cymulate

Native Cloud Defenses vs Kubernetes Attacks

Extended analysis of cloud native security tools against Kubernetes attack simulations

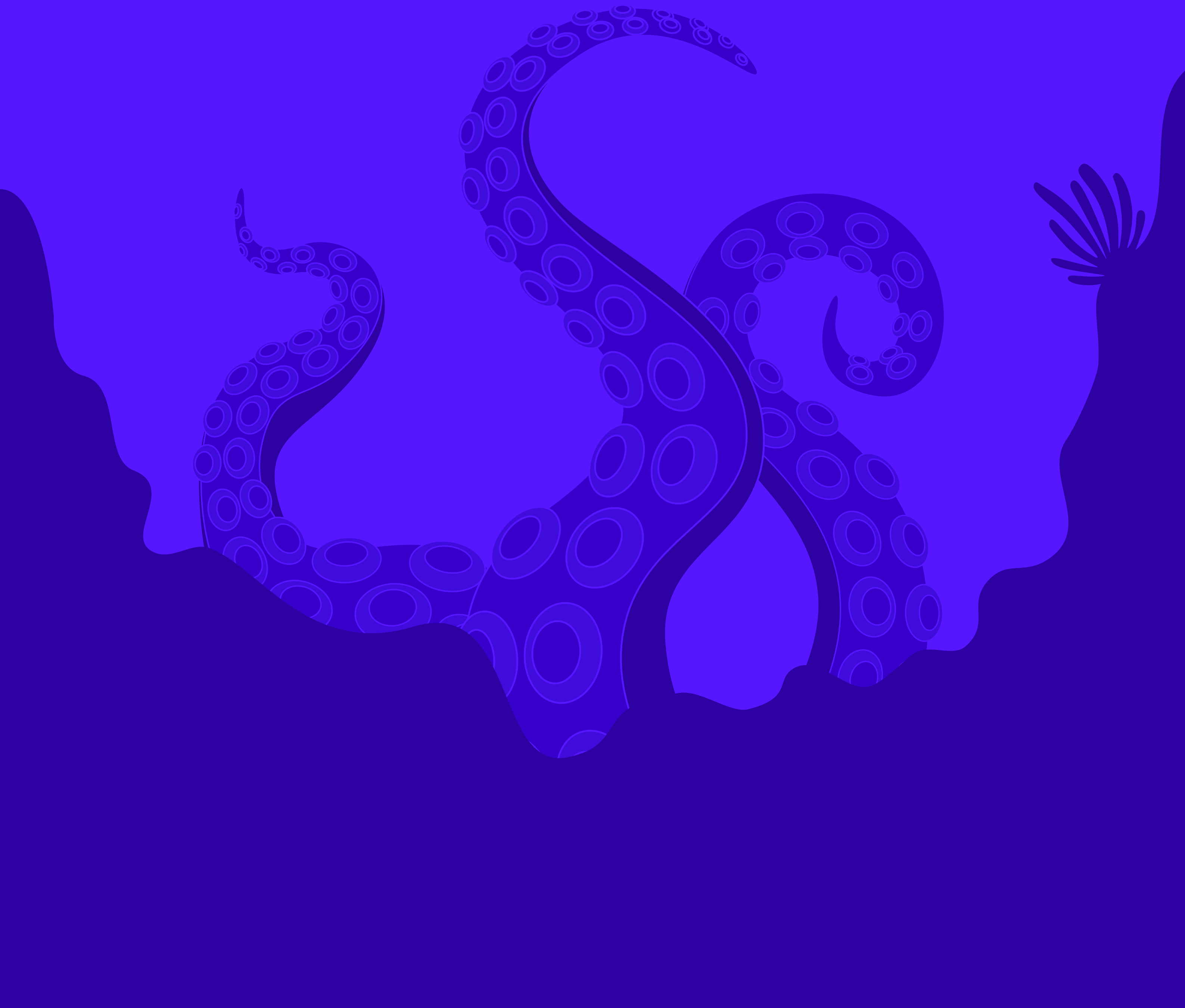


Table of Contents

01 Introduction

Kubernetes has become the de facto standard for deploying applications in the cloud as organizations of all sizes (from small to medium-sized businesses and large enterprises and multi-nationals) run business-critical applications on Kubernetes clusters hosted by leading cloud services providers like Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP).

However

The “standard” suite of native security tools offered by these leading cloud providers is simply not sufficient to detect many of the latest cyberattacks targeting your Kubernetes (K8) clusters.

While any cloud-based Kubernetes defense should start with the available native security tools from these cloud providers, it cannot end there. Businesses that place their trust solely in these tools for monitoring and alerting potential attacks on their Kubernetes containers are placing themselves at risk of a breach. These tools alone failed to detect more than half of the Kubernetes-specific threat activities our threat research team conducted.

Put to the Test

The **Cymulate Threat Research Group** decided to once again put the standard security tools of the three major cloud providers to the test by running a series of Kubernetes attack simulations to determine if their native security tools for monitoring and alerting these attacks could prevent and detect the threat activity. This time, we expanded the test scenarios from 14 to 29 for a more comprehensive test of the native security tools.

And, while the native tools do have value in enhancing cybersecurity resilience, they exhibit significant gaps in detection and alerting on Kubernetes-specific threat activity by themselves.

The results of our latest breach and attack simulation tests highlight a varying and concerning low degree of efficacy when used as the sole monitoring and alerting system for Kubernetes environments.

Continue reading to find out the results of our tests and what steps you can take to prevent this from happening in your Kubernetes environment.

Native cloud provider tools alone are **not sufficient** to detect many attacks on business-critical applications and workloadsrunning in Kubernetes clusters.

Native tools alone exhibit significant gaps with **less than 50% detection rate** for Kubernetes-specific threat activity.

The failure of these native tools to detect threat activity in K8 clusters **could leave your business KO'd** by a threat actor if used as the sole defense mechanism for Kubernetes.

And at Cymulate, we believe

that is not OK

02

K8 Clusters Under Attack

The past year has seen a wave of attacks targeting Kubernetes clusters, including role-based access control (RBAC) backdoor attacks, the infamous Scarleteel and AWS infrastructure attacks, and attacks involving Dero and Monero crypto miners.

These attacks are complex operations that highlight the sophisticated tactics used by threat actors targeting cloud environments to steal credentials and exploit common cloud misconfigurations like over-provisioned access rights and permissions.

The role-based access control backdoor attacks occurred from a misconfigured API server that allowed unauthenticated requests from anonymous users to get information about the K8 cluster, which they used to exploit the environment further and gain persistence.

These attack campaigns underscore the critical importance of securing Kubernetes clusters against common misconfigurations and highlight the need for ongoing vigilance, regular security validation, and the application of security best practices in cloud environments.

Need to secure Kubernetes clusters against the latest threat actor tactics

Role-based access control (RBAC) backdoor attacks

- Misconfigured K8 API server
- Allows unauthenticated requests from anonymous users
- Gain information about K8 cluster
- Used to exploit the environment further and gain persistence

Scarleteel and AWS infrastructure attacks

- Exploit vulnerable public-facing service within a K8 cluster
- Deploy a crypto miner as a decoy to distract from primary objectives
- Establish persistence, exfiltrate data, steal credentials

Dero and Monero crypto miner attacks

- Locates K8 clusters with anonymous access enabled
- Elevate privileges using DaemonSets and host root mount
- Deploy crypto mining operations consuming cloud resources

03

The Kubernetes Test Environment

The Cymulate Threat Research Group established Kubernetes environments in each of the three leading cloud providers – Amazon Web Services (AWS), Microsoft Azure, and Google Cloud Platform (GCP). Each cloud provider had its native defense mechanisms enabled:

- Azure Cloud Defender
- AWS GuardDuty
- GCP Command Center

More than half (57%) of the threat activity tested was not detected by the native cloud defense mechanisms.

