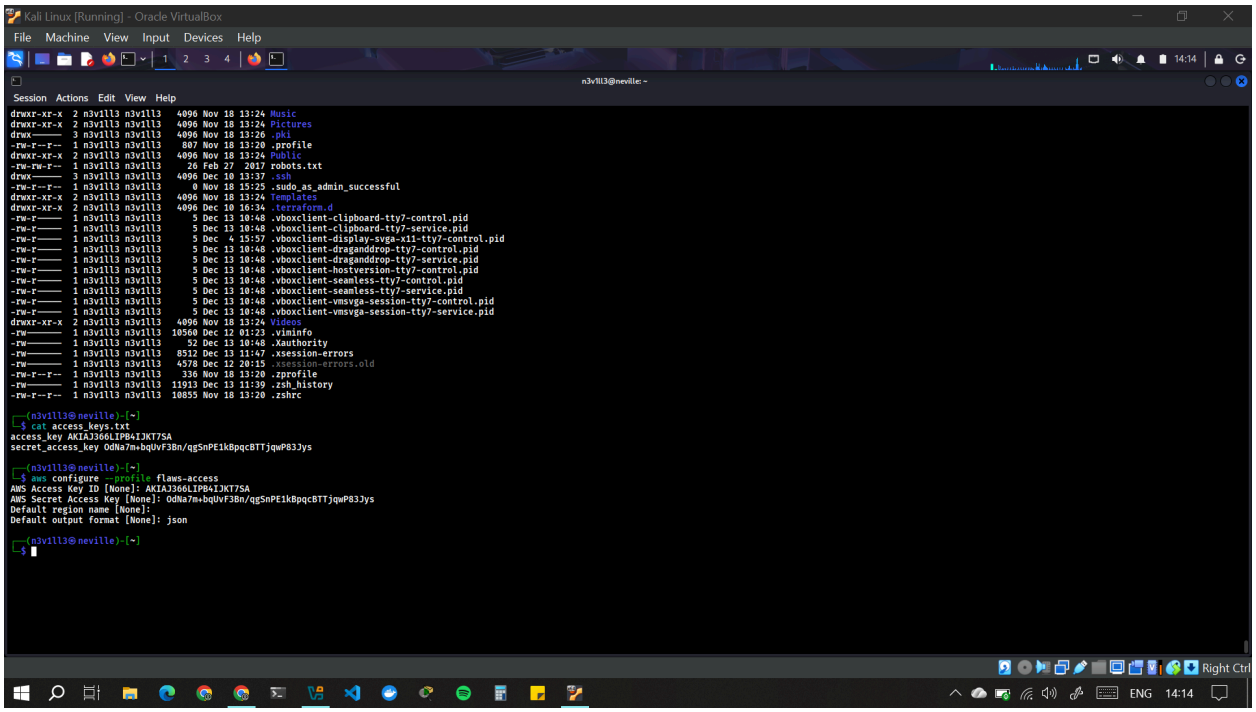


- It contained the following credentials:

  access_key: AKIAJ366LIPB4IJKT7SA

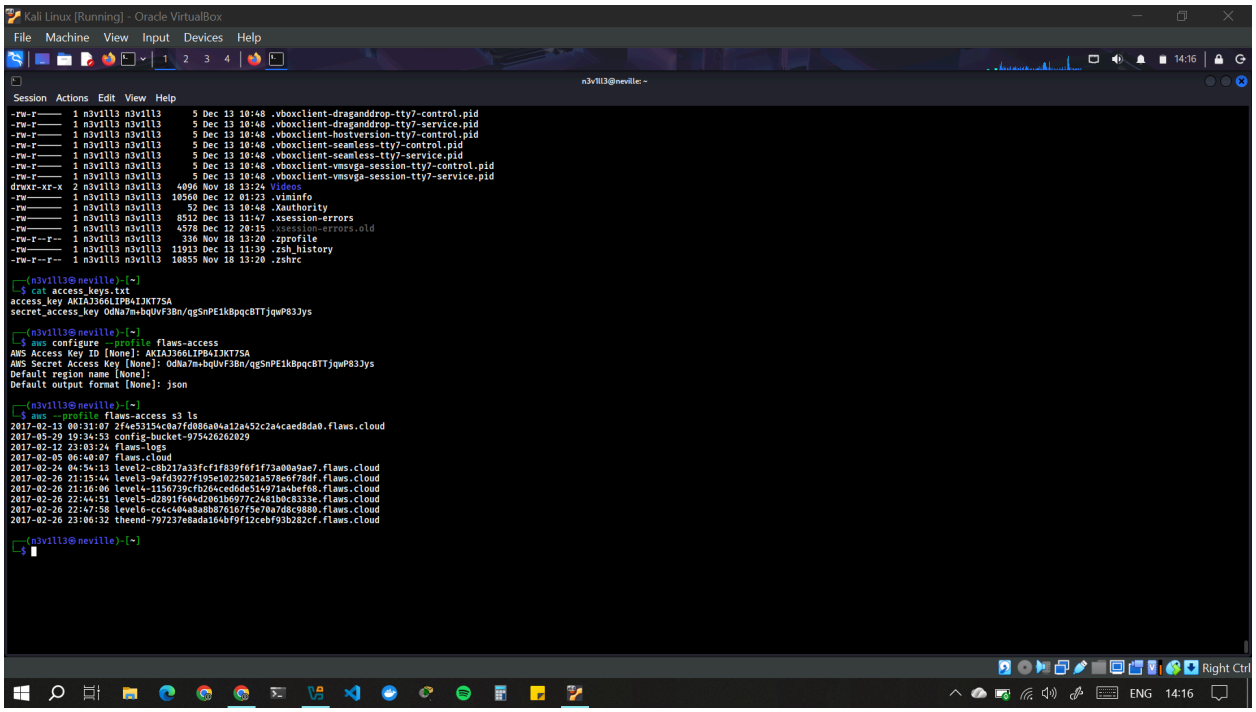
  secret_access_key: OdNa7m+bqUvF3Bn/qgSnPE1kBpqcBTTjqwP83Jys

6. I created a new profile using *aws configure* and the keys I just accessed:



7. Then to list S3 buckets using that profile, I run:

   aws --profile flaws-access s3 ls



Lesson learned

People often leak AWS keys and then try to cover up their mistakes without revoking the keys. You should always revoke any AWS keys (or any secrets) that could have been leaked or were misplaced. Roll your secrets early and often.

Examples of this problem

- Instagram's Million Dollar Bug: In this must read post, a bug bounty researcher uncovered a series of flaws, including finding an S3 bucket that had .tar.gz archives of various revisions of files. One of these archives contained AWS creds that then allowed the researcher to access all S3 buckets of Instagram. For more discussion of how some of the problems discovered could have been avoided, see the post "Instagram's Million Dollar Bug": Case study for defense

Another interesting issue this level has exhibited, although not that worrisome, is that you can't restrict the ability to list only certain buckets in AWS, so if you want to give an employee the ability to list some buckets in an account, they will be able to list them all. The key you used to discover this bucket can see all the buckets in the account. You can't see

what is in the buckets, but you'll know they exist. Similarly, be aware that buckets use a global namespace meaning that bucket names must be unique across all customers, so if you create a bucket named `merger_with_company_Y` or something that is supposed to be secret, it's technically possible for someone to discover that bucket exists.

Avoiding this mistake

Always roll your secrets if you suspect they were compromised or made public or stored or shared incorrectly. Roll early, roll often. Rolling secrets means that you revoke the keys (ie. delete them from the AWS account) and generate new ones.

