Security for
Amazon Elastic Kubernetes Service
Workloads

By: Aman Kumar Singh

Table of Content

Introduction

Security can help secure the system from being misused. This document gives the information about security related tools and technology that can secure your Kubernetes cluster and containers. The document covers some of the well known open-source tools like Calico and Falco in addition to AWS specific services like IAM and GuardDuty. A brief description of these tools are mentioned later in the Problems and Solution section of the document.

The main motto of this document is to enable Field teams (for example: AWS Solutions Architect) to understand and show various ways to secure a Kubernetes cluster and containers. Therefore, the document is to prepare SAs for customer conversations around specific scenarios on L300.

Security of Containers vs Virtual Machine

Virtual machines are highly isolated between each other. Since each virtual machine has its own hypervisor and they don't share any resources with each other, security compromise in one virtual machine don't affect other virtual machines. For example, if an attacker successfully breaches a system, they won't be able to manipulate other Virtual Machines. In case of containers, the attack surface is less as each container runs only a single application. Vulnerabilities in containers are most likely caused by the application running in them, which is also true in case of virtual machines. But containers share an underlying operating system. They're isolated from each other by a kernel, not a hypervisor. If an attacker compromises a container, they have a better chance of affecting the other containers than they do VMs in a similar scenario.

Problem statement

Isolating or securing containers from each other play a vital role in the system's security. This could be done using a tool like Calico which is a open source networking and network security solution for containers, virtual machines, and native host-based workloads. A proper access control is required when accessing AWS Services from a Kubernetes pod. AWS IAM provides this by setting and managing fine grained access control to your workloads on AWS. Apart from this, it is always beneficial to have a threat detection system to detect malicious activity in your Kubernetes cluster. An AWS service GuardDuty and an open source tool Falco can be integrated with Kubernetes to solve this purpose.

Solutions

We saw the problems and some tools that can solve these problems. Therefore, this project explains the reasons, benefits and drawback of using these tools/services. Following are various sections of the document.

- EKS Cluster + Workloads : Demonstrates the process of deploying an EKS cluster with sample workload.
- IAM Roles for pods : Demonstrates how IAM can be used to provide or restrict access to a pod.
- Network controls and Calico : Explains the use of Calico and some sample network rules for Kubernetes cluster.
- GuardDuty EKS Protection : Demonstrates, how GuardDuty can be used to protect EKS cluster.
- Falco installed on compute nodes: Gives a overview on differences between Falco and GuardDuty. Additionally, this section demonstrates on how to customize threat detection rules based on your requirement.

Chapter 0 (Prerequisite) : Deploying an EKS Cluster and sample workloads

To move forward with implementations of various tools and workloads on EKS cluster, first we need to deploy a cluster. This chapter discusses about deploying a EKS cluster along with a sample Nginx pod.

Deploying an EKS cluster:

We deploy an EKS cluster and add node group to them. There are many ways to do so, but I suggest some of the following method:
1) AWS Console - https://docs.aws.amazon.com/eks/latest/userguide/create-cluster.html
2) eksctl - https://eksctl.io/
3) AWS Cloudformation
4) Terraform

A node group with one node should be fine as of now to handle the load. [Recommended t3.large (2 CPU, 8 GB Memory) and 50 GB storage]

Deploying services to the cluster.

The user then connects to the cluster. On how to connect to the cluster, follow this document - https://docs.aws.amazon.com/eks/latest/userguide/create-kubeconfig.html

Make sure you have kubectl installed before proceeding to the next step. On more information to install kubectl follow this link - https://kubernetes.io/docs/tasks/tools/

Deploy a sample microservice to your EKS cluster using the following step.

1) Deploy a Nignx pod using the command

```
kubectl run nginx --image=nginx
```

2) Expose the pod as a Load Balancer service using the following command. We are using LoadBalancer because we want to access the service via the internet/outside world.

```
kubectl expose po nginx --port=80 --type=LoadBalancer
```

3) To get the link of the deployed service run the command:

```
kubectl get svc -A
```

4) In output, select the row with column values as nginx in NAME, default in NAMESPACE and LoadBalancer in TYPE. For the selected row, check for column entry EXTERNAL-IP. This is the Load

Balancer URL. Copy this URL and paste it on your browser. (Example URL: somethingsomething.us-east-1.elb.amazonaws.com)
For example: In the image below, select the load balancer under EXTERNAL-IP starting with *acb6567....com*

5) If you see the webpage with text “Welcome to Nginx!” we successfully deployed the pod.

