2023 LONDON

4 - 6 October, 2023 / London, United Kingdom

TALES FROM A CLOUD CSIRT – LET'S DEEP DIVE INTO A KUBERNETES (K8S) INFECTION

Santiago Abastante

Solidarity Labs, Argentina

sabastante@solidaritylabs.io

ABSTRACT

Kubernetes (K8s) is an orchestration system for automating software deployment, scaling and management, and if you don’t already know… it’s really hot right now.

When implemented in a cloud environment, it allows a service to grow almost without limit because the K8s cluster can create and destroy servers at will, based on the load of the containers running. Imagine what can go wrong when attackers get hold of this power… you’re right, lightspeed growth equals a lot of destructive power.

In this paper we will analyse a real example of an *AWS* Kubernetes cluster infection through a software development supply chain compromise. The attackers were able to get *AWS* credentials from a DevOps workstation and use them to introduce a poisoned docker image into a Kubernetes cluster. It allowed them to move laterally within the cluster and to the cloud provider, retrieving secrets, passwords, tokens, and a bunch of other data.

Luckily, we were able to detect them just in time, as they had retrieved secrets that would have allowed them to move laterally to other companies or execute a new docker image with nastier results.

We will present the examples using a real-time lab, offering examples for incident responders and malware analysts to understand how to investigate these techniques, running through the cyber kill chain and explaining what went wrong and what could have been done better.

TECHNICAL CONCEPTS AND INTRODUCTION

Kubernetes: an overview

Kubernetes, often referred to as K8s, is an open-source container orchestration platform that simplifies the management and deployment of containerized applications. It provides a scalable and resilient framework for automating the deployment, scaling and management of containerized workloads across clusters of machines. Kubernetes abstracts away the complexities of infrastructure management and allows developers and administrators to focus on defining and running their applications efficiently. With its robust set of features, Kubernetes enables organizations to achieve high availability, fault tolerance and scalability for their applications, making it a popular choice for modern cloud-native architectures. By leveraging Kubernetes, teams can streamline their development and deployment processes, maximize resource utilization, and effectively manage complex containerized environments.

Key Kubernetes components

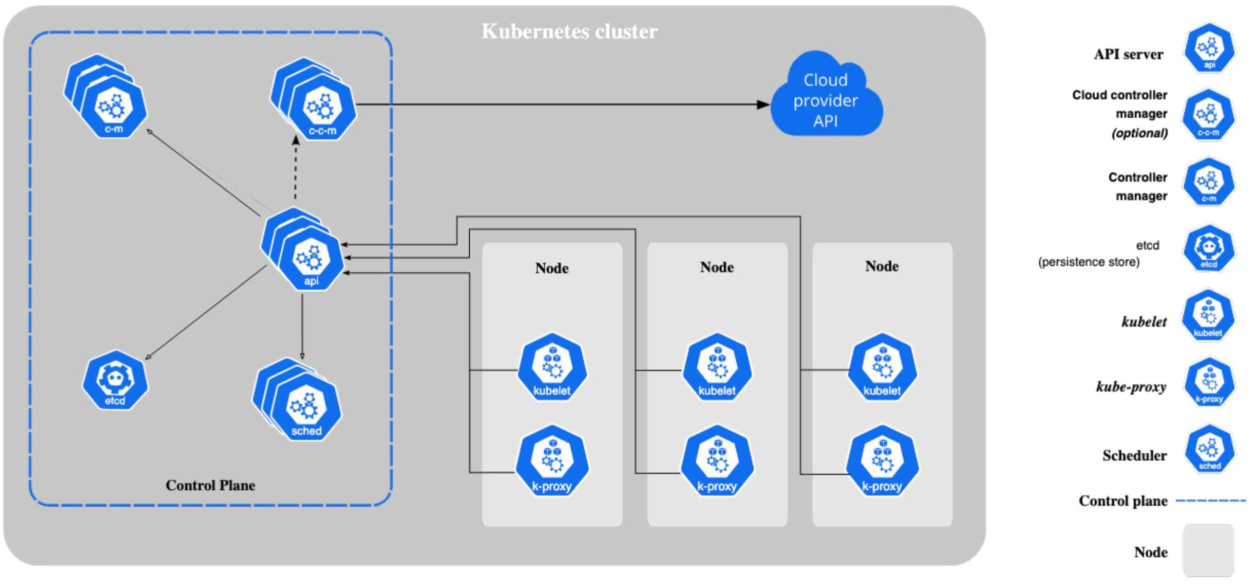


Figure 1: Kubernetes components (source: [1]).

Kubernetes consists of several key components that work together to manage and orchestrate containerized applications.

At the heart of the cluster is the API server, which serves as the central interface for communication and interaction with the cluster. It validates and processes requests, ensuring the cluster’s desired state is maintained. The scheduler is responsible for assigning workloads to available nodes based on resource requirements and constraints. The controller manager oversees various controllers that monitor the cluster’s state and take actions to ensure it matches the desired configuration. The etcd datastore serves as a distributed key-value store, storing the cluster’s configuration and runtime data.

Nodes, or worker machines, host the actual application workloads and are managed by the control plane. Each node runs a kubelet, a Kubernetes agent responsible for managing containers, and a container runtime, such as Docker, to run the application containers.

These key components work together to provide a scalable and resilient platform for deploying and managing containerized applications in a Kubernetes cluster.

Kubernetes security

Kubernetes security is of paramount importance in ensuring the protection and integrity of containerized applications and the underlying infrastructure. With the dynamic and distributed nature of Kubernetes deployments, there are specific security considerations to address.

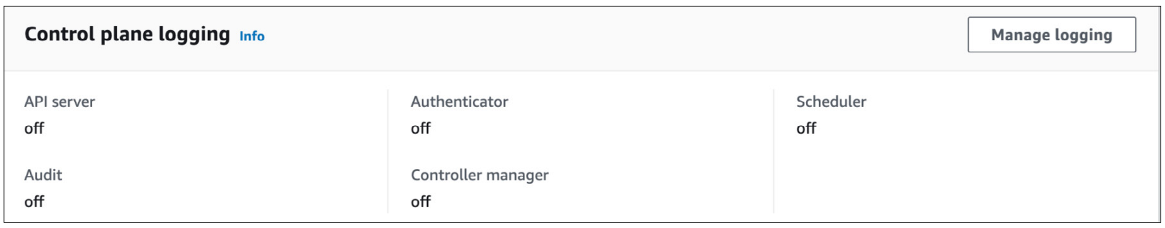
From the hardening perspective, secure configuration and access control are crucial aspects, involving the proper configuration of Kubernetes components and enforcing granular access controls to limit unauthorized access. Secure container images and runtime isolation are also vital, with the implementation of container image scanning, vulnerability management, and runtime security measures such as pod security policies and network policies.

