

Secure and Scalable Policy Management in Cloud Native Networking

Gerald Budigiri
gerald.budigiri@kuleuven.be
imec-DistriNet
Leuven, Belgium

ABSTRACT

Support for Ultra-Reliable Low-Latency Communication (URLLC) in 5G networks has made the adoption of edge computing and cloud-native computing imperative. However, URLLC applications require low-latency security defenses against attackers to maximize resilience without impacting performance. In this PhD project, we explore the use of standardized cloud-native network segmentation solutions to achieve this goal and we develop zero-cost techniques for the enforcement and verification of consistency of network segmentation policies across different system layers of the cloud-native stack that is based on Kubernetes (K8s), a de-facto standard for container-based orchestration.

We first evaluate network policies, a K8s-specific network segmentation solution on eBPF-based network plugins configured to avoid the use of network encapsulation techniques such as VxLAN or IP-in-IP. Our findings show negligible overheads. However, without network encapsulation, every protocol and port that is allowed in a container network policy must also be allowed at the underlying system layer, which is typically a VM-based solution with a statically-defined security group per K8s cluster. To drastically reduce the huge network attack surface that ensues from a static solution, we develop two dynamic cross-layer policy management approaches, each tailored to a particular level of trust between the administrators of these different system layers. A preliminary evaluation shows these techniques incur no application-level performance overhead and significantly reduce the network attack surface between VMs by 75-to-99% for low-latency applications.

CCS CONCEPTS

• **Security and privacy → Firewalls**; • **Computer systems organization → Cloud computing**.

KEYWORDS

kubernetes, network isolation, network policies, security groups.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
Middleware Demos, Posters and Doctoral Symposium'23 , December 11–15, 2023, Bologna, Italy
© 2023 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 979-8-4007-0429-1/23/12…$15.00
https://doi.org/10.1145/3626564.3629092

1 INTRODUCTION

1.1 Context

Ultra-Reliable Low-Latency Communication (URLLC) applications like Vehicle-to-Everything (V2X) and Remote Surgery (RS) require substantial computational resources due to strict performance demands. Service providers are turning to edge- and cloud computing, and containerisation to meet these requirements. Deploying low-latency applications with Kubernetes (K8s) rather than resource-intensive VMs boosts efficiency but heightens security concerns, especially since such applications demand low-overhead security solutions that do not compromise their efficiency. These concerns are even higher in multi-tenant K8s environments, where microservices belonging to different tenants lack default isolation. There are some efforts made to improve container security [8, 9] but these mostly focus on container image, runtime, and kernel OS with little or no consideration for network security, yet this lack of sufficient network segmentation is an often cited high-risk vulnerability that has been exploited in large cybercrimes such as the Equifax data breach [4]. In multi-tenant environments, strong network isolation is crucial, requiring explicit allow-policies for inter-container communication.

Even though containers are more efficient than VMs, they are typically deployed on top of VMs to protect the cloud providers' infrastructure. In this multi-layered setup, network policies restrict container communication while security groups perform a similar role at the VM node level. Security groups however often do not adhere to the principle of least privilege. To minimize the network attack surface between VMs, it's desirable to also configure security groups according to the least-privilege principle, where connections between VMs are only opened if they host containers that require these connections according to verified network policies. Without such measures in the event of a container escape, attackers may move laterally across worker nodes and, e.g., obtain API tokens that give them access to the API of the K8s control plane [2].

1.2 Problem domain

The pervasive use of containers has led to a wide range of security breaches on containerized applications. With potential flaws stemming directly from container images [2], it is evident that vulnerabilities exist in containers, posing a risk of exploitation by external attackers. It is thus crucial to have post-exploitation defenses to prevent spread and cluster compromise even in case of a container breakout or a privilege escalation. In the default security group setup, an attacker who gains access to a vulnerable pod in a cluster susceptible to container breakouts can establish a foothold across the entire cluster by exploiting a vulnerability [3]

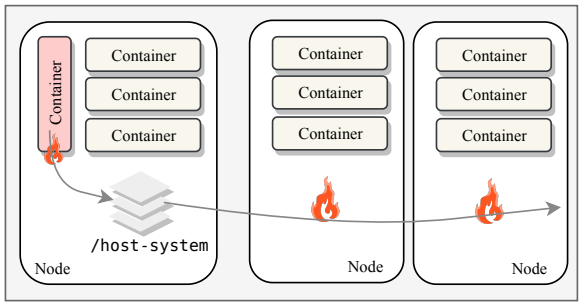


Figure 1: Attack scenario 1: Assuming that the attacker has gained access to a pod vulnerable to container escape.