The configurations for each security solution were verified by third parties outside of Cymulate: either by authorized partners, affiliated and certified by the respective cloud provider; or directly by the support teams of the cloud provider in question.

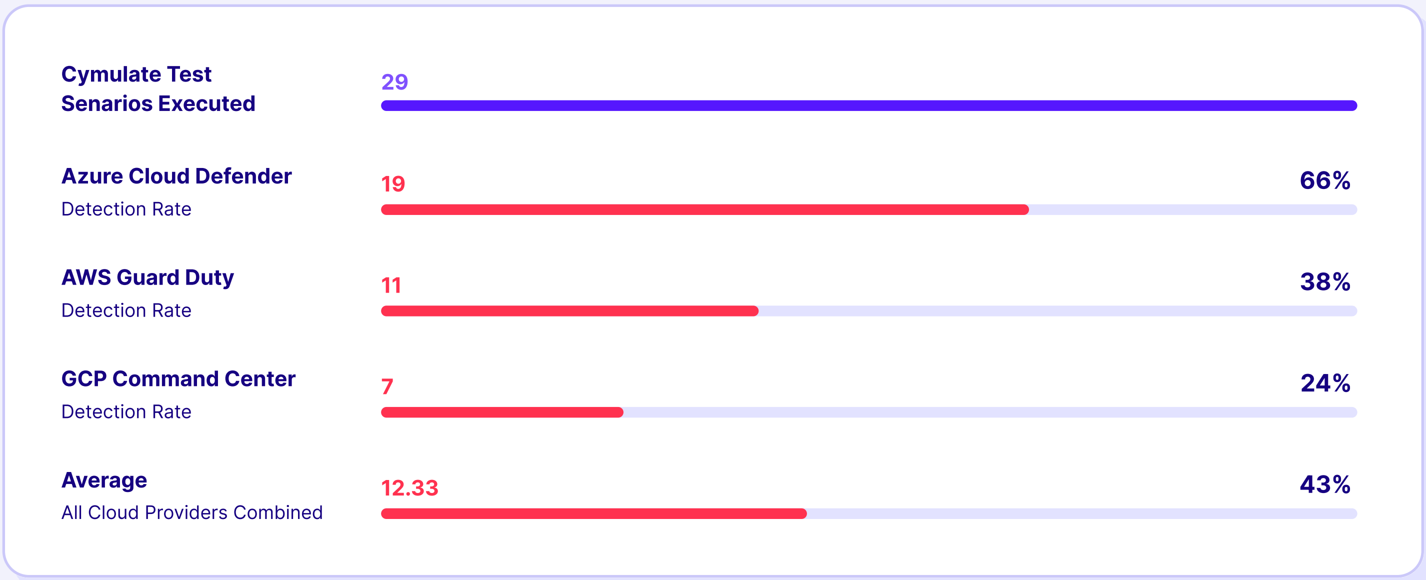
Cymulate then used our Breach & Attack Simulation solution to conduct 29 automated attacks targeting the Kubernetes containers in each cloud environment.

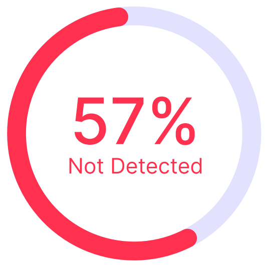
The results below were both illuminating and concerning for any organization using a cloud-hosted Kubernetes environment and relying solely on the cloud native security tools for protection.

04

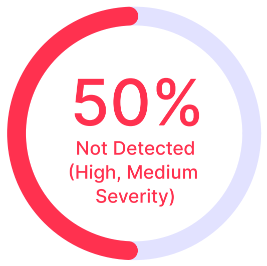
Analysis of Results: Low Detection Rates

The chart below summarizes the detection rates of the standard native cloud security tools used to defend Kubernetes clusters for each cloud service provider.





Of the 29 test scenarios executed by Cymulate, the overall detection rate for all three cloud providers scored between 24% and 66% successful defense, with an **average score of just 43%**. This means that **more than half (57%)** of the simulation attack scenarios carried out by Cymulate, were **not detected** by the native security tools across all cloud providers combined (Azure, AWS, GCP).



Of the 29 tests conducted, 15 received a Cymulate severity score of **High or Medium**, and the average across all cloud providers security tools detected only half (50%) of those higher priority attack simulations, highlighting their potential exposure to a real attack. The outcome of all 29 tests conducted indicates that the native tools for each of the three main cloud vendors **are not fully capable** of meeting the specific needs of successfully defending cloud-hosted Kubernetes clusters on their own.

View the complete list of 29 attack scenarios in the Kubernetes Testing Protocols Appendix at the end of this document.

Kubernetes Test Scenarios Using Threat Actor Tactics and Techniques

Cloud data breaches are often the result of misconfigured settings in the cloud infrastructure and Kubernetes environment. Threat actors use various tactics and techniques that take advantage of these misconfigurations to exploit Kubernetes clusters.

The Cymulate Threat Research Team has established advanced test scenarios in the Cymulate platform that are carefully crafted to simulate the tactics and techniques used by threat actors to exploit common misconfigurations in a Kubernetes environment.

The research team created 29 test cases using 15 different attack techniques that cover the seven tactics from initial access to discovery across the **MITRE ATT&CK** matrix. Threat actors commonly use these tactics and techniques to gain initial access, execute payloads, establish persistence, escalate privileges, evade defenses, access credentials, and discover resources in a Kubernetes environment.

The ability to detect these malicious behaviors is critical to stopping a cloud data breach before the threat actors reach the final stages of lateral movement, command and control, exfiltration, and final impact.

Kubernetes Test Scenarios Using Threat Actor Tactics and Techniques

Initial Access

T0001
1 Technique
1 Test Scenario

Exploit Public Facing Application
- Anonymous Access Granted

Execution

T0002
3 Techniques
11 Test Scenarios

Container Administration Command
- Stop Apt-Daily Upgrade
- Create Docker In Docker
- Execute Commands
- Exploit Nohup
- Terminate Processes
- Digital Currency Mining
- Suspicious File Download

Deploy Container
- Create Container in Kube-System Namespace

Inter-Process Communication
- Host IPC Privilege
- Host PID Privilege
- Insecure Capabilities

Persistence

T0003
1 Technique
1 Test Scenario

Create Account
- Exploit Useradd

Privilege Escalation

T0004
4 Techniques
8 Test Scenarios

Abuse Elevation Control Mechanism
- Anomalous Role Binding Behavior
- Create High Privilege Role

Escape to Host Create Linux Namespace
- Create Privileged Container
- Host Path Mount
- Writeable Host Path Mount
- Create Linux Namespace

Exploitation for Privilege Escalation
- Role Binding: Cluster - Admin Role

Valid Accounts
- Admin Access Default Service Account

Defense Evasion

T0005
1 Technique
2 Test Scenarios

Indicator Removal (Clear Command History)
- Clear History File
- Delete Events

Credential Access

T0006
2 Techniques
3 Test Scenarios

Steal Application Access Token
- App Credentials In Configuration Files

Unsecured Credentials
- Access To Kubelet Kubeconfig File
- List Secrets

Discovery

T0007
3 Techniques
3 Test Scenarios

Cloud Service Discovery
- Successful Anonymous Access

Container & Resource Discovery
- Anonymous Access To Kubelet Service

Network Service Discovery
- Host Network Access

05 Kubernetes Research Findings

This research analyzes the ability of native cloud security tools (Azure Cloud Defender, AWS GuardDuty, and GCP Command Center) to detect common attack techniques under an assumed breach mindset. The tests focus on the detection of these attacks and alerting security operations teams to the fact that stealthy threat actors are operating inside their Kubernetes environment.

Below is a summary of the key findings identified by this Kubernetes research:

Access Rights and Permissions

Over-provisioned access rights and permissions continue to be one of the leading causes of cloud data breaches and failed detections within this Kubernetes experiment.

