Five Best Practices for Kubernetes Network Security and Compliance

Modern application architectures leveraging Kubernetes have become mainstream as enterprises experience the benefits of rapid application delivery. While Kubernetes provides a tremendous business advantage for time to market when deploying new applications, increasingly sophisticated cyber-attacks demonstrate just how relentless attackers can be to uncover and exploit vulnerabilities, especially at the network level.

Introduction

Before Kubernetes, monolithic application components communicated via in-process library function calls. With Kubernetes, modern applications components communicate via network-based API calls. This difference means that security can no longer be enforced primarily by libraries and runtime frameworks. Instead, security must be applied at the network level. However, securing Kubernetes at the network level carries significant implications such as:

- Traditional security solutions like perimeter security, zone-based security, and static firewalls are not sufficiently scalable or flexible enough to meet security controls for Kubernetes
- Monitoring tools do not provide context into microservices traffic
- Compliance tools designed to provide audit trails were also designed for static applications and environments and won't work properly for Kubernetes
- Since microservices use the network, several teams – development, platform, networking, security – now need access to security tools that let them work together in an agile manner

Kubernetes changes the way that we implement security controls. Here are five best practices for securing the network in those environments.

1 Embrace Zero Trust Network Security

According to a recent PwC report, 44% of breaches originate from within the network[1]. Adding Kubernetes traffic makes the network even more complicated to secure. Kubernetes workloads use the internal network extensively as inter-application interactions shift from within computing resources to the network. This shift has caused an explosion of east-west traffic, forcing the concept of workload identity in Kubernetes environment to be fundamentally reevaluated. What constitutes a workload in a Kubernetes environment? Do we base workload identity on network locality and what they advertise themselves to be? To compound the problem, the metadata of a workload can be exploited by attackers to mimic legitimate workloads.

The challenge for many businesses is that they are trying to solve Kubernetes network security issues by leveraging legacy network security models. Traditional security models are based on network location such as IP address. The underlying IP address for Kubernetes workloads are ephemeral and thus breaks traditional security models. Traditional network security was designed to block external threats assuming the internal network is trustworthy. Trusting the internal network equates to workloads being able to communicate and access resources without authentication. Without workloads having to authenticate, just like users do, it becomes impossible to validate the identity of workloads.

Zero Trust network security addresses the issue by establishing trust with multi-factor authentication for workloads. Workloads need to authenticate based on a

combination of cryptographic identifiers, network identifiers, and metadata such as labels and namespace, and layer 7 request attributes. Anyone of these attributes in isolation cannot verify the true identity of the workload. Understanding the workload's metadata along with network locality can stitch together trusted workload identifies that can't be forged by attackers. Trust can then be applied as much as needed to a given workload in order to get the task done.

By establishing an identity for each workload, the Zero Trust model can validate trust at multiple enforcement points in and outside of the pod and hosts. Implementing multiple and interlocking security controls at every level of the stack eliminates the possibility that a single compromise can defeat the entire environment.

2 Make Sure Logs are Meaningful and Durable

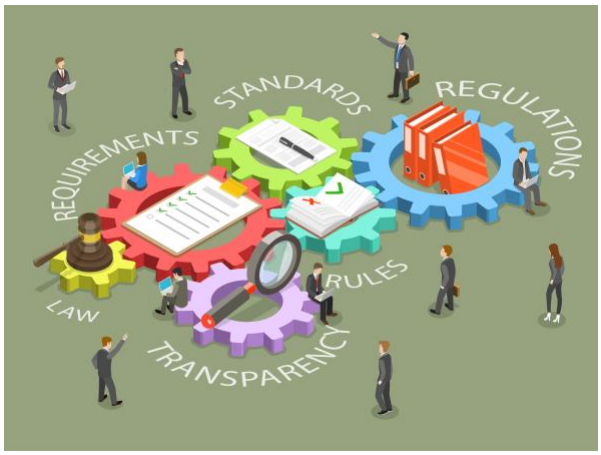
IP addresses for Kubernetes workloads change frequently making IP address ineffective as an identifier for logging. The challenge with traditional logging methods is that they rely solely on IP address and provide limited 5-tuple flow information. These dated methods do not capture denied traffic at the container level and required additional context to be useful. Monitoring Kubernetes workloads is significantly different than monitoring traditional host-based systems and requires visibility into the Kubernetes constructs. These constructs where the application runs are entirely invisible to traditional network monitoring systems that monitor at the host layer. Kubernetes context such as containers, pods, nodes, namespaces, and labels are needed along with the IP addresses.

Network flow logs need to be captured at the container level and appended with workload metadata. Workload identity and metadata provide visibility into which workload did what for compliance auditing and forensics purposes.

Beyond logging of network traffic, logging is required for Kubernetes network adjustments such as network policy changes. Logging of network policy change is necessary as attackers can alter the network policies if they are successful with their attacks. Network policy changelogs should also be tamperproof to limit attackers from covering up their malicious activities.

3 Ensure Deployments are in Compliance

Maintaining control and meeting compliance mandates is a requirement for most production deployments. Many compliance standards and guidelines, such as Payment Card Industry Data Security Standard (PCI DSS)[2] and Center for Internet Security (CIS) benchmarks [3], require and/or recommend proper segmentation at the network level. This is a crucial step in preventing attacks from moving laterally and also protecting sensitive data. However, traditional network segmentation with firewalls and VLAN does not scale with dynamic Kubernetes environments. New workloads or changes to workloads could equate to long change control cycles or even changing the entire network. Furthermore, traditional network segmentation provides coarse-grained control, only able to separate groups of hosts or workloads. This is where network policy can address the shortcomings.