Ongoing monitoring and logging are essential for detecting and responding to security incidents, while encryption of sensitive data and communication channels adds an extra layer of protection. Regular updates and patches, along with conducting periodic security assessments and audits, contribute to maintaining a robust and secure Kubernetes environment. By adopting a comprehensive security approach, organizations can mitigate risks and safeguard their Kubernetes deployments against potential threats and vulnerabilities.

Kubernetes logging and monitoring

In a Kubernetes environment, logs refer to the recorded events and messages generated by various components and applications running within the cluster. Kubernetes provides a centralized logging mechanism that allows you to collect, store, and analyse logs from different sources, including pods, containers, and control plane components. Logs are essential for understanding the behaviour and performance of your applications, diagnosing issues, and monitoring the health of your Kubernetes infrastructure.

- **Container logs**: These logs contain the output and error messages generated by individual containers running within a pod. They provide insights into the application's behaviour, including status, events, and any issues encountered by the container.
- **Kube-apiserver logs**: The kube-apiserver logs record activities and events related to the Kubernetes API server. These logs are crucial for monitoring API requests, authentication, authorization, and any errors or warnings related to the API server's functionality.
- **Kube-controller-manager logs**: The kube-controller-manager logs capture information about the Kubernetes controller manager. These logs provide details on various controllers, including node controller, replication controller, endpoint controller, and others. They help monitor the behaviour and health of controller processes.
- **Kube-scheduler logs**: The kube-scheduler logs contain information about the Kubernetes scheduler, which assigns pods to nodes based on resource requirements, affinity rules, and other constraints. These logs provide insights into the scheduling decisions made by the scheduler.
- **Kube-proxy logs**: The kube-proxy logs capture events and activities related to the Kubernetes network proxy running on each node. These logs provide details on network routing, load balancing, and any errors or warnings related to the proxy's operation.
- **Ingress controller logs**: If you are using an ingress controller for managing external access to your cluster, the logs from the ingress controller provide insights into the routing, load balancing, and SSL termination processes for incoming traffic.
- **Application logs**: These logs are specific to your applications running within Kubernetes pods. They capture application-specific events, errors, and informational messages. Application logs are vital for monitoring application behaviour, diagnosing issues, and troubleshooting application-level problems.



API server
off

Authenticator
off

Scheduler
off

Audit
off

Controller manager
off

Figure 2: EKS logging configuration.

Benefits and challenges of Kubernetes

Kubernetes enables horizontal scaling of applications by efficiently distributing workloads across multiple containers and nodes. It simplifies the process of adding or removing resources as needed, allowing applications to handle increased demand seamlessly.

It also provides robust mechanisms for ensuring high availability of applications. It can automatically detect and recover from node failures or pod disruptions by rescheduling or replicating affected containers on healthy nodes and optimizes resource utilization by intelligently scheduling and allocating resources based on application requirements. It allows for fine-grained control over resource allocation and can automatically scale resources up or down based on workload demands.

Kubernetes provides built-in service discovery and load balancing mechanisms. It allows applications to easily discover and communicate with other services within the cluster, abstracting away the complexities of network routing and load balancing, it continuously monitors the health of applications and automatically restarts or replaces failed containers to maintain the desired state. It also provides features like readiness and liveness probes to detect and recover from application-level failures.

On the other hand, Kubernetes is a complex platform with a steep learning curve. Setting up and configuring a Kubernetes cluster requires an understanding of various concepts, components, and configuration options. Managing and troubleshooting issues within a distributed system can also be challenging for administrators and operators. Deploying and managing Kubernetes clusters requires dedicated resources, including skilled personnel and infrastructure. Organizations need to invest in training, continuous monitoring, upgrades and maintenance to ensure the proper functioning of the Kubernetes environment.

Cloud computing fundamentals

Cloud fundamentals

Cloud computing is a model for delivering on-demand computing resources over the internet. It enables organizations to access a wide range of services, such as servers, storage, databases, networking, and software applications, without the need for upfront infrastructure investment or extensive on-premises hardware. Cloud computing offers several key concepts:

- **On-demand self-service**: Users can provision and access computing resources, such as virtual machines or storage, as needed, without requiring manual intervention from service providers. This allows for scalability and flexibility in resource allocation.
- **Broad network access**: Cloud services are accessible over the internet using standard protocols and can be accessed from various devices, including laptops, tablets and smartphones. Users can connect to the cloud from anywhere, anytime, as long as they have an internet connection.
- **Resource pooling**: Cloud providers consolidate computing resources to serve multiple customers, using virtualization techniques to create virtual instances that are logically isolated. These resources are dynamically allocated based on demand, ensuring efficient utilization and cost-effectiveness.
- **Rapid elasticity**: Cloud services can quickly scale up or down based on demand. This enables users to easily increase or decrease their resource usage to match their requirements, providing agility and cost optimization.
- **Measured service**: Cloud providers track resource usage and provide detailed billing information, allowing users to pay only for the resources they consume. This pay-per-use model helps optimize costs and provides transparency in resource utilization.

Security considerations

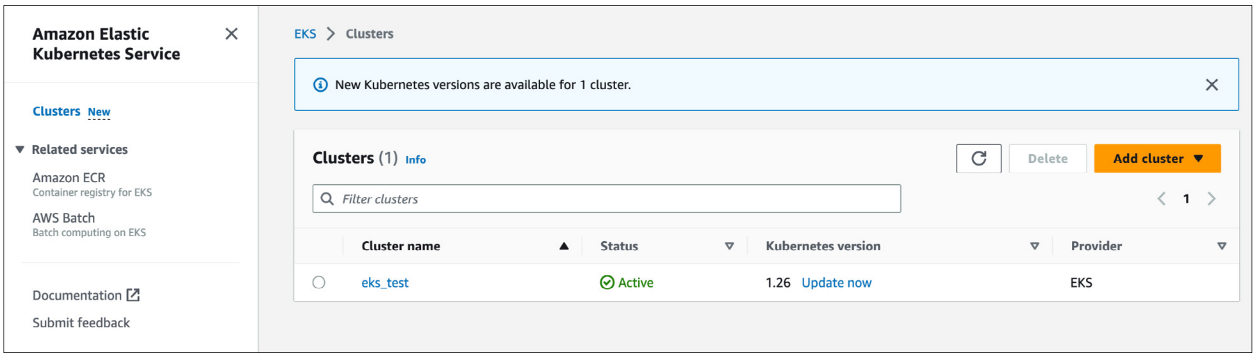
While the cloud offers numerous benefits, it also introduces unique security challenges. Organizations must carefully evaluate and address these considerations to ensure the confidentiality, integrity and availability of their data and applications. Key areas of focus include secure access controls, robust authentication and authorization mechanisms, and encryption of sensitive data in transit and at rest.

