Understanding the threat landscape for Kubernetes and containerized assets | Microsoft Security Blog

By Microsoft Threat Intelligence

Published: 2025-04-23 · Archived: 2026-04-02 12:45:58 UTC

The dynamic nature of containers can make it challenging for security teams to detect runtime anomalies or pinpoint the source of a security incident, presenting an opportunity for attackers to stay undetected. Microsoft Threat Intelligence has observed threat actors taking advantage of unsecured workload identities to gain access to resources, including containerized environments. Microsoft data showed that in the past year, 51% of workload identities were completely inactive, representing a potential attack vector for threat actors.

Microsoft released and updated the threat matrix for Kubernetes, an active knowledge base for security threats that target Kubernetes clusters, to systematically map the attack surface of Kubernetes. We also worked with MITRE to develop the ATT&CK® for Containers matrix in 2021. As the adoption of containers-as-a-service among organizations rises, Microsoft Threat Intelligence continues to monitor the unique security threats that affect containerized environments.

Threats in Kubernetes environments

Containerized assets (including Kubernetes clusters, Kubernetes nodes, Kubernetes workloads, container registries, container images, and more) are at risk of several different types of attacks. To fully secure containerized workloads, organizations must secure the containers and the code running within them, software dependencies and libraries, continuous integration and continuous delivery (CI/CD) pipelines, runtime, and more.

Threats in Kubernetes environments can come from six primary areas:

- **Compromised accounts**: In cases where Kubernetes clusters are deployed in public clouds (such as Azure Kubernetes Service (AKS) or Google Kubernetes Engine (GKE)), compromised cloud credentials could lead to cluster takeover, as attackers who have access to account credentials can get access to the cluster's management layer.
- **Vulnerable or misconfigured images**: Images that are not updated regularly might contain vulnerabilities that can be exploited in malicious attacks.
- **Environment misconfigurations**: An attacker with access to the Kubernetes API, either through exposed management interfaces or lack of appropriate authentication/authorization controls, could completely take down the server, deploy malicious containers, or hijack the entire cluster.
- **App-level attacks**: Applications could be exploited through several typical methods, such as SQL injection, cross-site scripting, and remote file inclusion.
- **Node-level attacks**: Attackers can gain initial access through nodes (host machines that containers run on) that run on vulnerable code or software, have open management interfaces such as SSH, or run commands from the cloud control plane. There is also the risk of pod escape, where a compromised pod can provide access to the node or to other pods in the cluster.

- **Unauthorized traffic**: Insecure networking between the different containers within the cluster and between the pods and outside world could be subject to malicious traffic if not secured.

