Amazon GuardDuty & Kasten K10

Quick Start Guide

January 2024

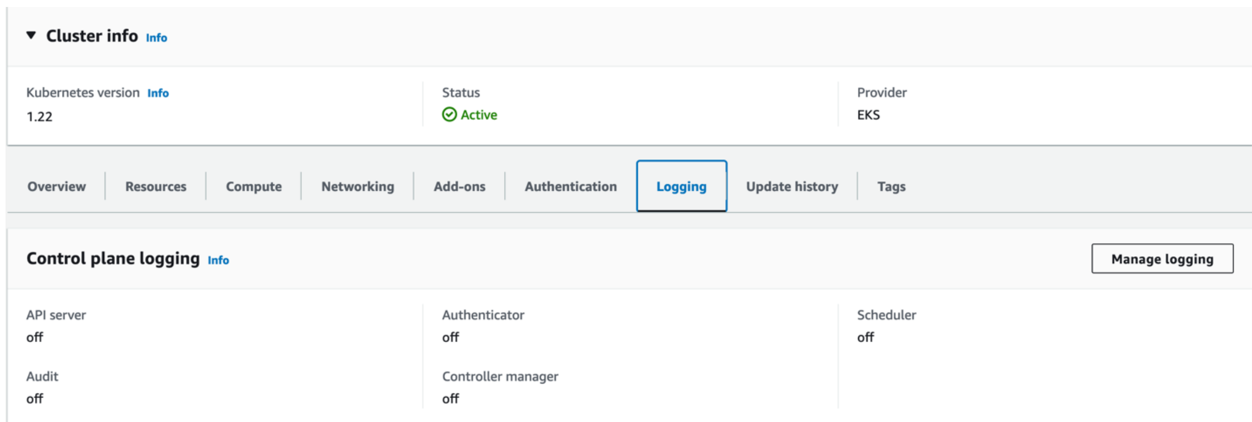
Amazon GuardDuty & Kasten K10

Below is an example guide of how someone may be able to deploy Amazon GuardDuty in conjunction with Kasten by Veeam's Kasten K10 in an Amazon Elastic Kubernetes Service (EKS) cluster to ensure a security posture fit for your deployment that correlates and heightens events that are associated with custom Kasten K10 resources.

Configuration of Amazon GuardDuty

First, make sure you have access rights to GuardDuty in your AWS account. Once you do, go to GuardDuty and turn on "EKS Protection".

Next, deploy a cluster in EKS; here we're deploying a simple cluster with three `m6a.large` nodes. Once the cluster has been set up, navigate to "Logging" as seen below, and click "Manage logging" in the upper right corner:



Turn on "audit" logs, and click save as seen below:

Control plane logging Info

Send audit and diagnostic logs from the Amazon EKS control plane to CloudWatch Logs.

API server
Logs pertaining to API requests to the cluster.

Audit
Logs pertaining to cluster access via the Kubernetes API.

Authenticator
Logs pertaining to authentication requests into the cluster.

Controller manager
Logs pertaining to state of cluster controllers.

Scheduler
Logs pertaining to scheduling decisions.

Cancel Save changes

This could take a few minutes, but once done you should have a new CloudWatch group called `/aws/eks/CLUSTER_NAME/cluster` with two log streams prefixed with `kube-apiserver-audit-*`.

Next, let's deploy the latest version of Kasten K10 and wait for all the pods to be running.

It's as simple as that to start getting `kube audit` logs from your cluster into CloudWatch and GuardDuty.

Seeing Kasten K10 events in CloudWatch Logs

Now that we have Kasten K10 installed in our EKS cluster, let's see how we can get audit logs to show up in CloudWatch. Let's first list all the available passkeys and then describe the `K10MasterKey`:

```
kubectl get passkeys
kubectl describe passkeys k10MasterKey
```

Navigate to the CloudWatch logs as defined above, and search for `/apis/vault.kio.kasten.io/v1alpha1/passkeys/k10MasterKey` (You may need to look in the second log stream if not found in the current one as EKS produces two log streams for the audit logs).