Chapter 1 : IAM for Pod Security

Now let's start with security for pods. This chapter aims to secure your pod and AWS services by restricting the interaction between them.

Prerequisites

I recommend going through the following links before proceeding with the document:

1) Kubernetes Service Account - https://kubernetes.io/docs/tasks/configure-pod-container/configure-service-account/
2) Amazon S3 - https://aws.amazon.com/s3/
3) IAM Roles - https://docs.aws.amazon.com/IAM/latest/UserGuide/id_roles.html

Why IAM role?

IAM roles can be added to a Kubernetes service account. Only pods that use that service account have access to those permissions. We show the same here in this document. To show the behavior of pods based on the attached IAM role, we use an AWS Service called Amazon Simple Storage Service (S3).

IAM Role implementation

We are using S3 service to show the IAM role implementation. We will try accessing S3 from our pod with and without IAM role attached to it and see the result.

a) Create S3 resources

- We create a s3 bucket as described here - https://docs.aws.amazon.com/AmazonS3/latest/userguide/create-bucket-overview.html
- We then create a file named confidential_data.txt in it (the file can be empty).

b) Creating AWS Roles and Policies

We then create required roles to give K8s cluster access to AWS services. We create and attach a suitable policy to access S3 to get the files from the bucket. Follow this link to create a role and attach policies to it.
- https://docs.aws.amazon.com/IAM/latest/UserGuide/access_policies_job-functions_create-policies.html.
Attach the following policy to the role.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "VisualEditor0",
            "Effect": "Allow",
            "Action": [
                "s3:ListStorageLensConfigurations",
                "s3:ListAccessPointsForObjectLambda",
                "s3:GetAccessPoint",
                "s3:PutAccountPublicAccessBlock",
                "s3:GetAccountPublicAccessBlock",
                "s3:ListAllMyBuckets",
                "s3:ListAccessPoints",
                "s3:PutAccessPointPublicAccessBlock",
                "s3:ListJobs",
                "s3:PutStorageLensConfiguration",
                "s3:ListMultiRegionAccessPoints",
                "s3:CreateJob"
            ],
            "Resource": "*"
        },
        {
            "Sid": "VisualEditor1",
            "Effect": "Allow",
            "Action": "s3:*",
            "Resource": [
                "arn:aws:s3:::*/*",
            ]
        }
    ]
}
```

c) Creating K8s Resources

Without IAM role.

IAM roles are required to access AWS Services. In case you want to restrict access to AWS services then a service account with no access to AWS Service can be created. Here we create a Service Account in K8s that has no defined access to S3 bucket.

Run the files for :
Namespace - https://github.com/aman-2812/EKS-Security/blob/main/pod-iam/namespace.yaml
Service Account without IAM - https://github.com/aman-2812/EKS-Security/blob/main/pod-iam/sa-without-iam.yaml
Deployment - https://github.com/aman-2812/EKS-Security/blob/main/pod-iam/deployment.yaml

Using `kubectl apply -f filename` command.

After deploying the workload (K8s deployment) and attaching the above mentioned service account to the deployment, we check the pod logs and we get Access denied error as shown in the image below.

Run Command:

```
kubectl logs eks-sample-linux-deployment-5664cd7489-dqqp8 -n eks-sample-app -
c my-aws-cli
```

The output is:

```
An error occurred (AccessDenied) when calling the ListObjectsV2 operation: Access Denied
```

With IAM role.

1) We create a Service Account from a following configuration in K8s that has access to S3 bucket by attaching appropriate role to Service Account.

Service Account with S3 Access - https://github.com/aman-2812/EKS-Security/blob/main/pod-iam/sa-with-iam.yaml

2) We deploy the workload (K8s deployment) and attach this service account to the deployment. Post deployment we check the pods and we see files in the S3 bucket as shown below.

Run Command:

```
kubectl logs eks-sample-linux-deployment-5664cd7489-qhvxc -n eks-sample-app -
c my-as-cli
```

The output is:

