AWS ECS Fargate security analysis

Jaume Boniquet Travila 01 June 2022

Abstract

Kubernetes (K8s) has been for several years the king of container orchestration. It distributes application workloads across clusters and automates container creation, management and governance. Kubernetes also allocates storage and volumes to running containers, provides automatic scaling, and works continuously to maintain the state of applications. It is well known for being a robust and secure environment (as long as it is deployed and constructed following some security principles, best practices, and coherent architecture design).

The process of securitization of standard K8s is out of the scope of this research. This research focuses on analyzing from an infosec and perspective, new adaptations of K8s that could potentially introduce some weaknesses to a mature software solution that was released about 8 years ago.

Some vendors have developed new alternatives based on the source code of K8s. One of the most noticeable ones is “Azure Kubernetes Service (AKS)” and “Amazon Elastic Container Service (ECS)”. For the sake of clarity and time, this research will focus on the latter platform, and to be even in more detail, to the relatively new implementation of ECS and ECS “*Fargate*” which is a fully managed orchestration platform provided by the vendor Amazon Web Services. (AWS).

Many vendors advertise that their new solutions are reinforced, more agile, easier to administer, cost effective and a plethora of advantages, but what about the new potential security issues that may be introduced in non-mature platforms compared to the horse of battle K8s ?

In this paper some attack scenarios will be devised and tested out, with the objective to determine any potential vulnerability, issue or room for improvement compared to the traditional K8s. Also it covers how to protect the infrastructure and objects created within ECS and finally, an additional section will cover some examples and scenarios on how a malicious attacker could leave a backdoor or a routine to access in a stealthy way. Where applicable, mitigation key points will be covered.

1. Introduction

Due to the complexity and similarities between the different types of orchestrators based on K8s, a comprehensive introduction and

definitions are required to fully provide meaning to the paper.

Not long ago Kubernetes (also known as K8s) was one of the most revolutionary Open Source solutions for container orchestration. Minded to orchestrate (automating deployment, scaling, and management of containerized applications.) Docker application containers are still heavily being used by the most demanding and modular applications developed today.

Kubernetes (K8s) basically eliminates many of the manual processes involved in deploying and scaling containerized applications, providing deployment speed, workload portability, simplification of provisioning resources, application development and delivery. Can also perfectly integrate with any kind of applications and cloud environments.

As described in the Abstract, this paper focuses on AWS ECS and ECS Fargate.
Fargate is not a Kubernetes distribution.
Fargate is an operational mode within Amazon Elastic Container Service (ECS) that abstracts container host clusters and servers away from the user of the service, meaning that there is no need to configure anything further down the infrastructure stack making the infrastructure management simple.

AWS ECS is different because it delivers more control over the infrastructure, but the trade-off is the added management that comes with it as it is not a fully managed service.

In short, this means that AWS ECS Fargate imposes some limitations not allowing some custom personalization but in exchange makes the deployment of containerized applications simpler, without complex infrastructure management.

2. Related work

Related work to the attack scenarios tested out and documented in this paper are scarce and many of them non-existent.
Notwithstanding, there are a few publications covering some security best practices in a low level non deep technical way, threat models and vendors attempting to sell some security software with their publications.

After additional research, no tools or proof of concepts were found capable of breaking or compromising the ECS Fargate security model.

3. EKS, ECS vs ECS Fargate

In order to shed some light on these confusing concepts and terminologies, a brief explanation of the three is below.

Amazon Elastic Container Service (ECS): Amazon's own container platform.

Amazon Elastic Kubernetes Service (EKS): Amazon's managed

Kubernetes platform based on Open source code.

AWS Fargate: Amazon own serverless container platform that works with ECS and EKS.

Amazon ECR: is an AWS managed container image registry storage service. Docker images can be pushed, pulled or managed.

Amazon ECS

- Launches ECS Tasks on ECS clusters

- Two types of instances provisioning:
    - EC2 Launch Type:
        - You must provision & maintain the infrastructure (the EC2 instances).