Gaps in Detection Capabilities

The findings underscore a significant gap in the detection capabilities of native cloud defense mechanisms against Kubernetes attack vectors. More than half of the attacks went undetected, potentially leaving cloud environments susceptible to sequential attack attempts.

Complexity of Kubernetes

Kubernetes, while powerful, is inherently complex. Its dynamic nature, native elasticity, and the rapid rate of both code and platform changes mean that traditional defense mechanisms might not be sufficient. This experiment highlights the importance of specialized tools or additional layers of security, especially for Kubernetes deployments.

Global Exploits

If such a small fraction of attacks can be detected in a controlled environment with verified configurations, it raises a significant concern about the broader state of Kubernetes deployments worldwide. If these findings are indicative of the global situation, countless Kubernetes clusters could be at risk both on-premise and in the cloud if specialized defensive tools are not implemented to strengthen Kubernetes security.

False Sense of Security

Organizations might have a false sense of security, believing that native cloud defense mechanisms provide comprehensive protection. Continuous monitoring, penetration testing, and threat modeling specific to Kubernetes are crucial. Specialized detection and alerting systems are necessary to ensure that threat activity is identified, and appropriate personnel are made aware – even in cases where some of the threat activity is blocked or otherwise stopped.

The Dangers of a Silent Threat

The absence of detection does not imply the absence of an attack. Silent breaches, where the victim remains unaware of an ongoing compromise, can be the most devastating. Over time, undetected malicious actors can gain deeper access, exfiltrate sensitive data, or lay dormant only to activate at a strategically opportune moment. Even when looking at the results of this round of testing, actions that could have alerted organizations to targeting and progressive infiltration attempts were not detected – information that could have allowed an organization to take proactive steps to head off an attacker.

06

Cymulate Recommendations

The number one recommendation from Cymulate:

Third Party Security Tools Are Needed

Organizations need to supplement the native defense mechanisms of all three cloud providers with additional third-party security tools for better detection and prevention of their Kubernetes environments.

In addition, Cymulate also recommends:

Privilege & Access Rights

Manage and maintain permissions and access rights using the principle of least privilege.

Continuous Training & Awareness

Keeping teams updated about the latest attack vectors and defense mechanisms is critical.

Regular Audits

Beyond initial setup verification, periodic audits, and red team exercises can help identify potential blind spots within the organization's security posture. This is especially critical for Kubernetes environments, where frequent code updates and the elastic nature of the platform itself ensure that the attack surface will change on an ongoing basis.

Multi-Layered Security

Employ a multi-layered approach that includes network segmentation, role-based access controls, and runtime security. Invest in advanced monitoring tools and techniques to identify unusual patterns, signaling potential breaches.

Ongoing Testing

The utilization of platforms like Cymulate can allow organizations to perform regular automated testing on a continuous basis. This permits the organization to safely simulate threat activity and observe the results, confirming where controls are acting as expected and where changes are required to defend the organization.

07
Conclusion: Additional Tooling Required

While the major cloud providers offer robust and evolving security solutions, the fast-paced and intricate world of Kubernetes requires a more flexible and dedicated approach.

The findings of this experiment show the need for a call to action from organizations, cloud providers, and the security community at large. With Kubernetes becoming a mainstay in modern infrastructure, ensuring its security is not just an organizational concern - it is a global imperative.

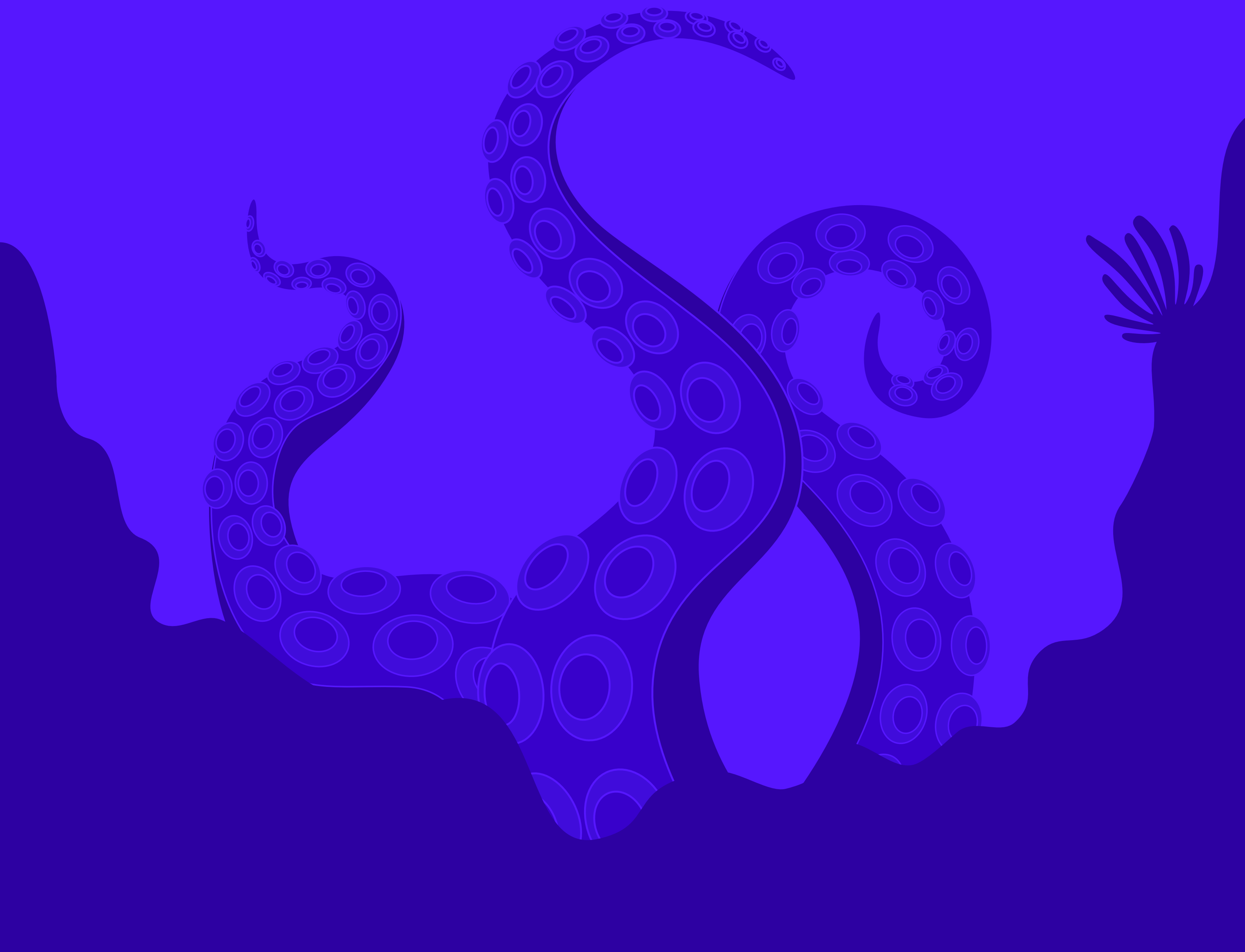
The Kubernetes-using world must recognize and act on these vulnerabilities before they manifest into larger, more catastrophic events. As supply-chain attacks continue to gain popularity and frequency, organizations should also mandate that vendors and suppliers remain aware of these concerns around Kubernetes.

Cloud-native Kubernetes clusters require similar tooling as on-premise deployments to stay protected.

Ensuring Kubernetes security is, and will remain, of key importance to all organizations, regardless of how little or how much they use the platform themselves and regardless of the cloud provider(s) they choose to host the platform on.

Appendix

Kubernetes Testing Protocol



The following testing operations were executed across the Mitre Att&ck chain against the Kubernetes environments configured for each cloud provider: Azure, AWS, and GCP.