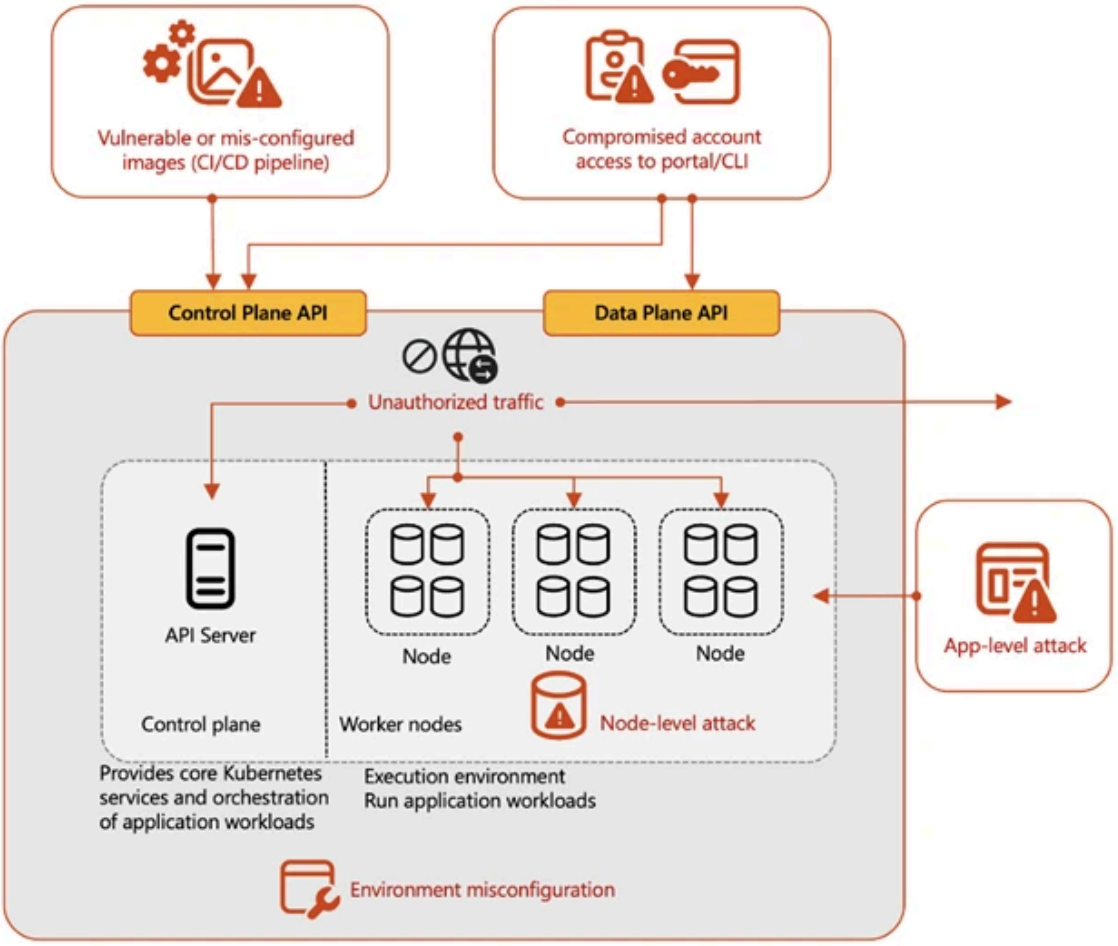

Figure 1. Overview of attacks against Kubernetes environments

Case study: Password spray attack leads to containers being used for cryptomining

In the past year, Microsoft Threat Intelligence has observed AzureChecker threats (tracked as Storm-1977) launching password spray attacks against cloud tenants in the education sector. The attack involves the use of *AzureChecker.exe*, a Command Line Interface (CLI) tool that is being used by a wide range of threat actors.

We observed that *AzureChecker.exe* connected to *sac-auth[.]nodefunction[.]vip* to download AES-encrypted data that when decrypted reveals the list of password spray targets. The tool then also accepted the file *accounts.txt*, which contained the username and password combinations to be used for the attack, as input. The threat actor then used the information from both files and posted the credentials to the target tenants for validation.

Microsoft Threat Intelligence was able to observe an instance of successful account compromise and found that the threat actor leveraged a guest account to create a resource group within the compromised subscription. The threat actor then created more than 200 containers within the resource group and used them for cryptomining activity.

Securing containerized environments

The following best practices can help secure containerized assets against commonly observed threats.

Secure code prior to deployment

Ensuring that containers have secure code prior to deployment is essential to preventing issues during deployment and runtime. To facilitate this, Microsoft Defender for Cloud scans container images for vulnerabilities and misconfigurations and alerts customers of issues before a container is deployed.

Defender for Cloud DevOps also provides visibility into the security posture of the CI/CD platform. Additional best practices such as restricting access to DevOps tooling, using a secret store instead of hard-coding secrets in code or documentation, and using hardened DevOps workstations to build and deploy code can help prevent security issues before code is deployed.

Secure container deployment and runtime

Container deployment refers to the phase of the lifecycle where container images are pulled from the static container registry to be run on virtual machines hosts. During deployment, you should ensure the following best security practices:

- **Ensure containers are immutable:** Prevent patches from running containers whenever possible. As best practice, if you notice that a running container needs updates, you should rebuild the image and deploy the new container. Introducing new code in running containers can introduce new vulnerabilities, bypass secure development lifecycle protections, as well as pose an operational risk in case a container is restarted and run again with the original container image content without any runtime modifications.
- **Leverage Admission Controllers:** Configure policies to prevent containers from being deployed from untrusted registries, from running out of alignment with the minimal Pod Security Standard that fits the pod requirement (such as restricting root privileges), and from utilizing too many resources in the event of a denial-of-service attack. These can be enforced with Azure Policy Add-On for Kubernetes.
- **Gate deployments of vulnerable images:** Ensure that the containers being deployed are free of vulnerabilities and misconfigurations by running a vulnerability scan in the Build and Ship phases. Any image with high or critical severity vulnerabilities should be blocked from deployment.

