Final Project Report
Cilium

Zachary Yeo
zyeo@utexas.edu
UT Austin, USA

Abstract

This paper is a study of Cilium being deployed in a Kubernetes cluster. Cilium is a networking and security solution used by many major cloud providers including Google Cloud, AWS, and Azure. This popularity is a testament to its scalability, performance, and rich feature set. This paper aims to study how Cilium is deployed in a Kubernetes cluster to examine its ability to enhance the security and networking capabilities of containerized applications. It also aims to validate current benchmarks that have been provided by the Cilium team. Through this study, we not only validate previous findings but also extend our understanding of Cilium's efficacy in Kubernetes deployments.

ACM Reference Format:
Zachary Yeo. 2025. Final Project Report Cilium. In *Proceedings of ACM Conference (Conference'17).* ACM, New York, NY, USA, 5 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

With the exponential growth of containerization technologies such as Docker and Kubernetes, the demand for robust network security solutions has surged. Cilium is currently a key player in this landscape, offering API-aware network security filtering tailored for container frameworks. This surge in interest can be attributed to the evolving technologies, where modern cloud computing demands scalable, performant, and feature-rich solutions for securing containerized applications. What makes Cilium stand out from its competition is the user of Enhanced Berkeley Packet Filter (eBPF) technology. This technology allows Cilium to indirectly control logic inside the Linux kernel without writing a kernel module. This safer approach also means that we are able to gain fine-grain and quick insights into network activities at multiple layers of the networking stack. Cilium's widespread adoption by major cloud providers attests to its effectiveness in addressing the security,networking, and observability challenges posed by contemporary container orchestration environments.

This study delves into the deployment of Cilium within a Kubernetes cluster, aiming to validate its efficacy in enhancing the security and networking capabilities. This study will be conducted using MicroK8s v1.28.3 for Kubernetes and Cilium version 1.13.4 with Hubble UI enabled and Cilium Service Mesh disabled. This is all run on a VirtualBox VM with Ubuntu version 20.04.1. The following sections will go over networking, observability, security, and benchmarks. First, we will look at how Cilium CNI uses eBPF technology to simplify networking within a Kubernetes Cluster. Then, we will see how Cilium provides fine-grained visibility into network traffic using Cilium's Hubble UI. Next, we will take a look at how Cilium provides security using L3/L4, L7, and DNS based networking policies. Lastly we will run a few benchmarks to see how Cilium eBPF performs compared to other networking solutions. These findings hope to extend the understanding of Cilium's role in fortifying the security of modern cloud infrastructures.

2 Motivation

I have chosen to do a study on Cilium to gain deeper insight on containerized technology and the tooling used within them. With Docker and Kubernetes becoming mainstream, understanding how tools like Cilium secure containerized applications has become crucial. My motivation is simple: I want to know how Cilium works, how it contributes to making containerized applications more secure, and why it is competitive to other networking options out there.

In addition, this study gives me an opportunity to learn about the use of eBPFs. This technology not only allows Cilium to indirectly control logic within the Linux kernel without the complexities of writing a kernel module but also introduces a level of safety and efficiency that is revolutionary in the realm of container network security.

This study is a practical exploration. I want to grasp how Cilium's technical aspects, particularly eBPF, make container security efficient and fine-tuned. My goal is not just to gain knowledge but also to see how this understanding can be applied to enhance the security of containerized applications. In essence, this study is a step towards becoming well-versed

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

in securing modern cloud-native architectures and leveraging tools like Cilium to do so.

3 Understanding Cilium

In this section we will be going over the architecture of Cilium CNI, how ebpf technology is used in Cilium, Hubble UI, and how security policies are implemented in Cilium.

Firstly, what is Cilium CNI? Cilium CNI is a container networking interface. This means that Cilium is responsible for managing the network connectivity between containers within a Kubernetes cluster. As a CNI, Cilium ensures that communication between different containers is seamless, efficient, and secure. Cilium also leverages Kubernetes labels and selectors to define fine-grained policies based on pod identity, namespace, labels, and other attributes to enforce network policies, perform load balancing, and enable service discovery within the cluster.

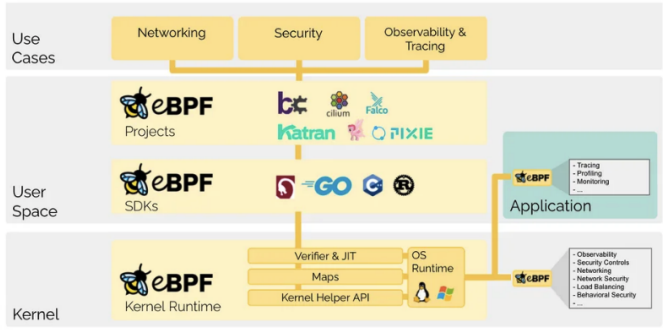

Figure 1. eBPF diagram.

