## Introduction

This challenge was designed as a **cloud threat hunting exercise** against an AWS environment. My objective was to analyze AWS CloudTrail logs, identify suspicious IAM user activity and trace the steps of a potential insider threat.

Initially, I planned to spend the weekend understanding CloudTrail logs more deeply , focusing on IAM users, the policies attached to them, and their interactions with AWS services. What began as a log analysis task evolved into a hands-on **cloud threat hunt**.

Now, I've worked with **Splunk logs** before, and I'm comfortable navigating through dashboards and filtering events. But **sifting through raw CloudTrail logs with `jq` and `grep` was a whole different beast**. There's no UI here , just layers of nested JSON, and you have to know exactly what to look for and how to get to it efficiently.

While I had prior experience reading CloudTrail logs, this challenge pushed me further. It demanded an attacker's mindset:

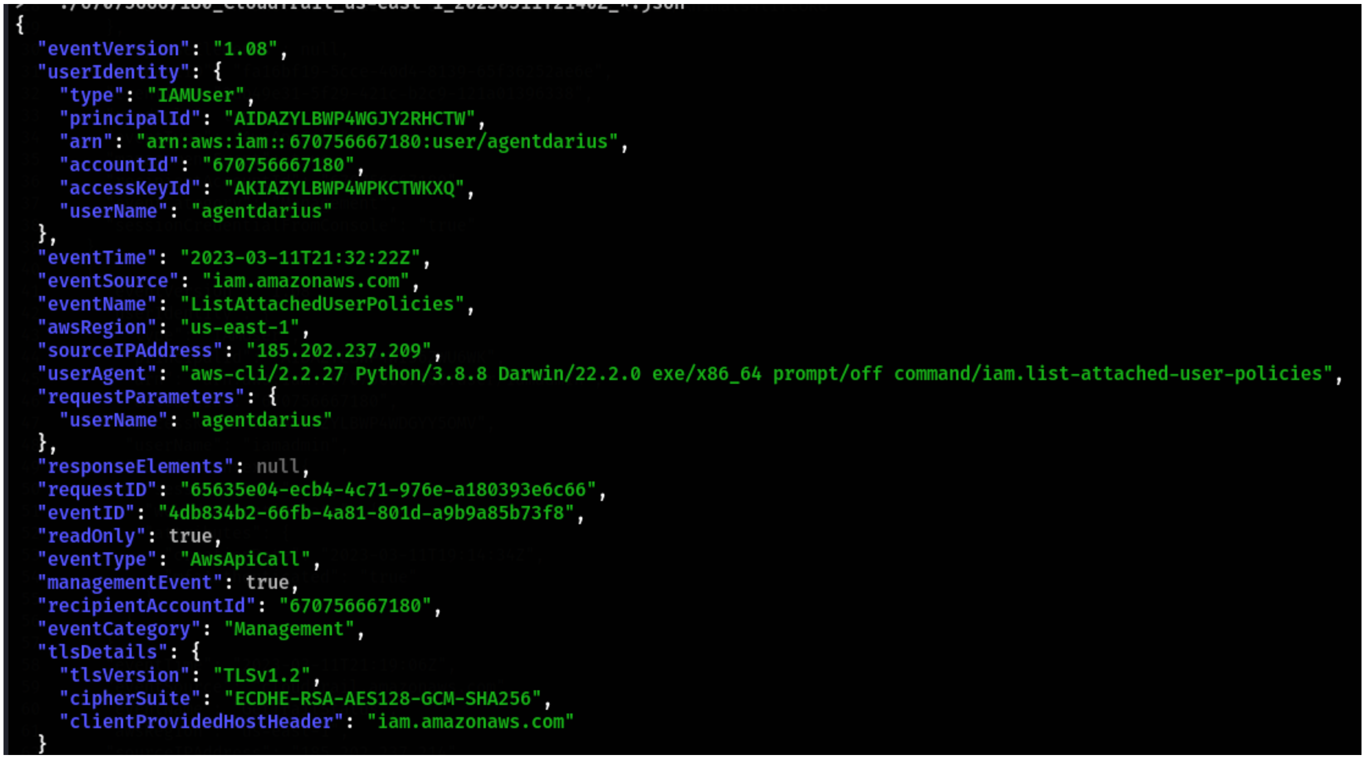
- How would someone enumerate IAM users and attached policies?
- What signs of privilege escalation or persistence should I look for?
- How can data exfiltration be quietly hidden inside legitimate-looking events?

This write-up documents the entire process i.e. from spotting the suspicious IAM user, to tracking how they moved through the environment, escalated access, and ultimately exposed data.

## Pre-requisites

To begin the investigation, I accessed the lab environment using an **RDP connection**, which conveniently dropped me into a **Kali Linux VM**. This was ideal, since I'm already comfortable working in Kali , it's my go-to environment for security analysis. Having access to familiar Linux tools like `jq` , `grep` , and shell scripting made it easy to sift through raw CloudTrail logs without the need for a fancy UI. I was able to move quickly and efficiently through the data using the terminal.

```
cd /home/analyst/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11/
```



```
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAZYLBWP4WGJY2RHCTW",
    "arn": "arn:aws:iam::670756667180:user/agentdarius",
    "accountId": "670756667180",
    "accessKeyId": "AKIAZYLBWP4WPKCTWKXQ",
    "userName": "agentdarius"
  },
  "eventTime": "2023-03-11T21:32:22Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "ListAttachedUserPolicies",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "185.202.237.209",
  "userAgent": "aws-cli/2.2.27 Python/3.8.8 Darwin/22.2.0 exe/x86_64 prompt/off command/iam.list-attached-user-policies",
  "requestParameters": {
    "userName": "agentdarius"
  },
  "responseElements": null,
  "requestID": "65635e04-ecb4-4c71-976e-a180393e6c66",
  "eventID": "4db834b2-66fb-4a81-801d-a9b9a85b73f8",
  "readOnly": true,
  "eventType": "AwsApiCall",
  "managementEvent": true,
  "recipientAccountId": "670756667180",
  "eventCategory": "Management",
  "tlsDetails": {
    "tlsVersion": "TLSv1.2",
    "cipherSuite": "ECDHE-RSA-AES128-GCM-SHA256",
    "clientProvidedHostHeader": "iam.amazonaws.com"
  }
```

After decompressing the logs, I began by opening a few of the `.json` files to get a feel for the structure and the type of data available. Since these were **AWS CloudTrail logs**, each file consisted of a top-level `Records` array, with each element representing a single API call made within the AWS environment.