or powerful pod credentials [2]. Powerful pods may however not always be present on a given node, necessitating lateral movement to explore other worker nodes as shown in Fig. 1. With default security group settings, an adversary can spread to other nodes by; (1) leveraging host credentials, (2) exploiting improperly configured kubelets, (3) targeting other applications for remote code execution, and (4) initiating malicious processes on a node e.g. through a reverse shell attack [6]. Even when an attacker has gained access to a container but not escaped from it, they can scan within the container's scope for vulnerable nodes, retrieve the pod tokens on these nodes, which can then be used for authentication and authorization to the K8s API server [2].

The aforementioned scenarios would greatly benefit from a substantial reduction in the network attack surface achieved through the automatic generation and enforcement of security groups from verified least-privilege container network policies. This approach can effectively block superfluous ports and protocols while entirely eliminating unnecessary access to neighboring nodes.

1.3 Summary of contributions

In the first part of this PhD project, we evaluate network policies for their efficacy as a low-overhead security solution for URLCC applications, specifically for eBPF-based CNI solutions without encapsulation. Additionally, we identify potential security challenges that could weaken defenses against attackers, a key weakness being performance and security conflicts within the system layers of the cloud-native stack.

In the second part, we focus on inconsistencies between network policies and security groups in K8s clusters where network encapsulation is avoided to attain low-latency requirements. We introduce, GrassHopper, a fast cross-layer network policy enforcement solution that dynamically reacts to container (de-)scheduling events and generates consistent and least privilege security group configurations from on-the-fly verified relevant network policies.

In the last part, we recognize that GrassHopper's assumption that applications and cluster orchestrators are managed by the same admin or mutual trust between them is not universally applicable in production environments. We thus propose extending the reactive core of GrassHopper with a proactive scheduler that enforces consistent configurations through proper scheduling of dependent application containers to compatible node segments.

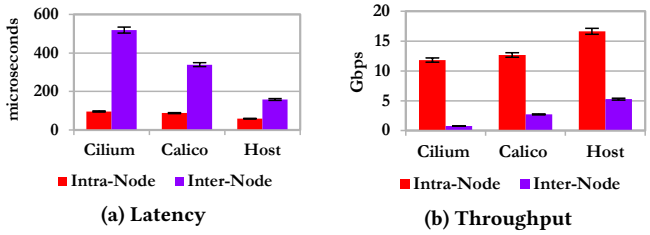


Figure 2: eBPF: Calico vs Cilium

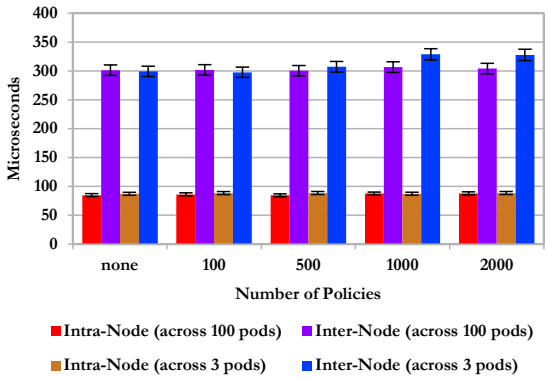


Figure 3: Latency overhead for increasing number of policies

2 ACHIEVED AND PLANNED CONTRIBUTIONS

In this section we further elaborate on the the main contributions delivered so far, including a prototypical solution named GrassHopper. We also present the improved and extended version of GrassHopper.

2.1 Study of cloud-native network solutions

We assess the effectiveness of K8s network policies as a low-overhead security solution for URLLC applications. K8s relies on customizable Container Network Interface (CNI) plugins for the enforcement of network policies. Calico and Cilium CNI plugins support enforcement of network policies with eBPF, a high-performance alternative to iptables. Indeed, when comparing eBPF with iptables, we observed a latency improvement of 0.7X and 0.8X, and a throughput improvement of 3.5X and 1.2X for inter- and intra-node scenarios respectively. We compared Calico eBPF and Cilium eBPF performances against host mode. The results (see Fig.2) indicate that despite the benefits of eBPF data plane, the overheads of CNI plugins remain significant, especially for inter-node communication. The results also show that in default settings, Calico eBPF outperforms Cilium eBPF notably in the inter-node scenario. Calico eBPF was thus used for the evaluation of network policies overheads on Netperf application [5] where we observed no significant effect as shown in Fig. 3.