At its core, Cilium uses eBPF technology. By leveraging eBPF, Cilium can intercept and process network packets at various stages within the Linux kernel, enabling quick decision-making based on predefined policies. This is achieved by writing custom eBPF programs that exist in the User space that utilize designated "hooks" to interact directly to the eBPF runtime operating in the Kernel space. These hooks serve as entry points for executing the custom logic, facilitating the enforcement of security policies, load balancing, and other network-related tasks directly within the kernel.

In addition, the eBPF programs in Cilium can be dynamically loaded and updated without requiring kernel module changes or system restarts. This dynamic adaptability ensures that Cilium can swiftly respond to changes in network conditions, security requirements, or the addition/removal of containers within the Kubernetes cluster. The efficiency of eBPF's in-kernel execution minimizes context-switching overhead, contributing to Cilium's high-performance network management.

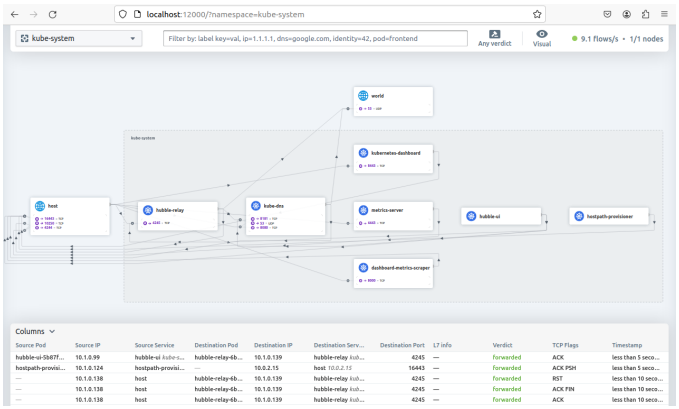

Figure 2. Hubble UI.

Cilium also provides observability through the Hubble UI, a component that provides a visual interface for monitoring and understanding network traffic within the Kubernetes cluster. Hubble complements Cilium's capabilities by offering insights into communication patterns, allowing for real-time visibility into how containers interact with each other. This feature is especially useful when you need to troubleshoot complex network issues in your kubernetes infrastructure.

Lastly, Cilium provides comprehensive security policies that enable administrators to define and enforce granular controls over communication between microservices. Understanding how Cilium enforces security policies at different layers of the OSI model is crucial for ensuring the integrity and confidentiality. Specifically, Cilium allows configuration of L3/L4, L7, and DNS based networking policies. L3/L4 policies enable administrators to define rules based on network and transport layer attributes, while L7 policies allow for application-specific controls, such as routing based on URL paths or HTTP methods. The inclusion of DNS-based policies introduces a service-centric approach, where policies are defined based on domain names. Since all these policies are enforced in the Kernel space, Cilium ensures a consistent and efficient application of security controls, regardless of the specific policy type. This approach is particularly advantageous in dynamic Kubernetes environments, where container workloads can change rapidly.

4 Our Architecture

In this section, we will go over the architecture used for a demo showing Cilium networking policies in action. We will also cover the architectural setup for running benchmark testing for Cilium's ebpf host routing.

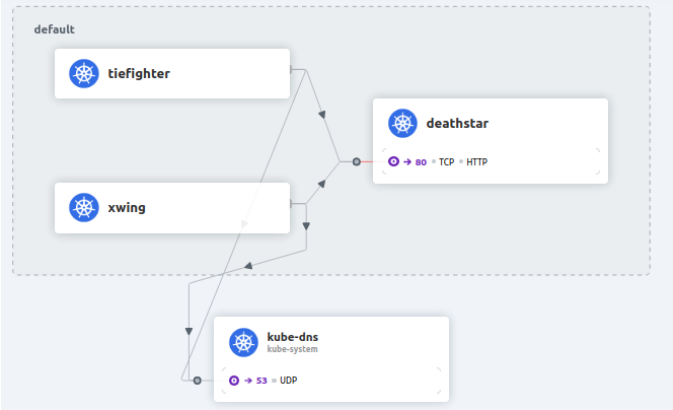

Figure 3. Hubble view of StarWars microservices within our cluster.