Implementing proper network security measures, such as firewalls, intrusion detection systems and network segmentation, is vital to protect against unauthorized access and potential attacks. Regularly monitoring and logging activities within the cloud environment helps detect and respond to security incidents promptly. Additionally, organizations should ensure compliance with industry regulations and standards, conduct regular security assessments and audits, and maintain clear incident response plans to mitigate and address potential security breaches effectively.

AWS services supporting Kubernetes

AWS provides a managed Kubernetes service called *Amazon Elastic Kubernetes Service* (*Amazon EKS*) [2], which allows customers to deploy, manage and scale Kubernetes clusters easily on the *AWS* platform.

Amazon EKS takes care of the underlying Kubernetes infrastructure management, including the control plane, ensuring high availability, scalability, and security. Customers can focus on deploying and managing their applications without the operational burden of managing the Kubernetes control plane. It also leverages the scalability and high availability capabilities of AWS to provide a reliable Kubernetes environment. It allows customers to easily scale their Kubernetes clusters up or down based on demand, ensuring optimal resource utilization and cost efficiency.



New Kubernetes versions are available for 1 cluster.

Cluster name	Status	Kubernetes version	Provider
eks_test	Active	1.26 Update now	EKS

Figure 3: AWS EKS management plane.

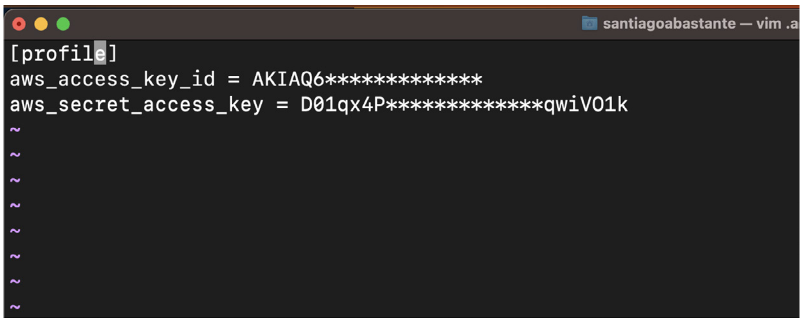
Identity and Access Management

AWS Identity and Access Management (*IAM*) [3] is a service provided by *Amazon Web Services* (*AWS*) that allows you to manage access and permissions to *AWS* resources. *IAM* enables you to securely control who can access your *AWS* services and resources and what actions they can perform.

IAM allows you to create individual *IAM* users for each person in your organization who needs access to *AWS* resources. You can also organize users into groups and assign permissions to groups, making it easier to manage access at scale. It also integrates with various *AWS* services, allowing you to control access to services like *Amazon S3*, *Amazon EC2*, *AWS Lambda*, and many others. By leveraging *IAM*, you can manage access at both the *AWS* account level and the individual resource level.

AWS enables users, applications, or services to interact with *AWS* resources programmatically through API calls or command-line tools. It involves the use of access keys (Access Key ID and Secret Access Key) associated with an *IAM* user or role, which are used to authenticate and authorize requests made to *AWS* APIs.

While *IAM* programmatic access is essential for automated interactions with *AWS* services, it also introduces potential security risks if not properly managed. If access keys are compromised, an attacker could gain unauthorized access to *AWS* resources. Access keys are similar to a username and password combination, and if they are inadvertently exposed or shared insecurely, malicious actors can use them to perform unauthorized actions.



```
[profile]
aws_access_key_id = AKIAQ6*************
aws_secret_access_key = D01qx4P*************qwiVO1k
~
~
~
~
~
~
~
~
```

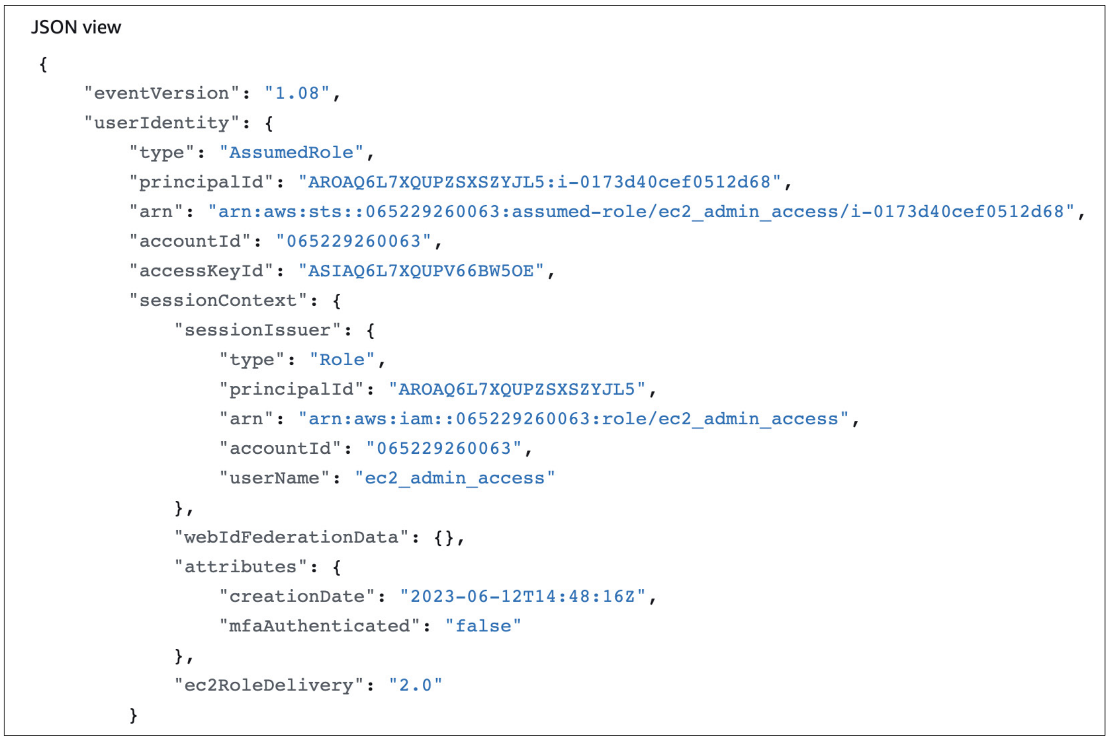
Figure 4: Example of AWS hard-coded credentials.