```
2022-08-19 06:01:09          0 confidential_data.txt
```

Chapter 2 : Calico

Prerequisites

I recommend it to go through the following topics before proceeding with the document:

- Kubernetes Namespace - https://kubernetes.io/docs/concepts/overview/working-with-objects/namespaces/
- Kubernetes Service - https://kubernetes.io/docs/concepts/services-networking/service/

What is Calico?

Calico is an open source networking and network security solution for containers, virtual machines, and native host-based workloads. Calico supports a broad range of platforms including Kubernetes, OpenShift, Mirantis Kubernetes Engine (MKE), OpenStack, and bare metal services.

Why Calico?

Calico Open Source was born out of this project and has grown to be the most widely adopted solution for container networking and security, powering 2M+ nodes daily across 166 countries. Other benefits of using calico is Performance, ability to scale, full Kubernetes network policy support, various choice of dataplanes, Interoperability etc. The detailed explaination is mentioned here - https://projectcalico.docs.tigera.io/about/about-calico.

Why should you invest your time on calico?

Calico provides a rich network security model that allows operators and developers to declare intent-based network security policy that is automatically rendered into distributed firewall rules across a cluster of containers, VMs, and/or servers.

Calico enables Kubernetes workloads and non-Kubernetes or legacy workloads to communicate seamlessly and securely. All workloads have the same network policy model, so the only traffic that may flow is the traffic you expect to flow.

Calico Cloud builds on top of open source Calico to provide Kubernetes security and observability features on the cloud (AWS EKS). Also, being open source, it has a very active development community.

Problems and Solutions

Features of Network Policy: If we choose to go with the default Kubernetes Network policy, following are the possibilities:

- Policies are limited to a single environment and are applied only to pods marked with labels.
- You can only apply rules to pods, environments, or subnets.
- Rules can only contain protocols, numerical ports, or named ports.

Features with Calico: With Calico following are the additional benefits:

- Policies can be applied to pods, containers, virtual machines, or interfaces.
- Rules can contain a specific action (such as restriction, permission, or logging).
- Rules can contain ports, port ranges, protocols, HTTP/ICMP attributes, IPs, subnets, or selectors for nodes (such as hosts or environments).
- Interoperability with current non-K8s workloads.
- Full Kubernetes network policy support.
- Very active development community as calico is open source with 200+ contributors across a broad range of companies.

Deploying Calico on EKS

Follow the steps mentioned in this link, https://projectcalico.docs.tigera.io/getting-started/kubernetes/managed-public-cloud/eks to deploy Calico on EKS cluster. Delete the nodes before deploying calico or deploy a cluster without nodes and add nodes post Calico deployment.

Applying Calico Network Policy and Securing container communication using Network Policies

1) Deny all access

- First we deploy a sample nginx pod using the following command

```
kubectl run nginx —image=nginx
```

Post this run the command `kubectl get po -A` to see the deployed pod.

```
NAMESPACE         NAME                                       READY   STATUS    RESTARTS        AGE
calico-system     calico-kube-controllers-56b6ddd94f-l6swj   1/1     Running   0               99m
calico-system     calico-node-cmq8m                          1/1     Running   0               109m
calico-system     calico-node-gz9gh                          1/1     Running   0               108m
calico-system     calico-typha-6497d7d7c6-ggsxj              1/1     Running   1 (108m ago)    108m
calico-system     csi-node-driver-jn77h                      2/2     Running   0               109m
calico-system     csi-node-driver-sdq2r                      2/2     Running   0               108m
default           nginx                                      1/1     Running   0               37s
kube-system       coredns-7f5998f4c-rvpnm                    1/1     Running   0               99m
kube-system       coredns-7f5998f4c-xtnx5                    1/1     Running   0               99m
kube-system       kube-proxy-jbgpd                           1/1     Running   0               109m
kube-system       kube-proxy-qlljd                           1/1     Running   0               108m
tigera-operator   tigera-operator-6f669b6c4f-6v2vj           1/1     Running   0               99m
```

- Expose pod to a LoadBalancer using the command:

```
kubectl expose po nginx —port=80 —type=LoadBalancer
```

Once done run the command `kubectl get svc -A` to see the services. There you see nginx with a LoadBalancer attached as shown in the figure below.