Container runtime refers to the phase of the lifecycle where containers are running on the virtual hosts. During runtime, monitor your running containers for any new vulnerabilities that might have been introduced during runtime. In cases where a container image was not scanned in build time or in registry before being deployed to the cluster, Microsoft Defender Vulnerability Management supports Azure vulnerability assessments.

Additionally, monitor each node, pod, and container during runtime for any sort of anomalous or malicious activity that may be occurring:

- Look for malicious API calls and unusual activity using a monitoring system to identify any unusual Kubernetes API server requests for malicious activity. Defenders can query Kubernetes API calls in Defender XDR advanced hunting using the *CloudAuditEvents* table.
- For AKS clusters, Container Insights offers the ability to collect Syslog events from Linux nodes, to then be accessed within Azure's built-in workbooks.

Defender for Containers' Agentless discovery for Kubernetes provides API-based discovery of Kubernetes clusters, their configurations, and deployments. Defender for Cloud also identifies runtime threats at both the API level and

the workload level. Additionally, organizations can use Microsoft Defender for Cloud to identify and remediate attack paths to address any potential attack vectors.

Secure user accounts and permissions

Attackers are increasingly using compromised identities for initial access and for establishing long-term persistence within an environment. If a compromised user has access to Kubernetes services, an attacker could use that identity to access those services using portal access or the command-line interface. In cases where Kubernetes clusters are deployed in public clouds (such as AKS in Azure or GKE in Google Cloud Platform (GCP)), compromised cloud credentials could lead to cluster takeover as attackers who have access to account credentials can get access to the cluster's management layer.

The following recommendations, focused on requiring strong authentication to services and following the principle of least privilege, can help secure cloud credentials from compromise:

- Use strong authentication when exposing sensitive interfaces to the internet. For example, attacks were observed against exposed Kubeflow and Argo workloads that were not configured to use OpenID Connect or other authentication methods.
- Use strong authentication methods to the Kubernetes API to help prevent attackers from gaining access to the cluster even if valid credentials such as *kubeconfig* were achieved. For example, in AKS use Entra ID authentication instead of basic authentication. By using Entra ID authentication, a short-lived credential of the cluster is retrieved after authenticating to Entra ID.
- Avoid using the read-only endpoint of Kubelet in port 10255, which doesn't require authentication. In newer versions of managed clusters, this port is disabled.
- Implement multifactor authentication (MFA).
- Configure the Kubernetes role-based access controls (RBAC) for each user and service accounts to have only necessary permissions. This applies also to other external authorization providers such as Azure RBAC in AKS.
- In a managed cluster, Kubernetes credentials are often retrieved or generated by the cloud provider through API call. To reduce the attack surface, grant permissions to the cloud provider API only to necessary accounts. In the case of Azure, make sure that only required identities have permissions to call: */subscriptions/resourceGroups/providers/Microsoft.ContainerService/managedClusters/listClusterUserCredential*
- The *kubeconfig* file can contain credentials of accounts that allow interaction with a cluster. By applying the least privilege principle to all accounts, you can limit the impact of an account compromised through the *kubeconfig* file. To further limit misuse of the *kubeconfig* file, enable Microsoft Entra-based authentication to AKS and disable the local admin account, avoiding the use of the *kubeconfig* file altogether.

The Kubernetes project also lists the following recommendations for permissions and role assignment best practices:

- Avoid wildcard permissions, especially to all resources.
- Use *RoleBinding* instead of *ClusterAdminBinding* to give access within a namespace.
- Avoid adding users to the *system:master* group as it bypasses RBAC.
- Use impersonation rights for admins instead of adding to the cluster admin role. Audit and monitor when impersonation is being done.