AWS logging and monitoring

AWS offers various monitoring and logging services, such as *Amazon CloudWatch* [4] and *AWS CloudTrail* [5], which can be integrated with *Amazon EKS* clusters. These services provide visibility into the performance, health and activities within the Kubernetes environment, enabling customers to monitor and troubleshoot their applications effectively. Figure 5 shows an example of an *AWS CloudTrail* log.

AWS cloud workload protection

AWS GuardDuty [6] is a threat detection service that continuously monitors your *AWS* accounts and workloads for malicious activity and unauthorized behaviour. While *GuardDuty* is not specifically designed for Kubernetes clusters, it can



JSON view

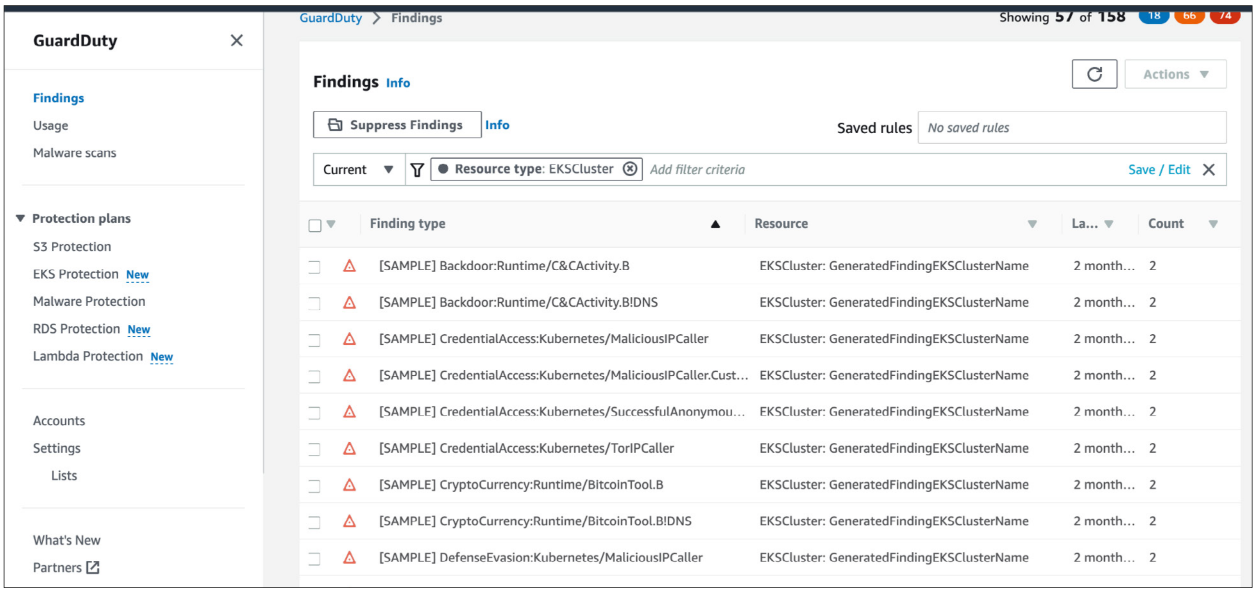
```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROAQ6L7XQUPZSXSZYJL5:i-0173d40cef0512d68",
        "arn": "arn:aws:sts::065229260063:assumed-role/ec2_admin_access/i-0173d40cef0512d68",
        "accountId": "065229260063",
        "accessKeyId": "ASIAQ6L7XQUPV66BW5OE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROAQ6L7XQUPZSXSZYJL5",
                "arn": "arn:aws:iam::065229260063:role/ec2_admin_access",
                "accountId": "065229260063",
                "userName": "ec2_admin_access"
            },
            "webIdFederationData": {},
            "attributes": {
                "creationDate": "2023-06-12T14:48:16Z",
                "mfaAuthenticated": "false"
            },
            "ec2RoleDelivery": "2.0"
        }
```

Figure 5: Example of an AWS CloudTrail log.

provide valuable insights and alerts for securing your overall *AWS* environment, including the underlying infrastructure and services that support your Kubernetes clusters.

GuardDuty uses machine learning algorithms and threat intelligence feeds to analyse *AWS CloudTrail* logs, VPC Flow Logs, and DNS logs to detect a wide range of threats, such as compromised credentials, unauthorized access attempts, and network attacks. By monitoring and analysing these logs, *GuardDuty* can detect potential security issues and generate actionable findings, including detailed alerts and recommendations for remediation.

To protect Kubernetes clusters specifically, *GuardDuty* can detect anomalies or suspicious activity related to the control plane infrastructure, such as unauthorized access attempts to the cluster's API server or suspicious changes to cluster configurations. *GuardDuty* can also identify potentially malicious network traffic between Kubernetes nodes, such as lateral movement attempts or communication with known malicious IP addresses.



Finding type	Resource	La...	Count
[SAMPLE] Backdoor:Runtime/C&CActivity.B	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] Backdoor:Runtime/C&CActivity.B!DNS	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CredentialAccess:Kubernetes/MaliciousIPCaller	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CredentialAccess:Kubernetes/MaliciousIPCaller.Cust...	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CredentialAccess:Kubernetes/SuccessfulAnonymou...	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CredentialAccess:Kubernetes/TorIPCaller	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CryptoCurrency:Runtime/BitcoinTool.B	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] CryptoCurrency:Runtime/BitcoinTool.B!DNS	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2
[SAMPLE] DefenseEvasion:Kubernetes/MaliciousIPCaller	EKSCluster: GeneratedFindingEKSClusterName	2 month...	2

Figure 6: Samples of Guard Duty findings.