What you should see is the following:

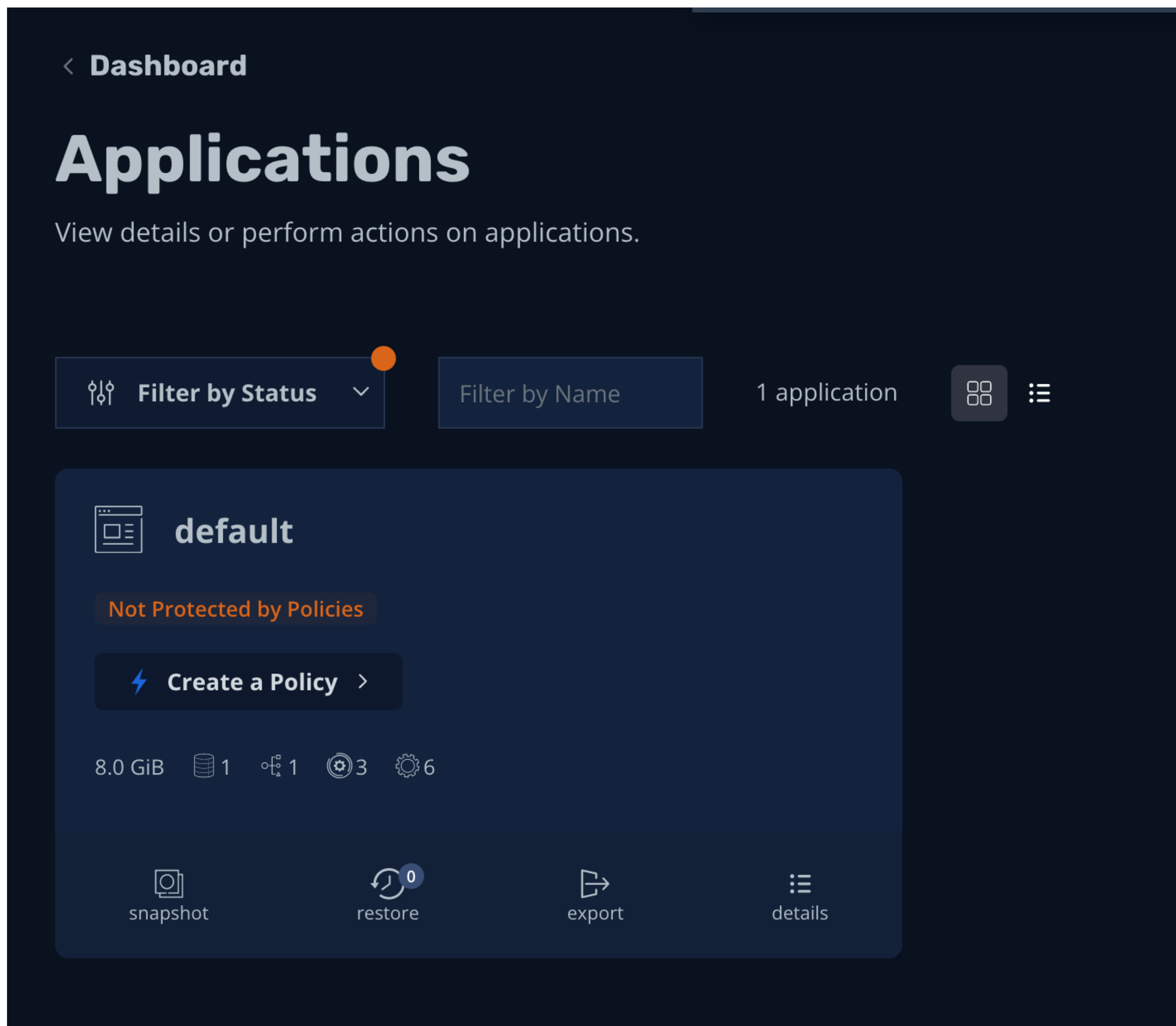
```
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "level": "Metadata",
    "auditID": "a3371b5b-4d23-41d7-b7c6-ab8d50ac6a02",
    "stage": "ResponseComplete",
    "requestURI":
"/apis/vault.kio.kasten.io/v1alpha1/passkeys/k10MasterKey",
    "verb": "get",
    "user": {
        "username": "kubernetes-admin",
        "uid": "aws-iam-authenticator:{ACCOUNT_ID}:{ACCESS_KEY_ID}",
        "groups": [
            "system:masters",
            "system:authenticated"
        ],
        "extra": {
            "accessKeyId": [
                "{ACCESS_KEY_ID}"
            ],
            "arn": [
                "arn:aws:sts:: {ACCOUNT_ID}:assumed-
role/EKSAdminRole/EKSGetTokenAuth"
            ],
            "canonicalArn": [
                "arn:aws:iam:: {ACCOUNT_ID}:role/EKSAdminRole"
            ],
            "sessionName": [
                "EKSGetTokenAuth"
            ]
        }
    },
    "sourceIPs": [
        "{IP_ADDRESS}"
    ],
    "userAgent": "kubectl/v1.25.0 (darwin/arm64) kubernetes/a866cbe",
    "objectRef": {
        "resource": "passkeys",
        "name": "k10MasterKey",
        "apiGroup": "vault.kio.kasten.io",
        "apiVersion": "v1alpha1"
    },
    "responseStatus": {
        "metadata": {},
        "code": 200
    },
    "requestReceivedTimestamp": "2022-12-22T22:04:47.213440Z",
    "stageTimestamp": "2022-12-22T22:04:47.216982Z",
    "annotations": {
        "authorization.k8s.io/decision": "allow",
        "authorization.k8s.io/reason": ""
    }
}
```