- Avoid granting the escalate or bind permissions to roles when not needed, audit and monitor when escalation is being made.
- Avoid adding users to the *system:unauthenticated* group.
- Limit permissions to issue certificate signing requests (CSR) and certificates.
- Avoid granting users with *create* rights on service accounts/token, which could be exploited to create *TokenRequests* and issue tokens for existing service accounts.
- Users with control over *validatingwebhookconfigurations* or *mutatingwebhookconfigurations* can control webhooks that can read any object admitted to the cluster, and in the case of mutating webhooks, also mutate admitted objects

Secure container images

- Secure the CI/CD environment. Secure code repositories and CI/CD environment by placing gates to restrict unauthorized access and modification of content. This can include enforcing RBAC permissions to access and make changes to code, artifacts and build pipelines, ensure governed process for pull-request approval, apply branch policies and others.
- Apply image assurance policy to evaluate container images against vulnerabilities, malware, exposed secrets or other policies. By ensuring consistent and comprehensive image assurance policy across the build, ship, and run development stages. One approach of ensuring images pass assurance or compliance checks it to sign the container images, so the image signature can be checked downstream when deploying to Kubernetes clusters at runtime.
- Take and store data backups from pod-mounted volumes for critical workloads. Ensure backup and storage systems are hardened and kept separate from the Kubernetes environment to prevent compromise.

Restrict network traffic

The Kubernetes API server is the gateway to the cluster. Restricting access to the API server, as well as restricting how pods can communicate, can prevent unwanted access to the clusters management, even if an adversary gained valid credentials to the cluster. The following best practices can help harden clusters against attacks.

- Restrict access to the API server using intrusion detection signatures, network policies, and a web application firewall to block traffic at network boundaries to pods and services in a Kubernetes cluster. In managed clusters, cloud providers often support native built-in firewalls, which can restrict the IP addresses that are allowed to access the API server.
    - Implement a Next-Generation Firewall (NGFW) such as Azure Firewall, which offers a solution for AKS to monitor inbound and outbound traffic. You can integrate Azure Firewall as well as several third-party firewalls into Azure Sentinel to accumulate all logs into a centralized location. AKS also supports Retina, a cloud-agnostic platform for analysis of Kubernetes' workload traffic.
    - Use Kubernetes network policies to control traffic and manage how pods communicate.
    - Adapt a network intrusion prevention solution to a Kubernetes environment if needed, in order to route network traffic destined to services through the security solution. In some cases, this can be done by deploying a containerized version of a network intrusion prevention solution to the Kubernetes cluster and be part of the cluster network, and in some cases, routing ingress traffic to Kubernetes services through an external appliance, requiring that all ingress traffic only come from such an appliance.

- Enable Just In Time (JIT) access to the API server through Microsoft Entra conditional access. Employing JIT elevated access to the Kubernetes API server helps reduce the attack surface by allowing access only at specific times, and through a governed escalation process. Enabling JIT access in Kubernetes is often done together with OpenID authentication, which includes processes and tools to manage JIT access. One example of such OpenID authentication is Azure Active Directory authentication to Kubernetes clusters. The JIT approval is performed in the cloud control plane level. Therefore, even if attackers have access to account credentials, their access to the cluster is limited.
- Limit access to services over network. Avoid exposing sensitive interfaces insecurely to the internet or limit access to it. Sensitive interfaces include management tools and applications that allow the creation of new containers in the cluster. Some of those services do not use authentication by default and are not intended to be exposed. Examples of services that were exploited include Weave Scope, Apache NiFi, and more.
    - If services need to be exposed to the internet and are exposed using a LoadBalancer service, use IP restriction (*loadBalancerSourceRanges*) when possible. This reduces the attack surface of the application and can prevent attackers from being able to reach the sensitive interfaces.

Detection details

Microsoft Defender for Cloud

Microsoft Defender for Containers provides security alerts on the cluster level and on the underlying cluster nodes by monitoring both the control plane (the API server) and the containerized workload itself.