As I skimmed through the entries, I started to notice a consistent structure across each record. Some of the key fields that stood out were:

- `eventTime` – The timestamp of the action
- `userIdentity` – Who performed the action (IAM user, role, federated user, etc.)
- `eventName` – The actual AWS API call that was made (e.g., `CreateUser`, `ListBuckets`)
- `eventSource` – The AWS service being interacted with (e.g., `iam.amazonaws.com`, `s3.amazonaws.com`)
- `sourceIPAddress` – The IP address from which the request originated
- `userAgent` – The client or tool used to issue the request (e.g., AWS CLI, internal service, console)
- `requestParameters` – The parameters submitted with the API call (e.g., user name, bucket name)
- `responseElements` – The response returned by the AWS API.

This initial review helped set the stage for what to look out for. I now had a clear understanding of **how user behavior and AWS operations were recorded**, and could start hunting for suspicious actions based on fields like `eventName`, `userIdentity`, and `sourceIPAddress`.

## What is the name of the IAM user account being used by the SIA agent?

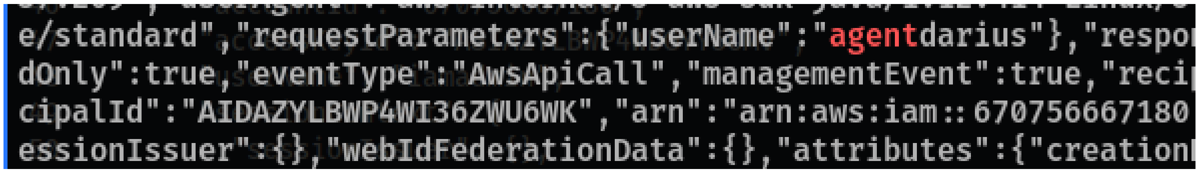
The challenge provided a subtle hint:

> *"He's an agent."*

That led me to try a simple keyword search for anything related to **"agent"** across all the decompressed CloudTrail logs. I used the following command:

```
grep "agent" ./
```

Sure enough, a whole lot of lines were returned. Upon inspecting them, I noticed that the IAM user involved was named: agentdarius. So this user is now our subject of interest.

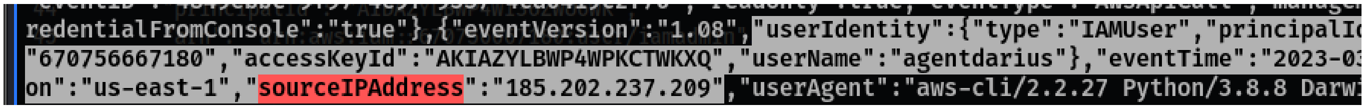


```
e/standard","requestParameters":{"userName":"agentdarius"},"respo
dOnly":true,"eventType":"AwsApiCall","managementEvent":true,"reci
cipalId":"AIDAZYLBWP4WI36ZWU6WK","arn":"arn:aws:iam::670756667180
essionIssuer":{},"webIdFederationData":{},"attributes":{"creation
```

## What is the source IP the SIA agent is authenticating from?

In AWS CloudTrail logs, every API call includes a `sourceIPAddress` field that shows the origin of the request. To find this, I began by searching across all the log files using:

```
grep -r "sourceIPAddress" ./
```



```
redentialFromConsole":"true"},{"eventVersion":"1.08","userIdentity":{"type":"IAMUser","principalI
"670756667180","accessKeyId":"AKIAZYLBWP4WPKCTWKXQ","userName":"agentdarius"},"eventTime":"2023-0
on":"us-east-1","sourceIPAddress":"185.202.237.209","userAgent":"aws-cli/2.2.27 Python/3.8.8 Darw
```

This command helped surface all entries where a `sourceIPAddress` was recorded. From there, I filtered down to just the entries involving `agentdarius`.Upon reviewing those matching logs, I discovered that all of `agentdarius`'s requests originated from the same IP address: 185.202.237.209

## Diving Deeper : Tracking the Attacker's Activity

With both the **IAM user** ( `agentdarius` ) and their **source IP address** ( `185.202.237.209` ) identified, the investigation naturally progressed into a more detailed phase.

The goal from here was to track the user's movement across the AWS environment, understand what permissions they had, and uncover how they might have established **persistence** or performed **privilege escalation**.

At first glance, it might seem like a simple task. Just follow the user’s activity and look at what they did. But in reality, CloudTrail logs require careful interpretation. Each field like `eventName` , `eventSource` , `requestParameters` , and `responseElements` can reveal something critical about the user's intent or the impact of their actions.

Missing or misreading a field could mean overlooking a major part of the attack. So I took the time to read through several entries, align them by time, and get a clear sense of the attack sequence.

With that mindset, I started putting the pieces together. The real investigation was just beginning.

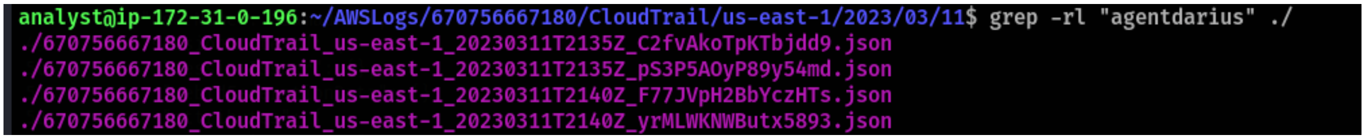
## Investigation: What was the SIA agent’s activity related to enumerating identities & permissions?

**Step 1: Identify Files with Activity from the SIA Agent ( `agentdarius` )**

To begin the investigation, I needed to locate all CloudTrail log files in a specific directory that referenced the SIA agent of interest — in this case, `agentdarius` .