        - Each EC2 instance must run the ECS Agent to register in the ECS cluster

        - AWS takes care of starting/stopping containers as needed

    - Fargate Launch type:
        - You do not need to provision the EC2 infrastructure (but of course there are servers behind but it is transparent.

        - 100% serverless

        - You just need to create task definitions

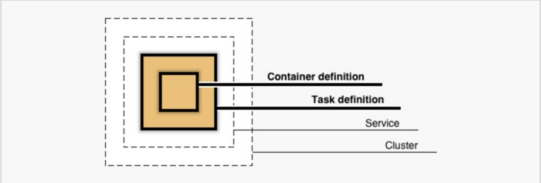
        - AWS runs ECS tasks according to the needed CPU/RAM and auto-scales tasks (by defining the amount you desire in the settings) automatically, without the need of provisioning or creating new EC2 instances.

        - Easier to manage and maintain than “EC2 Launch type”.

4. ECS Fargate Concepts and overview

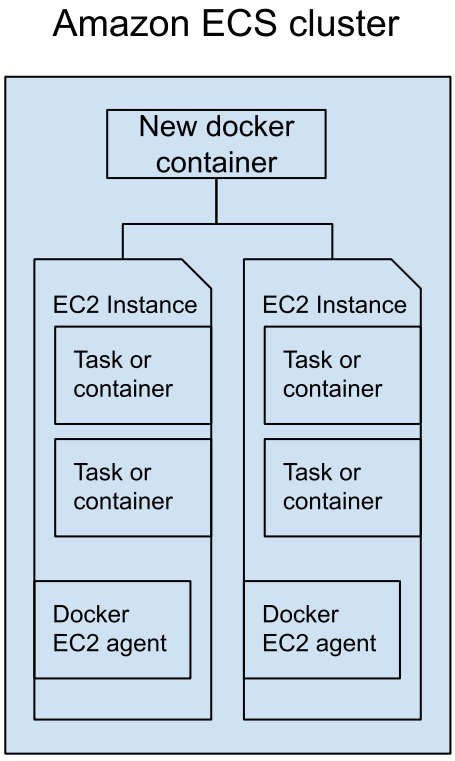
AWS ECS Fargate was released in 2017 to simplify the workflow involved in running containerized workloads. Fargate acts as an abstraction layer of the orchestrator ECS. With the AWS Fargate abstracted model, the underlying nodes are not visible, and also not searchable in the VPC (very likely they are EC2 instances).

ECS model diagram:



* For a detailed explanation of each element and equivalent term in a standard K8s orchestrator, check the "*Acronyms and definitions*" section of this paper.

ECS clusters overview:



A high-level summary of the steps involved in deploying a container image in Fargate are as follows:

1. Push the customized Docker image to Amazon Elastic Container Registry (ECR) or pull an image from a public image repository (https://hub.docker.com) or (https://gallery.ecr.aws/)
2. Create a task definition with the chosen container image along with the desired CPU, memory, and networking ports.
3. Create a Fargate cluster associated with a VPC and subnet which are used for routing the traffic to the workload.
4. (OPTIONAL) Launch an ALB and point the listener to the container port.
5. Finally, create a service definition with desired task count and associate it with the ALB (if any).

5. Advantages and disadvantages of AWS ECS Fargate

Pros:

- Easy to deploy any application and containers (low requirement of Know-How).
- No need to select the right server type.
- Pay only for the underlying resources needed by each container
- Enhanced security model

Cons:

- Lower control of your infrastructure
- Hard to predict the cost
- Less Customization

- For additional storage (more than 20gb, you need EFS which incurs in more costs)
- Service not available in all regions.

6. Types of attackers

Attackers typically fall into one of three areas:

- Amateur: Amateurs are curious individuals who carry out attacks just to "*see if it can be done.*"

- Expert: Experts attack under the auspices of scientific institutions and universities studying the technology.