AWS ECR and container registries

Amazon Elastic Container Registry (*ECR*) [7] acts as a repository for storing container images. When using *EKS*, you can push your container images to *ECR*, either manually or as part of your CI/CD pipeline. *ECR* provides secure and highly available storage for your container images, ensuring they are readily accessible to your *EKS* clusters. You can define *IAM* policies to manage who has permissions to push, pull, or manage container images in *ECR*. This allows you to control access to the images used by your *EKS* clusters and enforce security best practices.

ECR seamlessly integrates with *EKS*, making it easy to deploy your containerized applications. You can directly reference *ECR* images in your Kubernetes deployment manifests or Helm charts, allowing *EKS* to pull the required images and create the necessary containers.

It also includes a built-in image scanning capability that can analyse container images for vulnerabilities and security risks. When pushing an image to *ECR*, you can enable image scanning, which automatically scans the image for known vulnerabilities based on the Common Vulnerabilities and Exposures (CVE) database. This helps ensure that only secure and trusted container images are deployed within your *EKS* clusters.

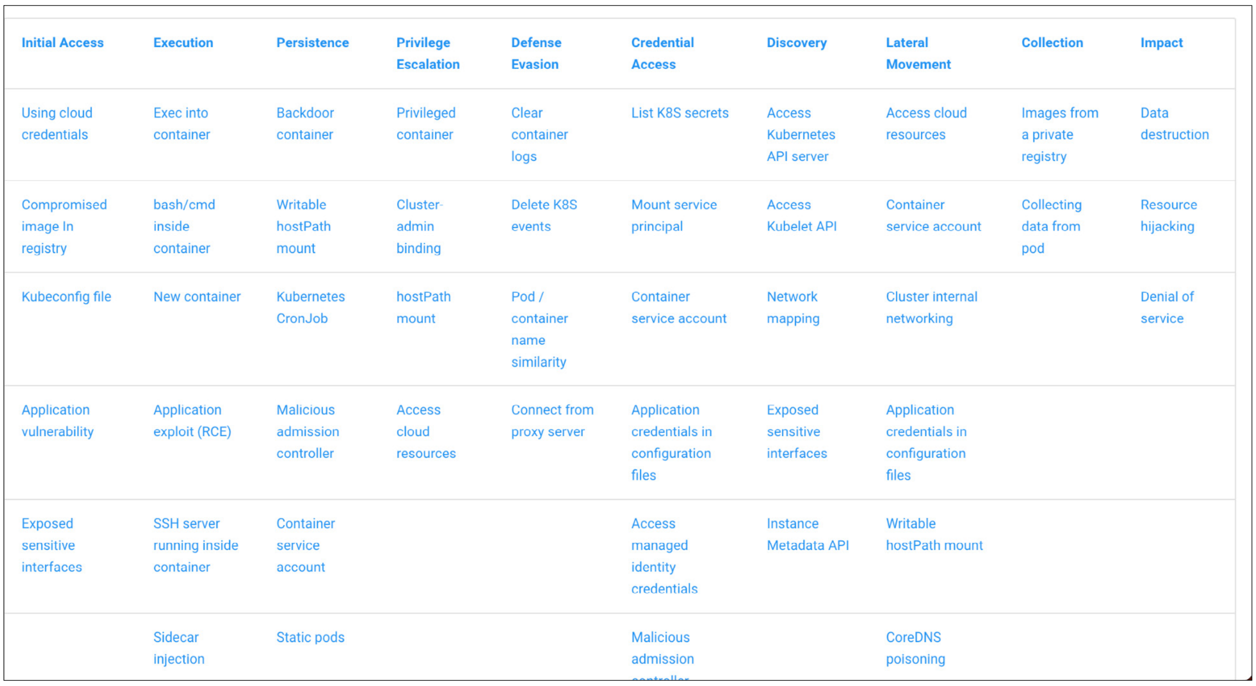
Understanding Kubernetes infections

Kubernetes infections refer to instances where malicious actors gain unauthorized access or exploit vulnerabilities within Kubernetes clusters, compromising the integrity, availability, or security of the containerized applications and the underlying infrastructure.

These infections can manifest in various forms, including the deployment of malicious containers, unauthorized privilege escalations, or the manipulation of cluster configurations.

Kubernetes infections can lead to data breaches, service disruptions, or unauthorized activities within the cluster, highlighting the critical importance of implementing strong security measures, such as proper access controls, container image scanning, and regular vulnerability patching, to mitigate the risk of infections and protect the overall Kubernetes environment.

Common attack vectors



Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access Kubernetes API server	Access cloud resources	Images from a private registry	Data destruction
Compromised image In registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account	Collecting data from pod	Resource hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from proxy server	Application credentials in configuration files	Exposed sensitive interfaces	Application credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container	Container service account			Access managed identity credentials	Instance Metadata API	Writable hostPath mount		
	Sidecar injection	Static pods			Malicious admission		CoreDNS poisoning		

Figure 7: Kubernetes Threat Matrix (source: [8]).

For the sake of this paper we will review those that are related to the security incident:

- **Using cloud credentials [9]**: In cases where the Kubernetes cluster is deployed in a public cloud (e.g. *AKS* in *Azure*, *GKE* in *GCP*, or *EKS* in *AWS*), compromised cloud credentials can lead to cluster takeover. Attackers who have access to the cloud account credentials can get access to the cluster's management layer.
- **Access Kubernetes API server [10]**: The Kubernetes API server is the gateway to the cluster. Actions in the cluster are performed by sending various requests to the RESTful API. The status of the cluster, which includes all the components that are deployed on it, can be retrieved by the API server. Attackers may send API requests to probe the cluster and get information about containers, secrets, and other resources in the cluster.

- **New container [11]**: Attackers may attempt to run their code in the cluster by deploying a container. Attackers who have permissions to deploy a pod or a controller in the cluster (such as DaemonSet \ ReplicaSet\ Deployment) can create a new resource for running their code.
- **Resource hijacking [12]**: Attackers may abuse a compromised resource for running tasks. A common abuse is to use compromised resources for running digital currency mining. Attackers who have access to a container in the cluster or have permissions to create new containers may use them for such activity.

Potential risks and impact

Kubernetes introduces potential risks and impacts that organizations need to be aware of. Misconfigurations, vulnerabilities, and unauthorized access can lead to data breaches, compromising sensitive information and violating compliance requirements.