As you can see here unlike the local `k3d` cluster audit log, AWS provides the account id and access key id of the requester IAM. In this case we assumed the `EKSAdminRole` and so the

access key id was that of this associated IAM. Note that this was done from the cli and that's why the IAM information is present.

We can clearly see this was a `GET` operation on the `vault.kio.kasten.io` API group's `passkeys` resource to get the named resource `k10MasterKey`.

Now, let’s create a new policy for our default application:



New Policy

Name
The display name for this policy
default-backup

Comments

Action
The action that should be taken when this policy is executed
Snapshot
Import

Use a Preset
Choose a pre-configured group of schedule and export settings

Backup Frequency
Hourly
Daily
Weekly
Monthly
Yearly
On Demand

Advanced Frequency Options
Backup Window

Snapshot at :00 each hour

Note: Times are stored in UTC, which does not change with Daylight Savings Time.

Create Policy
YAML
Cancel

Now when we search for `{($.requestURI = "/apis/config.kio.kasten.io/v1alpha1/namespaces/kasten-io/policies") && ($.verb = "create")}` we should see the following audit event:

```
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "level": "Metadata",
    "auditID": "9c884d9d-b68b-4b9d-a361-a47d1849ccf4",
    "stage": "ResponseComplete",
    "requestURI": "/apis/config.kio.kasten.io/v1alpha1/namespaces/kasten-
io/policies",
    "verb": "create",
    "user": {
        "username": "system:serviceaccount:kasten-io:k10-k10",
        "uid": "853192af-7f81-47b7-9558-a19ca20a5555",
        "groups": [
            "system:serviceaccounts",
            "system:serviceaccounts:kasten-io",
            "system:authenticated"
        ],
        "extra": {
            "authentication.kubernetes.io/pod-name": [
                "dashboardbff-svc-bb8799555-6254b"
            ],
            "authentication.kubernetes.io/pod-uid": [
                "3457698b-24ae-4990-9b9f-a47001083535"
            ]
        }
    },
    "sourceIPs": [
        "{IP_ADDRESS}"
    ],
    "userAgent": "dashboardbff-server/v0.0.0 (linux/amd64)
kubernetes/$Format",
    "objectRef": {
        "resource": "policies",
        "namespace": "kasten-io",
        "name": "default-backup",
        "apiGroup": "config.kio.kasten.io",
        "apiVersion": "v1alpha1"
    },
    "responseStatus": {
        "metadata": {},
        "code": 201
    },
    "requestReceivedTimestamp": "2022-12-22T22:15:27.150188Z",
    "stageTimestamp": "2022-12-22T22:15:27.191005Z",
    "annotations": {
        "authorization.k8s.io/decision": "allow",
        "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding
\"kasten-io-k10-k10-cluster-admin\" of ClusterRole \"cluster-admin\" to
ServiceAccount \"k10-k10/kasten-io\""
    }
}
```


Unlike the `passkeys` describe action which was done using the cli, this was done using the UI and so we see extra information such as the new `userAgent` and extra user information detailing the specific pod this request came from, namely, the `dashboard-bff` pod.

GuardDuty pulls from these CloudWatch logs and analyzes them against the finding types that are listed for the Kubernetes Audit logs data source. If any of them are triggered, it will show up in your dashboard.

Triggering Amazon GuardDuty Kubernetes Findings

The final step is to trigger a finding in Amazon GuardDuty that relates to a Kasten K10 custom resource. Here we'll use that `passkeys` resource again and aim to trigger the `Discovery:Kubernetes/SuccessfulAnonymousAccess` finding by creating a `clusterrolebinding` to the `k10-admin clusterrole` with the `system:anonymous` user; let's create this by running:

```
kubectl create clusterrolebinding anonymous-view --clusterrole=k10-admin --
user=system:anonymous
```