```
NAMESPACE       NAME                              TYPE           CLUSTER-IP       EXTERNAL-IP                                                              PORT(S)
AGE
calico-system   calico-kube-controllers-metrics   ClusterIP      10.100.241.178   <none>                                                                   9094/TCP
100m
calico-system   calico-typha                      ClusterIP      10.100.128.24    <none>                                                                   5473/TCP
146m
default         kubernetes                        ClusterIP      10.100.0.1       <none>                                                                   443/TCP
3h31m
default         nginx                             LoadBalancer   10.100.88.89     acb6567e1b8f148ada797564febf3e68-1230036249.us-east-1.elb.amazonaws.com   80:32750/TCP
85s
kube-system     kube-dns                          ClusterIP      10.100.0.10      <none>                                                                   53/UDP,53/TCP
3h31m
```

- Run curl command on the LoadBalancer to see a HTML Nginx welcome page. For the LoadBalancer above run the command

curl acb6567e1b8f148ada797564febf3e68-1230036249.us-east-1.elb.amazonaws.com

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

- Now apply the NetworkPolicy that restricts all incoming access to the service.

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
```

```
      metadata:
        name: default-deny-ingress
      spec:
        podSelector: {}
        policyTypes:
          - Ingress
      EOF
```

- Now, again, run the curl command `curl acb6567e1b8f148ada797564febf3e68-1230036249.us-east-1.elb.amazonaws.com` You won't see any response as all the incoming traffic is denied.

2) Block access from a particular Namespace

- Now we create two new nginx pods in two different Kubernetes Namespaces using the following command

```
kubectl create ns nginx
kubectl run nginx —image=nginx -n nginx
```

```
kubectl create ns server
kubectl run nginx image=nginx -n server
```

```
NAMESPACE         NAME                                       READY   STATUS    RESTARTS       AGE
calico-system     calico-kube-controllers-56b6ddd94f-l6swj   1/1     Running   0              4h56m
calico-system     calico-node-cmq8m                          1/1     Running   0              5h5m
calico-system     calico-node-gz9gh                          1/1     Running   0              5h5m
calico-system     calico-typha-6497d7d7c6-ggsxj              1/1     Running   1 (5h4m ago)   5h5m
calico-system     csi-node-driver-jn77h                      2/2     Running   0              5h5m
calico-system     csi-node-driver-sdq2r                      2/2     Running   0              5h4m
default           nginx                                      1/1     Running   0              3h16m
kube-system       coredns-7f5998f4c-rvpnm                    1/1     Running   0              4h56m
kube-system       coredns-7f5998f4c-xtnx5                    1/1     Running   0              4h56m
kube-system       kube-proxy-jbgpd                           1/1     Running   0              5h5m
kube-system       kube-proxy-qlljd                           1/1     Running   0              5h5m
nginx             nginx                                      1/1     Running   0              44s
server            nginx                                      1/1     Running   0              7s
tigera-operator   tigera-operator-6f669b6c4f-6v2vj           1/1     Running   0              4h56m
```

- Expose these two pods as ClusterIP service so that they could be accessed within the cluster.

```
kubectl expose po nginx —port=80 -n server
```

```
kubectl expose po nginx —port=80 -n nginx
```

- We now access the nginx pod in server namespace from nginx pod in nginx namespace using following command to see the following output.

```
kubectl exec -it -n nginx nginx curl nginx.server.svc.cluster.local
```

```
<!DOCTYPE html>
<html>
<head>
<title>Welcome to nginx!</title>
<style>
html { color-scheme: light dark; }
body { width: 35em; margin: 0 auto;
font-family: Tahoma, Verdana, Arial, sans-serif; }
</style>
</head>
<body>
<h1>Welcome to nginx!</h1>
<p>If you see this page, the nginx web server is successfully installed and
working. Further configuration is required.</p>

<p>For online documentation and support please refer to
<a href="http://nginx.org/">nginx.org</a>.<br/>
Commercial support is available at
<a href="http://nginx.com/">nginx.com</a>.</p>

<p><em>Thank you for using nginx.</em></p>
</body>
</html>
```

Now we restrict access to server namespace from any other name space except default namespace using the following Kubernetes Network Policy and command.