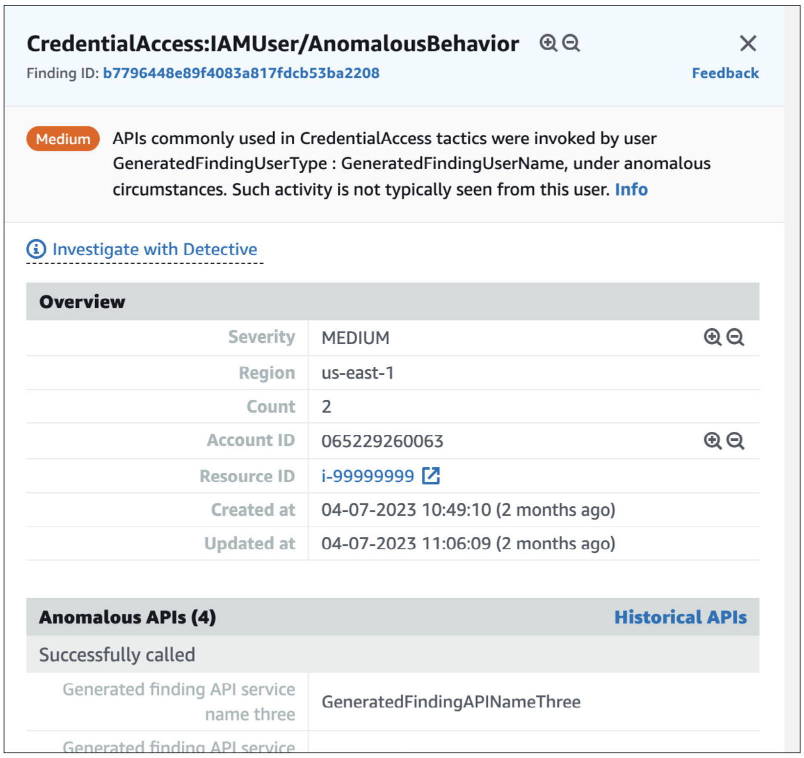
Downtime and service disruptions can occur due to cluster failures, resource exhaustion, or malicious attacks, resulting in financial losses, reputational damage, and customer dissatisfaction. Unauthorized modifications to cluster configurations or unauthorized access to production environments can lead to unauthorized activities, data tampering, or unauthorized privilege escalations.

Additionally, the complex nature of Kubernetes requires organizations to invest in skilled personnel, training, and ongoing maintenance, increasing operational overhead. It is crucial for organizations to address these risks proactively through robust security measures, proper configuration management, regular vulnerability assessments, and monitoring to ensure the security, availability and integrity of their Kubernetes environments.

THE INCIDENT

Detection

Our CSIRT Oncall process was triggered because of a security alert in one of our client's *AWS* tenants. It was an *AWS GuardDuty* alert related to *IAM* misbehaviour:



CredentialAccess:IAMUser/AnomalousBehavior

Finding ID: b7796448e89f4083a817fdcb53ba2208

Feedback

Medium APIs commonly used in CredentialAccess tactics were invoked by user GeneratedFindingUserType : GeneratedFindingUserName, under anomalous circumstances. Such activity is not typically seen from this user. Info

Investigate with Detective

Overview

Severity	MEDIUM
Region	us-east-1
Count	2
Account ID	065229260063
Resource ID	i-99999999
Created at	04-07-2023 10:49:10 (2 months ago)
Updated at	04-07-2023 11:06:09 (2 months ago)

Anomalous APIs (4) **Historical APIs**

Successfully called

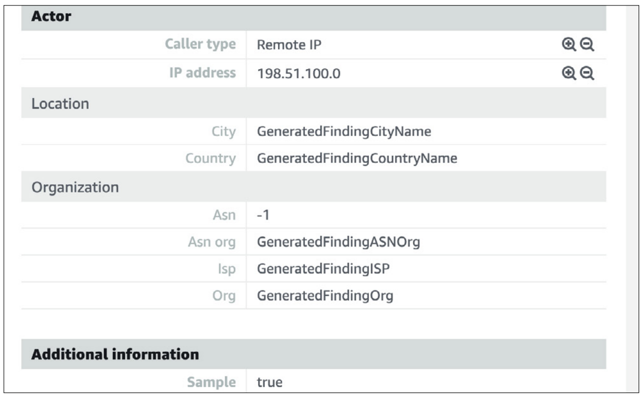
Generated finding API service name three	GeneratedFindingAPINameThree

Figure 8: Example.

The alert was letting us know that a user API key (programmatic access) was being executed in an unusual way – so, it could be an attack.

API key compromises are dangerous because if access permissions are not correctly configured, the attacker can gain access to the whole tenant.

From the alert, we also were able to get important information like the source IP address from the request.


Actor

Caller type	Remote IP
IP address	198.51.100.0

Location

City	GeneratedFindingCityName
Country	GeneratedFindingCountryName

Organization

Asn	-1
Asn org	GeneratedFindingASNOrg
Isp	GeneratedFindingISP
Org	GeneratedFindingOrg

Additional information

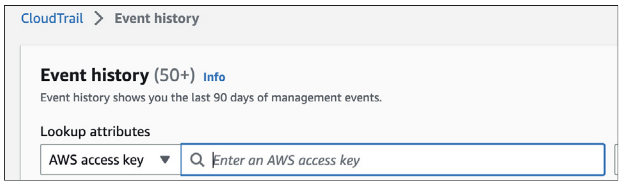
Sample	true

Figure 9: Example of IP address.

The access key was owned by a DevOps based in Argentina, and the request source IP was executed from a European country. So it was clearly something real.

Analysis and containment

The first thing we did was to try digging into *AWS CloudTrail* logs to see what actions had been executed by the attacker, to understand the criticality and define a response plan. Our client didn't have *CloudTrail* enabled, so we needed to search *AWS Event History*, a narrowed API logging feature that is always on. We can filter the results based on different parameters.