Network policies are however not a silver bullet. Thus we also defined an attacker model for network policies and highlighted corresponding threats and vulnerabilities [1]. Notably, misconfiguration surfaced as a critical concern due to the numerous opportunities for weak or erroneous manual configurations.

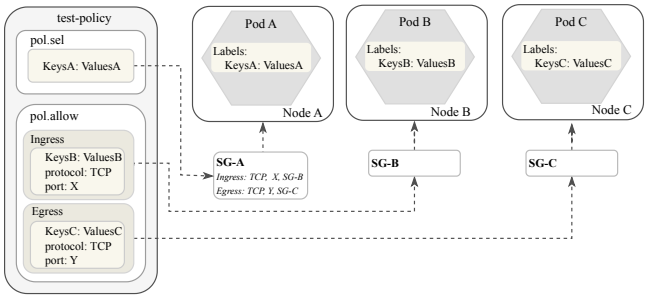


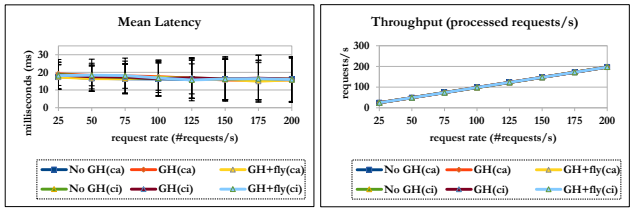
Figure 4: Policy to SG mapping

2.2 Cross- layer policy enforcement

To address inconsistencies between container and VM connections especially in low-latency CNI setup where container layer protocols and ports must also be opened at VM layer, we introduce GrassHopper for cross-layer enforcement of network policies in response to dynamic container scheduling. It ensures connections align between VM security groups and container network policies, while also upholding the principle of least privilege. When a new container p is scheduled on VM N, GrassHopper automatically verifies relevant network policies and generates consistent security group rules for VM N so that N may *only* communicate with VMs that run containers for which communication with p is allowed, and *only* using the specified traffic direction, ports, and protocols. A caveat to the least-privilege property is that cloud platform admins may create extra security group rules unrelated to the K8s data plane but necessary for other node workloads. We assume that these additional rules do not conflict with our objectives

2.2.1 GrassHopper design. Network policies use label sets such as 'keysA:valuesA' in Fig. 4 to match containers in the cluster. GrassHopper consolidates multiple network policies with a common label set into a single security group, minimizing time-consuming operations like creating, modifying, or attaching security groups. This ensures that all configurations are completed before container is fully started, eliminating any performance overhead or additional latency. GrassHopper checks network policies for broad access permissions, conflicts, and redundancy with any of the previously added policies before generating security group configurations, rejecting any offending policies. Without such policy checking GrassHopper can be used by attackers to generate very permissive security groups. We implement an incremental verification performed dynamically whenever a new policy is submitted to the K8s API. Given a network policy *test-policy* allows TCP ingress on port X from pod B to pod A and TCP egress on port Y from pod A to pod C as shown in Fig. 4. In response, security groups *SG-A*, *SG-B* and *SG-C* are created and *SG-A* is augmented with rules targeting *SG-B* and *SG-C* as *remote security groups*. As pod A is deployed on node A, the corresponding hashmap entry *Map*('keysA:valuesA') represents security group *SG-A* with its rules. GrassHopper also handles K8s *service* resources, adding rules for *node port* services to open ports on all nodes.

2.2.2 GrassHopper evaluation. We evaluated GrassHopper for both calico and cilium eBPF solutions across three applications: netperf [5], teastore [11], and an adaptive SaaS application [10] on K8s



(a) Latency **(b) Throughput**

Figure 5: Evaluation of GH in an agressive autoscaling scenario of the SaaS app; error bars indicate standard deviation

Table 1: System-wide average CPU and Memory utilization for the K8s control plane node.

Application	Netperf			SaaS		
	No GH	GH	GH+fly	No GH	GH	GH+fly
Average CPU usage	10.03%	10.18%	10.90%	10.52%	10.96%	11.13%
Average Memory usage	20.27%	20.31%	20.34%	19.21%	20.12%	20.28%

Table 2: SG operation times (s) by GH for netperf and SaaS application.