I used the following `grep` command to **recursively search for JSON files** that include the string `agentdarius` :

```
grep -rl "agentdarius" ./
```



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ grep -rl "agentdarius" ./
./670756667180_CloudTrail_us-east-1_20230311T2135Z_C2fvAkoTpKTbjdd9.json
./670756667180_CloudTrail_us-east-1_20230311T2135Z_pS3P5AOyP89y54md.json
./670756667180_CloudTrail_us-east-1_20230311T2140Z_F77JVpH2BbYczHTs.json
./670756667180_CloudTrail_us-east-1_20230311T2140Z_yrMLWKNWButx5893.json
```

I see there's 4 log files that's returned back to us.
**Step 2 : Identify the EventName from the returned logs

To begin mapping out the attacker’s behavior, I first focused on **enumeration**. In the context of AWS, enumeration typically involves API calls that allow a user to list resources, roles, permissions, or policies. These are usually the **first steps** an attacker takes after gaining access ,to understand the scope of their permissions and what they can potentially exploit.

Since CloudTrail logs capture every API call under the field `eventName`, I used that to extract a complete list of actions performed by `agentdarius`. Here's the command I used:

```
jq -r '.Records[]
  | select(.userIdentity.userName == "agentdarius")
  | .eventName' \
  ./670756667180_CloudTrail_us-east-1_20230311T2135Z_C2fvAkoTpKTbjdd9.json \
  ./670756667180_CloudTrail_us-east-1_20230311T2135Z_pS3P5AOyP89y54md.json \
  ./670756667180_CloudTrail_us-east-1_20230311T2140Z_F77JVpH2BbYczHTs.json \
  ./670756667180_CloudTrail_us-east-1_20230311T2140Z_yrMLWKNWButx5893.json \
  | sort | uniq
```

This command pulled all `eventName` fields from the logs where the user was `agentdarius`, then sorted and removed duplicates. Since `eventName` directly maps to the AWS API operation performed, it gave a precise view of the user's actions within the environment.

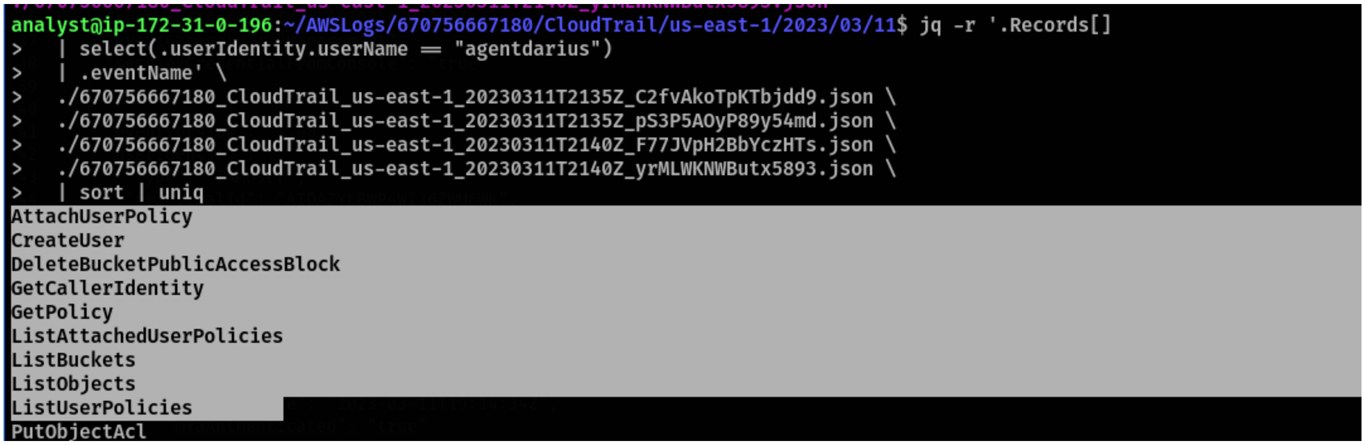
From the output, I reviewed which API calls were specifically related to **enumerating identities and permissions**. These included:

- `ListUserPolicies`
- `ListAttachedUserPolicies`
- `GetPolicy`
- `GetCallerIdentity`

These actions suggested that `agentdarius` was probing the IAM configuration , likely trying to understand what policies were available, what permissions were already assigned, and what opportunities existed for escalation.

Identifying enumeration is a crucial early step in understanding attacker objectives. In this case, it revealed the groundwork being laid before further movement or privilege abuse. Although the focus of the next phase would be on **IAM policies and privileges**, it was clear that `agentdarius` was also exploring **S3 buckets**, likely scouting for sensitive data to exfiltrate later.

With enumeration behavior confirmed, the next step was to understand **what permissions this user had**, and whether they attempted **privilege escalation or persistence** within the AWS environment.



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(.userIdentity.userName == "agentdarius")
>   | .eventName' \
>   ./670756667180_CloudTrail_us-east-1_20230311T2135Z_C2fvAkoTpKTbjdd9.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2135Z_pS3P5AOyP89y54md.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2140Z_F77JVpH2BbYczHTs.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2140Z_yrMLWKNWButx5893.json \
>   | sort | uniq
AttachUserPolicy
CreateUser
DeleteBucketPublicAccessBlock
GetCallerIdentity
GetPolicy
ListAttachedUserPolicies
ListBuckets
ListObjects
ListUserPolicies
PutObjectAcl
```

**Step 3 : Determining the IAM User's Attached Managed Policy

After identifying that `agentdarius` had been performing IAM enumeration, the next goal was to figure out **what permissions he actually had**.