Kubernetes network security policies (referred to as NetworkPolicy in this paper) controls how workloads are allowed to communicate with each other and other network endpoints based on a declarative specification. NetworkPolicy is the cornerstone to providing fine-grained control for Kubernetes and defines rules that specify what traffic is allowed or denied. Workload identity information and labels are used by NetworkPolicy to identify what each workload can communicate with. Labels are not only used for identifying workloads for communication but also to help establish a structure for compliance control.

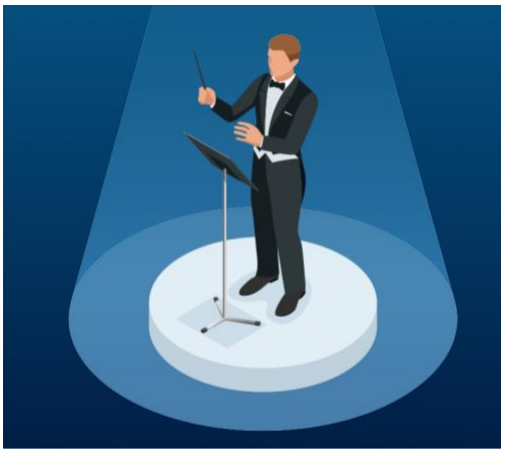
To illustrate how compliance is achieved with NetworkPolicy, suppose an application needs to be PCI DSS compliant. Workloads involved in processing financial transactions or handling sensitive financial information must be prohibited from interacting with other workloads or resources that are not PCI compliant.

In order to apply policies that meet PCI DSS requirements, PCI labels are applied to PCI-certified workloads. The policy will restrict any service labeled PCI from interacting with any other service that is missing the PCI label. This would institute a global policy ensuring PCI workloads only interact with other PCI workloads.

In addition, NetworkPolicy models with tiering capability should be used to align with organizational needs. A tiered NetworkPolicy model separates concerns of different functional teams in a hierarchical manner. Policy tiers enable businesses to safely delegate policy specification to downstream teams. By leveraging tiered NetworkPolicy, development teams can plan and manage their network security posture in a standard and autonomous way. Furthermore, each policy should be tied to one or more users and/or teams in the Kubernetes Role Based Access Control System (RBAC). By tying policies to RBAC, team members can make the changes they need, without being able to change higher-priority policies set by other teams.

4 Orchestrate Security Just Like You Orchestrate Code

One of the top challenges for DevOps teams in the continuous integration/continuous deployment (CI/CD) pipeline[4] is an inconsistent approach to implementing security tools and testing. Achieving security objectives is often difficult when relying on traditional security tools outside of the CI/CD pipeline. The concept of "policy as code" treats the Kubernetes NetworkPolicy artifact an immutable fragment of code. Security can be built into the CI/CD pipeline by checking NetworkPolicies into version control system. As the policy refers to workloads using metadata rather than specific instance data such as IP address, it is possible to apply a set of policies and continuously achieve the same security state in the network. Policy as code also allows the network infrastructure to revert to a known security state by simply deploying the same set of policy artifacts.



5 Know What is Deployed Onto the Network

Nearly a quarter of all container images [5] from popular public repositories were recently identified with significant vulnerabilities. Containers could potentially be built from images obtained from these public repositories and then orchestrated by Kubernetes into the network. Containers running with vulnerable code are easily exploitable

by attackers over the network. Regardless of whether you use public or private repositories, it is crucial to practice good security hygiene and understand if vulnerabilities exist within your containers.

Container scanners can screen out container images with known major Common Vulnerabilities and Exposures (CVE)s. Evaluate your existing scanning or open source tools to discover known CVEs. It is important to note that container scanning should be integrated with the CI/CD pipeline.

If the container image fails during the scanning process, the continuous deployment process must fail as well. This ensures that vulnerable containers cannot propagate into the network for exploitation. Container images that are scanned with vulnerabilities need to be replaced with newly built images, without the known issues, within the continuous integration process.

Summary

As businesses look to modernize their applications, many have turned to Kubernetes because of the benefits containerized workloads bring. While Kubernetes enables applications to be deployed in a rapid manner, businesses need to think about securing their applications, especially at the network level.

Kubernetes environments have more moving parts that introduce new network security implications, particularly around East-West traffic. A new way of managing workload trust and identity is needed to secure the network. Monitoring network traffic needs to account for Kubernetes constructs that are invisible to traditional monitoring tools. Compliance responsibilities with Kubernetes can span across different teams such as applications, infrastructure, and networks. Each functional area must be involved in the definition and enforcement of compliance and security policies, ideally in an autonomous manner. Lastly, network security and container images need to be accounted for in the CI/CD pipeline to ensure addressing security issues at build-time and redeployment.

About Tigera

Tigera delivers solutions for secure application connectivity for the cloud native world. Tigera technology is used by the world's largest enterprises and public cloud providers to power connectivity for application development and deployment and to address the connectivity and security challenges that arise in at-scale production. Tigera Secure meets enterprise needs for zero trust network security, multi-cloud and legacy environment support, organizational control and compliance, and operational simplicity. Tigera Secure builds on leading open source projects Kubernetes, Calico, and Istio, which Tigera engineers help maintain and contribute to as active members of the cloud native community.

tigera.io

email: contact@tigera.io

phone: +1.415.612.9546

Tigera, Inc. 58 Maiden Lane, Fifth Floor, San Francisco CA 94018 USA

References:

1. PwC, US State of Cybercrime Survey - Insider Threat
2. PCI, DSS v3.2.1 1.2.1 Restrict inbound and outbound traffic
3. CIS, Kubernetes Benchmark 1.6.3 Create network segmentation using Network Policies
4. 451 RESEARCH, DevSecOps Realities and Opportunities
5. InfoSecurity, Securing Docker Containers Can Begin With a Clean Image