- Professional: Professional attackers motivated for financial rewards or to obtain sensitive data and compromise a system.

7. Attacks

Before attempting to reproduce any test, make sure your aws-cli is updated to the latest version. Some commands do not work properly with older versions, even if recently updated.

6.1 Attack vector #1: Access to the VPC/Host

AWS Fargate launch type pre-provisions a fleet of EC2 instances within a VPC which are not accessible. By issuing the "*ecs execute-command*" it was possible to access the instances, but not to edit the settings of the underlying EC2 instances or VPC settings (actually is possible, but when the task is relaunched, all changes are gone).

This attack is deemed not successful, but with some additional research may leverage some weakness.

Attack vector mitigations: None

6.2 Attack vector #2: Attack on the Agent.

Each EC2 instance launched in the Fargate runs Amazon Linux 2 that has Docker runtime along with an agent that manages the two-way communication with the control plane. This agent is responsible for pulling the images from the registry and calling the Docker APIs to manage the lifecycle of each container defined in the task.

Some basic ELF debugging was performed, along with some basic binary scanners. At a first glance nothing relevant was observed to easily exploit any potential vulnerability existing in the binary files. A deeper binary debugging is required to analyze any potential hidden vulnerability among the binary files.

The files can be found in this folder, placed in the root:

```
/managed-agents
```

Attack vector mitigations: Due to the limitation of time, no deep reversing was able to be performed, so a feasible attack was not succeeded via this method.

6.3 Attack vector #3 Exec commands

Simply try to execute commands if you already have access to the infrastructure or into a compromised account.
In a traditional K8s deployment you can connect to the containers by using `”docker exec -it”` but in ECS is slightly different, as there are two options to do:

[1.] ECS Exec:

```
aws ecs execute-command \
    --region us-east-2 \
    --cluster <cluster_name> \
    --task <task_id> \
    --container <container_name> \
    --command "/bin/bash" \
    --interactive
```

[2.] Systems Manager:

```
aws --profile <local_profile> ssm
start-session --target <instance-id>
```

Remember that in order to be able to connect using an interactive shell to the containers, you need two requisites:

- Have permissions of:

```
"ssmmessages:CreateControl
Channel",
"ssmmessages:CreateDataC
hannel",
"ssmmessages:OpenControl
Channel",
"ssmmessages:OpenDataCh
annel"
```

- Enable the Execute Command in the Service Task Definition. This can not be done via web CLI, only via command line

  Check if ECS Exec is enabled by ensuring that the response matches this value: “`"enableExecuteCommand": true,”`:

```
aws ecs describe-tasks \
--cluster your-cluster-name \
--tasks your-task-id
```

  Enable ECS exec in a service:
  You can’t enable ECS Exec for existing tasks. It can only be enabled for new tasks.

```
aws ecs update-service \
  --cluster your-cluster-name \
  --service your-service-name
\
  --task-definition
your-td-family-name:Number \
  --force-new-deployment \
  --enable-execute-command
```

Exec to enter to an ECS container:

```
aws ecs execute-command
--cluster cluster-name \
    --task task-id \
    --container container-name
\
    --interactive \
    --command "/bin/sh"
```

SSM enter to an ECS container:

```
aws ssm start-session --target
ecs:ClusterName_Task_id_C
ontainer_ID
```

Attack vector mitigations: If for some reason an attacker has gained access to execute commands into a container, this will cause no major impact on the service itself because the attacker has no access to the ECS Task Definitions in order to permanently modify or alter the settings.

On the other hand if the container execution role has excessive permissions, those could be abused to jump into other vulnerable containers or reuse the Temporary Access Tokens of that role associated with the container to further escalate privileges or perform any other malicious actions.