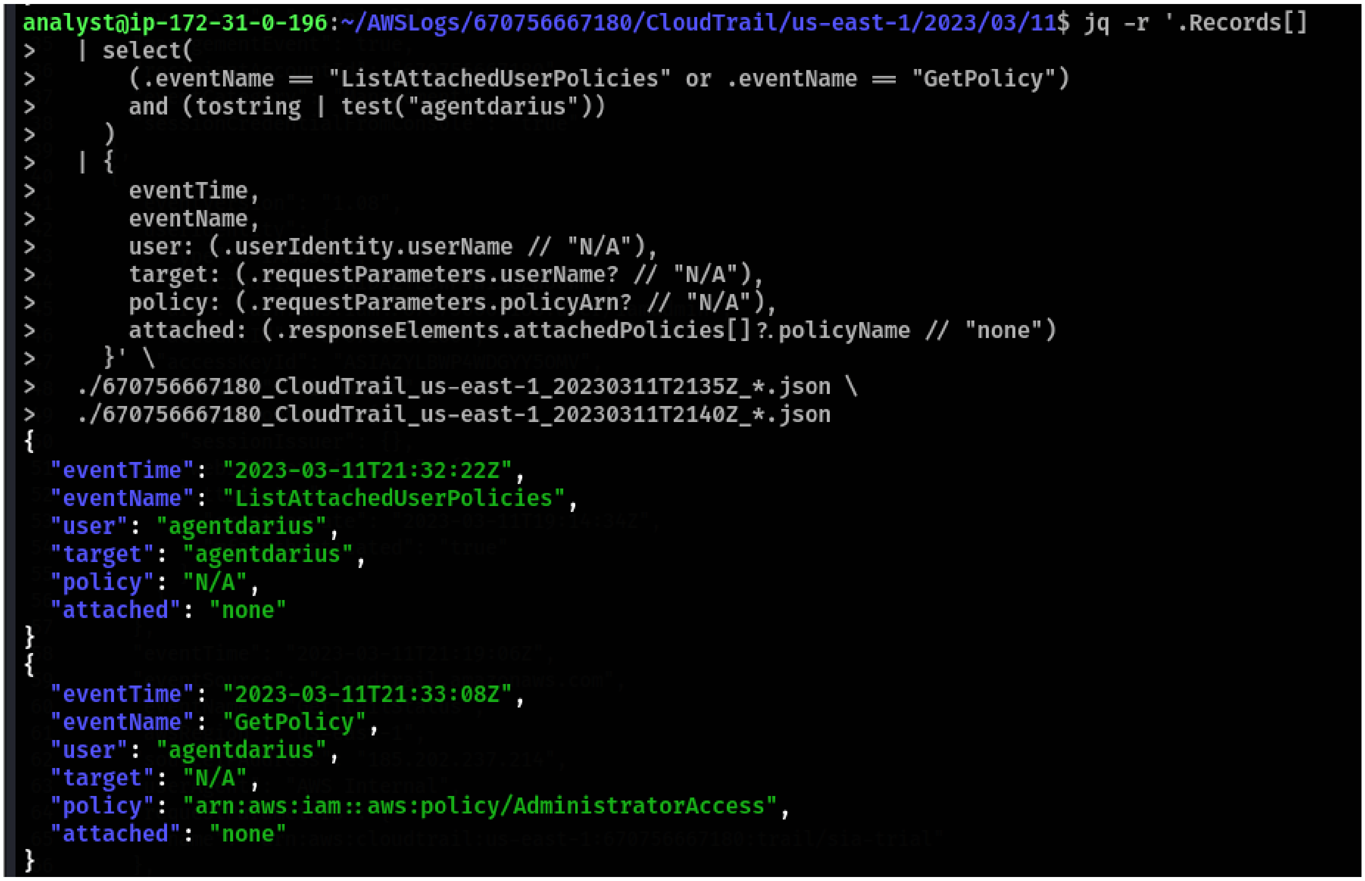
To answer this, I focused on two specific API operations recorded in the CloudTrail logs:

1. `ListAttachedUserPolicies`
   This API call returns the list of **managed IAM policies** that are attached to a specific IAM user. It's a direct way to see what permissions the user has access to through reusable AWS-managed or customer-managed policies.
   For Example: *"What rules does this person have?"*
   This tells you what set of rules (called policies) are given to a user.

2. `GetPolicy`
   This API retrieves detailed metadata about a specific managed policy including its ARN (Amazon Resource Name) and policy document allowing us to understand **what that policy actually allows**.
   For Example: *What do those rules say?"*
   This shows you what those rules actually allow the person to do.

Since these two operations provide direct insight into the **permissions** tied to the user, I filtered the logs for just these events where `agentdarius` was either the **caller** or the **target** of the request.
``

```
jq -r '.Records[]
  | select(
      (.eventName == "ListAttachedUserPolicies" or .eventName == "GetPolicy")
      and (tostring | test("agentdarius"))
    )
  | {
      eventTime,
      eventName,
      user: (.userIdentity.userName // "N/A"),
      target: (.requestParameters.userName? // "N/A"),
      policy: (.requestParameters.policyArn? // "N/A"),
      attached: (.responseElements.attachedPolicies[]?.policyName // "none")
    }' \
  ./670756667180_CloudTrail_us-east-1_20230311T2135Z_*.json \
  ./670756667180_CloudTrail_us-east-1_20230311T2140Z_*.json
```