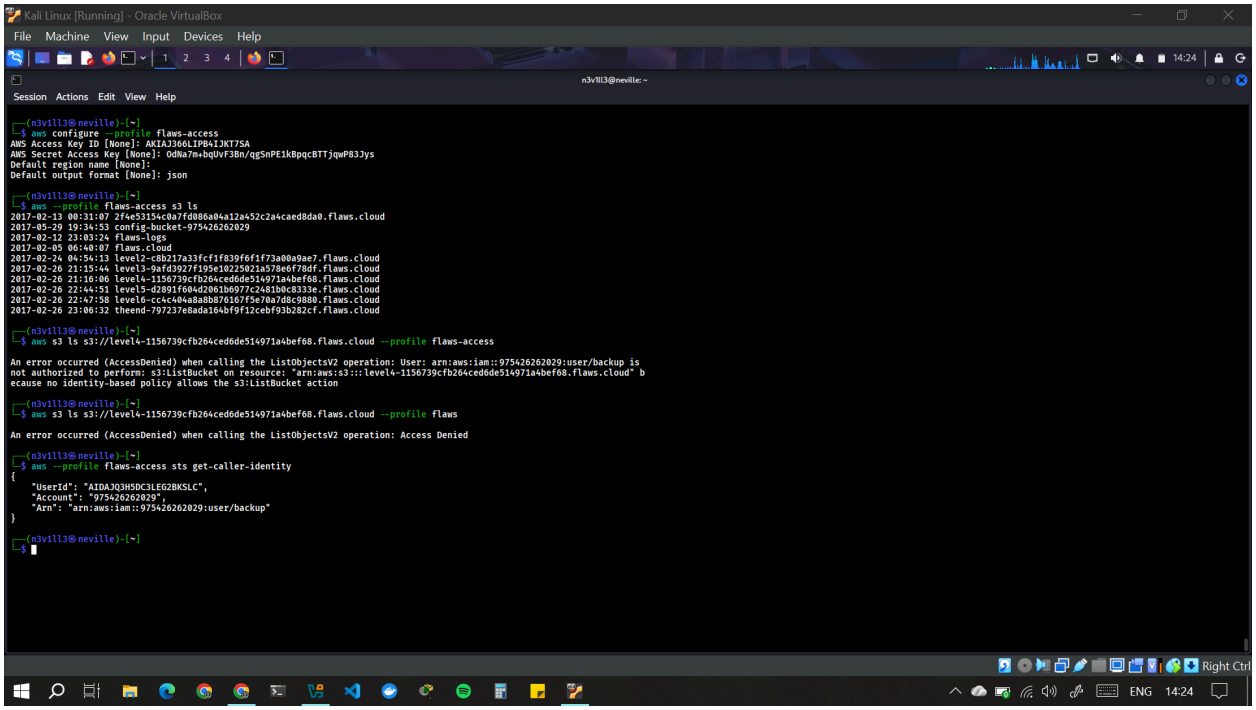
Level 4

For the next level, you need to get access to the web page running on an EC2 at 4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud

It'll be useful to know that a snapshot was made of that EC2 shortly after nginx was setup on it.

1. You can snapshot the disk volume of an EC2 as a backup. In this case, the snapshot was made public, but you'll need to find it.

   To do this, first we need the account ID, which we can get using the AWS key from the previous level: ***aws --profile flaws-access sts get-caller-identity***

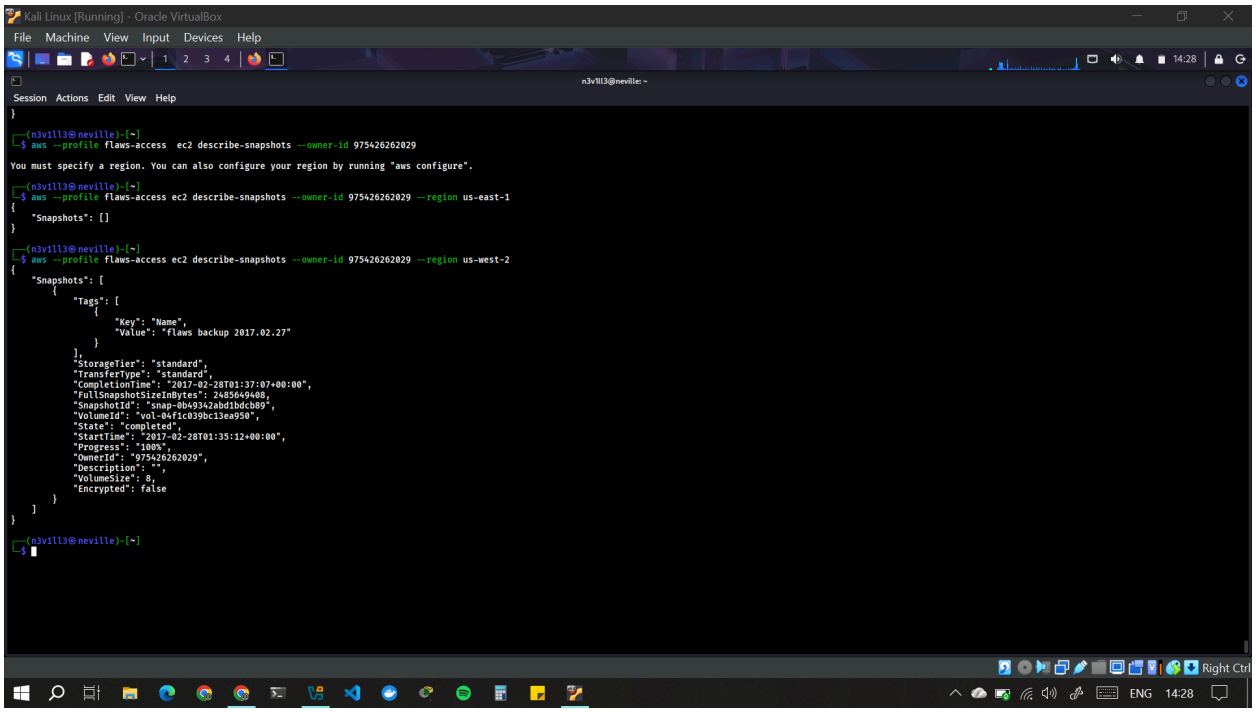


- Using that command also tells you the name of the account, which in this case is named "backup". The backups this account makes are snapshots of EC2s.

2. discover the snapshot:

   Command: ***aws --profile flaws-access ec2 describe-snapshots --owner-id 975426262029 --region us-west-2***

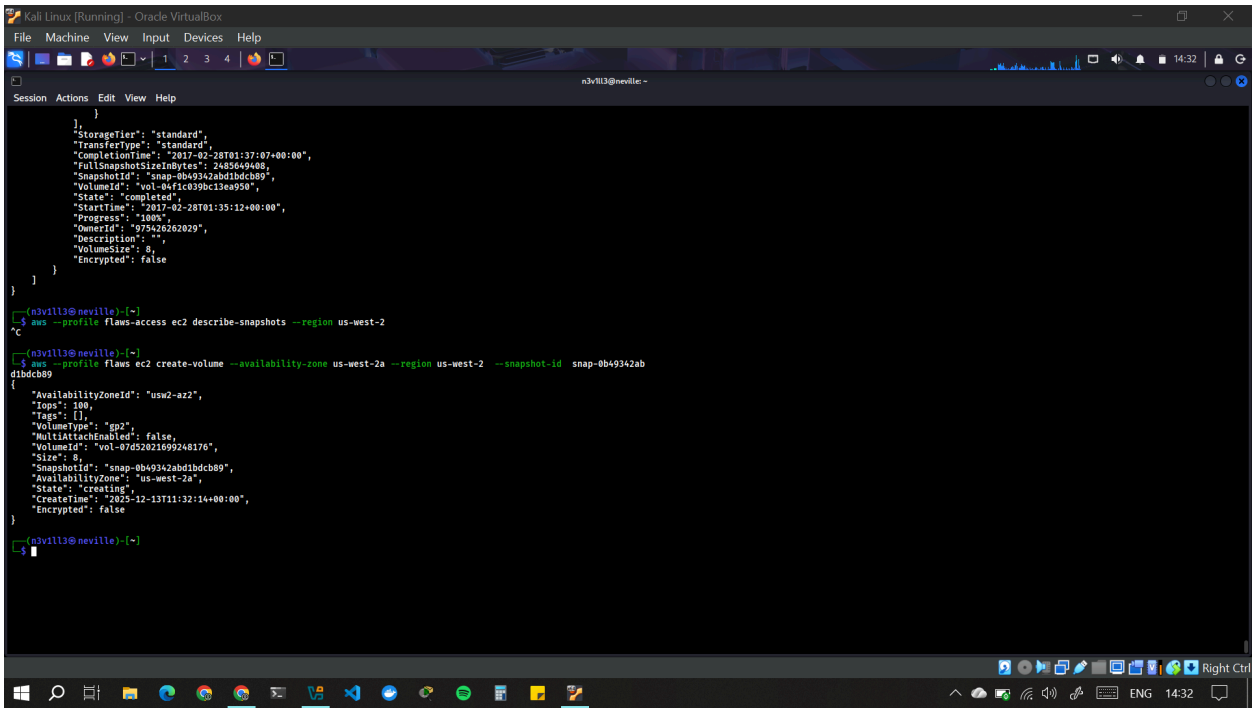
- We specify the owner-id just to filter the output. For fun, run that command without the owner-id and notice all the snapshots that are publicy readable. By default snapshots are private, and you can transfer them between accounts securely by specifiying the account ID of the other account, but a number of people just make them public and forget about them it seems.