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: allow-namespace
  namespace: server
spec:
  podSelector:
    matchLabels:
      run: nginx
  policyTypes:
    - Ingress
  ingress:
    - from:
        - namespaceSelector:
            matchLabels:
                kubernetes.io/metadata.name: default
EOF
```

- Now again, run –

```
kubectl exec -it -n nginx nginx curl nginx.server.svc.cluster.local
```

- This won't work this time and the access to nginx pod in server ns will be blocked
- Now access nginx pod in server namespace from default namespace using the command

```
kubectl exec -it nginx curl nginx.server.svc.cluster.local
```

- You are now able to see the HTML nginx welcome page because the above mentioned policy grants access to traffic from default namespace.

3) Deny all egress traffic from a port

- Now we block all outgoing traffic from nginx in default namespace from port number 80. Before that we try accessing ngnix pod in server and nginx namesapce using commands

```
kubectl exec -it nginx curl nginx.nginx.svc.cluster.local
kubectl exec -it nginx curl nginx.server.svc.cluster.local
```

Both the command gives the output as nginx HTML Welcome page.

- Now block all egress from default nginx pod from port 80 using command:

```
cat <<EOF | kubectl apply -f -
apiVersion: networking.k8s.io/v1
kind: NetworkPolicy
metadata:
  name: egress-test-network-policy
  namespace: default
spec:
  podSelector:
    matchLabels:
      run: nginx
  policyTypes:
    - Egress
  ingress:
  egress:
    - ports:
        - protocol: TCP
          port: 80
EOF
```

Run the following command to see no response. Therefore, egress traffic from nginx pod in default namespace at port 80 was successfully blocked.

```
kubectl exec -it nginx curl nginx.nginx.svc.cluster.local
kubectl exec -it nginx curl nginx.server.svc.cluster.local
```

Chapter 3 : Threat Detection with EKS GuardDuty

Prerequisite

I recommend it to go through the following topics before proceeding with the document:

- Kubernetes Service - https://kubernetes.io/docs/concepts/services-networking/service/

What is GuardDuty?

Guard duty continuously monitor your AWS accounts, instances, container workloads, users, and storage for potential threats. In this document, we focus only on threat detection for AWS EKS.

Why Guard Duty?

- GuardDuty EKS Protection enables Amazon GuardDuty to detect suspicious activities and potential compromises of your Kubernetes clusters within Amazon Elastic Kubernetes Service (Amazon EKS).
- With one click in the AWS Management Console or a single API call, you can enable Amazon GuardDuty on a single account.

Problems and Solutions:

Problems:

- How do we detect malicious activity in EKS cluster like access from a tor network, anonymous access to the cluster, exposing Kubernetes dashboard etc.
- Developing and deploying the above mentioned threat detection system on premise would require a lot of resource, overhead and maintenance.
- Deploying a similar system across various accounts or places could be even more time consuming and complicated.
- User/Customer needs to define threat findings and their meaning. For example, *Kubernetes/TorIPCaller* - This finding informs you that an API was invoked from a Tor exit node IP address.

Solutions:

- Guard duty detects malicious activity in your EKS cluster once you enable it.
- We can enable guard duty using one click in the AWS Management Console on a single account. AWS manages the resources related overhead and maintenance (AWS Managed Service).
- Guard duty has a set of predefined finding mentioned in this link : https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-kubernetes.html.

Enabling Guard Duty for EKS

- Open the GuardDuty console at https://console.aws.amazon.com/guardduty/.
- In the navigation pane, under **Settings**, choose EKS Protection.
- The EKS Protection pane lists the current status of Kubernetes protection for your account. You may enable or disable it by selecting **Enable** or **Disable** respectively, then confirming your selection.

Note: For more details and updates on enabling guard duty refer this AWS doc: https://docs.aws.amazon.com/guardduty/latest/ug/kubernetes-protection.html.

Generating Sample findings – On AWS Console

- Open the GuardDuty console at https://console.aws.amazon.com/guardduty/.
- In the navigation pane, choose **Settings**.
- On the **Settings** page, under **Sample findings**, choose **Generate sample findings**.
- In the navigation pane, choose **Findings**. The sample findings are displayed on the **Current findings** page with the prefix **[SAMPLE]**.

For more details and updates on generating sample finding refer this aws doc: https://docs.aws.amazon.com/guardduty/latest/ug/sample_findings.html.

Generate EKS related Finding "Manually"

Prerequisites

Guard Duty enabled on the cluster. To enable Guard Duty refer this link - https://docs.aws.amazon.com/guardduty/latest/ug/kubernetes-protection.html

Findings

Policy:Kubernetes/ExposedDashboard: (severity Medium)

This find can be generated by exposing your K8s dashboard using a load balancer. Kubernetes dashboard can be deployed using this link - https://kubernetes.io/docs/tasks/access-application-cluster/web-ui-dashboard/. Once done now we need to expose it as a LoadBalancer using the following command:

1. `kubectl edit svc kubernetes-dashboard -n kubernetes-dashboard`
2. This opens a VI editor. Under spec edit type to LoadBalancer.

Execution:Kubernetes/ExecInKubeSystemPod (severity Medium)

Run the exec command on a pod in kube-system Namespace. For example, if the name of the pod is kube-proxy-jbgpd then run the command