6.4 Attack vector #4 Instance Metadata Attacks.

ECS Fargate Launch type:
After some testing, all attempts to call the metadata service via ECS Exec did not retrieve the session values on the regular Metadata Endpoint IMDSv1. It turns out that ECS had a different IMDSv1 called "*Task metadata endpoint*". It was possible to retrieve the Metadata Temporary Access Keys, which can also be abused as normal and regular IMDSv1. Details about the specific ECS metadata endpoint can be found in the appendix #12 or here.

```
curl
169.254.170.2$AWS_CONTAINER_
CREDENTIALS_RELATIVE_URI
```

ECS EC2 Launch type:
When using an IAM role with your tasks that are running on Amazon EC2 instances, the containers aren't prevented from accessing the credentials that are supplied to the Amazon EC2 instance profile. Via access to the container or by wrong web/proxy configurations, it is possible to steal the credentials metadata. These credentials can then be used in the AWS CLI or other means to make API calls as the IAM role.

Bear in mind that if you use those credentials outside the EC2, an alert will trigger in AWS Trusted Advisor !

```
curl
http://169.254.169.254/latest/meta-d
ata/
```

Attack vector mitigations:
To mitigate this attack, implement IMDSv2, to prevent external users from receiving credentials, allowing only application resources to receive them.

6.5 Attack vector #5 Remount, reuse ephemeral storage.

According to the documentation, the Ephemeral Storage is automatically assigned to all containers with a capacity of 20 GB and is encrypted. Textually it indicates "*The ephemeral storage is encrypted with an AES-256 encryption algorithm, which uses an AWS owned encryption key*."

During the tests performed it was not possible to determine if the ephemeral storage was really encrypted. After issuing some of the most common commands to determine this status, none was successful. As probably AWS uses a custom CMK, this could be transparent to the linux instances and do not reflect as it is not encrypted by the host but with an external abstraction layer.

```
df -h
lsblk -o
lsblk –all -T
dmsetup status
ecryptfs-verify --home
```

The second most common mounted filesystem is AWS EFS because it is compatible with both, "*EC2 Launch Type*" and *"Fargate Launch type*". There are some chances that the volumes used for data persistence are shared among tasks/containers and can be read/mounted within the same multi-AZ. As this is defined within the Task Definition, it is expected that hosts with the proper definition can access these volumes.

Attack vector mitigations:
It could not be irrefutably determined that the volumes are really encrypted. Also all attempts to remount volumes from other tasks (active and expired) did not succeed, so this attack scenario, with the set of tests performed, could not be exploited, so no mitigations are needed.

8. Hardening

Regardless of the attack tests results performed in this essay, there are some general best practices to make your environment a bit more secure. It consists of applying some specific settings. There is no silver bullet solution to protect against attacks or backdoors, but rather a set of measures that all together provides a robust ecosystem capable of stopping specialized attackers. Some of these best practices are as follows:

1. Follow the AWS ECS security best practices The information can be found here.

2. Implement IMDSv2 to prevent attacks related to stealing the instances metadata Temporary Access keys, migrate to IMDSv2 and deny the usage of the old version IMDSv1.
   More details can be found in the appendix #9 or by clicking on this site or on this other site.

3. Do not assign VPC with public access/routing to the Task Definitions. Instead use a

LoadBalancer (ALB) in front.

4. Recurrent infra checks with IaC:
   If your infrastructure is deployed as code (IaC), it is highly recommended to run at least a daily check against the infrastructure in order to verify that no new malicious code or by mistake, any changes were introduced.
   Please note that such a solution will only detect changes to the already existing infrastructure that was created by the IaC solution. If an attacker creates new resources, those will not be scanned nor detected by the IaC run/validate/init/plan/apply commands.

5. Deny access to the Metadata Temporary Access Keys:
   Restrict IMDSv1 availability by locking down the metadata endpoint so it is only accessible to specific O.S. users. On Linux machines, there are two ways of achieving it:

```
ip-lockdown 169.254.169.254 root

ip-lockdown 169.254.170.2 root
```

or

```
iptables -A OUTPUT -m owner ! --uid-owner root -d 169.254.169.254 -j DROP

iptables -A OUTPUT -m owner ! --uid-owner root -d 169.254.170.2 -j DROP
```

An additional step to prevent containers run by tasks that use the "*awsvpc*" network mode, from accessing the credential information supplied to the Amazon EC2 instance profile, while still allowing the permissions that are provided by the task role, set the:

```
ECS_AWSVPC_BLOCK_IMDS
```

agent configuration variable to "*true*" in the agent configuration file and restart the agent.