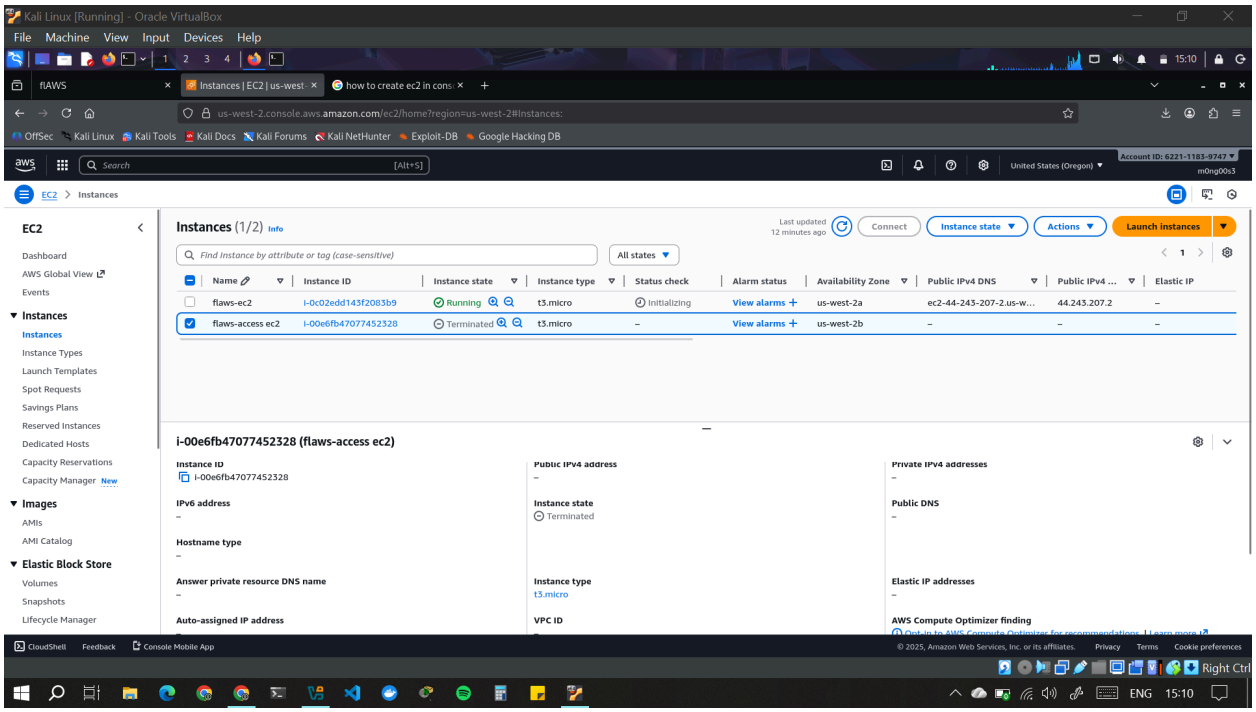
3. With knowledge of the snapshot ID, i needed to mount it using my own AWS account.

   First, create a volume using the snapshot:

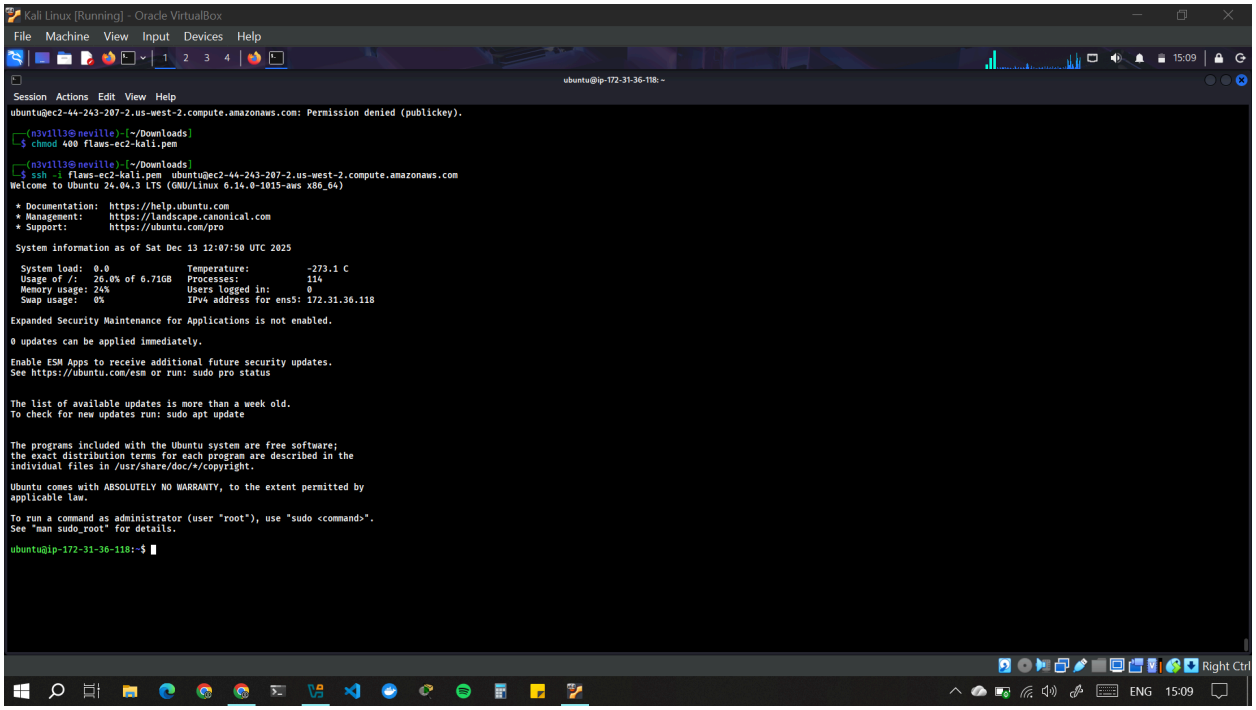
   Command: ***aws --profile YOUR_ACCOUNT ec2 create-volume --availability-zone us-west-2a --region us-west-2 --snapshot-id snap-0b49342abd1bdcb89***



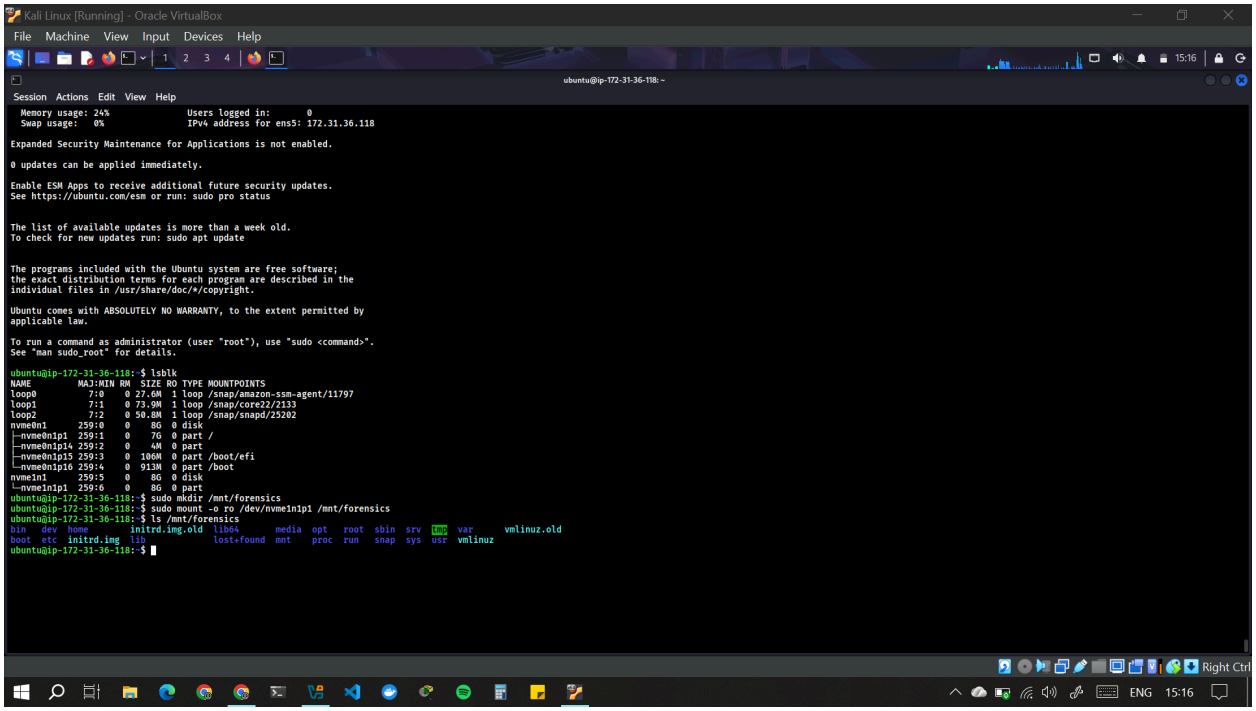
4. Now in the console you can create an EC2 (I prefer ubuntu, but any linux will do) in the us-west-2 region and in the storage options, choose the volume you just created.