Initial Access (TA0001)

Anonymous Access Granted

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Not Detected

- **Category: Privilege Escalation**
- **Attack Technique: Exploit Public Facing Application**
- **Type of Attack:** The attack involves creating a ClusterRole with read-only permissions and binding it to the system:anonymous user. This allows unauthenticated users to access sensitive cluster information, which can facilitate further attacks or reconnaissance activities. The consequences include unauthorized data access, potential information leakage, and an increased risk of subsequent attacks.
- **Use Case Flow:**
  - Initialization. The attack initializes by loading the Kubernetes configuration and creating an API client for RBAC operations.
  - ClusterRole Creation. The attack creates a read-only ClusterRole with permissions to get, list, and watch various resources such as pods, services, deployments, jobs, and more.
  - ClusterRoleBinding Creation. The attack binds the newly created ClusterRole to the system:anonymous user, allowing unauthenticated access to the specified resources.
  - Cleanup (Optional). If specified, the attack cleans up by deleting the created ClusterRole and ClusterRoleBinding to remove traces of the attack.
- **Research Notes:** This attack highlights the risks of granting broad access permissions to unauthenticated users. By binding a ClusterRole with read-only permissions to the system:anonymous user, an attacker can gain unauthorized visibility into the cluster’s operations and configurations. This can lead to information leakage and provide the attacker with the necessary insights to plan further attacks. Mitigation strategies include enforcing strict RBAC policies, restricting anonymous user permissions, and continuously monitoring and auditing role bindings to detect and prevent unauthorized access. Strengthening RBAC configurations and employing least privilege principles are essential steps to secure Kubernetes environments against such privilege escalation attempts.

Execution (TA0002)

Attempt To Stop Apt-Daily Upgrade

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** An Attempt to Stop Apt-Daily Upgrade attack targets Kubernetes clusters by attempting to stop the apt-daily.timer service on the host system. This timer triggers daily updates and upgrades of packages on Debian-based systems. By stopping this service, an attacker could disrupt the regular update process, potentially leaving the system vulnerable to unpatched security flaws and reducing the system's overall stability and reliability.
- **Use Case Flow:** The script exploits this misconfiguration by executing a command within a container to stop the apt-daily.timer service on the host system. The steps are as follows:
  - Loading the Kubernetes configuration using the kubernetes Python client.
  - Executing a command inside a specified pod to stop the apt-daily.timer service using systemctl.
- **Research Notes:** This attack illustrates the potential risks associated with allowing containers to execute commands on the host system. Disabling system maintenance tasks like apt-daily.timer can expose the host to unpatched vulnerabilities, increasing the risk of exploitation. Researchers should focus on securing the interaction between containers and the host system by restricting access to critical system files and services. Implementing strict Pod Security Policies (PSPs) and minimizing the use of privileged containers are essential steps to mitigate such risks. This attack underscores the importance of maintaining system update mechanisms and ensuring that containers cannot interfere with critical host system operations.

Create Docker In Docker (DinD) - Pod

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** A Create Docker In Docker (DinD) - Pod attack targets Kubernetes clusters by deploying a privileged pod that runs Docker inside a container. This misconfiguration can allow an attacker to gain significant control over the underlying node by leveraging Docker commands to manage images and containers within the pod. Such an attack can lead to unauthorized access to sensitive data, execution of arbitrary commands, and potentially full control over the node.

- **Use Case Flow:** The script exploits this misconfiguration by creating a DinD pod and executing Docker commands within it. The steps are as follows:
    - Loading the Kubernetes configuration using the kubernetes Python client.
    - Creating a new pod in the default namespace with a privileged container running Docker.
    - Waiting for the pod to reach the "Running" state.
    - Executing a series of commands inside the container to create a Dockerfile, build a Docker image, and optionally clean up by removing the image and deleting the pod.
- **Research Notes:** This attack illustrates the potential risks associated with allowing containers to execute commands on the host system. Disabling system maintenance tasks like apt-daily.timer can expose the host to unpatched vulnerabilities, increasing the risk of exploitation. Researchers should focus on securing the interaction between containers and the host system by restricting access to critical system files and services. Implementing strict Pod Security Policies (PSPs) and minimizing the use of privileged containers are essential steps to mitigate such risks. This attack underscores the importance of maintaining system update mechanisms and ensuring that containers cannot interfere with critical host system operations.

Execute Command In Kube System

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Detected	Not Detected

- **Category: Privilege Escalation**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** The attack targets the ability to run arbitrary commands within the highly privileged kube-system namespace. By creating and deploying a Pod in the kube-system namespace, an attacker can execute commands within the container, potentially gaining control over critical system components. This can lead to unauthorized access, data exfiltration, or disruption of Kubernetes operations.
- **Use Case Flow:**
    - Initialization. The attack initializes by loading the Kubernetes configuration and creating an API client for Core operations.
    - Pod Creation. The attack creates a new Pod in the kube-system namespace using a specified container image (e.g., busybox). The Pod runs a sleep command to keep it active.
    - Wait for Pod to Run. The attack waits until the Pod reaches the Running state, ensuring it is fully operational before executing further commands.
    - Command Execution. Once the Pod is running, the attack executes a specified command (echo 'Executing command inside the pod') within the container using the Kubernetes API.
    - Cleanup (Optional). If specified, the attack cleans up by deleting the created Pod from the kube-system namespace to remove traces of the attack.
- **Research Notes:** This type of attack highlights the critical importance of securing the kube-system namespace and monitoring for unauthorized Pod creation and command execution. The ability to run commands in this namespace can lead to significant security breaches, as it often contains core components of the Kubernetes cluster. Effective defenses should include strict access controls, regular audits of namespace activities, and implementing policies to prevent unauthorized Pod deployments.

Detected Suspicious Use Of The Nohup Command

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Misconfiguration Exploitation**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** By using the nohup command, an attacker can start a script that continues to run even after the user has logged out, allowing them to maintain a foothold within the container and execute long-running tasks without interruption. This attack targets the persistence and stealthiness of malicious activities within containerized environments.
- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes a command to create a script, make it executable, and run it in the background using nohup. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.
  - Executing the command within the container to create a script file, modify its permissions, and run it with nohup.
- **Research Notes:** This type of attack highlights the importance of monitoring and controlling background processes within container environments. The use of nohup to run scripts can allow attackers to perform continuous, undetected actions, complicating efforts to secure and audit containerized applications. Researchers and security professionals should focus on detecting and mitigating such persistence techniques, employing real-time monitoring and alerting systems to identify suspicious use of commands like nohup. Understanding the potential for such attacks is crucial for enhancing the security and resilience of containerized infrastructures.

Security-Related Process Termination Detected

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Process Manipulation**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** By killing processes that are critical for monitoring and auditing, such as auoms, an attacker can evade detection and disable security mechanisms within the container. This attack targets the operational integrity of security tools, potentially allowing unauthorized actions to go unnoticed.
- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes a command to update package lists, install necessary tools, and terminate a security-related process. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.
  - Executing the command within the container to update the package list, install procps (which includes the pkill command), and terminate the auoms process with the pkill command.

- **Research Notes:** This type of attack highlights the importance of robust monitoring and alerting for the termination of critical security processes within container environments. By terminating the auoms process, the attacker disables a key component of the auditing infrastructure, thereby reducing visibility into their activities. Researchers and security professionals should prioritize the detection of process manipulation attempts and ensure that critical security processes are resilient to termination. Additionally, implementing layered defense mechanisms and real-time alerts for process termination events can help mitigate the impact of such attacks.