CloudTrail > Event history

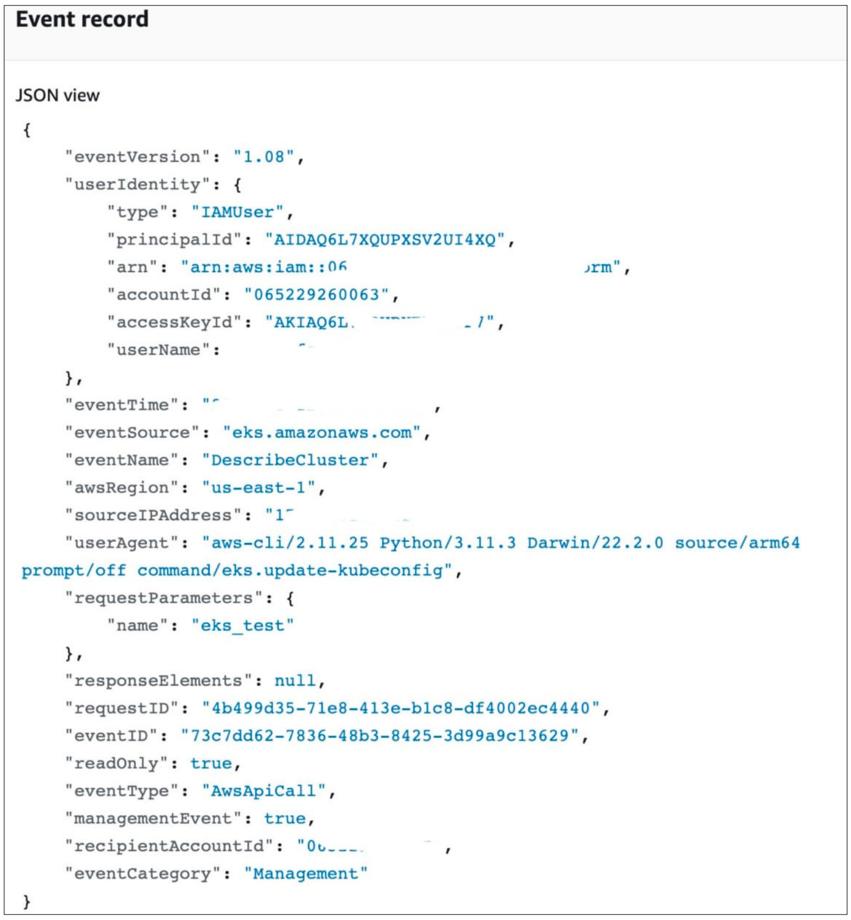
Event history (50+) Info

Event history shows you the last 90 days of management events.

Lookup attributes

AWS access key ▼ | Enter an AWS access key

Figure 10: Filtering by access keys in AWS Event History.


Event record

JSON view

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "AIDAQ6L7XQUPXSV2UI4XQ",
        "arn": "arn:aws:iam::06[redacted]rm",
        "accountId": "065229260063",
        "accessKeyId": "AKIAQ6L[redacted]",
        "userName": [redacted]
    },
    "eventTime": "[redacted],
    "eventSource": "eks.amazonaws.com",
    "eventName": "DescribeCluster",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "1[redacted]
    "userAgent": "aws-cli/2.11.25 Python/3.11.3 Darwin/22.2.0 source/arm64
prompt/off command/eks.update-kubeconfig",
    "requestParameters": {
        "name": "eks_test"
    },
    "responseElements": null,
    "requestID": "4b499d35-71e8-413e-b1c8-df4002ec4440",
    "eventID": "73c7dd62-7836-48b3-8425-3d99a9c13629",
    "readOnly": true,
    "eventType": "AwsApiCall",
    "managementEvent": true,
    "recipientAccountId": "0[redacted],
    "eventCategory": "Management"
}
```

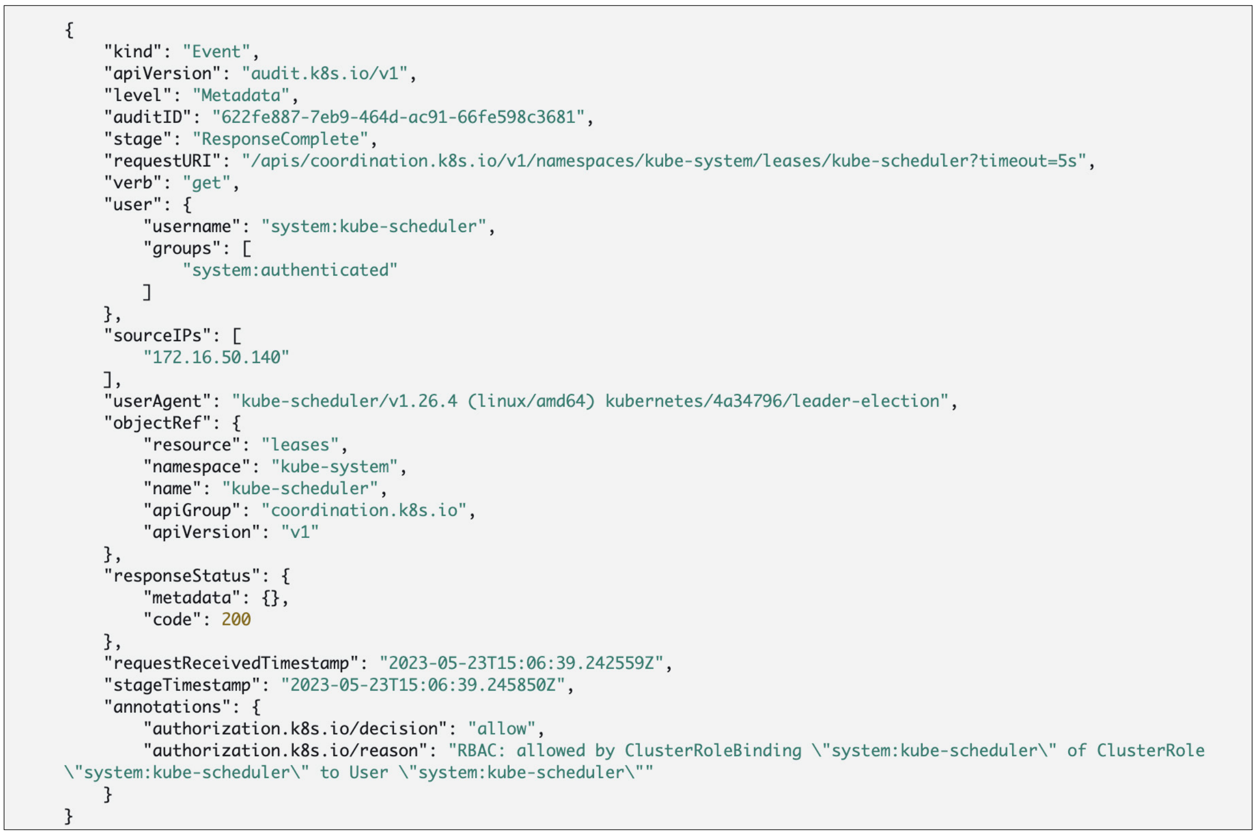
Figure 11: Example of DescribeCluster Event History event.

The attacker was able to enumerate the *AWS* account, and as we can see in this example they were interested in the Kubernetes cluster, and were executing tasks using *AWS* CLI, which means they were accessing it programmatically.

Based on this information, we urged the client to disable the access key, stopping the attacker, and in the meantime we would continue with the research.

We then found that the attacker was attempting to create a login profile, which means they were trying to elevate privileges to keep attacking using the *AWS* console.

But the threat didn't stop there. We discovered from *CloudTrail* logs that they were trying to interact with the *AWS EKS* Kubernetes cluster, so we dug into the *EKS* logs to understand the impact. What we found was scary. Figure 12 shows an example of an *EKS* log.