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(
>       (.eventName == "ListAttachedUserPolicies" or .eventName == "GetPolicy")
>       and (tostring | test("agentdarius"))
>     )
>   | {
>       eventTime,
>       eventName,
>       user: (.userIdentity.userName // "N/A"),
>       target: (.requestParameters.userName? // "N/A"),
>       policy: (.requestParameters.policyArn? // "N/A"),
>       attached: (.responseElements.attachedPolicies[]?.policyName // "none")
>     }' \
>   ./670756667180_CloudTrail_us-east-1_20230311T2135Z_*.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2140Z_*.json
{
  "eventTime": "2023-03-11T21:32:22Z",
  "eventName": "ListAttachedUserPolicies",
  "user": "agentdarius",
  "target": "agentdarius",
  "policy": "N/A",
  "attached": "none"
}
{
  "eventTime": "2023-03-11T21:33:08Z",
  "eventName": "GetPolicy",
  "user": "agentdarius",
  "target": "N/A",
  "policy": "arn:aws:iam::aws:policy/AdministratorAccess",
  "attached": "none"
}
```

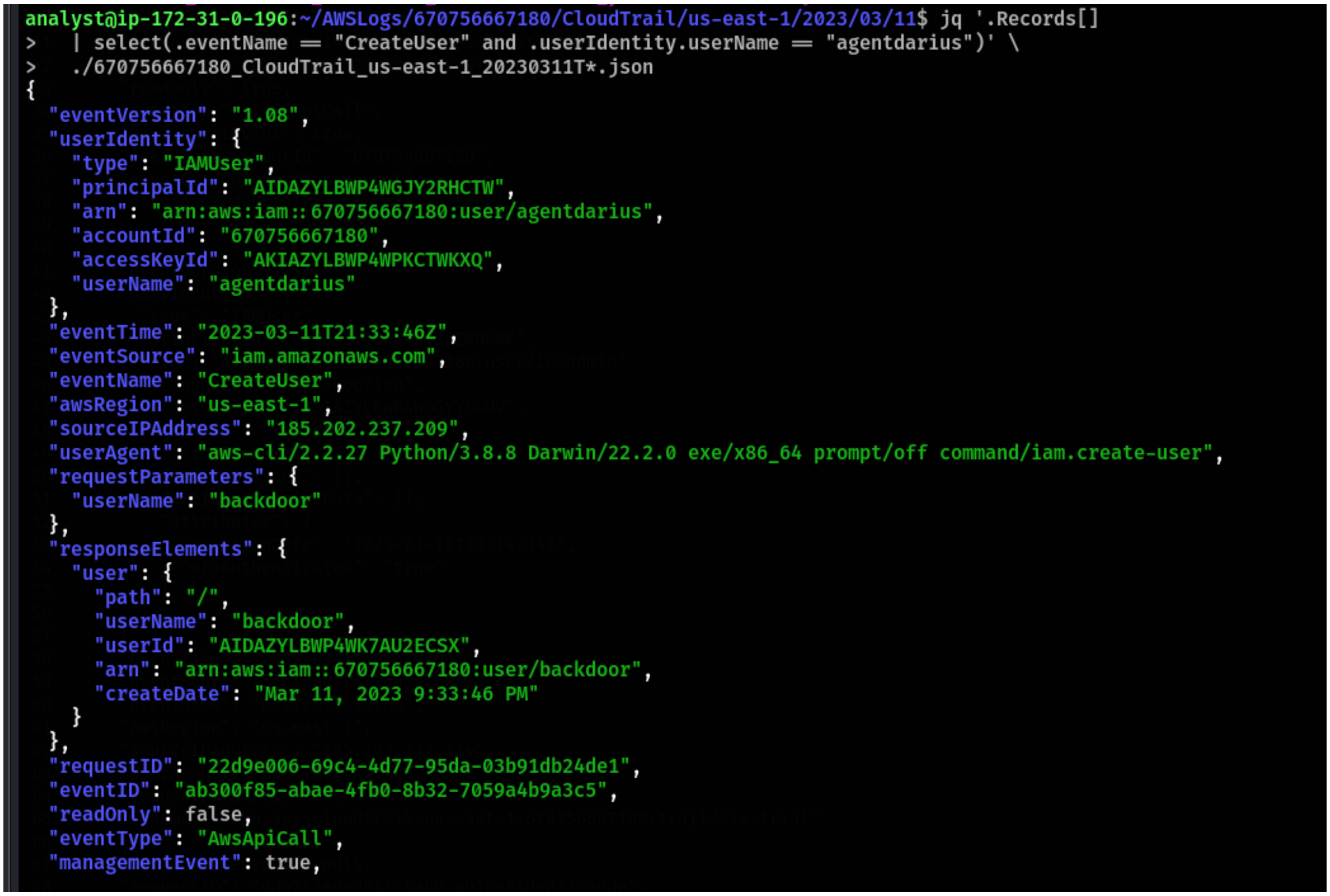
This command revealed that `agentdarius` had the `AdministratorAccess` managed policy attached , giving him **full permissions** across the AWS environment. That explains how he was able to create users, list objects, and modify bucket ACLs later in the timeline.

**Step 4 : What Permissions Were Associated with the Persistence Attempt?

From the earlier list of API operations extracted for `agentdarius` , one stood out immediately `CreateUser` . This strongly indicated that the attacker was attempting to **establish persistence** by creating a new IAM identity. To confirm this, I located and expanded the log entry that recorded this `CreateUser` operation.

The log confirmed that a new IAM user named `backdoor` was created by `agentdarius`. This is a classic persistence tactic creating an additional access point that can be used later, especially if the original credentials are revoked or detected.

```
jq -r '.Records[]
  | select(.eventName == "CreateUser" and (tostring | test("agentdarius")))
  | {
      eventTime,
      user: (.userIdentity.userName // "N/A"),
      target: (.requestParameters.userName? // "N/A"),
      attached: (.responseElements.attachedPolicies[]?.policyName // "none")
    }' <your-files>
```



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq '.Records[]
>   | select(.eventName == "CreateUser" and .userIdentity.userName == "agentdarius")' \
>   ./670756667180_CloudTrail_us-east-1_20230311T*.json
{
  "eventVersion": "1.08",
  "userIdentity": {
    "type": "IAMUser",
    "principalId": "AIDAZYLBWP4WGJY2RHCTW",
    "arn": "arn:aws:iam::670756667180:user/agentdarius",
    "accountId": "670756667180",
    "accessKeyId": "AKIAZYLBWP4WPKCTWKXQ",
    "userName": "agentdarius"
  },
  "eventTime": "2023-03-11T21:33:46Z",
  "eventSource": "iam.amazonaws.com",
  "eventName": "CreateUser",
  "awsRegion": "us-east-1",
  "sourceIPAddress": "185.202.237.209",
  "userAgent": "aws-cli/2.2.27 Python/3.8.8 Darwin/22.2.0 exe/x86_64 prompt/off command/iam.create-user",
  "requestParameters": {
    "userName": "backdoor"
  },
  "responseElements": {
    "user": {
      "path": "/",
      "userName": "backdoor",
      "userId": "AIDAZYLBWP4WK7AU2ECSX",
      "arn": "arn:aws:iam::670756667180:user/backdoor",
      "createDate": "Mar 11, 2023 9:33:46 PM"
    }
  },
  "requestID": "22d9e006-69c4-4d77-95da-03b91db24de1",
  "eventID": "ab300f85-abae-4fb0-8b32-7059a4b9a3c5",
  "readOnly": false,
  "eventType": "AwsApiCall",
  "managementEvent": true,
```

**Step 5 : What Permissions Were Associated with the Persistence Attempt?

To understand what permissions were associated with this attempt, I used the following command to enumerate the policies attached at the time:

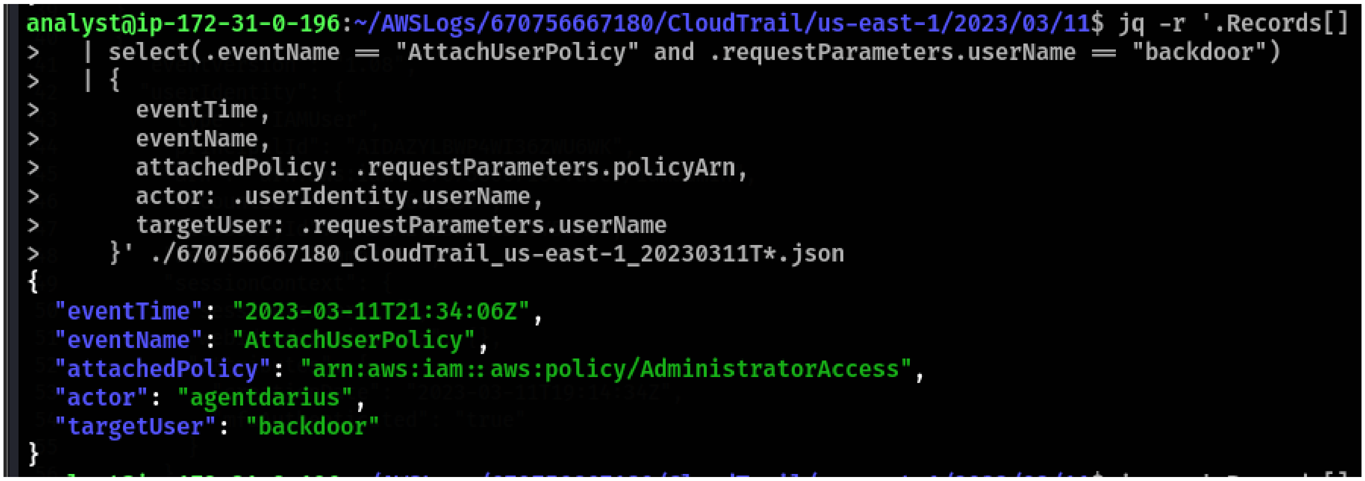
``

```
jq -r '.Records[]
  | select(.eventName == "AttachUserPolicy" and .requestParameters.userName ==
"backdoor")
  | {
      eventTime,
      eventName,
      attachedPolicy: .requestParameters.policyArn,
      actor: .userIdentity.userName,
      targetUser: .requestParameters.userName
    }' ./670756667180_CloudTrail_us-east-1_20230311T*.json
```

This helped validate that the attacker had access to `AdministratorAccess`, which includes permissions like:

- `iam:CreateUser`
- `iam:AttachUserPolicy`
- `iam:CreateAccessKey`

These permissions allowed the creation of the `backdoor` user laying the foundation for persistent, privileged access to the environment.



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(.eventName == "AttachUserPolicy" and .requestParameters.userName == "backdoor")
>   | {
>       eventTime,
>       eventName,
>       attachedPolicy: .requestParameters.policyArn,
>       actor: .userIdentity.userName,
>       targetUser: .requestParameters.userName
>     }' ./670756667180_CloudTrail_us-east-1_20230311T*.json
{
  "eventTime": "2023-03-11T21:34:06Z",
  "eventName": "AttachUserPolicy",
  "attachedPolicy": "arn:aws:iam::aws:policy/AdministratorAccess",
  "actor": "agentdarius",
  "targetUser": "backdoor"
}
```

## What else ????

Earlier, when I extracted all API operations initiated by `agentdarius`, I observed two key entries related to **S3 activity**:

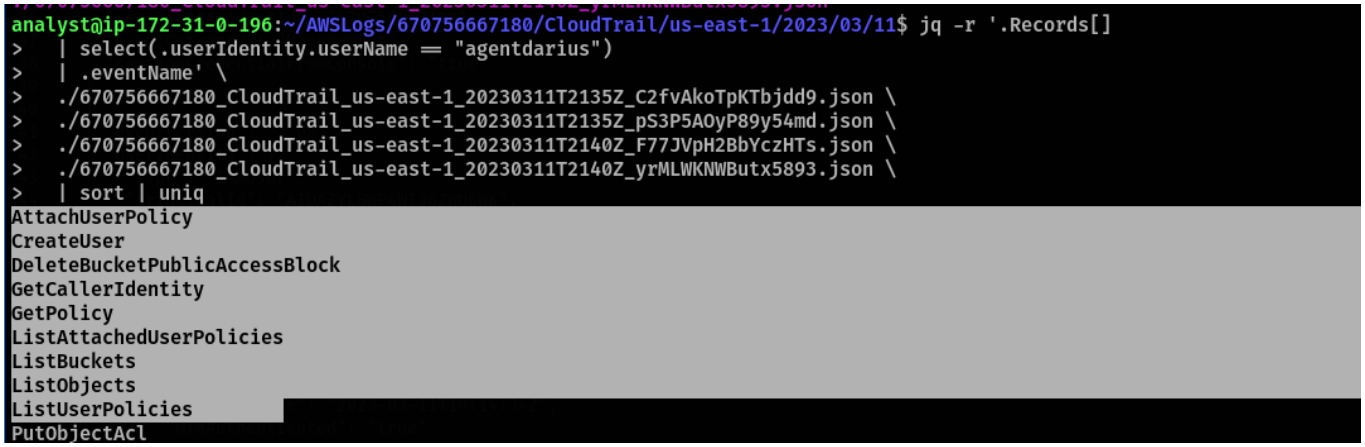
- `ListBuckets`
- `ListObjects`

- `DeleteBucketPublicAccessBlock`

These API operations clearly indicate that the attacker was performing **S3 enumeration**, likely to identify accessible buckets and inspect their contents.

- `ListBuckets` is used to retrieve a list of **all S3 buckets** within the AWS account.
- `ListObjects` allows listing the **files inside a specific bucket**, helping the attacker assess what kind of data is stored there.

This behavior typically follows IAM enumeration. Once an attacker knows they have sufficient permissions, they begin **probing data storage services** like S3 , which often contain logs, source code, credentials, or other sensitive information.



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(.userIdentity.userName == "agentdarius")
>   | .eventName' \
>   ./670756667180_CloudTrail_us-east-1_20230311T2135Z_C2fvAkoTpKTbjdd9.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2135Z_pS3P5AOyP89y54md.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2140Z_F77JVpH2BbYczHTs.json \
>   ./670756667180_CloudTrail_us-east-1_20230311T2140Z_yrMLWKNWButx5893.json \
>   | sort | uniq
AttachUserPolicy
CreateUser
DeleteBucketPublicAccessBlock
GetCallerIdentity
GetPolicy
ListAttachedUserPolicies
ListBuckets
ListObjects
ListUserPolicies
PutObjectAcl
```