	Netperf	SaaS
Time for pod ready	3 to 4	3 to 4
Create and attach SG to node	0.234 (server pod) 0.333 (two client pods)	0.357 (first SaaS pod on node x) 0.084 (first scaled pod on node y) 0.075 (second scaled pod on node z)
Detach and remove SG	0.05	0.05

running on OpenStack, which defined the de-facto security group API for all major cloud providers.

We measured application performance without GrassHopper (No GH), with GrassHopper (GH), and when security groups for other applications are added to the cluster during the evaluation (GH+fly). We assessed; (1) GrassHopper's impact on K8s application performance with respect to latency, throughput, and resource utilization, (2) time taken by GrassHopper to configure least privilege security groups, (3) performance impact on the application when additional security groups are added to the cluster.

The results as observed in Fig. 5 show that GrassHopper does not affect application performance for both Calico and Cilium CNI plugins, even when security groups for other applications were dynamically added during the evaluation (GH+fly). We found similar results for the other test applications [2]. Since GrassHopper runs on the K8s control plane node, we observed (as shown in Table 1) a negligible increase in the average CPU and memory usage of the K8s control plane node for GH and GH+fly scenarios, when compared to no GH scenario. Table 2 shows that GrassHopper configures security groups much faster than it takes for application pods to start and become operational. The running of GrassHopper operations in parallel with pod start-up ensures that security groups are set up before pods are ready, preserving application performance.

2.2.3 Reduction of attack surface. We analyzed the reduction of the network attack surface as a ratio of the *connectivity density* (CD) achieved with GrassHopper to statically defined security groups for all cluster nodes. A statically defined security group for N cluster nodes (CD_{noGH}) corresponds to opening P ports of M application

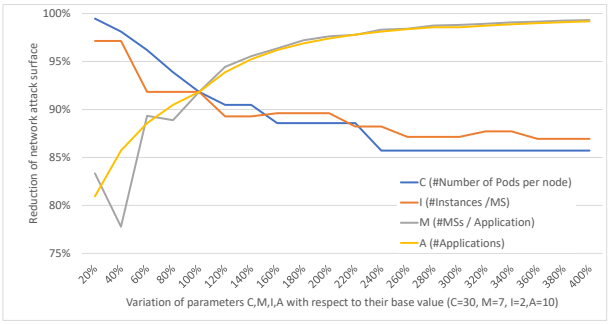


Figure 6: Network attack surface reduction with d=2

components of A applications at a cluster, and allowing all possible connections between any pair of its N nodes in both directions (cfr. Eq. 1). GrassHopper CD (CD_{GH}) is derived from call graphs [2] and can be defined accordingly to Eq. 2. A scheduling complexity constant d is added when computing the number of nodes N in Eq. 3 to account for pods that cannot be scheduled on available nodes due to pod (anti-)affinity constraints. The reduction rate is defined by Eq. 4 as the ratio of the CD values.

$$CD_{noGH} = 2 \times \binom{N}{2} \times A \times M \times P \tag{1}$$

$$CD_{GH} = (min(N, M) - 1) \times I^2 \times P \times A \tag{2}$$

$$N = max(\lceil \frac{I \times M \times A}{C} \rceil + d, I) \tag{3}$$

$$Reductionrate = (1 - \frac{CD_{GH}}{CD_{noGH}}) \times 100 \tag{4}$$

Fig. 6 shows a 94% reduction rate when setting at the following base values: $C = 30$, $M = 7$, $I = 2$, and $A = 10$. Varying each parameter individually results in reduction rates between 75% and 99%. A more detailed analysis is presented in our recent work [2].

2.3 Conflicts management in edge-to-cloud platforms

GrassHopper's assumption of same administrator for applications and cluster orchestrators is not always true in edge-to-cloud platforms offering K8s-compatible workflows but limiting underlying infrastructure access. With different admins, inconsistencies between network policies and security groups can lead to security conflicts. As an extension, we propose a concept of *node segments* to denote trust boundaries between different set of nodes and group nodes permitted to communicate with each other where dependent containers can be scheduled. This requires the reactive core of GrassHopper to be extended with a proactive scheduler Diktyo [7] that ensures the scheduling of containers on node segments with compatible network bandwidth and latency guarantees.