5. SSH in with something like:
    - ssh -i YOUR_KEY.pem ubuntu@ec2-<public ip address with - instead of .>.us-west-2.compute.amazonaws.com



6. I mounted this extra volume by running:
    - ***Lsblk***
    - ***sudo mkdir /mnt/forensics***
    - ***sudo mount -o ro /dev/nvme1n1p1 /mnt/forensics*** (`-o ro` ensures **no changes are written** to the disk.)

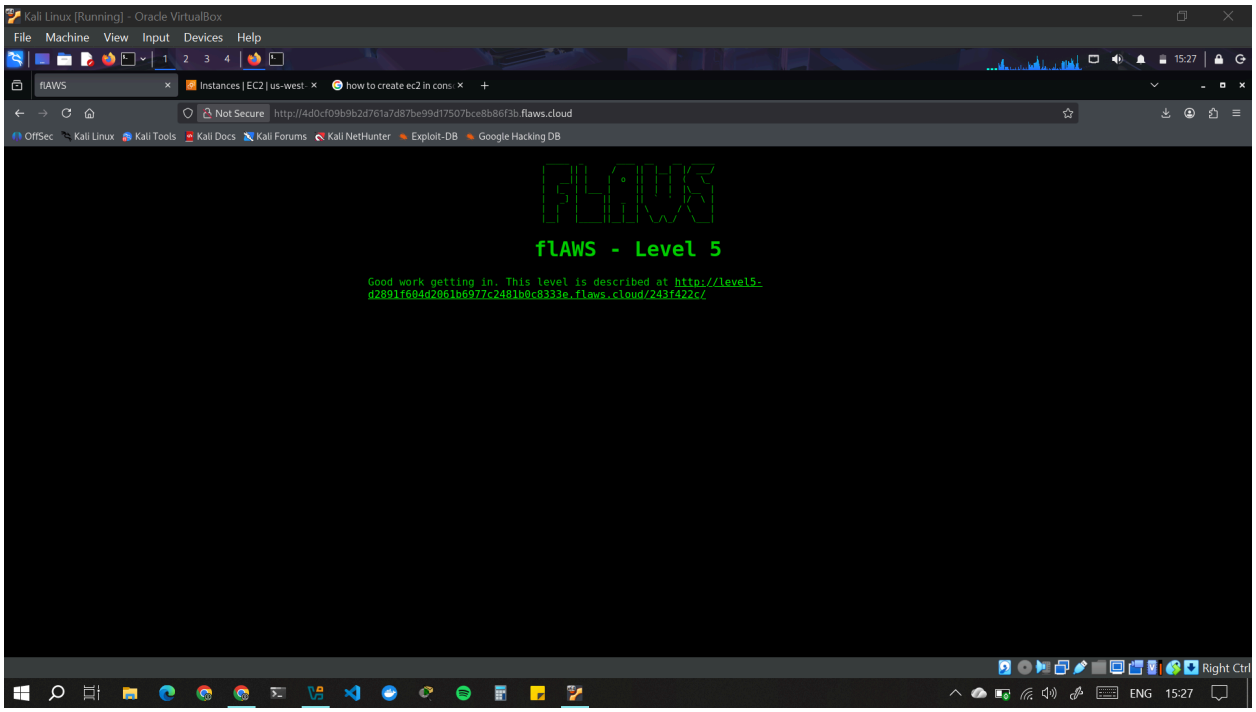


In the ubuntu user's home directory is the file: `/home/ubuntu/setupNginx.sh`

This creates the basic HTTP auth user:

htpasswd -b /etc/nginx/.htpasswd flaws nCP8xigdjpjyiXgJ7nJu7rw5Ro68iE8M

That is the username and password for the user. Enter those at 4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud



Lesson learned

AWS allows you to make snapshots of EC2's and databases (RDS). The main purpose for that is to make backups, but people sometimes use snapshots to get access back to their own EC2's when they forget the passwords. This also allows attackers to get access to things. Snapshots are normally restricted to your own account, so a possible attack would be an attacker getting access to an AWS key that allows them to start/stop and do other things with EC2's and then uses that to snapshot an EC2 and spin up an EC2 with that volume in your environment to get access to it. Like all backups, you need to be cautious about protecting them.

Level 5

This EC2 has a simple HTTP only proxy on it. Here are some examples of it's usage:

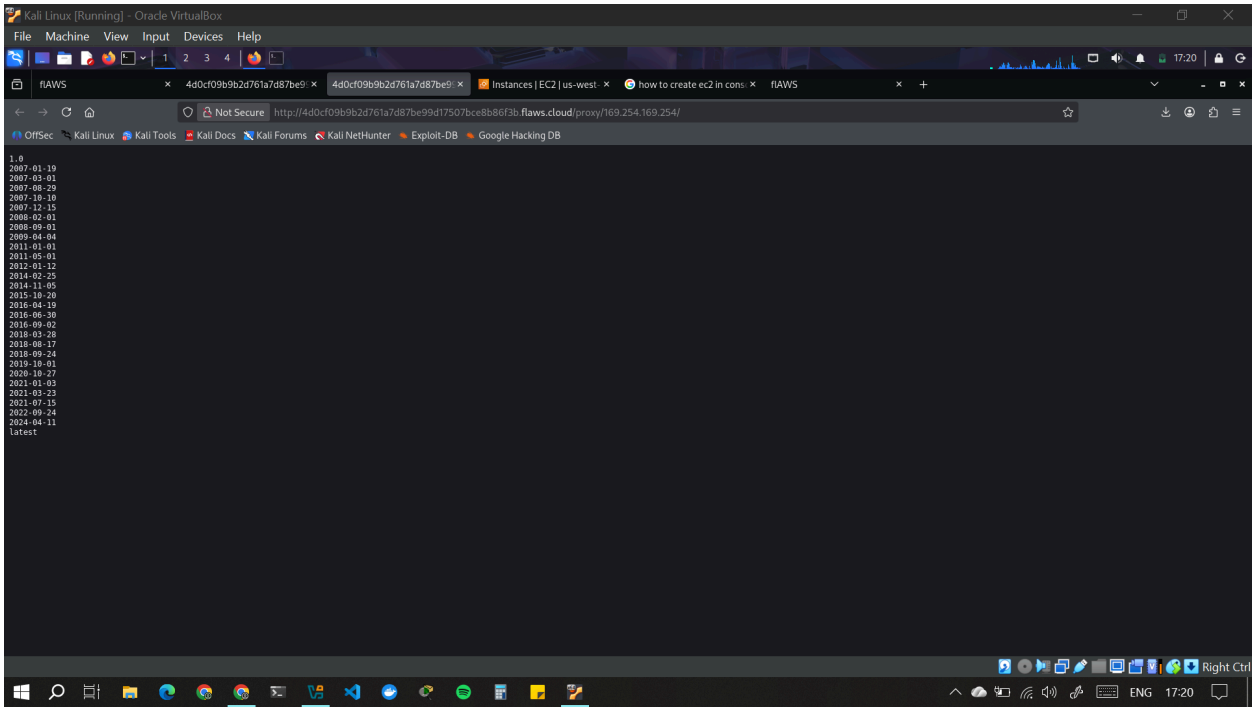
- http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/flaws.cloud/
- http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/summitroute.com/blog/feed.xml
- http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/neverssl.com/

See if you can use this proxy to figure out how to list the contents of the level6 bucket at level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud that has a hidden directory in it.