```
{
    "kind": "Event",
    "apiVersion": "audit.k8s.io/v1",
    "level": "Metadata",
    "auditID": "622fe887-7eb9-464d-ac91-66fe598c3681",
    "stage": "ResponseComplete",
    "requestURI": "/apis/coordination.k8s.io/v1/namespaces/kube-system/leases/kube-scheduler?timeout=5s",
    "verb": "get",
    "user": {
        "username": "system:kube-scheduler",
        "groups": [
            "system:authenticated"
        ]
    },
    "sourceIPs": [
        "172.16.50.140"
    ],
    "userAgent": "kube-scheduler/v1.26.4 (linux/amd64) kubernetes/4a34796/leader-election",
    "objectRef": {
        "resource": "leases",
        "namespace": "kube-system",
        "name": "kube-scheduler",
        "apiGroup": "coordination.k8s.io",
        "apiVersion": "v1"
    },
    "responseStatus": {
        "metadata": {},
        "code": 200
    },
    "requestReceivedTimestamp": "2023-05-23T15:06:39.242559Z",
    "stageTimestamp": "2023-05-23T15:06:39.245850Z",
    "annotations": {
        "authorization.k8s.io/decision": "allow",
        "authorization.k8s.io/reason": "RBAC: allowed by ClusterRoleBinding \"system:kube-scheduler\" of ClusterRole
\"system:kube-scheduler\" to User \"system:kube-scheduler\""
    }
}
```

Figure 12: EKS CloudWatch log example.

From the log we were able to understand that, in a short amount of time, the attacker was able to:

- Create a new service account
- Add a new infected image to the ECR Repository
- Run it into the Kubernetes cluster

This was huge, because the attacker was able to move laterally in a way that is not intended, getting around our initial containment process of removing the access keys.

From here, we needed to remove the service account and isolate the running pod – first in order to stop it, and then for forensic purposes.

LESSONS LEARNED AND CONCLUSION

The main conclusion we can draw from this security incident is that cloud environments are complex and it's hard to be aware of all the boundaries that exist among different services. The rapid elasticity that cloud providers offer to companies can be exploited by technical attackers in order to gain an extra advantage in their activities.

That's why it's important to understand how our system behaves, what the critical components are, and how they relate with each other and with the Mitre Threat Matrix [13].

This is hard in the real world when rapidly growing companies often don't have security teams and rely on security being implemented in the technology areas, which often tend to give greater priority to time-to-market than to shipping a secure product.

Besides that, it’s important to know that there are security practices that can be enabled and configured with minimal effort that can help those teams to achieve both objectives.

Regarding Kubernetes and *AWS* specifics, we outline some good practices that could have prevented this attack:

- **AWS IAM Entities Least Privilege**: The principle of least privilege refers to the practice of granting the minimum set of permissions necessary for a user, group, or role to perform their required tasks. By applying the principle of least privilege, you ensure that entities have only the permissions needed to accomplish their specific responsibilities and nothing more.
- **EKS Private Endpoint Configuration**: The *EKS* endpoint allows a user to interact with the Kubernetes cluster API. This should always be private and only accept requests from trusted internal networks.
- **Kubernetes Cluster Hardening:** Out of the box, the *EKS* cluster doesn’t have security practices implemented. This is why it’s critical to apply security considerations like network segmentation, namespaces segmentation, service account creation prevention, etc.
- **ECR Image Scan on Push:** *ECR* repositories can be configured to scan images before they are stored in order to detect vulnerabilities and misconfigurations.
- **Cloud Workload Protection:** *AWS GuardDuty* or *Falco* [14] allow us to rapidly detect workload attacks in *AWS* and *EKS* environments, like service account creations, malicious images being executed, assets connecting with malicious IPs like TOR, etc.

REFERENCES

[1] Kubernetes Components. Kubernetes. https://kubernetes.io/docs/concepts/overview/components/.

[2] Amazon Elastic Kubernetes Service Documentation. Amazon. https://docs.aws.amazon.com/eks/index.html.

[3] AWS Identity and Access Management Documentation. Amazon. https://docs.aws.amazon.com/iam/index.html.

[4] Amazon CloudWatch Documentation. Amazon. https://docs.aws.amazon.com/cloudwatch/index.html.

[5] AWS CloudTrail Documentation. Amazon. https://docs.aws.amazon.com/cloudtrail/index.html.

[6] Amazon GuardDuty. Amazon. https://docs.aws.amazon.com/guardduty/index.html.

[7] Amazon Elastic Container Registry Documentation. Amazon. https://docs.aws.amazon.com/ecr/index.html.

[8] Tactics. Microsoft. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/.

[9] Using cloud credentials. Microsoft. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/techniques/Using%20Cloud%20Credentials/.

[10] Access the K8S server. Microsoft. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/techniques/Access%20the%20K8S%20API%20server/.

[11] New Container. Microsoft. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/techniques/New%20Container/.

[12] Resource hijacking. Microsoft. https://microsoft.github.io/Threat-Matrix-for-Kubernetes/techniques/Resource%20hijacking/.

[13] MITRE ATT&CK. Cloud Matrix. https://attack.mitre.org/matrices/enterprise/cloud/.

[14] Sysdig. Threat Detection Built on Falco. https://sysdig.com/opensource/falco/.