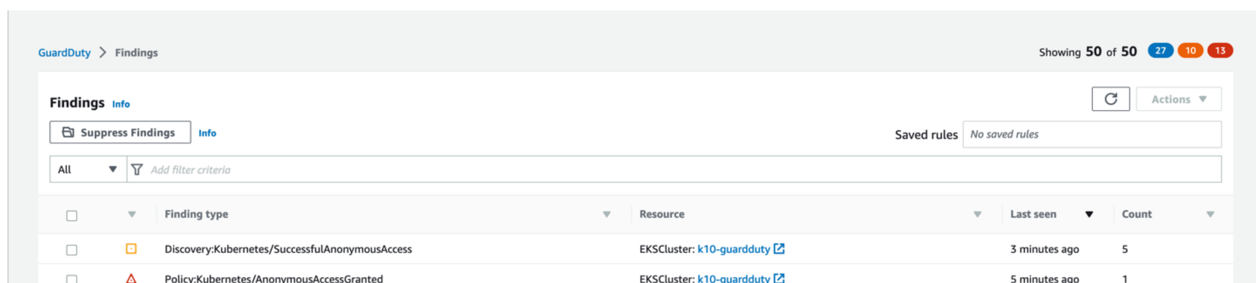
This will trigger the `Policy:Kubernetes/AnonymousAccessGranted` Amazon GuardDuty finding showing that we granted anonymous access within a Kubernetes cluster. Next, we obtain the query endpoint of our cluster since we'll need to use `curl` instead of `kubectl` to ensure we aren't authorized to call the api endpoint:

```
aws eks describe-cluster --name {CLUSTER_NAME} --query "cluster.endpoint" --
region {AWS_REGION} --output text
```

With that endpoint in hand let's now call the `passkeys` with a GET:

```
 curl -k
https://{SPECIFIC_ID}.{AWS_REGION}.eks.amazonaws.com/apis/vault.kio.kasten.io
/v1alpha1/passkeys\?limit\=500
```

Doing this will return the information requested, and it will also trigger the Amazon GuardDuty finding:



If you click on the finding, you'll be shown a lot more information including the Resource affected, Actor, and Action. You'll find information about your EKS cluster, as well as the action showing it was a call to listing `K10 passkeys`:

Discovery:Kubernetes/SuccessfulAnonymousAccess
Finding ID: 28c32e273dde714a30e4aa60df3d7ba6
Feedback

Medium Kubernetes API commonly used in Discovery tactics was invoked on cluster k10-guardduty by the anonymous user system:anonymous. Info

Investigate with Detective

Overview	
Severity	MEDIUM
Region	us-east-1
Count	5
Account ID	
Resource ID	k10-guardduty
Created at	02-16-2023 14:18:12 (6 minutes ago)
Updated at	02-16-2023 14:21:18 (3 minutes ago)

Resource affected	
Resource role	TARGET
Resource type	EKSCluster
EKS cluster details	
Name	k10-guardduty
ARN	arn:aws:eks:us-east-1: cluster/k10-guardduty
VPC ID	
Status	ACTIVE
Created at	02-16-2023 21:00:39 UTC
Kubernetes user details	
Username	system:anonymous
Tags	
AWS :cloudformation:stack-name	eksctl-k10-guardduty-cluster
AWS :cloudformation:stack-id	arn:aws:cloudformation:us-east-1: :stack/eksctl-k10-guardduty-cluster/
Ephemeral-role-name	21b97-K10_CUST_PERMS-k10-guardduty
EKS ctl.cluster.k8s.io/v1alpha1/cluster-name	k10-guardduty
Env	
Renewals	0
Name	eksctl-k10-guardduty-cluster/ControlPlane
Alpha.eksctl.io/cluster-name	k10-guardduty
Created by	
Alpha.eksctl.io/cluster-oidc-enabled	true
AWS :cloudformation:logical-id	ControlPlane
Alpha.eksctl.io/eksctl-version	0.105.0
Expiration date	1680469201

Action	
Action type	KUBERNETES_API_CALL
Request uri	/apis/vault.kio.kasten.io/v1alpha1/passkeys
Verb	list
Status code	200
First seen	02-16-2023 14:17:12 (19 hours ago)
Last seen	02-16-2023 14:21:14 (19 hours ago)

Actor	
IP address	
Location	
City	
Country	United States
Lat	
Lon	
Organization	
Asn	20001
Asn org	TWC-20001-PACWEST
Isp	Spectrum
Org	Spectrum

Additional information	
Archived	false

Conclusion

Amazon GuardDuty configured in conjunction with Kasten K10 for Kubernetes allows for an enhanced security posture to further protect Kasten K10 data from unknown attacks from malicious users.

Footnotes

1. https://www.statista.com/statistics/871513/worldwide-data-created/
2. https://www.securitymagazine.com/articles/97046-over-22-billion-records-exposed-in-2021