- Exposed Postgres service with trust authentication configuration in Kubernetes detected (Preview)
- Exposed Postgres service with risky configuration in Kubernetes detected (Preview)
- Attempt to create a new Linux namespace from a container detected
- A history file has been cleared
- Abnormal activity of managed identity associated with Kubernetes (Preview)
- Abnormal Kubernetes service account operation detected
- An uncommon connection attempt detected
- Attempt to stop apt-daily-upgrade.timer service detected
- Behavior similar to common Linux bots detected (Preview)
- Command within a container running with high privileges
- Container running in privileged mode
- Container with a sensitive volume mount detected
- CoreDNS modification in Kubernetes detected
- Creation of admission webhook configuration detected
- Detected file download from a known malicious source
- Detected suspicious file download
- Detected suspicious use of the nohup command
- Detected suspicious use of the useradd command
- Digital currency mining container detected
- Digital currency mining related behavior detected
- Docker build operation detected on a Kubernetes node
- Exposed Kubeflow dashboard detected

- Exposed Kubernetes dashboard detected
- Exposed Kubernetes service detected
- Exposed Redis service in AKS detected
- Indicators associated with DDOS toolkit detected
- K8S API requests from proxy IP address detected
- Kubernetes events deleted
- Kubernetes penetration testing tool detected
- New container in the kube-system namespace detected
- New high privileges role detected
- Possible attack tool detected
- Possible backdoor detected
- Possible command line exploitation attempt
- Possible credential access tool detected
- Possible Cryptocoinminer download detected
- Possible Log Tampering Activity Detected
- Possible password change using crypt-method detected
- Potential port forwarding to external IP address
- Potential reverse shell detected
- Privileged container detected
- Process associated with digital currency mining detected
- Process seen accessing the SSH authorized keys file in an unusual way
- Role binding to the cluster-admin role detected
- Security-related process termination detected
- SSH server is running inside a container
- Suspicious file timestamp modification
- Suspicious request to Kubernetes API
- Suspicious request to the Kubernetes Dashboard
- Potential crypto coin miner started
- Suspicious password access
- Possible malicious web shell detected
- Burst of multiple reconnaissance commands could indicate initial activity after compromise
- Suspicious Download Then Run Activity
- Access to kubelet kubeconfig file detected
- Access to cloud metadata service detected
- MITRE Caldera agent detected

Recent updates to Microsoft Defender for Cloud enhance its container security capabilities from development to runtime. Defender for Cloud now offers enhanced discovery, providing agentless visibility into Kubernetes environments, tracking containers, pods, and applications. The updates also strengthen security posture through continuous and granular scanning from build to runtime, helping maintain compliance and secure configurations across the SDLC.

Defender for Cloud's native integration with Defender XDR enables threat protection with real-time monitoring, prioritizing vulnerabilities based on risk and enabling SOC analysts to detect and respond to threats faster through

rich contextual insights and cloud-native response tools

Microsoft Defender for Endpoint

Microsoft Defender for Endpoint also detects threats on endpoints running container hosts, focusing on suspicious behavior commonly observed on endpoints, including stealing locally stored credentials for accessing the cloud, downloading and running malicious images, and privilege escalation from dockers to hosts.

Microsoft Defender External Attack Surface Management

Microsoft Defender External Attack Surface Management detects Docker and Kubernetes instances with known vulnerabilities or misconfigurations using the following alerts:

- ASI: Open Docker Daemon API Service
- ASI: Unauthenticated Kubelet API

Microsoft Security Copilot

Security Copilot customers can use the standalone experience to create their own prompts or run the following pre-built promptbooks to automate incident response or investigation tasks related to this threat:

- Incident investigation
- Microsoft User analysis
- Threat actor profile
- Threat Intelligence 360 report based on MDTI article
- Vulnerability impact assessment

Note that some promptbooks require access to plugins for Microsoft products such as Microsoft Defender XDR or Microsoft Sentinel.

Hunting queries

In addition to the below hunting queries, the open-source tool KubiScan, developed by CyberArk Labs, can be used to scan clusters for risky permissions and users. Results can be used to manage RBAC within the environment and eliminate unnecessary permissions; it can also be used in incident response to identify the potential exposure of compromised users.