For additional details on alternative environment variables to protect access, please check the appendix #13 or click here.

Note: During the tests performed, for some unknown reason, this environment variable protection does not work with ECS Fargate launch type instances !!

6. Platform version.
   Use the latest platform version. If AWS releases a new version, it will not automatically migrate, unless a "*update-service*" command is issued. As of today, the latest version is (1.4.0).

7. Exec command
   Regular users shall not be able to log-in (get a remote interactive shell) to the containers in production environments. Proper mechanisms should be put in place for monitoring, debugging, and log analysis.

   It is suggested to tag tasks and create IAM policies by specifying the proper conditions on those tags.

```
...
"Action": [
"ecs:ExecuteCommand"
            ],
            "Condition": {
                "StringEquals": {

"aws:ResourceTag/tag-key":
"tag-value"",
                    "StringEquals": {

"ecs:container-name":
"<container_name>"
                    }
                }
            },
"Resource":"arn:aws:ecs:<reg
ion>:<aws_account_id>:clust
er/<cluster_name>"
 ...
```

8. Strict permissions, segregation and naming convention
   Tasks roles shall be segregated as much as possible for the purpose of the job. For example, if we assign the same Security Group (with permissions to S3, DynamoDB, etc) to several other services (via Task Definitions), we may be risking giving too many permissions to a container that could be potentially exploited by an attacker and then he could abuse those roles and permissions.

   Plan a good strategy for creating the roles and their corresponding permissions. As per any specific and individual needs, strategies may different one from each other, but a generalistic good example naming convention is aso detailed for each ECS role type.

   Task role: permissions granted in this IAM role are assumed by the containers running in the task.

   Applicable to both, the EC2 Launch Type and ECS Fargate Launch Type, this role is used to allow each task/container to have a specific or different role as may be needed by the requirements of the application or service running inside the container. These settings are defined in the "*Task Definition*".

   Make sure NOT to grant "*Task role*" permissions to Secrets Manager / SSM or other broad permissions as they can be abused if by some technique the container is compromised by an attacker.

An example of a good role naming convention is:

ecs-<ENVIRONMENT>-<SERVICE_NAME>-task-role

Task Execution IAM role: role required to perform some tasks on your behalf such as:

Makes API calls to ECS service, Sends task/container logs to CloudWatch Events, Pulls docker images from the AWS ECR repository, Reference credentials data from Secrets Manager or SSM Parameter Store

Try to give the least permissions as possible.

- *AWS ECS Fargate Launch types:* Ensure to make use of the minimal list of permissions used in the managed policy

“AmazonECSTaskExecutionRolePolicy”

- *AWS ECS EC2 Launch types*, ensure to make use of the minimal list of permissions used in the managed policy

“AmazonEC2ContainerServiceforEC2Role”

Some common additional policies attached along with the previous roles could be SecretsManager, DataDog, CloudWatch, etc.

An example of a good role naming convention is:

ecs-<ENVIRONMENT>-task-execution-role

9. Sensitive data in Task Definitions
   ECS task definitions are metadata in JSON format to tell ECS how to run a Docker container.
   Some of the information it may contain is: image name, port binding for container and host, RAM & CPU, environment variables, network details, IAM roles, Logging setup, etc.

   Do not put sensitive environment variables, don’t use public not reputable docker images, and unnecessary ports mapping.

10. VPC and Security groups (During the creation of a Service or task definition)

    a. Ensure “**Auto-assign public IP** is “*DISABLED*”

on all services.

b. When setting up an ECS in "*awsvpc*" network mode, ensure not to attach to the service public facing subnets.