```
kubectl exec -it -n kube-system kube-proxy-jbgpd ls
```

Policy:Kubernetes/AdminAccessToDefaultServiceAccount (severity High)

Add default service account to your ClusterRoleBinding `cluster-admin`. This can be done by the command `kubectl edit clusterrolebinding cluster-admin`. This opens up a VI editor. Add the following snippet of code under `-subject`

```
- kind: ServiceAccount
  name: default
  namespace: default
```

Policy:Kubernetes/AnonymousAccessGranted (severity High)

Grand anonymous access to the users by adding this configuration using the command

```
kubectl apply -f crb-anonymous.yaml
```

crb-anonymous.yaml

```
apiVersion: rbac.authorization.k8s.io/v1
kind: ClusterRoleBinding
metadata:
  name: anonymous-binding
roleRef:
  apiGroup: rbac.authorization.k8s.io
  kind: ClusterRole
  name: anonymous-role
subjects:
- apiGroup: rbac.authorization.k8s.io
```

```
  kind: User
  name: system:anonymous
```

Tor related finding

For this steps you need to torify your environment, for which you are free to use your own method. My blog (https://amansingh.blog/2023/07/06/torify-your-machine/) describes one of the easiest way to do so.

CredentialAccess:Kubernetes/TorIPCaller (severity High)

With you torified kubectl run the command:

```
kubectl get secrets -A
```

Impact:Kubernetes/TorIPCaller (severity High)

Run command to edit a kube-system namespace secret from the output of the command in previous step (`kubectl get secrets -A`). For example:

```
kubectl edit secret node-controller-token-8lvdb -n kube-system
```

The command above opens a VI editor. Under metadata - annotation add a test annotation as below.