Digital Currency Mining Related Behavior Detected

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Resource Abuse**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** By installing and running a mining application, an attacker can hijack the container's resources to mine digital currency, which can lead to significant resource consumption and degrade the performance of the affected container and potentially the entire cluster. This attack targets the computational resources of the Kubernetes environment, potentially causing financial and operational impacts.
- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes commands to install necessary software and initiate cryptocurrency mining. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.
  - Executing the command within the container to update the package list, install GCC (a compiler), and initiate mining by connecting to a mining pool with specified credentials.
- **Research Notes:** This type of attack highlights the importance of monitoring and securing resource usage within Kubernetes environments. The unauthorized use of container resources for cryptocurrency mining can lead to increased operational costs and degraded performance of legitimate applications. Researchers and security professionals should focus on implementing resource limits, monitoring for unusual resource usage, and employing security policies to prevent the installation and execution of unauthorized software. Understanding the potential for such attacks is crucial for maintaining the performance and security of Kubernetes clusters.

Detected Suspicious File Download

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Malicious File Download**
- **Attack Technique: Container Administration Command**
- **Type of Attack:** By using tools like curl to fetch and download files from untrusted sources, an attacker can introduce malware into the container environment. This attack targets the security and integrity of the container by adding executable malware that can perform unauthorized actions.

- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes commands to install necessary tools and download a suspicious file. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.
  - Executing the command within the container to update the package list, install curl, and download a file from a specified URL.
- **Research Notes:** This type of attack highlights the critical need for robust security controls and monitoring of network activities within container environments. The unauthorized download of files, especially executables, poses a significant security risk as it can lead to the introduction of malware, data breaches, and further exploitation. Researchers and security professionals should focus on implementing strict network policies, monitoring file download activities, and employing intrusion detection systems to prevent and detect such activities. Understanding the potential for such attacks is essential for safeguarding Kubernetes clusters against malware and ensuring the integrity of containerized applications.

Create Container In Kube-System Namespace

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Display Container**
- **Type of Attack:** A Create Container in Kube-System Namespace attack targets the highly privileged kube-system namespace within a Kubernetes cluster. By creating a container in this namespace, an attacker can leverage elevated privileges to perform unauthorized actions, potentially compromising the entire cluster. This type of misconfiguration can allow attackers to disrupt system operations, exfiltrate sensitive data, or gain control over critical components.
- **Use Case Flow:** The script exploits the misconfiguration by creating a new pod within the kube-system namespace and executing commands inside the container. The steps are as follows:
  - Loading the Kubernetes configuration using the kubernetes Python client.
  - Creating a new pod in the kube-system namespace with a specified container image (busybox).
  - Waiting for the pod to reach the “Running” state.
  - Executing a command (/bin/sh -c “echo ‘Executing command inside the pod’”) inside the container.
  - Optionally cleaning up by deleting the created pod.

  The script provides detailed logging for each step, including the creation and execution phases.
- **Research Notes:** This attack emphasizes the critical importance of securing the kube-system namespace. Creating a pod in this privileged namespace can give an attacker substantial control over the cluster, as this namespace often contains system-level components and controllers. Researchers should prioritize securing access to the kube-system namespace by implementing strict Role-Based Access Control (RBAC) policies and regularly auditing namespace configurations. This attack demonstrates the potential consequences of misconfigurations and highlights the need for robust security practices to protect against unauthorized access and privilege escalation within Kubernetes environments.

Host IPC Privilege

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Detected

- **Category: Find Misconfiguration and Exploit**
- **Attack Technique: Inter-Process Communication**
- **Type of Attack:** A Host IPC Privilege attack on a Kubernetes container takes place when the container is configured to share the host's Inter-Process Communication (IPC) namespace. This setting grants the container the ability to communicate directly with processes running on the host system, introducing a significant security vulnerability. An attacker who gains access to such a container could potentially tamper with or eavesdrop on IPC communications of the host, compromising system integrity and security. The potential impact could range from unauthorized data access to more extensive system-level manipulations.
- **Use Case Flow:** The script scans all or specified Kubernetes namespaces to identify pods where host IPC is enabled. After identifying these vulnerable pods, it moves to the exploit phase. During exploitation, for each identified vulnerable pod, it creates a new pod in the same node, writes a file to shared memory (/dev/shm), and checks if the same file can be accessed from the vulnerable pod.
- **Research Notes:** Misconfiguration alerts and alerts regarding writing file to shared memory would be expected. While no misconfiguration alert was seen in any of the systems, an attack alert in 1 of the 3 tested systems was found.

Host PID Privilege

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Inter-Process Communication**
- **Type of Attack:** A Host PID Privilege attack on a Kubernetes container occurs when a container is configured to share the host's process ID namespace. This grants the container visibility into all the host's processes, posing a significant security risk. An attacker who gains access to such a container can potentially monitor or manipulate host processes, thereby affecting the integrity and security of the entire system. The impact could range from unauthorized data access to complete control over the host machine.
- **Use Case Flow:** The script scans a Kubernetes cluster to identify Pods with host PID set to true, indicating these potential security risks. It then validates whether these Pods can access specific host processes, such as the kubelet. Results of the scan and exploit phases are logged for review.
- **Research Notes:** The expected outcome would be an alert about both misconfiguration and active command invocation.

Insecure Capabilities

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Not Detected	Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Inter-Process Communication**
- **Type of Attack:** An Insecure Capabilities attack involves assigning elevated capabilities to a container within a Kubernetes pod. Capabilities such as SYS_ADMIN grant the container extensive privileges, potentially allowing an attacker to perform unauthorized actions that could compromise the host system or other containers. This misconfiguration can lead to severe security risks, including privilege escalation and system compromise.
- **Use Case Flow:**
    - Loading the Kubernetes configuration using the kubernetes Python client.
    - Creating a new pod in the default namespace with a container that has the SYS_ADMIN capability.
    - Waiting for the pod to reach the “Running” state.
    - Optionally cleaning up by deleting the created pod.
    - The specific capability assigned is SYS_ADMIN, which provides extensive administrative privileges on the host system.
- **Research Notes:** This attack underscores the dangers of assigning insecure capabilities to containers. Elevated capabilities can grant a container excessive control over the host system, leading to potential security breaches. Researchers should focus on enforcing the principle of least privilege by restricting capabilities granted to containers. Implementing strict Pod Security Policies (PSPs), regularly auditing container configurations, and ensuring that only necessary capabilities are assigned are essential steps to mitigate such risks. This attack highlights the importance of securing container configurations to prevent unauthorized access and privilege escalation within Kubernetes clusters.

Persistence (TA0003)

Detected Suspicious Use Of The Useradd Command

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category:** Privilege Escalation
- **Attack Technique: Create Account**
- **Type of Attack:** By adding a new user with a user ID of 0, which is typically reserved for the root user, an attacker can gain elevated privileges within the container, allowing them to execute any command with root-level access. This attack targets the integrity and security of container operations, leading to potential full system compromise.
- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes a command to add a new user with root privileges. The process involves:
    - Loading Kubernetes configuration to access the cluster.
    - Creating an API client to interact with the Kubernetes API.
    - Executing the command within the container to add a new user with a user ID of 0.
- **Research Notes:** This type of attack underscores the critical need for stringent security controls and monitoring of user management within container environments. The ability to add users with root privileges poses a significant security risk, as it can lead to unauthorized access and control over the container. Researchers and security professionals should focus on enforcing least privilege principles, monitoring suspicious user management activities, and implementing strong authentication and authorization mechanisms. Understanding the potential for such attacks is essential for improving the security posture of containerized applications and preventing privilege escalation.