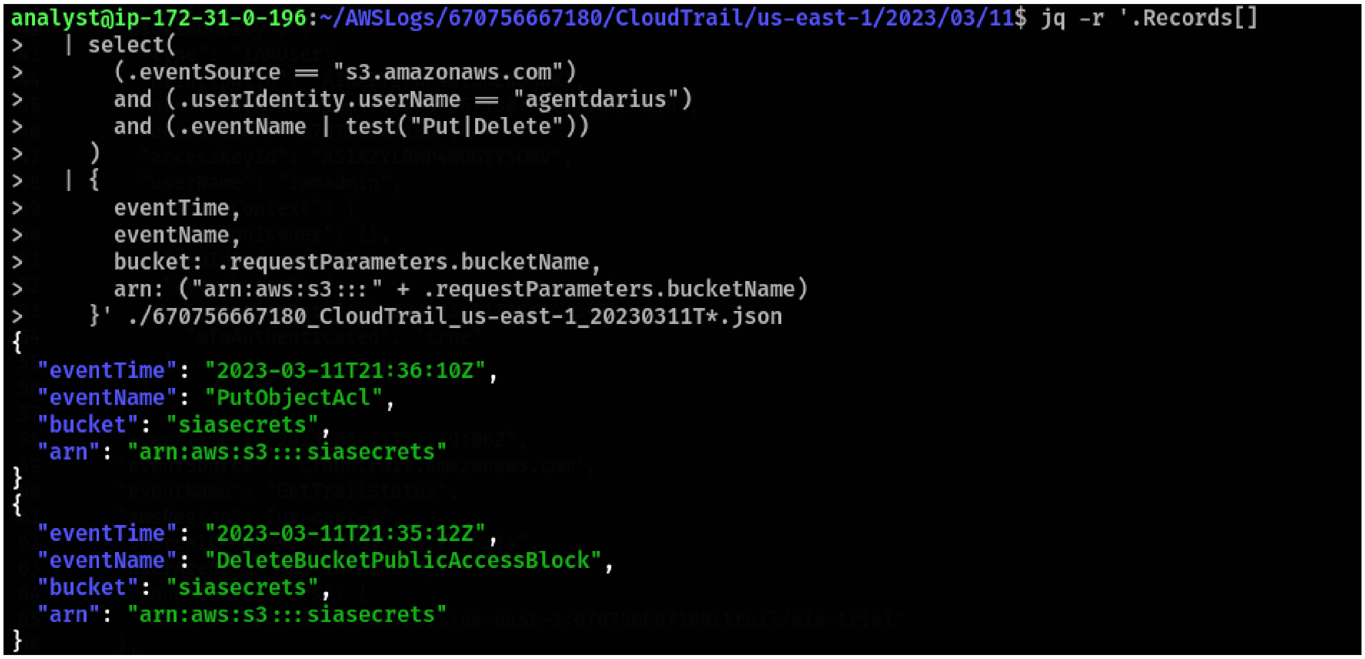
After confirming S3 enumeration activity with `ListBuckets` and `ListObjects`, the next step was to determine whether `agentdarius` performed **any tampering** , especially actions that modified bucket contents or permissions.

To uncover this, I searched for API operations like:

- `PutObjectAcl` – modifies access permissions on an object.
- `DeleteBucketPublicAccessBlock` – removes restrictions preventing public exposure.
- Any `Put*` or `Delete*` actions targeting S3 buckets.

```
jq -r '.Records[]
  | select(
      (.eventSource == "s3.amazonaws.com")
      and (.userIdentity.userName == "agentdarius")
      and (.eventName | test("Put|Delete"))
    )
  | {
      eventTime,
      eventName,
      bucket: .requestParameters.bucketName,
      arn: ("arn:aws:s3:::" + .requestParameters.bucketName)
    }' ./670756667180_CloudTrail_us-east-1_20230311T*.json
```

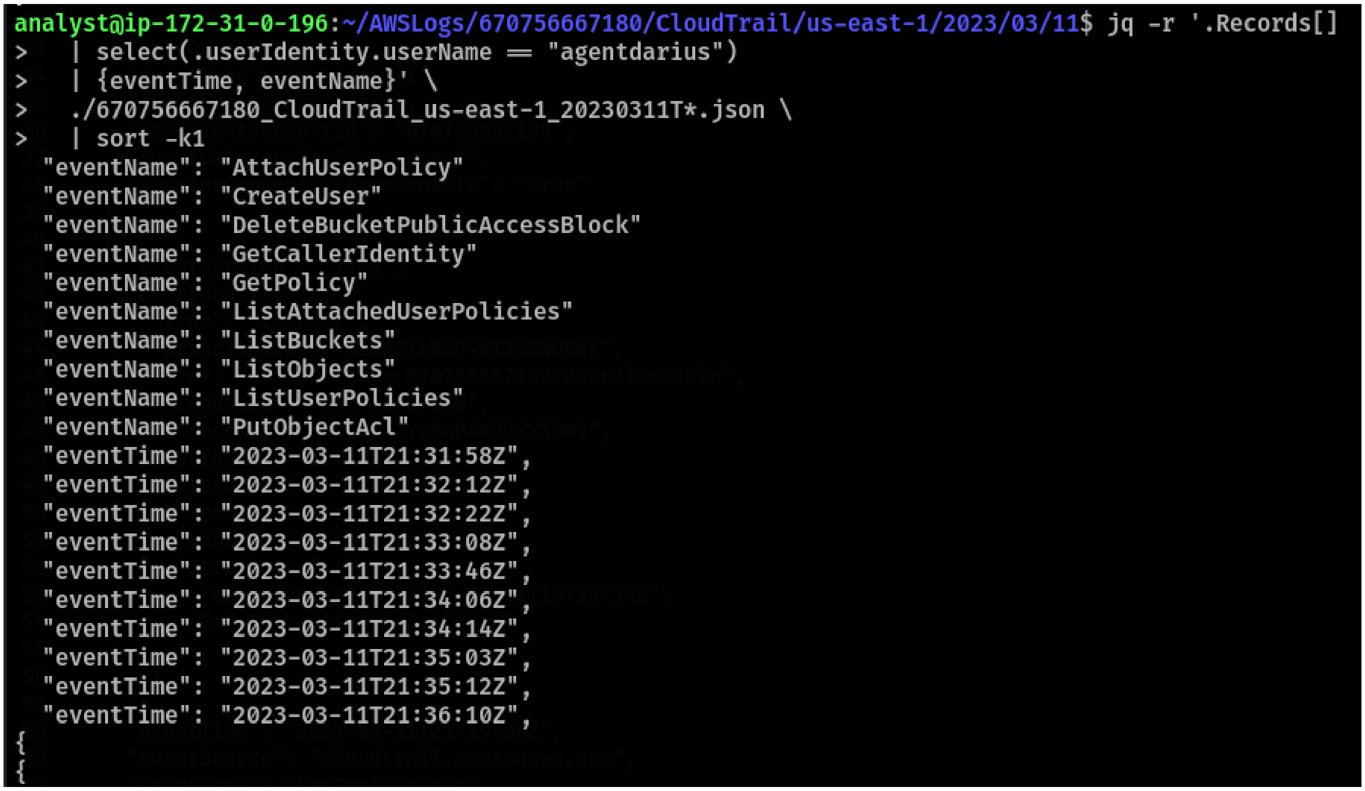
This gave me the **exact S3 bucket name and ARN** that `agentdarius` tampered with.
It confirmed that the attacker didn't just look around ,they took action. The use of `PutObjectAcl` or `DeleteBucketPublicAccessBlock` suggests an attempt to **modify access controls**, possibly to exfiltrate or publicly expose sensitive data.