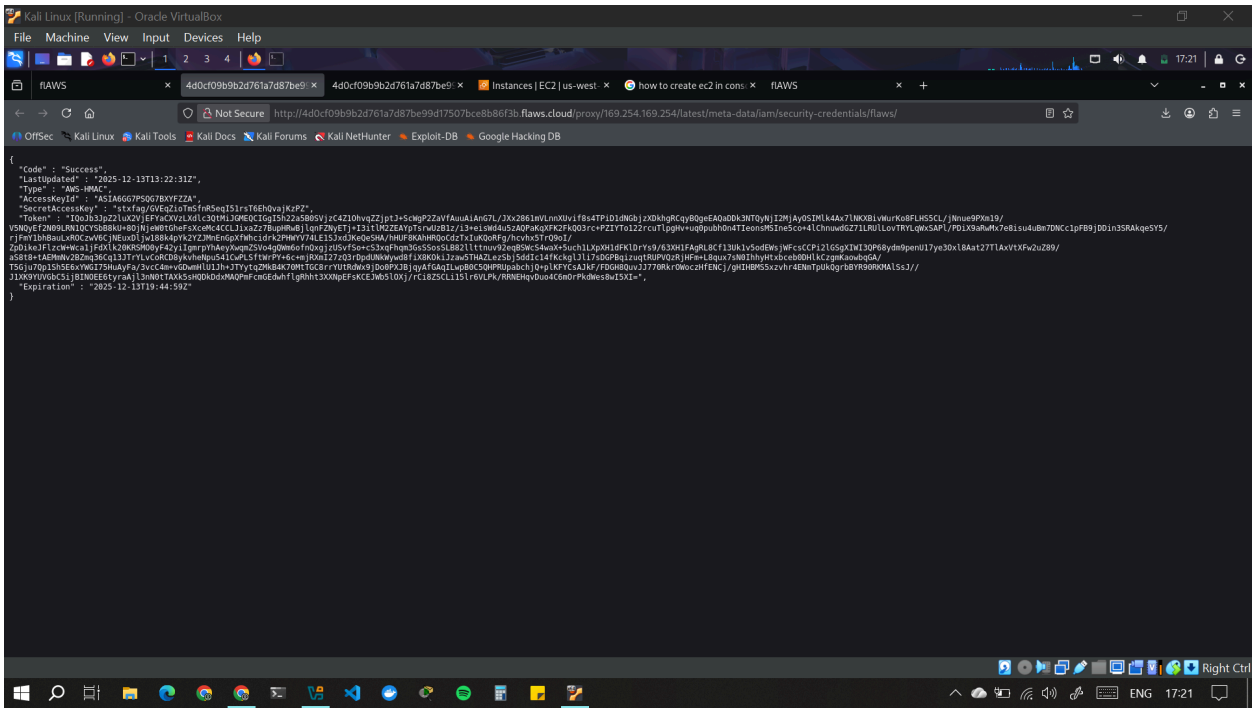
On cloud services, including AWS, the IP 169.254.169.254 is magical. It's the metadata service.

There is an RFC on it (RFC-3927), but you should read the AWS specific docs on it here.

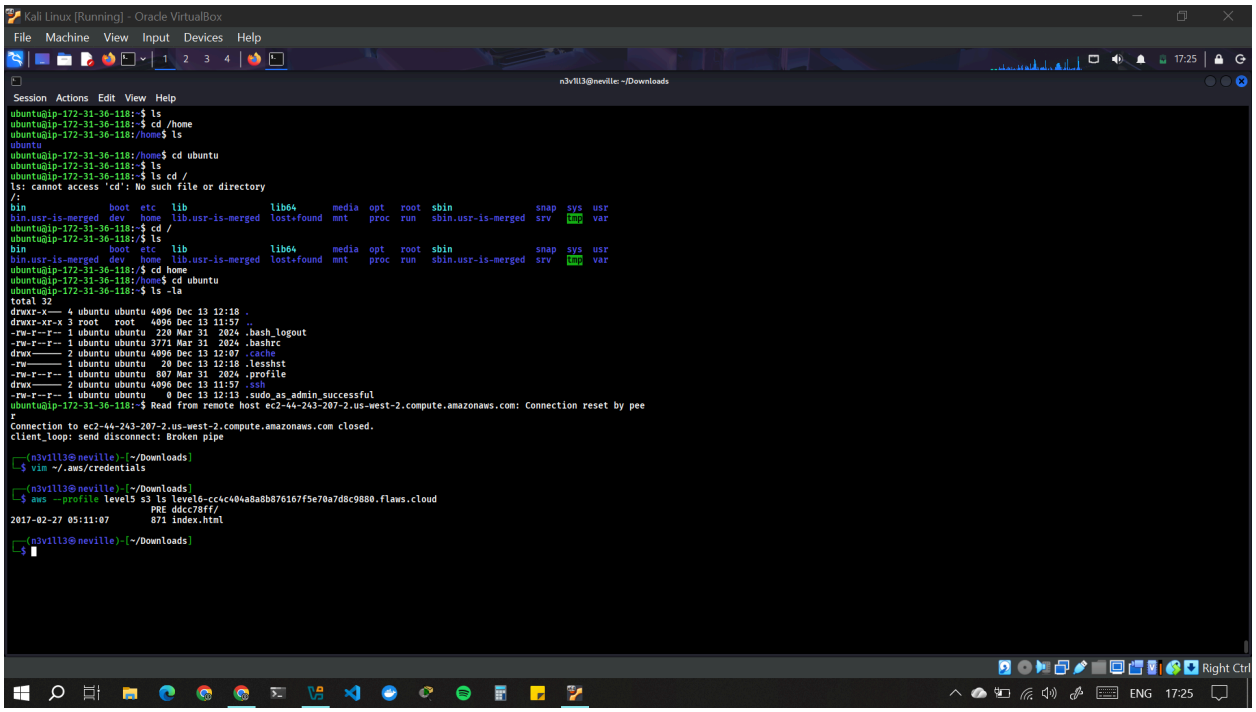
- Combining the proxy and the Ip address, I visited the link in my browser.



The latest directory had a bunch of other folders, one of which had security credentials provided by the IAM role of the EC2 that allowed you to list the contents of the level 6 bucket, which you knew existed from the .git repo key



Looking in the bucket shows:



The IP address 169.254.169.254 is a magic IP in the cloud world. AWS, Azure, Google, DigitalOcean and others use this to allow cloud resources to find out metadata about themselves. Some, such as Google, have additional constraints on the requests, such as requiring it to use `Metadata-Flavor: Google` as an HTTP header and refusing requests with an `X-Forwarded-For` header. AWS has recently created a new IMDSv2 that requires special headers, a challenge and response, and other protections, but many AWS accounts may not have enforced it. If you can make any sort of HTTP request from an EC2 to that IP, you'll likely get back information the owner would prefer you not see.

Examples of this problem

- Nicolas GrÃ©goire discovered that prezi allowed you point their servers at a URL to include as content in a slide, and this allowed you to point to 169.254.169.254 which provided the access key for the EC2 intance profile (link). He also found issues with access to that magic IP with Phabricator and Coinbase.

A similar problem to getting access to the IAM profile's access keys is access to the EC2's user-data, which people sometimes use to pass secrets to the EC2 such as API keys or credentials.

Avoiding this mistake

Ensure your applications do not allow access to 169.254.169.254 or any local and private IP ranges. Additionally, ensure that IAM roles are restricted as much as possible.

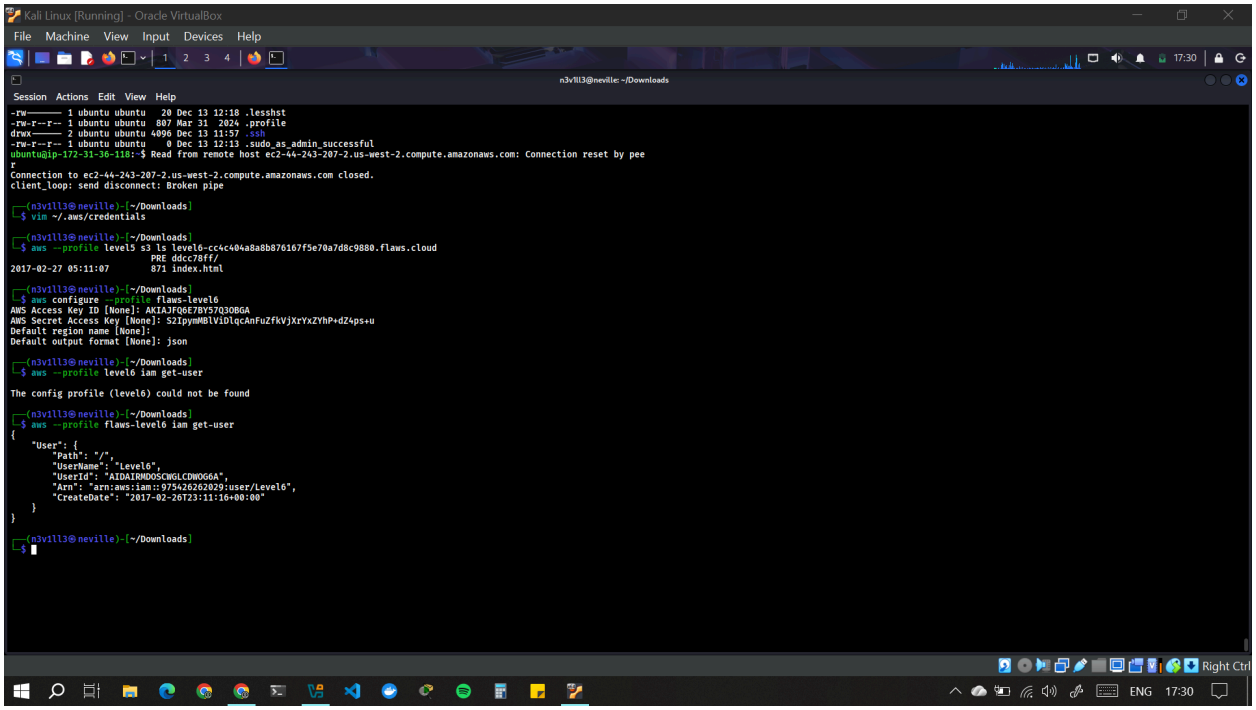
Level 6

For this final challenge, you're getting a user access key that has the SecurityAudit policy attached to it. See what else it can do and what else you might find in this AWS account.