Privilege Escalation (TA0004)

Anomalous Behavior, Role Binding Created

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Detected	Not Detected

- **Category: Privilege Escalation**
- **Attack Technique: Abuse Elevation Control Mechanism**
- **Type of Attack:** The attack involves creating a ClusterRole with read-only permissions and binding it to the system:anonymous user. This can allow unauthenticated users to access sensitive cluster information, facilitating further attacks or reconnaissance activities. The consequences include unauthorized data access, potential information leakage, and an increased risk of subsequent attacks.
- **Use Case Flow:**
    - Initialization. The attack initializes by loading the Kubernetes configuration and creating an API client for RBAC operations.
    - ClusterRole Creation. The attack creates a read-only ClusterRole with permissions to get, list, and watch various resources such as pods, services, deployments, jobs, and more.
    - ClusterRoleBinding Creation. The attack binds the newly created ClusterRole to the system:anonymous user, allowing unauthenticated access to the specified resources.
    - Cleanup (Optional). If specified, the attack cleans up by deleting the created ClusterRole and ClusterRoleBinding to remove traces of the attack.
- **Research Notes:** This attack demonstrates the dangers of creating and binding a ClusterRole to the system:anonymous user, allowing unauthorized and unauthenticated access to cluster resources. By assigning read-only permissions across various resource types (pods, services, deployments, jobs, etc.), an attacker can gain valuable insights into the cluster's configuration and operations. This type of misconfiguration can lead to information leakage and reconnaissance activities, potentially setting the stage for more severe attacks. To mitigate such risks, it is crucial to implement strict RBAC policies, limit the permissions of anonymous users, and continuously audit role bindings to detect and prevent unauthorized access.

New High Privileges Role Detected

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Abuse Elevation Control Mechanism**
- **Type of Attack:** A High Privileges Role attack on a Kubernetes cluster involves creating and binding a ClusterRole with extensive permissions to a ServiceAccount. By exploiting this misconfiguration, an attacker can gain elevated privileges, potentially allowing them to perform any action within the cluster. The consequences of this attack could include unauthorized access to sensitive data, disruption of services, or even full control over the Kubernetes environment.

- **Use Case Flow:**
    - Initialization. The attack initializes by loading the Kubernetes configuration and creating API clients for RBAC and Core operations.
    - ClusterRole Creation. The attack creates a ClusterRole with broad permissions (apiGroups: ["*"], resources: ["*"], verbs: ["*"]), allowing any action on any resource.
    - ServiceAccount Creation. A new ServiceAccount is created in the default namespace.
    - ClusterRoleBinding Creation. The attack binds the newly created ClusterRole to the ServiceAccount, effectively granting it the high-level permissions defined in the ClusterRole.
    - Cleanup (Optional). If specified, the attack cleans up by deleting the created ClusterRole, ServiceAccount, and ClusterRoleBinding to remove traces of the attack.
- **Research Notes:** This attack underscores the risks associated with improperly configured role-based access control (RBAC) settings within a Kubernetes cluster. By creating a ClusterRole with extensive permissions and binding it to a ServiceAccount, an attacker can escalate privileges and gain broad access to cluster resources. This vulnerability highlights the need for rigorous RBAC policies, continuous monitoring, and auditing of role bindings to ensure that high-privilege roles are only assigned where absolutely necessary. Implementing least privilege principles and regularly reviewing RBAC configurations can significantly mitigate the risk of such attacks.

Create Linux Namespace

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Escape to Host**
- **Type of Attack:** A Kubernetes Create Linux Namespace attack leverages the ability to create a new namespace within the Linux environment of a Kubernetes container. By exploiting this capability, an attacker can isolate a set of processes from the rest of the system, potentially bypassing security controls and gaining unauthorized access to system resources. This misconfiguration can lead to a situation where the attacker can perform privileged operations that should be restricted.
- **Use Case Flow:** The script targets a Kubernetes cluster, aiming to create a new Linux namespace within a container. It does so by executing the unshare command, which isolates the container's processes from the rest of the system. The process involves:
    - Loading the Kubernetes configuration using the kubernetes Python client.
    - Connecting to the specified pod, namespace, and container.
    - Executing the command unshare --mount /bin/bash to create a new mount namespace and start a new shell session within the container.
- **Research Notes:** This attack highlights a significant misconfiguration risk in Kubernetes environments. By creating a new namespace, attackers can effectively segregate their activities, making it harder to detect and respond to malicious actions. The ability to execute privileged operations within an isolated namespace can lead to various security breaches. Researchers should focus on identifying and mitigating such misconfigurations by enforcing strict namespace policies and monitoring for unusual namespace creation activities. This underscores the importance of proper configuration management and continuous monitoring to ensure the security and integrity of Kubernetes deployments.

Create Privileged Container

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Detected

- **Category: Find Misconfiguration & Exploit**
- **Attack Technique: Escape to Host**
- **Type of Attack:** A privileged container is a container that inherits the full range of capabilities from the host machine. Practically speaking, this enables the privileged container to execute nearly any operation that could be done directly on the host system.
- **Use Case Flow:** The script initiates by scanning a Kubernetes cluster for containers running with privileged settings. It gathers details from either all namespaces or a specific list, storing the results. The script then proceeds to exploit these privileged containers. It performs various actions like mounting devices and creating files in the host file system to demonstrate the exploit.
- **Research Notes:** The expected outcome would be an alert about both misconfiguration and active command invocation

Host Path Mount

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Detected	Detected

- **Category: Find Misconfiguration & Exploit**
- **Attack Technique: Escape to Host**
- **Type of Attack:** A Host Path Mount attack on a Kubernetes container occurs when the container is improperly configured to mount directories or files from the host system. This configuration exposes the container to elevated risks, as it gains direct access to crucial parts of the host file system. An attacker who gains control of such a container could potentially read, modify, or delete sensitive data on the host, affecting system integrity and security. The consequences could range from data loss and unauthorized data access to full system compromise.
- **Use Case Flow:** The script scans a Kubernetes cluster for Pods that use hostPath volume mounts. It leverages the Kubernetes API to get Pods, then iterates through their containers to look for hostPath volume mounts. Containers with hostPath mounts are considered vulnerable. During the exploit phase, the script further validates the volume mounts in the containers of the flagged Pods by using the Kubernetes API client and a stream object to execute the command, returning the response, followed by an execution of a specified command in a specified container within a specified pod.
- **Research Notes:** The expected outcome would be an alert about both misconfiguration and active command invocation. In testing, expected alerts regarding misconfigurations only occurred in 1 of the 3 tested solutions, with alerting about command invocation in none.

Writeable Host Path Mount

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Detected	Detected

- **Category: Find Misconfiguration & Exploit**
- **Attack Technique: Escape to Host**
- **Type of Attack:** A Writable Host Path Mount attack targets containerized environments by exploiting the ability to mount host system directories (paths) into a container with write permissions. In a successful attack, the attacker could manipulate or replace files on the host machine from within the container.
- **Use Case Flow:** Scripting first initializes logging and scans a Kubernetes cluster to identify containers with writable host path mounts. It gathers pod and container details from specific namespaces or all detected namespaces. After the scan, it validates each potentially vulnerable container by executing specific commands to confirm the writability of discovered mounts. Summaries of both the scanning and validation results are printed to the console.
- **Research Notes:** The expected outcome would be an alert about both misconfiguration and active command invocation. In testing, expected alerts regarding misconfigurations only occurred in 1 of the 3 tested solutions, with alerting about command invocation in none.

Role Binding To The Cluster-Admin Role Detected

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Detected

- **Category: Privilege Escalation**
- **Attack Technique: Exploitation for Privilege Escalation**
- **Type of Attack:** Role Binding to the cluster-admin Role attack on a Kubernetes cluster involves exploiting a misconfiguration to bind a user or service account to the highly privileged cluster-admin role. By creating such role bindings, an attacker can gain elevated privileges, allowing them to perform administrative actions across the entire cluster. This attack targets the access control mechanisms within Kubernetes, leading to potential full control over cluster resources and operations.
- **Use Case Flow:** The script connects to a Kubernetes cluster, creates a RoleBinding and a ClusterRoleBinding to the cluster-admin role, and optionally cleans up the created bindings. The process involves:
    - Loading Kubernetes configuration to access the cluster.
    - Creating an API client to interact with the Kubernetes API.
    - Creating a RoleBinding in a specified namespace to bind a user to the cluster-admin role.
    - Creating a ClusterRoleBinding to bind a user to the cluster-admin role across the entire cluster.
    - Optionally deleting the created bindings to clean up the environment.