11. LB's secure connections and Security Groups
If Target Groups towards LoadBalancers are attached to ECS services, ensure those are using secure and encrypted connections and non plain protocols such as HTTP. For example implies the usage of MTLS between the applications and the Load Balancers (which likely requires modifying the source code of the applications or to set up a mesh).

Try not to directly expose open ports accessible from the internet to the services/containers during the ECS Service creation phase.

12. Protect the containers by granting read-only permissions to the root folder. Set "*Read only root file system*" to "*true*" inside:

task definition / <container_name> / containers definitions / storage and logging

13. By default ECS logs the invocation command along with the user that invoked it will be logged in AWS CloudTrail. To log all commands and their outputs inside the shell session, enable in SSM the sessions logging to CloudWatch or S3, under:

SSM / Node Management / Session Manager / Preferences

14. Assets tagging
An SCP (Service Control Policy via AWS Organizations) can be forged to protect from editing, deletion or any other sensitive API call that is not supposed to be common.

Tag all ECS clusters, tasks, tasks definitions, containers, IAM roles, and ECS services security groups to benefit from such SCP protection. An example of such a policy was created and tested. For the sake of clarity, just a few service API calls were documented, but additional ones can be added as required per each business case and specific needs.

```
…
"ecs:UpdateService",
"eks:TagResource",
“eks:UntagResource"
      ],
      "Resource": [
        "*"
      ],
      "Condition": {
        "StringEquals": {
"aws:ResourceTag/<key>": [
            "<value>"
```

```
            ]
          }
        }
    },
…
```

15. Limiting access to the Start Session action
    While starting SSM sessions on your container outside of ECS Exec is possible, this could potentially result in the sessions not being logged. Sessions started outside of ECS Exec also count against the session quota. Deny the `"ssm:start-session"` action directly for your Amazon ECS tasks using an IAM policy attached to the users, not to the container task roles.

    Policy example:

    ```
    "Version": "2012-10-17",
    "Statement": [ { "Effect":
    "Deny", "Action":
    "ssm:StartSession",
    "Resource":
    "arn:aws:ecs:*:111122223333:
    task/cluster-name/*" } ]
    ```

16. Implement a service connectivity mesh and service encryption
    A service mesh is a logical boundary for network traffic between the ECS services that reside within it. The objective is to prevent service communication among them. In a standard K8s, EKS, and ECS deployment, all containers or services can communicate among them, with no restrictions. If one service or container is compromised by an attacker, it makes the task of jumping across hosts (containers) easier.
    More details about service mesh here, and about AWS mesh solutions here and here.
    To enable TLS between connected services within a mesh, follow this tutorial.

17. Shared responsibility model
    Remember to follow the security best practices for Docker at all times. AWS ECR is not invulnerable and its design principles are based on the shared responsibility model.
    In short, this means that the service users are responsible for patching, hardening, the container, the network, data, etc. Additional details can be found here.
    Some of the most noticeable securitization examples that the vendor will not perform and is up to the service users are:

    - Do not run services inside pods with the root user. Then, you can run the application as a non-root user by using "*USER* " in Dockerfile. You must build the docker image with a user and upload it to ECR before you run the ECS tasks.

- Scan with a Vulnerability Scanner all packages and dependencies (ECR Scan, Trivy, etc)

- Do not use public untrusty docker images

9. Backdooring

There are many ways an attacker or an insider employee may leave a backdoor or some sort of access to later access the cluster or services. In some cases it is either possible to cover his tracks or to gain access to the infrastructure at some point. Below it is described some potential features and techniques that can be used to leave latent access.

1. Create an ECS service with a rogue task definition that includes an inline policy with administrator privileges. Then attach this task definition to the Task/Pod, run the containers with a reverse Netcat shell scheduled daily to reverse connect to another server (outside the AWS infrastructure) that the attacker has control. Then the attacker will have access to the container, which has assigned an execution role with excessive permissions, which can be used (for example by reusing the Temporary Access Tokens) to execute any commands to the AWS infrastructure with Administrator privileges.

