Extending Zero Trust architecture to Kubernetes sidecars

Aarni Halinen

Department of Computer Science
Aalto University

October 22, 2023

Supervisor: Prof. Mario Di Francesco
Advisor: M.Sc. (Tech.) José Luis Martin Navarro
Advisor: M.Sc. (Tech.) Jacopo Bufalino

Contents

Zero Trust Architecture

A security paradigm that focuses on the premise that trust must always be explicitly granted

- Move security boundaries to the most granular level and use fine-grained access rules
- A multi-layer, defense-in-depth approach
- Network communication, even if internal and behind a firewall, should not be trusted
- Services communicate securely (Mutual authentication, with mTLS), rely on more robust identifiers than IP addresses, and restrict traffic on L5-L7
- Modern service meshes help implementing the architecture in Kubernetes clusters, often using sidecars (like Envoy proxy)

Kubernetes sidecars

- Sidecar pattern allows isolating of peripheral tasks (logging, observability, Envoy proxies) from application to own helper containers called sidecars
- Pods consist of one or more tightly-coupled containers, sidecars are not technically distinguishable from application container
- Containers in Pod share Linux network namespace with each other

K8s networking model

- Addresses 4 different types of networking communication
  - Inside Pod's network namespace (localhost)
  - Pod-to-Pod, even accross different Nodes
  - Service-to-Pod
  - Cluster external sources to Services
- Pod-to-Pod connection is implemented by a CNI plugin that creates NIC and assigns IP addresses (IPAM)
- CNI plugins with operator daemons can also implement network rules (Network Policy resource)
- Calico, Cilium
- Meta-plugins such as Multus implement other features as part of the CNI chain

Sidecar threat modelling

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Instance Metadata API	Applications credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container				Access managed identity credential		Writable volume mounts on the host		
	Sidecar injection				Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

Figure: Kubernetes threat matrix (MITRE ATT&CK) [2]

- Initial access is already assumed
- Attacks are also experimented with custom container in a Minikube cluster

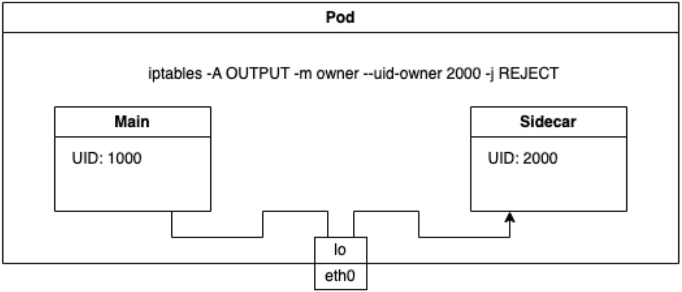
Permission related threats

- Most of the threats found are easily mitigated with Pod Security Admission Controller
- Other threats can be mitigated with custom admission controllers
  - Containers in a Pod share and automatically mount Service Accounts $\Rightarrow$ do not allow automatic SA mounting, manually mount to containers when needed
  - Resource limits are not enforced (denial of service) $\Rightarrow$ enforce with admission webhooks

Networking threats

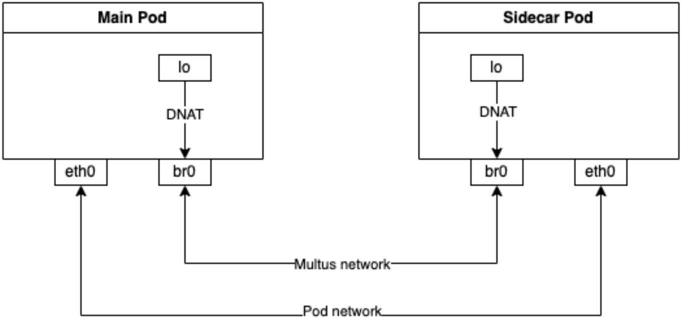
- The common network namespace in Pod's allows unlimited access to other containers
- Network policies apply to all containers in a Pod
- No built-in options for fine-grained network access within the network namespace
- Two different approaches invistigated
- Both approaches should not interfere with permission related mitigations

Approach 1: Injecting networking rules to Pod



- Inject IPTables rules to Pod net namespace after deployment
- Containers are distinguished from one another by using unique user IDs and IPTables owner-module

Approach 2: Split the Pod and rebuild sidecar-like connectivity



- All containers are inherently in own net NS
- Multus used for static IPs, NPs and isolation from Pod network
- Loopback connectivity with `net.ipv4.conf.all.route_localnet=1`

Findings

- Moving security boundaries to container-level is possible, but laborous to implement
  - Keeping access rules up-to-date requires a custom K8s operator
  - Multus is not yet a mature project
  - Multus approach breaks co-scheduling
  - mTLS between containers is not solved
- *Avoiding sidecars is the best mitigation*
- $\Rightarrow$ Use DaemonSets to run sidecars per-Node

Future development

- K8s v1.28 introduces SidecarContainers
  - Introduced for fixing existing lifecycle issues of initContainers
  - Proposal explicitly states a non-goal of enforcing different security regulations for sidecars
- Service meshes have introduced sidecarless architectures
- Cilium service mesh (eBPF) and Istio ambient mesh

Re-cap

Initial Access	Execution	Persistence	Privilege Escalation	Defense Evasion	Credential Access	Discovery	Lateral Movement	Collection	Impact
Using Cloud credentials	Exec into container	Backdoor container	Privileged container	Clear container logs	List K8S secrets	Access the K8S API server	Access cloud resources	Images from a private registry	Data Destruction
Compromised images in registry	bash/cmd inside container	Writable hostPath mount	Cluster-admin binding	Delete K8S events	Mount service principal	Access Kubelet API	Container service account		Resource Hijacking
Kubeconfig file	New container	Kubernetes CronJob	hostPath mount	Pod / container name similarity	Access container service account	Network mapping	Cluster internal networking		Denial of service
Application vulnerability	Application exploit (RCE)	Malicious admission controller	Access cloud resources	Connect from Proxy server	Applications credentials in configuration files	Instance Metadata API	Applications credentials in configuration files		
Exposed sensitive interfaces	SSH server running inside container				Access managed identity credential		Writable volume mounts on the host		
	Sidecar injection				Malicious admission controller		CoreDNS poisoning		
							ARP poisoning and IP spoofing		

Figure: Threat matrix. Attack techniques addressed are highlighted in green.

Re-cap

- The thesis investigated and found ways to mitigate sidecar related threats in K8s
- Most vulnerable configuration can be prevented with PSA
- Admission controller allows extending protections even further
- No existing way to implement ZTA in sidecar networking, but it is possible
- Implementing the network solutions are cumbersome and require extensive work
- Avoiding sidecars altogether is the easiest mitigation

References

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

2 Secure containerized environments with updated threat matrix for Kubernetes, By Yossi Weizman, Senior Security Researcher, Microsoft Defender for Cloud. [https://www.microsoft.com/en-us/security/blog/2021/03/23/secure-containerized-environments-with-updated-threat-matrix-for-kubernetes/]