To show Cilium networking policies in action we have a StarWars demo that replicates the interaction between different spaceships in the StarWars universe. We are running one MicroK8s cluster. Within, we have three microservices applications we will be using: deathstar, tiefighter, and xwing. The deathstar runs an HTTP webservice on port 80, and is load-balanced across two replica pods. The deathstar service provides "landing services" to the empire's spaceships so that they can request a landing port. The tiefighter pod represents a landing-request client service on a typical empire ship and xwing represents a similar service on an alliance ship. These two pods are for testing different security policies for access control to deathstar landing services.

```
security@security-VirtualBox:~$ kubectl -n kube-system exec cilium-n6dcx -- cilium policy get
Defaulted container "cilium-agent" out of: cilium-agent, config (init), mount-cgroup (init), a
it)
[
  {
    "endpointSelector": {
      "matchLabels": {
        "any:class": "deathstar",
        "any:org": "empire",
        "k8s:io.kubernetes.pod.namespace": "default"
      }
    },
    "ingress": [
      {
        "fromEndpoints": [
          {
            "matchLabels": {
              "any:org": "empire",
              "k8s:io.kubernetes.pod.namespace": "default"
            }
          }
        ],
        "toPorts": [
          {
            "ports": [
              {
                "port": "80",
                "protocol": "TCP"
              }
            ],
            "rules": {
              "http": [
                {
                  "path": "/v1/request-landing",
                  "method": "POST"
                }
              ]
            }
          }
```

Figure 4. L3/L4 and L7 policies for deathstar service.

In our demonstration, we implement both L3/L4 and L7 policies. Above we can see that we restrict ingress to the deathstar service with labels (L3/L4) and with API rules (L7). Only ships with the label "org:empire" are allowed to submit API calls to the deathstar. On top of that, only POST requests are allowed to be made to the deathstar service.

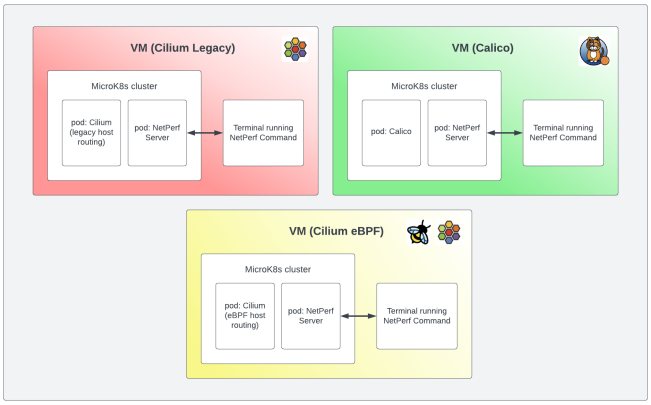

Figure 5. Block diagram for benchmark architecture.

We also wanted to conduct throughput and latency benchmarks to assess the overheads associated with using Cilium eBPF compared to other CNIs. To do this, we set up three separate VMs. The first VM runs microK8s with Cilium legacy host routing (without eBPF), the second VM runs microK8s with Cilium eBPF host routing, and the third VM runs microK8s with Calico. In each of these environments, we deploy a pod running a NetPerf net server listening on port 12865. This benchmark setup allows us to assess the impact of adopting Cilium's eBPF host routing on the overall performance.

Next, we ran two sets of tests (throughput and latency) on each of the VMs. We ran each of these tests 8 times using these two commands:

```
netperf -H 10.1.0.214 -l 30 -t TCP_STREAM
netperf -H 10.1.0.214 -l 30 -t TCP_RR
```

Figure 6. Netperf-Commands.

With the results, we created two box plots, one for throughput and another for latency. We chose to use box plots because they provide the most accurate visualization of our results given our small sample size. Both of which we will see in the next section.

5 Experimental Results

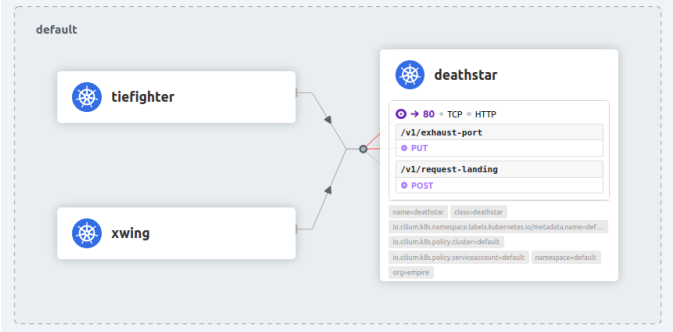

Figure 7. Hubble UI showing possible API connections to deathstar service.