   The more services are in the production environment, the more unlikely it is for IT administrators to find out who and what is the purpose of this service and can remain active for a long time.

2. In Systems Manager (SSM), surf to "*Sessions Manager*", then to the tab "*Preferences*", section "*Shell Profiles*" and add some rogue commands to be executed in the shell profile when connecting to hosts. More info here.

   After some testing, this property appears to only work in Standard ECS tasks running in EC2 containers or EKS instances, not in ECS Fargate launch types (when connecting via ECS Exec).

   Similar backdoors can be left in the ECS Task Definition, under the section "*Containers Definition / command*" where a custom command can be entered only when the task/container is launched. Remember that Task Definitions have versions and this change can be easily tracked down via CloudTrail/CloudWatch.
   Also note that in order to add any of these commands to the settings, IAM permissions of "*AmazonECS_FullAccess*" and "*AdministratorAccess*" are required.

3. Editing or adding backdoors to any already existing assets created by Terraform or CloudFormation stacks (IaS) will be likely detected or replaced in the next deployment or execution.

   Any assets that were not created by any IaC solution will not be managed, alerted nor detected.
   Being said that, if an attacker has IAM permissions to create a Lambda, it is possible to create a simple python code that is executed recurrently to perform any innocuous action. For example launching an ECS service which uses an already compromised container image from the public ECR repositories. This persistent hack is likely not to be easily detected for a long time.

10. Conclusions

After an intensive testing over the platform, several attack vectors were covered, with not much success.
The attack vectors “6.1 Attack vector #1: Access to the VPC/Host”, “6.2 Attack vector #2: Attack on the Agent”, “6.5 Attack vector #5 Remount, reuse ephemeral storage.” did not provide good results as not effective weaknesses could be found using the techniques described in this paper. It is not discarded that by using additional methodologies for testing, these may be exploited somehow.

On the other hand, the attack vectors “6.3 Attack vector #3 Exec commands” and
“6.4 Attack vector #4 Instance Metadata Attacks.” were quite successful. Both attack vectors require access to the existing infrastructure somehow, either because an attacker has gained access to a container, or by having credentials or API keys with enough IAM privileges to execute these actions. Wild exploitation of these attack vectors is not easy and not within the reach of a non highly skilled attacker. Usually these kinds of attacks could be exploited by insiders or when AWS API keys are leaked.

Regarding the Hardening section, a comprehensive set of best practices were found and documented. Not applying them does not mean that the environment or containers will be vulnerable, but in the author’s opinion of this research, it will slow down any potential attacker (insider or outsider) and is based on the principle of zero-trust security model. The author is a big fan of the “*Onion Theory of Data Security Layers*” which consists of multiple defensive layers that support each other.

Being said that, by applying all the security recommendations listed above, even if an attacker gets access to AWS API Keys, to a container, to an IAM role, etc, the impact of the malicious actions is greatly reduced and in some cases, early detection of a successful attack or backdoor placement.

Finally, regarding the section Backdooring, real interesting scenarios were devised and tested. Three scenarios were covered and in case of achievement, these backdoors would be hard to detect even for the most trained IT staff. They are not high-tech backdoors as could be by loading customized pieces of code in the shape of LKM's in a Linux kernel module, but given that the AWS infrastructure is so big, small changes to it may go undercover. It is implicit that in order to place any backdoors, it does not matter if it is AWS, Linux, or any kind of other system, some sort of access and privileges are required in order to place them, and in this case, it is not an exception.

11. Further research

Additional research is needed to figure out alternative ways to exploit IMDSv2 metadata temporary keys.

Also, during the elaboration of the hardening guidelines, when implementing the environment variable "ECS_AWSVPC_BLOCK_IMDS", it did not prevent calls to the metadata service on ECS Fargate Launch type instances. Additional research is needed to find alternative ways to block them.