- **Research Notes:** This type of attack emphasizes the importance of strict access control and monitoring within Kubernetes environments. By binding a user to the cluster-admin role, an attacker can gain unrestricted access to the cluster, posing a significant security risk. Researchers and security professionals should focus on enforcing the principle of least privilege, regularly auditing role bindings and cluster role bindings, and implementing real-time monitoring to detect and respond to suspicious privilege escalation attempts. Understanding the potential for such attacks is crucial for maintaining the security and integrity of Kubernetes clusters.

Admin Access To Default Service Account

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Not Detected

- **Category: Privilege Escalation**
- **Attack Technique: Valid Accounts**
- **Type of Attack:** The attack involves creating a ClusterRoleBinding that grants the cluster-admin role to the default ServiceAccount in the default namespace. This grants administrative privileges to any pod running with the default ServiceAccount, potentially allowing an attacker to take full control over the Kubernetes cluster. The consequences include unauthorized access, privilege escalation, and potential compromise of the entire cluster.
- **Use Case Flow:**
  - Initialization. The attack initializes by loading the Kubernetes configuration and creating API clients for RBAC and Core operations.
  - ServiceAccount Creation. The attack creates a new ServiceAccount in the default namespace.
  - ClusterRoleBinding Creation. The attack binds the cluster-admin role to the default ServiceAccount, granting it administrative privileges across the cluster.
- **Research Notes:** This attack demonstrates the dangers of misconfiguring role-based access control (RBAC) settings in Kubernetes. By binding the cluster-admin role to the default ServiceAccount, an attacker can easily escalate privileges and gain control over the entire cluster. Mitigation strategies include strict RBAC policies, limiting the permissions of default ServiceAccounts, and continuously auditing role bindings to detect and prevent unauthorized access. Implementing least privilege principles and regularly reviewing RBAC configurations are essential steps to secure Kubernetes environments against privilege escalation attempts.

Defense Evasion (TA0005)

A History File Has Been Cleared

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Misconfiguration Exploitation**
- **Attack Technique: Indicator Removal (Clear Command History)**
- **Type of Attack:** By manipulating the .bash_history file within a container, an attacker can clear the command history, making it difficult for administrators to track and investigate suspicious activities. This attack targets the auditability and transparency of container operations, potentially allowing unauthorized actions to go unnoticed.
- **Use Case Flow:** The script connects to a specified Kubernetes pod and executes a command to create and then remove the .bash_history file, effectively erasing any recorded commands. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.
  - Executing the command within the container to manipulate the .bash_history file.
- **Research Notes:** This type of attack emphasizes the critical need for robust auditing and monitoring mechanisms within container environments. By erasing the .bash_history file, the attacker ensures that their activities remain hidden, complicating forensic investigations. Researchers should focus on developing and implementing enhanced logging mechanisms and real-time monitoring solutions that can detect and alert on suspicious behavior, even in the absence of traditional log entries. Understanding the potential for such attacks is essential for improving the security posture of containerized applications and preventing similar threats.

Kubernetes Events Deleted

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Misconfiguration Exploitation**
- **Attack Technique: Indicator Removal (Clear Command History)**
- **Type of Attack:** By deleting these events, an attacker can remove evidence of their actions, making it difficult for administrators to detect and investigate suspicious activities. This attack targets the visibility and traceability of operations within the Kubernetes cluster, potentially allowing unauthorized actions to go unnoticed.
- **Use Case Flow:** The script connects to a Kubernetes cluster, creates a dummy event, and then deletes it. The process involves:
  - Loading Kubernetes configuration to access the cluster.
  - Creating an API client to interact with the Kubernetes API.

- Creating a dummy event in a specified namespace to simulate a legitimate event.
- Deleting the created event to demonstrate the ability to remove Kubernetes events.

- **Research Notes:** This type of attack underscores the importance of robust logging and monitoring mechanisms within Kubernetes environments. By deleting events, the attacker ensures that their activities remain hidden, complicating forensic investigations and incident response. Researchers and security professionals should focus on implementing immutable logging mechanisms, setting up alerts for event deletions, and regularly auditing event logs to detect suspicious behavior. Understanding the potential for such attacks is crucial for enhancing the security posture of Kubernetes clusters and preventing the manipulation or deletion of critical audit data.

Credential Access (TA0006)

Application Credentials In Configuration Files

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Not Detected	Not Detected

- **Category: Information Disclosure**
- **Attack Technique: Steal Application Access Token**
- **Type of Attack:** The attack targets the exposure of sensitive information, such as credentials, tokens, and passwords, stored in environment variables of Kubernetes Pods. By scanning and exploiting these misconfigurations, an attacker can gain unauthorized access to applications and services within the cluster. The consequences include potential data breaches, unauthorized access to resources, and further escalation of privileges.
- **Use Case Flow:**
    - Initialization. The attack initializes by loading the Kubernetes configuration.
    - Scanning for Sensitive Information. The Scanner class scans all or specified namespaces for Pods with environment variables containing sensitive keys (e.g., "credential", "bearer", "token"). If any such Pods are found, their names and namespaces are logged.
    - Exploitation. The Exploit attempts to execute a command (env) within the containers of the identified Pods to extract the environment variables. The attack logs any discovered sensitive information from these environment variables.
- **Research Notes:** This attack underscores the risks of storing sensitive information in environment variables within Kubernetes Pods. Such misconfigurations can be easily exploited to reveal critical credentials and tokens, leading to unauthorized access and potential breaches. Mitigation strategies include using Kubernetes Secrets for storing sensitive data, ensuring environment variables do not contain confidential information, and regularly auditing configurations for compliance.

Access To Kubelet Kubeconfig File

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration**
- **Attack Technique: Unsecured Credentials**
- **Type of Attack:** An Access to Kubelet Kubeconfig File attack targets Kubernetes clusters by accessing the Kubelet configuration file (kubeconfig). This file contains sensitive information, including credentials and configuration details, which can be exploited to gain unauthorized control over the Kubelet and potentially the entire cluster. This misconfiguration can lead to significant security breaches, allowing attackers to manage nodes, deploy workloads, and access sensitive data.

- **Use Case Flow:** The script exploits this misconfiguration by accessing and reading the Kubelet kubeconfig file from within a container. The steps are as follows:
    - Loading the Kubernetes configuration using the kubernetes Python client.
    - Executing a command inside a specified pod to read the contents of the kubeconfig file located at /host/var/lib/kubelet/kubeconfig.
- **Research Notes:** This attack highlights the critical importance of securing configuration files within Kubernetes clusters. Access to the Kubelet kubeconfig file can provide an attacker with the credentials and configuration details necessary to control Kubelet operations, which can lead to full cluster compromise. Researchers should prioritize securing access to the host file system, implementing strict file permissions, and ensuring that sensitive files are not exposed to containers. Regularly auditing Kubernetes node configurations and applying security best practices are essential to mitigate such risks. This attack underscores the necessity of protecting sensitive information within Kubernetes environments to prevent unauthorized access and potential privilege escalation.

List Secrets

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Not Detected	Not Detected

- **Category: Access Sensitive Data**
- **Attack Technique: Unsecured Credentials**
- **Type of Attack:** A List Secrets attack on a Kubernetes container involves exploiting improperly configured access controls to list or retrieve sensitive information stored as secrets within the Kubernetes cluster. When permissions are too lax, an attacker gaining access to a container or service account can enumerate these secrets, which may include database credentials, API keys, or other sensitive data. The attacker could then use these secrets to compromise applications or gain unauthorized access to resources. The impact could range from data breaches to full system takeovers.
- **Use Case Flow:** The script scans a Kubernetes cluster to identify users with permissions to read secrets by examining role-based access control settings. It logs users who have permissions to either 'get', 'list', or 'watch' secrets. In the exploit phase, the script attempts to list secrets using the service account tokens of the flagged users.
- **Research Notes:** Log entries identifying attempts to obtain sensitive information, such as secrets, were witnessed for other cloud services; but were not present for Kubernetes-specific attempts performed.