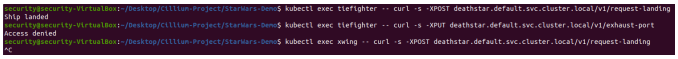

Figure 8. Result of applying Cilium networking policies.

Here are the results of the StarWars demo that aims to show Cilium networking policies in action.

Firstly, you can see that if the xwing, with the label "org=alliance", requests a landing on deathstar service, the request will hang. This is a demonstration of defining a L3/L4 "identity-aware" networking rule as the xwing pod isn't able to make a connection to the service.

You can also see that if a tiefighter pod submits a PUT request on the deathstar service, the pod is denied access. This is an example of a Cilium L7 security policy because even though the pod has access to the service, it is only permitted certain API resources. This is also known as a "least privilege" security approach for communication between microservices which is generally good practice.

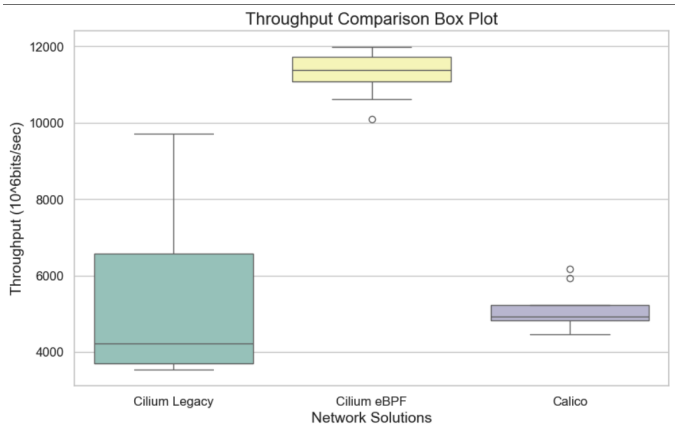

Figure 9. Box diagram for throughput benchmarks for three CNIs.

In Figure 9, we have a box plot of throughput benchmarks for the three configurations we mentioned before: Cilium with Legacy host routing, Cilium with eBPF host routing, and Calico. Throughput is the rate at which data is successfully transferred between endpoints in the network. From this figure, we can see that Cilium eBPF has, by far, the most throughput per second and thus the best performance compared to the other two configurations. These results are aligned with the benchmarks that Cilium has on its website[11].

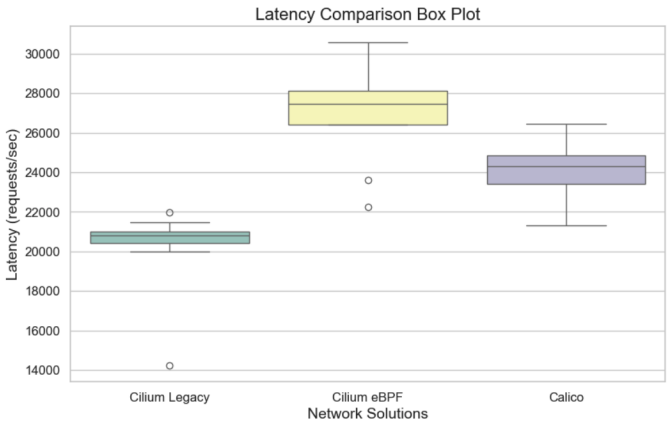

Figure 10. Box diagram for latency benchmarks for three CNIs.

In Figure 10, we have a box plot of latency benchmarks for the same three configurations. Latency measures how efficiently a single network packet can be processed. Lower latency is better. It also means that more packets are processed per second. From this figure, we can see that Cilium eBPF processes the most requests per second and thus has by far the best performance compared to the other two configurations. These results are also in line with the benchmarks that Cilium has on its website[11].

6 Related Work

One related work using eBPF technology is Pyroscope. Pyroscope is an open-source solution that specializes in continuous profiling[13]. Deployable on Linux, Docker, and Kubernetes, Pyroscope offers a unique capability to ingest profiles not only from its dedicated Pyroscope Agents but also from other profilers available on the market. This versatility makes Pyroscope a compelling choice for organizations seeking efficient and non-intrusive profiling methods. With the use of eBPFs Pyroscope lets you profile without the need to modify application code, thereby minimizing overhead. In addition, its integration with OpenTelemetry, support for a query language named FlameQL, and the ability to export metrics enhance its observability features, making Pyroscope a valuable tool for understanding resource consumption and

optimizing code performance in many production environments.