Access key ID: AKIAJFQ6E7BY57Q3OBGA
Secret: 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

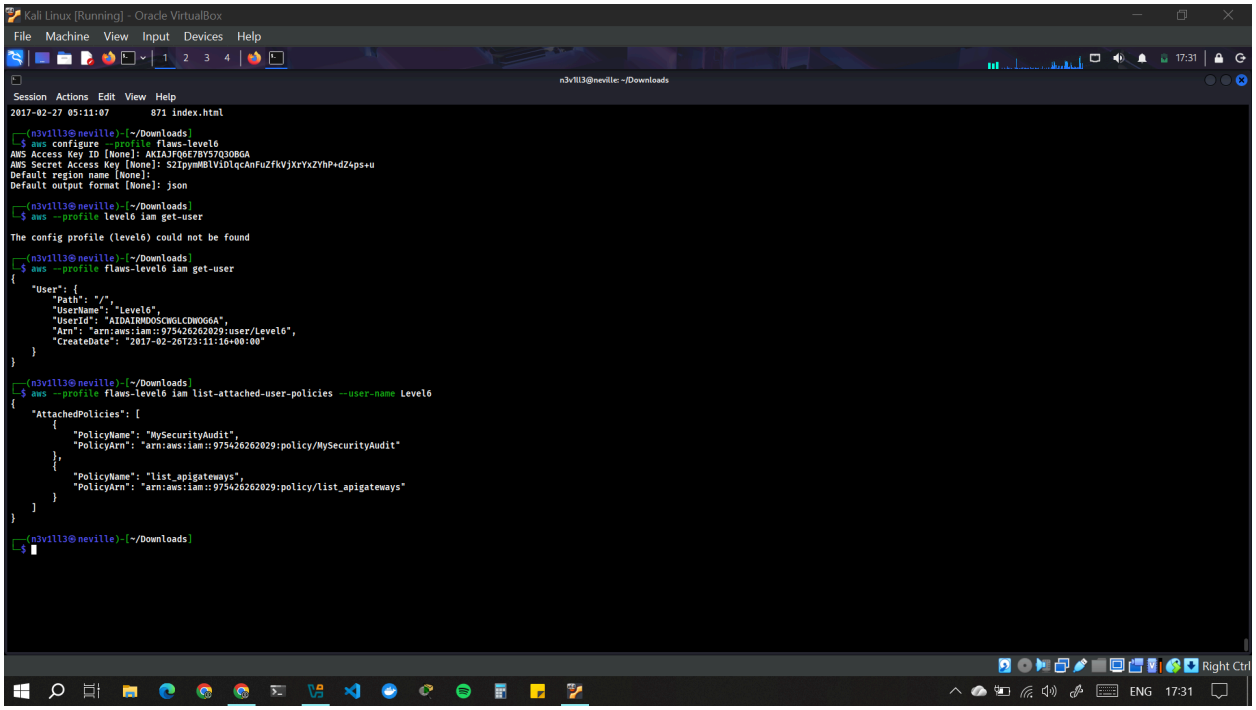
1. The SecurityAudit group can get a high level overview of the resources in an AWS account, but it's also useful for looking at IAM policies. First, find out who you are (assuming you named your profile "level6"):

Command: aws --profile flaws-level6 iam get-user



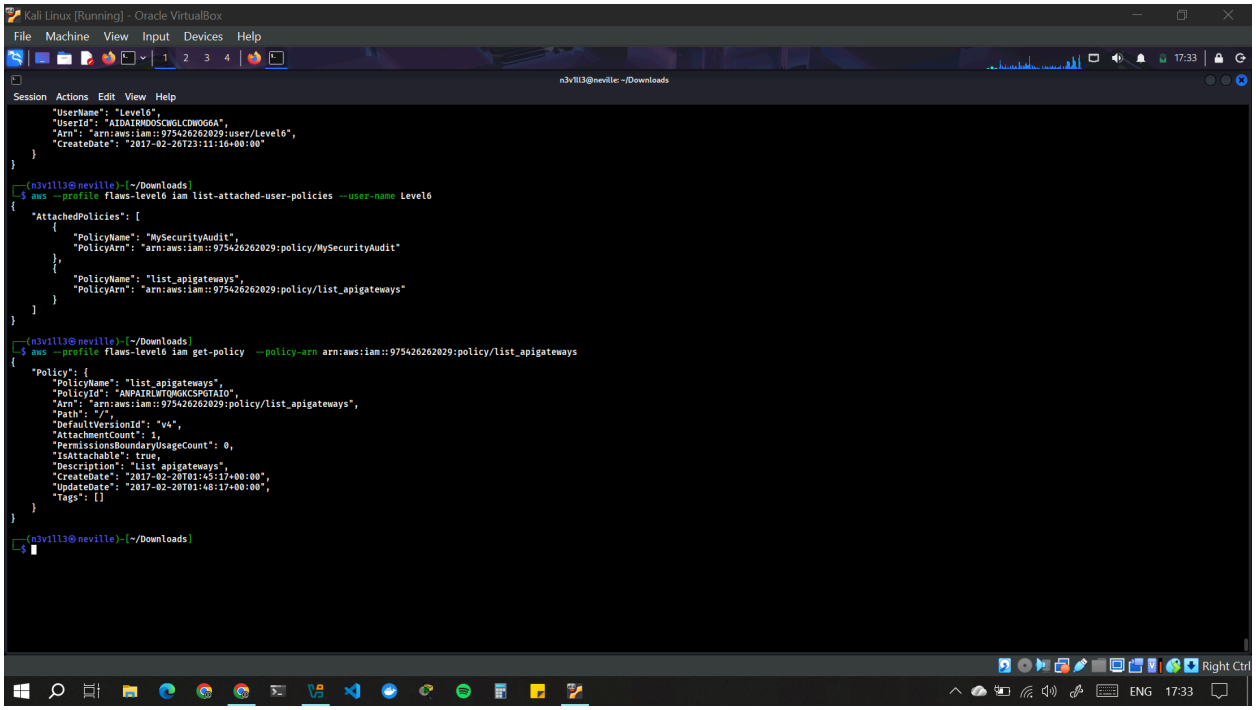
2. Now that you know your username is Level6 you can find out what policies are attached to it:

Command used: aws --profile flaws-level6 iam list-attached-user-policies --user-name Level6



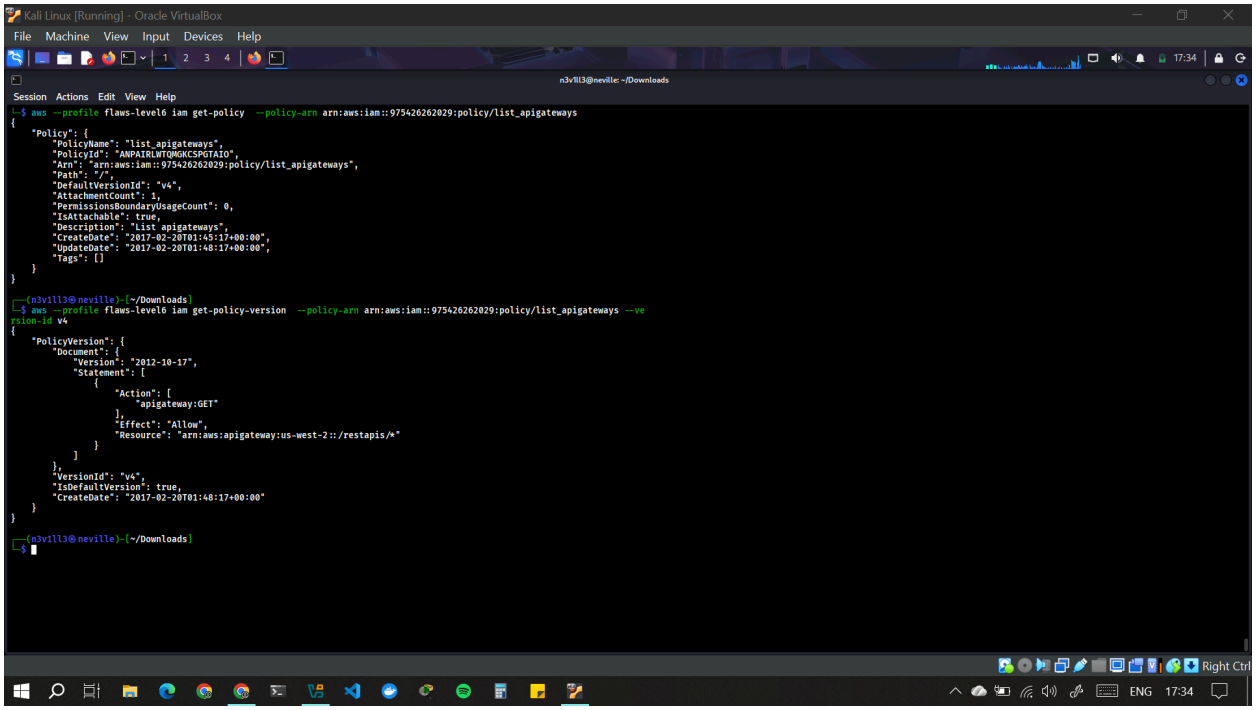
Once you know the ARN for the policy you can get it's version id:

aws --profile level6 iam get-policy --policy-arn arn:aws:iam::975426262029:policy/list_apigateways