Microsoft Defender XDR

In addition to viewing alerts and incidents within Defender XDR, you can now use Azure Resource Manager (ARM) logs as well as Kubernetes audits logs for further investigation using the advanced hunting capabilities.

If a hunting query provides a good indicator of malicious or unsanctioned activity in your environment, you can create a custom rule detection in the Defender XDR portal by going to the **Advanced unting** page > **Manage rules** > **Create custom detection**.

Privileged pod deployment

The following query surfaces deployment of a privileged pod:

```
CloudAuditEvents
| where Timestamp > ago(1d)
| where DataSource == "Azure Kubernetes Service"
| where OperationName == "create"
| where RawEventData.ObjectRef.resource == "pods" and isnull(RawEventData.ObjectRef.subresource)
| where RawEventData.ResponseStatus.code startswith "20"
| extend PodName = RawEventData.RequestObject.metadata.name
| extend PodNamespace = RawEventData.ObjectRef.namespace
| mv-expand Container = RawEventData.RequestObject.spec.containers
| extend ContainerName = Container.name
| where Container.securityContext.privileged == "true"
| extend Username = RawEventData.User.username
| project Timestamp, AzureResourceId , OperationName, IPAddress, UserAgent, PodName, PodNamespace, ContainerName, Username
```

Exec command

The following query identifies use of the *exec* command in the *kube-system* namespace:

```
CloudAuditEvents
| where Timestamp > ago(1d)
| where DataSource == "Azure Kubernetes Service"
| where OperationName == "create"
| where RawEventData.ObjectRef.resource == "pods" and RawEventData.ResponseStatus.code == 101
| where RawEventData.ObjectRef.namespace == "kube-system"
| where RawEventData.ObjectRef.subresource == "exec"
| where RawEventData.ResponseStatus.code == 101
| extend RequestURI = tostring(RawEventData.RequestURI)
| extend PodName = tostring(RawEventData.ObjectRef.name)
```

```
| extend PodNamespace = tostring(RawEventData.ObjectRef.namespace)

| extend Username = tostring(RawEventData.User.username)

| where PodName !startswith "tunnelfront-" and PodName !startswith "konnectivity-" and PodName !startswith "aks-link"

| extend Commands =  extract_all(@"command=([^\&]*)", RequestURI)

| extend ParsedCommand = url_decode(strcat_array(Commands, " "))

| project Timestamp, AzureResourceId , OperationName, IPAddress, UserAgent, PodName, PodNamespace, Username, ParsedCommand
```

Cluster-admin role binding

The following query identifies the creation of *cluster-admin* role binding:

```
CloudAuditEvents

| where Timestamp > ago(1d)

| where OperationName == "create"

| where RawEventData.ObjectRef.resource == "clusterrolebindings"

| where RawEventData.ResponseStatus.code startswith "20"

| where RawEventData.RequestObject.roleRef.name == "cluster-admin"

| mv-expand Subject = RawEventData.RequestObject.subjects

| extend SubjectName = tostring(Subject.name)

| extend SubjectKind = tostring(Subject["kind"])

| extend BindingName = tostring(RawEventData.ObjectRef.name)

| extend ActionTakenBy = tostring(RawEventData.User.username)

| where ActionTakenBy != "acsService" //Remove FP

| project Timestamp, AzureResourceId , OperationName, ActionTakenBy, IPAddress, UserAgent, BindingName, SubjectName, SubjectKind
```

References

- KubiScan
- Learn Kubernetes Basics

Learn more

For the latest security research from the Microsoft Threat Intelligence community, check out the Microsoft Threat Intelligence Blog: https://aka.ms/threatintelblog.

To get notified about new publications and to join discussions on social media, follow us on LinkedIn at https://www.linkedin.com/showcase/microsoft-threat-intelligence, and on X (formerly Twitter) at https://twitter.com/MsftSecIntel.

To hear stories and insights from the Microsoft Threat Intelligence community about the ever-evolving threat landscape, listen to the Microsoft Threat Intelligence podcast: https://thecyberwire.com/podcasts/microsoft-threat-intelligence.