Due to the limitation on time, not enough resources could be assigned to decompile and properly debug the binary files (Systems manager agents) used by AWS to manage the connectivity between docker containers and ECR. A large amount of time is required to debug them and attempt to find any potential vulnerabilities.

Regarding the section Backdooring, there was no information at all publicly documented, so this looks like an area where additional research may be performed in the future.

New attack vectors may exponentially grow as the ECS service and features are added over time by the vendor. This means that in the near future, potential new findings are to be found in further research.

12. Acknowledgements

The author would like to thank the reviewers for their helpful comments, critics and advice. Finally, thanks to my family and friends who helped me in ways unknown to them.

13. Acronyms and definitions

Cluster: a group or bunch of nodes that executes containerized applications
In Fargate, this term is also called "*Cluster*".

Node: are comprised of virtual or physical machines in a cluster; these "worker" machines contains all the necessary to run the application containers
In Fargate, this term is equal to "*Node*" but is transparently managed by the vendor.

Pod: a logical group of one or more containers running together in the same space/cluster
In Fargate, this term is equal to "*Task*".

Replica set / deployment: Set of pods/tasks that hosts an application.
In Fargate, this term is equal to "*Service*".

Kubernetes API: the application that serves Kubernetes functionality through a RESTful interface and stores the state of the cluster

Kubernetes Control Plane: maintains a record of all of the Kubernetes Objects in the system, and runs continuous control loops to manage those objects' state

Master: Every cluster has a master node, as well as several "worker" nodes. The master includes three critical processes for managing the state of your cluster: kube-apiserver, kube-controller-manager and kube-scheduler.

kubectl: command line interface for managing operations on K8s clusters and API.

Minikube: tool to locally run K8s for testing and development purposes.

Volume: a directory of data residing in a pod and can be accessed by any container running inside that pod.

Persistent volume: a directory of data which content persists after the pods and containers life.

Ephemeral volume: underlying hardware physically attached to the host for the instance used for temporary and not persistent data storage.

14. References

[1.]https://enterprisersproject.com/sites/default/files/kubernetes_glossary.pdf

[2.]https://docs.aws.amazon.com/AmazonECS/latest/developerguide/Welcome.html

[3.]https://docs.aws.amazon.com/AmazonECR/latest/public/public-repositories.html#public-repository-concepts

[4.]https://docs.aws.amazon.com/AmazonECS/latest/userguide/ecs-exec.html

[5.]https://aws.amazon.com/blogs/containers/new-using-amazon-ecs-exec-access-your-containers-fargate-ec2/

[6.]https://www.skyhighsecurity.com/en-us/about/newsroom/blogs/threat-research/how-an-attacker-could-use-instance-metadata-to-breach-aws.html

[7.]https://hackingthe.cloud/aws/exploitation/ec2-metadata-ssrf/

[8.]https://attack.mitre.org/matrices/enterprise/cloud/

[9.]https://aws.amazon.com/es/blogs/security/defense-in-depth-open-firewalls-reverse-proxies-ssrf-vulnerabilities-ec2-instance-metadata-service/

[10.]https://docs.aws.amazon.com/cli/latest/reference/ec2/modify-instance-metadata-options.html

[11.]https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-metadata-endpoint.html

[12.]https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-metadata-endpoint.html

[13.]https://docs.aws.amazon.com/AmazonECS/latest/developerguide/task-iam-roles.html

[14.]https://hackingthe.cloud/aws/exploitation/ec2-metadata-ssrf/

[15.]https://www.ernestchiang.com/en/posts/2021/using-amazon-ecs-exec/

[16.]https://docs.aws.amazon.com/AmazonECS/latest/bestpracticesguide/security.html

[17.]https://aws.amazon.com/about-aws/whats-new/2021/04/amazon-ecs-aws-fargate-configure-size-ephemeral-storage-tasks/?nc1=h_ls