```
metadata:
  annotations:
    test: "1"
```

Save the file.

Discovery:Kubernetes/TorIPCaller (severity Medium)

The above step automatically generates this finding (When you run kubectl edit command)

Understanding these finding:

Each finding generated by Guard Duty has a severity i.e. low, medium and high. For information on each finding refer this https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-kubernetes.html.

Some of these findings can tell the following information:

- CredentialAccess:Kubernetes/TorIPCaller - Tor network can be used for a lot of malicious activity and privacy related threat. Guard duty detects access from a tor exit node to your cluster.

- Policy:Kubernetes/ExposedDashboard - This tells that your cluster is exposed to the internet by a Kubernetes LoadBalancer service. This can help you take actions so that someone does not manipulate your EKS cluster using the dashboard.

To take action on above mentioned finding you can follow the **Remediation recommendations** mentioned here - https://docs.aws.amazon.com/guardduty/latest/ug/guardduty_finding-types-kubernetes.html#credentialaccess-kubernetes-toripcaller

Chapter 4 : Falco

What is Falco?

Falco, a Cloud Native Computing Foundation (CNCF) project, is a common open source tool used to perform similar threat detection capabilities within Kubernetes clusters. Falco monitors system calls from the Linux kernel for most of its analysis. It is also preloaded with some out of the box rule sets. Falco is like Amazon GuardDuty, a threat detection system. I explain later in the chapter the exact difference between the two.

Why Falco?

Strengthen container security: The flexible rules engine allows you to describe any type of host or container behavior or activity.

Reduce risk via immediate alerts: You can immediately respond to policy violation alerts and integrate Falco within your response workflows for example AWS CloudWatch.

Leverage most current detection rules: Falco's out-of-the box rules alert on malicious activity and Common Vulnerabilities and Exposures (CVE) exploits

Falco vs GuardDuty

Criteria	Guardduty	Falco
Ease of implementation	GuardDuty can be enabled by a single click. There is no implementation overhead involved.	Resource deployment is required. For Kubernetes, Falco should be deployed as deamonset using Helm or Kubernetes configuration file.
Customizing findings	GuardDuty finding can't be customized. It is possible to take actions on reducing sevirity of a finding using lambdas but it is not possible to cutomize the finding.	Findings in Falco can be customized by changing Falco rules.
Adding new	It is not possible to add new findings. At the time of writing	Using Falco you can always add new rules to generate a finding. Apart from this, Falco already

findings	this document there were 27 GuardDuty finding for EKS.	has a set of predefined rules.
Maintaining findings	For Guardduty, you don't need to maintain the findings rules. They are managed by AWS. For example Malicious IP address list is updated by AWS.	In Falco, you need to update the finding when required. For example, you need to manually update the Malicious IP address list when required. This could be challenging when you want the list to be updated automatically by someone. Additionally, this can be an advantage when you want to keep track of Malicious IP list and update it mannually.
Syscall ability	EKS GuardDuty only alerts on activities against the K8s Control Plane of the cluster.	Falco has the ability to ingest syscalls from containers on the cluster to detect malicious activity on the container itself.
Integrations	GuardDuty findings are sent to AWS services like SecurityHub and the default Eventbridge eventbus within the account.	Falco can emit its findings to stdout, a file, syslog, or custom endpoints with “Program Output” using bash. Also by using data collection tools like fluentbit you can send the findings to AWS cloudwatch, Elasticsearch etc.
Integration with Cloud Native platforms	GuardDuty is a AWS service and can be integrated only with AWS EKS	Falco can be integrated with any Kubernetes cluster.

GuardDuty EKS protection offers quick centralized enablement and is managed by AWS for updates to findings. Enabling this functionality organization-wide only costs money when an EKS cluster is present and has activity.

Comapred to GuardDuty Falco offers an increased level of customization and granularity. Falco should be enabled in organizations with the maturity to tune and monitor the alerts, as well as create defined runbooks for responders.

In the end, it is not usually an either-or situation for GuardDuty EKS and Falco. The two tools complement each other to supplement a defense in depth strategy securing container workloads on AWS.

Automated deployment of Falco

This blog https://aws.amazon.com/blogs/containers/implementing-runtime-security-in-amazon-eks-using-cncf-falco/ mentions the steps to deploy Falco on Amazon EKS.

I simplified this process to one command if the following perquisites are met.

Prerequisites:

- EKS Cluster with at least 1 node.
- awscli already configured with secret key. Refer this link for more information on how to configure it https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-quickstart.html#cli-configure-quickstart-creds.
- Connected to the eks cluster and you must already have kubectl installed

Deployment:

- Clone the repo - https://github.com/aman-2812/EKS-Security/tree/main/falco/scripts
- Run the bash file `deploy_falco.sh` using the command:

```
bash deploy_falco.sh EKS_NODE_ROLE_NAME
```

Here, replace `EKS_NODE_ROLE_NAME` with the name of the eks node role.
This script will deploy Falco, Fluentbit and start pushing findings (if any) to Cloudwatch.

Note: The bash script uses customized git repos and already has customized rules defined. We will see some findings generated by these rules in the next section.

Analysing Findings:

- Open AWS Cloudwatch and select Log groups. For more information on how to navigate to Log groups refer this link https://docs.aws.amazon.com/AmazonCloudWatch/latest/logs/Working-with-log-groups-and-streams.html.
- You will see a Log group with name Falco. Click on that.
- Under Log Streams select alerts.
- Now run the following command in your Kubernetes cluster.

```
kubectl exec nginx-pod -it touch /etc/1
```

  Here, replace nginx-pod with the nginx pod running in your cluster.
  This command generates a finding on the Log Stream alerts. This finding rule is provided by Falco.

- Now, run the following command to generate a self defined finding.

```
kubectl exec nginx-pod -it whoami
```

  Here, replace nginx-pod with the nginx pod running in your cluster.

This command generates a finding on the Log Stream alerts. This finding is generated by a custom written rule.

Conclusion

Security for AWS EKS (Elastic Kubernetes Service) is bolstered by a combination of powerful tools and services. IAM (Identity and Access Management) enables fine-grained control over user access, ensuring only authorized entities interact with EKS resources. EKS leverages Calico for network security, implementing network policies to enforce traffic isolation between pods. AWS GuardDuty actively monitors EKS clusters, detecting potential threats and vulnerabilities. Falco, an open-source runtime security tool, adds an extra layer of protection by providing real-time container-level monitoring and alerting for suspicious activities. Together, these services fortify EKS with robust security measures, enhancing the overall safety and integrity of containerized workloads.