This approach seeks to manage conflicts both proactively by enforcing consistent configurations through proper scheduling, and reactively by employing a verification approach to detect conflicts that cannot be prevented by a scheduler, such as the addition of network policies allowing two pods to communicate but located on nodes not connected to each other. Such conflicts must be swiftly detected to trigger rescheduling of containers to compatible nodes.

3 CONCLUSION AND FUTURE WORK

Deploying URLLC applications using containers, not VMs, boosts performance but raises security concerns. With such applications demanding low-overhead security solutions that do not compromise their efficiency, we have evaluated network policies, and observed no significant performance overheads, affirming their suitability for securing such applications. Network policies are however not a silver bullet. Misconfigurations and other vulnerabilities can still be exploited, potentially compromising the entire cluster. Running containers atop VMs only adds complexity, introducing inconsistencies and security conflicts between network policies and security groups. We have proposed GrassHopper, a novel cross-layer enforcement approach that automatically generates least privilege VM level security groups from at-run-time verified container level network policies, thereby reducing network attack surface for low-latency applications by 75–to–99% at zero cost, i.e, without significant overhead on container application performance.

Our next step involves extending GrassHopper with a proactive scheduler, followed by developing a fast consistency checker for detecting conflicts that may still arise despite proactive scheduling as future work.

ACKNOWLEDGMENTS

The studies presented in this paper are a result of collaborative team effort, and thus I would like to acknowledge the contributions of Dr. Eddy Truyen, Dr. Christoph Baumann, Prof. Jan Tobias Mühlberg, and Prof. Wouter Joosen. This research has received funding under EU H2020 MSCA-ITN action 5GhOSTS, grant agreement no. 814035. Partially funded by the Research Fund KU Leuven, and by the Flemish Research Programme Cybersecurity.

REFERENCES

[1] Gerald Budigiri, Christoph Baumann, Jan Tobias Mühlberg, Eddy Truyen, and Wouter Joosen. 2021. Network policies in kubernetes: Performance evaluation and security analysis. In *2021 Joint European Conference on Networks and Communications & 6G Summit (EuCNC/6G Summit)*. IEEE, 407–412.

[2] Gerald Budigiri, Christoph Baumann, Eddy Truyen, Jan Tobias Mühlberg, and Wouter Joosen. 2023. Zero-Cost In-Depth Enforcement of Network Policies for Low-Latency Cloud-Native Systems. In *2023 IEEE 16th International Conference on Cloud Computing (CLOUD)*. 249–261.

[3] Common Vulnerabilities and Exposures 2020. CVE-2020-8559: Privilege escalation from compromised node to cluster. https://groups.google.com/g/kubernetes-security-announce/c/JAIGG5yNROs?pli=1. [Accessed: 2022-05-12].

[4] EPIC. [n. d.]. Equifax Data Breach. https://archive.epic.org/privacy/data-breach/equifax/. [accessed 2023-10-25].

[5] HewlettPackard. 2022. Netperf. https://github.com/HewlettPackard/netperf/. [Accessed: 2022-04-11].

[6] Marco Lancini. 2020. Get Reverse Shell. https://cloudsecdocs.com/container_security/offensive/attacks/techniques/reverse_shell/. [Accessed: 2023-10-25].

[7] José Santos, Chen Wang, Tim Wauters, and Filip De Turck. 2023. Diktyo: Network-aware scheduling in container-based clouds. *IEEE Transactions on Network and Service Management* (2023).

[8] Murugiah Souppaya, John Morello, and Karen Scarfone. 2017. *Application Container Security Guide (2nd Draft)*. Technical Report. NIST.

[9] Sari Sultan, Imtiaz Ahmad, and Tassos Dimitriou. 2019. Container security: Issues, challenges, and the road ahead. *IEEE Access* (2019).

[10] Eddy Truyen, André Jacobs, Stef Verreydt, Emad Heydari Beni, Bert Lagaisse, and Wouter Joosen. 2020. Feasibility of container orchestration for adaptive performance isolation in multi-tenant SaaS applications. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing*. 162–169.

[11] Joakim Von Kistowski, Simon Eismann, Norbert Schmitt, André Bauer, Johannes Grohmann, and Samuel Kounev. 2018. Teastore: A micro-service reference application for benchmarking, modeling and resource management research. In *2018 IEEE 26th International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems (MASCOTS)*. IEEE, 223–236.