Discovery (TA0007)

Successful Anonymous Access

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Not Detected

- **Category: Unauthorized Access**
- **Attack Technique: Cloud Service Discovery**
- **Type of Attack:** The attack involves accessing the Kubernetes API server without authentication, allowing the attacker to retrieve information about the cluster's resources. This attack can expose sensitive data and provide insights into the cluster's configuration, potentially leading to further exploitation. The impact includes unauthorized data access, potential information leakage, and an increased risk of subsequent attacks.
- **Use Case Flow:**
  - Initialization: The script initializes by setting up a base URL for the Kubernetes API server.
  - Curl Command Execution: The script executes a curl command to access the API endpoint for listing pods in the default namespace. The command is run without any authentication, leveraging the assumption that anonymous access is permitted.
  - Result Handling: The output of the curl command is displayed, showing the retrieved information from the API server.
- **Research Notes:** This attack highlights the critical need to secure the Kubernetes API server against unauthenticated access. Allowing anonymous access to the API can expose sensitive information about the cluster's resources, configurations, and operations. To mitigate this risk, it is essential to enforce strong authentication mechanisms, restrict anonymous access, and continuously monitor API access logs for suspicious activity. Ensuring that the Kubernetes API server is not exposed to unauthenticated users is a fundamental security measure to protect the integrity and confidentiality of the cluster.

Anonymous Access To Kubelet Service

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Not Detected	Not Detected

- **Category: Find Misconfiguration & Exploit**
- **Attack Technique: Container & Resource Discovery**
- **Type of Attack:** An Anonymous Access to the Kubelet Service attack occurs when the Kubelet service running on Kubernetes nodes is misconfigured to allow anonymous access. This exposes a critical component of the cluster to unauthorized users, who could retrieve sensitive data, manipulate running containers, or execute malicious actions. The impact of such an attack could range from unauthorized data disclosure to potential control over the node or even the entire cluster.

- **Use Case Flow:** The script scans Kubernetes nodes to identify those that are misconfigured to allow anonymous access to the Kubelet service. During scanning, it deploys temporary pods on each node to gather information. If vulnerabilities are found, during the exploit phase the script takes advantage of anonymous access by creating a ClusterRoleBinding for anonymous users with admin rights. It then sends an HTTP request to validate the exploit.

- **Research Notes:** A log of the deployment of the pod (which exists in native Kubernetes), as well as attempts to access information and to provide roles to anonymous users are expected. One cloud provider logged pod creation; however, it was impossible to link it to our action, based on the provided information. All cloud providers did log activity when providing roles, though these log entries did not align with the timeframe of the attack protocol. This can lead to inaccurate forensics and the potential for not finding the correct root-cause of an attack. Logs are also expected for attempts to execute the exploit components of the script but were not found in any of the platforms used.

Host Network Access

Azure Cloud Defender	AWS GuardDuty	GCP Command Center
Not Detected	Detected	Not Detected

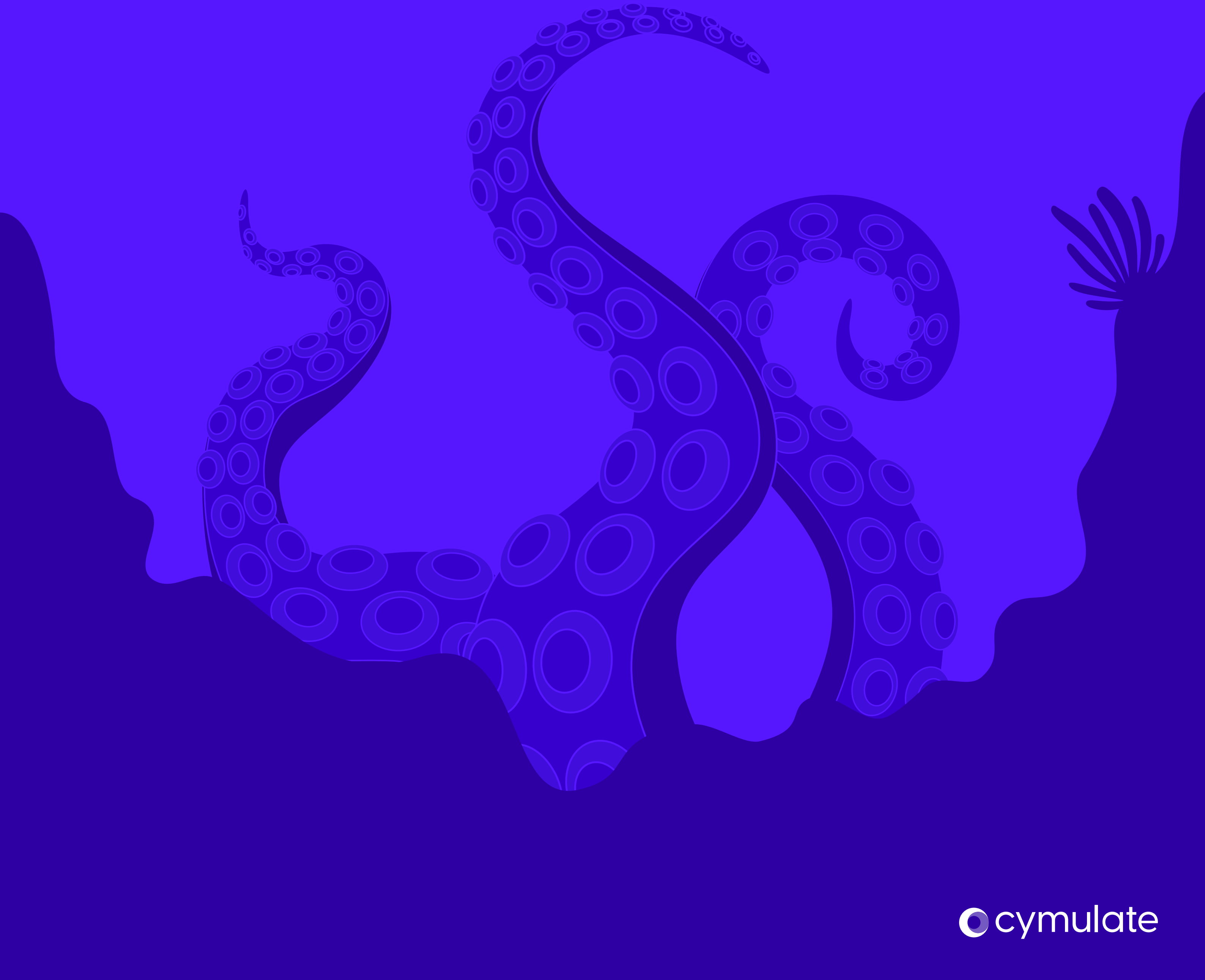
- **Category: Find Misconfiguration**

- **Attack Technique: Network Service Discovery**

- **Type of Attack:** A Host Network Access attack on a Kubernetes container occurs when the container is improperly configured to access the host's network namespace. This gives the container direct visibility into network traffic and resources on the host system, creating a significant security risk. An attacker gaining control of such a container could potentially eavesdrop on, intercept, or manipulate network communications, affecting both the host and other containers on the network. The impact could range from unauthorized data access to disruption of network services or even full system compromise.

- **Use Case Flow:** The script scans a Kubernetes cluster to identify pods that have host networking enabled. In the exploit phase, it checks if the pods with host networking enabled actually share the same IP address with their nodes, implying they are directly connected to the node's network.

- **Research Notes:** Logging and alerting to the misconfiguration were expected, but not found. In this specific technique, there would not be logging of the exploitation, making logging of the misconfiguration significantly more critical for defense.



cymulate