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(
>       (.eventSource == "s3.amazonaws.com")
>       and (.userIdentity.userName == "agentdarius")
>       and (.eventName | test("Put|Delete"))
>     )
>   | {
>       eventTime,
>       eventName,
>       bucket: .requestParameters.bucketName,
>       arn: ("arn:aws:s3:::" + .requestParameters.bucketName)
>     }' ./670756667180_CloudTrail_us-east-1_20230311T*.json
{
  "eventTime": "2023-03-11T21:36:10Z",
  "eventName": "PutObjectAcl",
  "bucket": "siasecrets",
  "arn": "arn:aws:s3:::siasecrets"
}
{
  "eventTime": "2023-03-11T21:35:12Z",
  "eventName": "DeleteBucketPublicAccessBlock",
  "bucket": "siasecrets",
  "arn": "arn:aws:s3:::siasecrets"
}
```

After tracing `agentdarius` through IAM enumeration, privilege escalation, persistence creation, and S3 tampering, it was time to answer the final and most critical question:
How did the SIA agent expose his country's secrets?
` To get to the bottom of this, I sorted all the activity from agentdarius` chronologically using the command:

```
jq -r '.Records[]
| select(.userIdentity.userName == "agentdarius")
| {eventTime, eventName}' \
./670756667180_CloudTrail_us-east-1_20230311T*.json \
 | sort -k1
```



```
analyst@ip-172-31-0-196:~/AWSLogs/670756667180/CloudTrail/us-east-1/2023/03/11$ jq -r '.Records[]
>   | select(.userIdentity.userName == "agentdarius")
>   | {eventTime, eventName}' \
>   ./670756667180_CloudTrail_us-east-1_20230311T*.json \
>   | sort -k1
  "eventName": "AttachUserPolicy"
  "eventName": "CreateUser"
  "eventName": "DeleteBucketPublicAccessBlock"
  "eventName": "GetCallerIdentity"
  "eventName": "GetPolicy"
  "eventName": "ListAttachedUserPolicies"
  "eventName": "ListBuckets"
  "eventName": "ListObjects"
  "eventName": "ListUserPolicies"
  "eventName": "PutObjectAcl"
  "eventTime": "2023-03-11T21:31:58Z",
  "eventTime": "2023-03-11T21:32:12Z",
  "eventTime": "2023-03-11T21:32:22Z",
  "eventTime": "2023-03-11T21:33:08Z",
  "eventTime": "2023-03-11T21:33:46Z",
  "eventTime": "2023-03-11T21:34:06Z",
  "eventTime": "2023-03-11T21:34:14Z",
  "eventTime": "2023-03-11T21:35:03Z",
  "eventTime": "2023-03-11T21:35:12Z",
  "eventTime": "2023-03-11T21:36:10Z",
{
{
```

The last two API calls in the timeline were:

- `DeleteBucketPublicAccessBlock`
  This removed the guardrails that prevent public access to S3 buckets. Essentially, it opened the door for public visibility.
- `PutObjectAcl`
  This explicitly **granted public permissions** on individual objects ,likely the files containing sensitive or classified information.
  Together, these two actions formed a deliberate move to **exfiltrate and expose** the data.

## Conclusion

These were the two API calls that led to the exposure of secrets , a chilling but realistic demonstration of how misused IAM permissions can be weaponized in a cloud environment.

This wasn't just a walkthrough it was more like a deep dive into **how IAM-based attacks unfold in the cloud**.

Throughout this challenge, I tracked an attacker's movement via CloudTrail logs:

- Starting with **enumeration of users and permissions**
- Escalating privileges by attaching **AdministratorAccess**
- Creating a **backdoor user** ( `backdoor` ) for persistence

- **Modifying S3 permissions** to expose sensitive data

Each API call told a story and understanding the sequence helped reconstruct how the compromise occurred, step-by-step.

From an IR perspective, this drove home a few key lessons:

- **Cloud attackers don't need malware just permissions.**
- IAM changes are subtle but powerful; they deserve close monitoring.
- Tools like `jq` are essential to quickly triage and isolate malicious activity.

While I've reviewed logs in Splunk before, working directly with raw JSON made the story clearer. It sharpened my investigative mindset and reinforced that **identity is both a target and a weapon** in cloud breaches.

This was more than just practice ,it was a reminder that **incident responders must understand IAM as deeply as attackers do**.