Now that you have the ARN and the version id, you can see what the actual policy is:

aws --profile level6 iam get-policy-version --policy-arn arn:aws:iam::975426262029:policy/list_apigateways --version-id v4



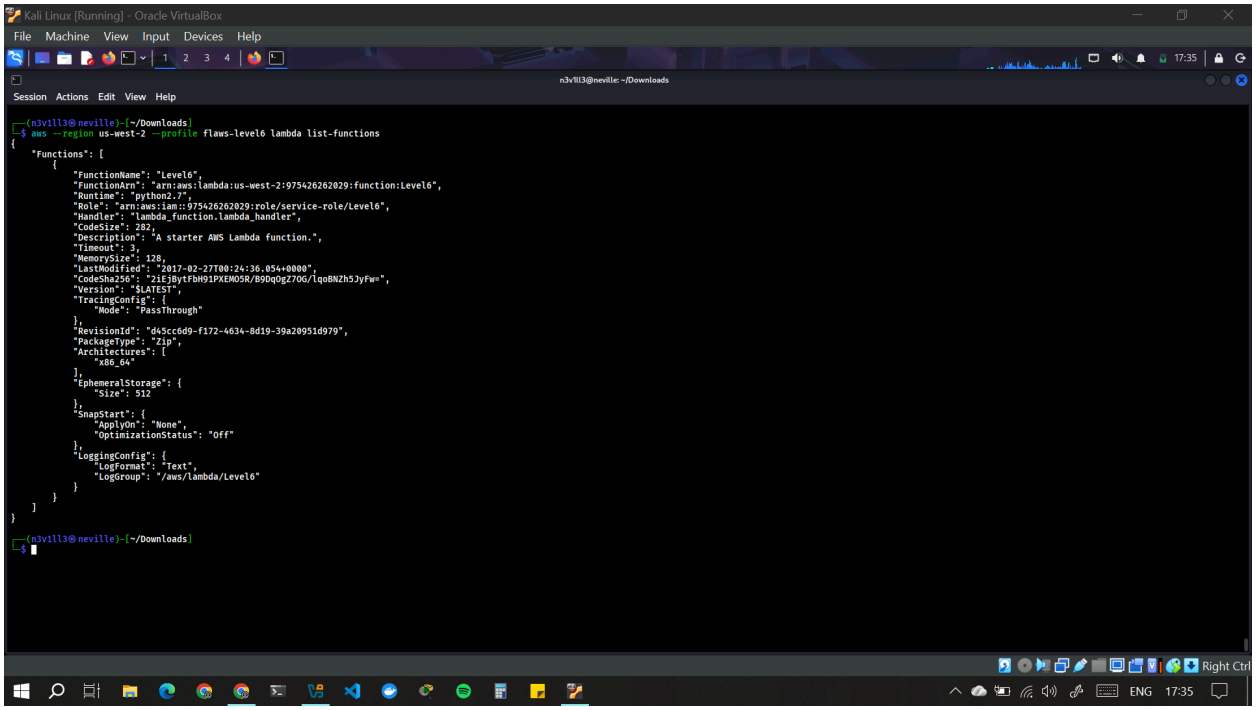
This tells us using this policy we can call "apigateway:GET" on "arn:aws:apigateway:us-west-2::/restapis/*"

I now have the ability to GET "arn:aws:apigateway:us-west-2::/restapis/*"

The API gateway in this case is used to call a lambda function, but you need to figure out how to invoke it.

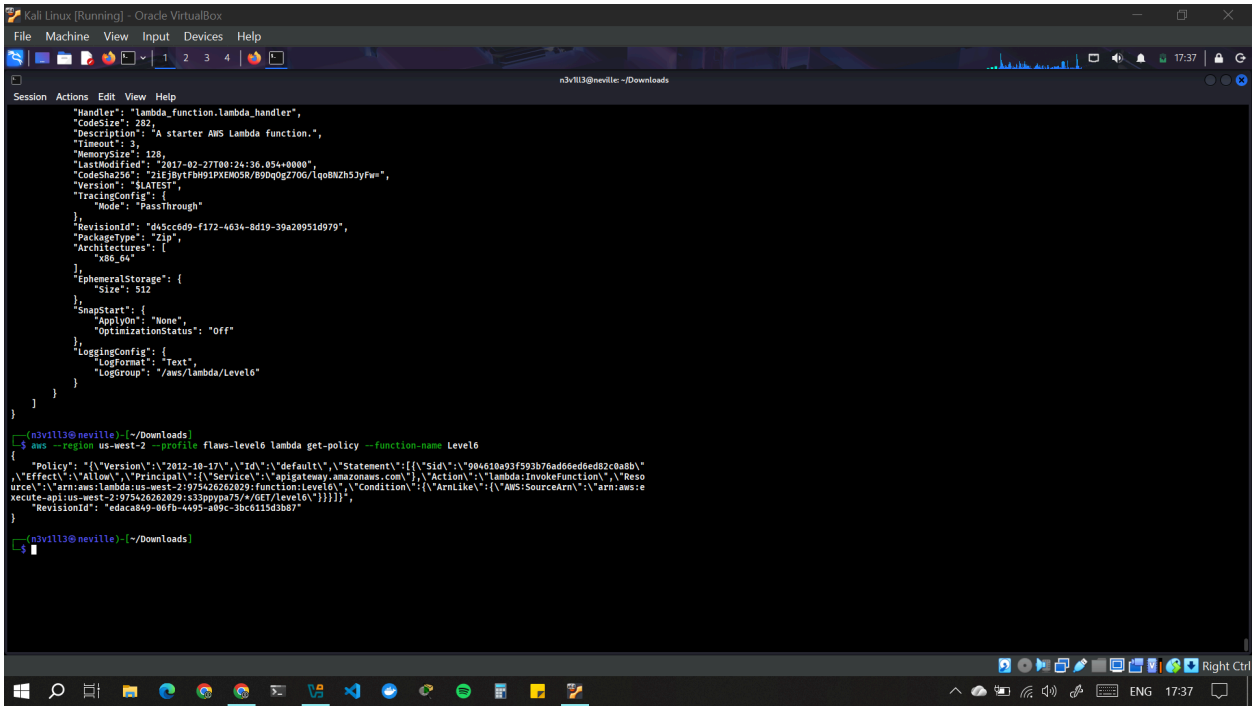
The SecurityAudit policy lets you see some things about lambdas:

aws --region us-west-2 --profile level6 lambda list-functions



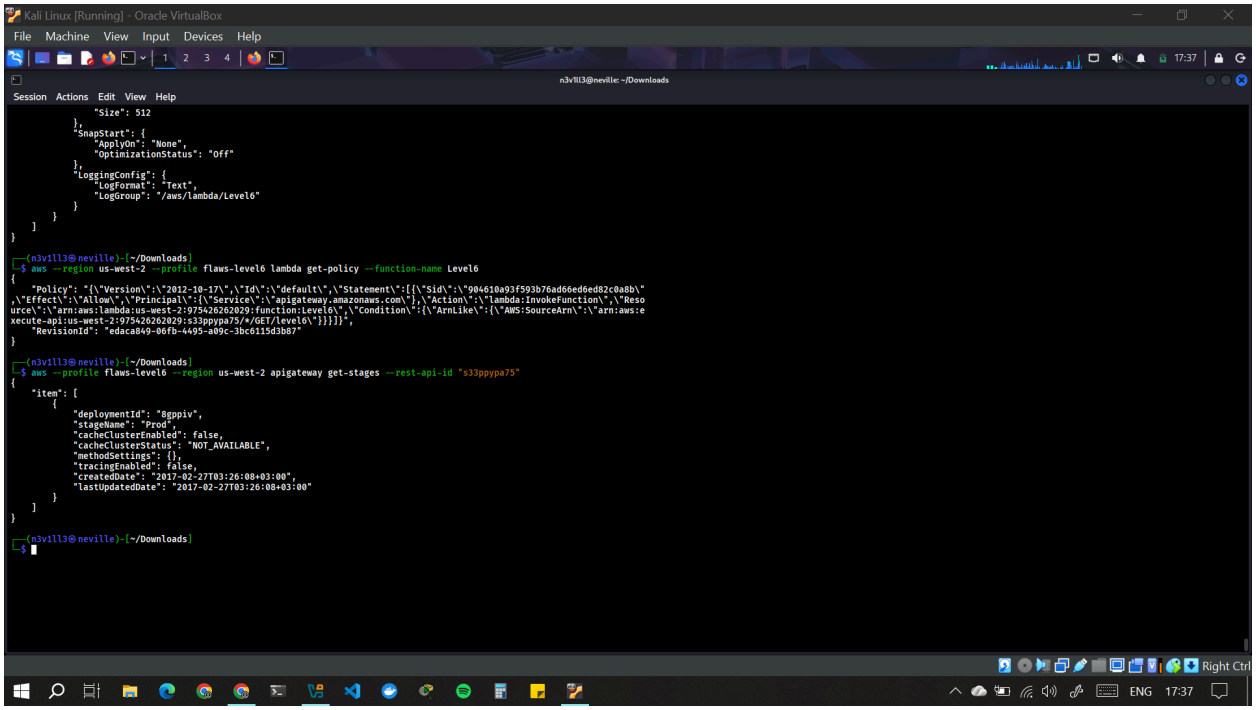
That tells you there is a function named "Level6", and the SecurityAudit also lets you run:

aws --region us-west-2 --profile level6 lambda get-policy --function-name Level6



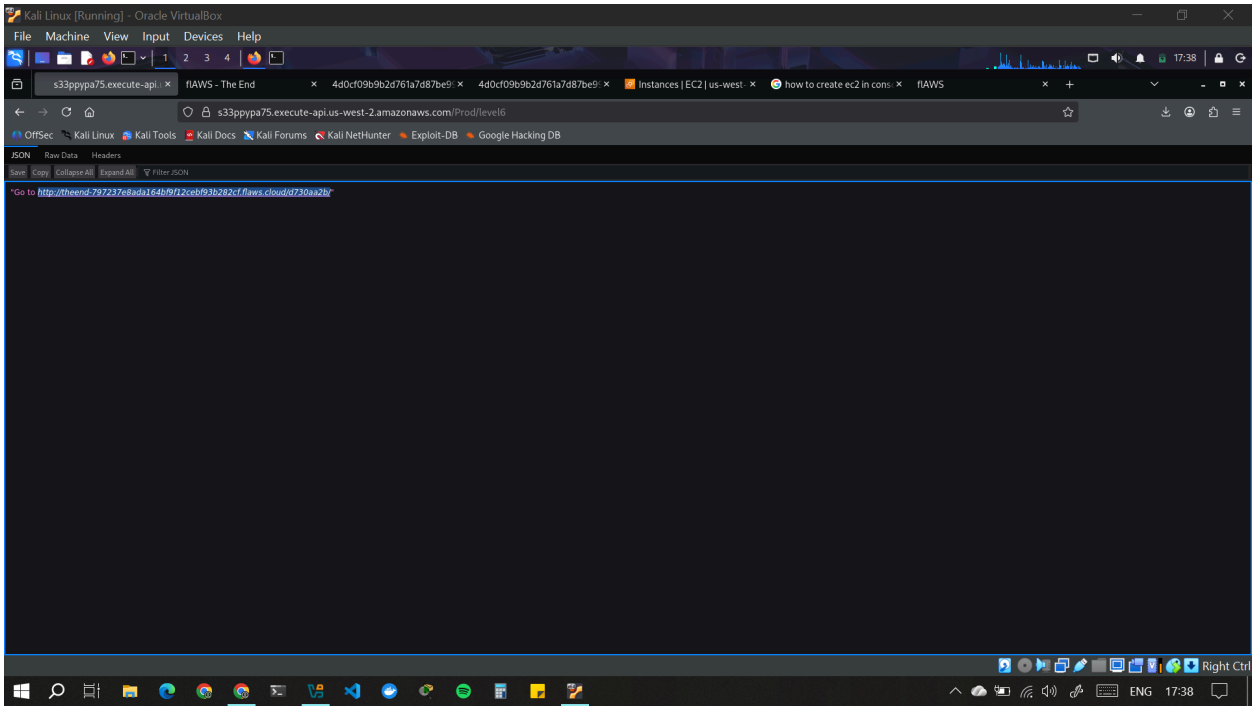
This tells you about the ability to execute `arn:aws:execute-api:us-west-2:975426262029:s33ppypa75/*/GET/level6\` That "s33ppypa75" is a rest-api-id, which you can then use with that other attached policy:

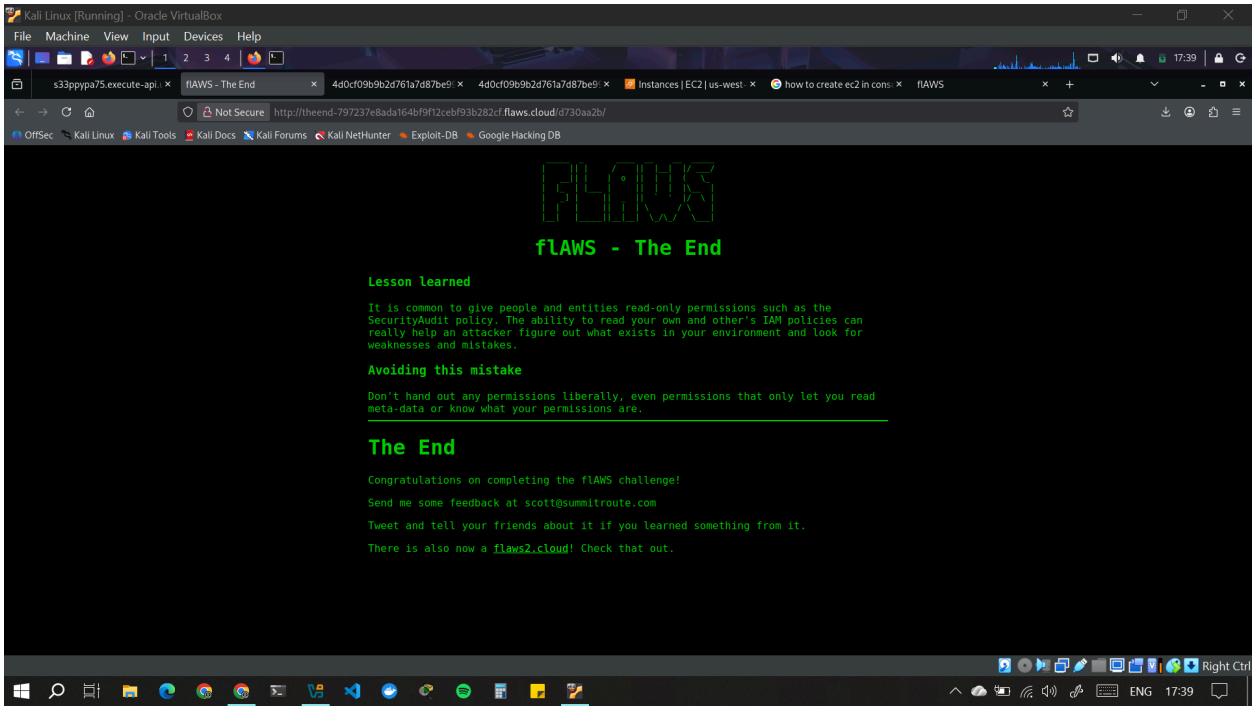
aws --profile level6 --region us-west-2 apigateway get-stages --rest-api-id "s33ppypa75"



That tells you the stage name is "Prod". Lambda functions are called using that rest-api-id, stage name, region, and resource as https://s33ppypa75.execute-api.us-west-2.amazonaws.com/Prod/level6

I visited that URL.





Lesson learned

It is common to give people and entities read-only permissions such as the SecurityAudit policy. The ability to read your own and other's IAM policies can really help an attacker figure out what exists in your environment and look for weaknesses and mistakes.

Avoiding this mistake

Don't hand out any permissions liberally, even permissions that only let you read meta-data or know what your permissions are.