Another tool that is starting to leverage the powers of eBPF technology is Datadog. Datadog has introduced Universal Service Monitoring (USM), a feature that automatically detects and monitors all services in complex, dynamic environments[14]. Through the use of eBPF, Datadog's USM enables automatic parsing of HTTP traffic, providing visibility into critical metrics such as request, error, and duration for every service. It also has a Service Map feature that allows users to visualize dependencies between services, aiding in incident resolution by understanding the relationships within a system. With USM, Datadog aims to simplify the monitoring of service health, streamline troubleshooting processes, and enhance overall observability.

Lastly, as with any technology, the landscape of network security is dynamic, making it important to remain aware of potential vulnerabilities. Resources such as the vulnerability database on Snyk (Cilium Vulnerability Database) provide insights into potential exploits and vulnerabilities associated with different versions of Cilium[12]. This awareness is crucial for maintaining the integrity and security of Cilium deployments, allowing organizations to make informed decisions about version compatibility and mitigation strategies.

7 Conclusions

This study has provided valuable insights into the deployment and performance of Cilium in a Kubernetes cluster. By examining Cilium's networking, security, and observability capabilities, we aimed to validate its efficacy and gain a broader understanding of its role in securing containerized applications.

The architecture of Cilium CNI, its utilization of eBPF technology, Hubble UI for observability, and the implementation of fine-grained security policies were thoroughly explored. The StarWars demo demonstrated how Cilium's policies can be applied to control access and interactions between microservices, showcasing its versatility in securing modern cloud-native architectures.

Benchmarking further confirmed Cilium's exceptional performance, especially with eBPF host routing, which outperformed legacy host routing and alternative solutions like Calico in terms of both throughput and latency. These results align with benchmarks provided by Cilium, reaffirming its reputation for scalability and efficiency.

By exploring Cilium and its use of eBPF technology, I have gained valuable insights into the intersection of containerized networking, security, and observability within Kubernetes environments. In addition, the practical understanding acquired through this study will enable me to more confidently navigate the complexities of modern cloud-native architectures in the future.

8 References

[1] A. Dinaburg, "Pitfalls of relying on EBPF for Security Monitoring (and some solutions)," Trail of Bits Blog, https://blog.trailofbits.
of-relying-on-ebpf-for-security-monitoring-and-some-solutions/ (accessed Dec. 13, 2023).

[2] A. Zhang, "Kubernetes Network Learning with Cilium and EBPF," Medium, https://addozhang.medium.com/kubernetes-network-learning-with-cilium-and-ebpf-aafbf3163840 (accessed Dec. 13, 2023).

[3] "What is eBPF? an introduction and deep dive into the EBPF technology," What is eBPF? An Introduction and Deep Dive into the eBPF Technology, https://ebpf.io/what-is-ebpf/ (accessed Dec. 13, 2023).

[4] "Linux runtime security agent powered by EBPF: Hacker News," Linux runtime security agent powered by eBPF | Hacker News, https://news.ycombinator.com/item?id=37942791: :text=

[5] D. GERMAIN, "Migrating cilium from legacy iptables routing to native EBPF routing in production," Medium, https://deezer.io/migrating-cilium-from-legacy-iptables-routing-to-native-ebpf-routing-in-production-84a035af1cd6: :text=This

[6] S. C. Amaechi, "Cilium: Empowering kubernetes networking and security," Medium, https://medium.com/cloud-native-daily/cilium-empowering-kubernetes-networking-and-security-9d25750e8f44 (accessed Dec. 13, 2023).

[7] D. Lewis, "Hubble series (part 3): Cilium Hubble and Grafana Better together," Isovalent, https://isovalent.com/blog/post/cilium-hubble-with-grafana/ (accessed Dec. 13, 2023).

[8] A. Gupta, Isovalent Enterprise for Cilium on EKS/EKS-A in AWS Marketplace, https://isovalent.com/blog/post/isovalent-aws-marketplace/ (accessed Dec. 13, 2023).

[9] J. Colvin, "What is Kube-proxy and why move from Iptables to ebpf?nbsp;," Isovalent, https://isovalent.com/blog/post/why-replace-iptables-with-ebpf/ (accessed Dec. 13, 2023).

[10] Cilium Authors, "Welcome to Cilium's documentation! - cilium 1.14.5 documentation," Welcome to Cilium's documentation! - Cilium 1.14.5 documentation, https://docs.cilium.io/en/st
(accessed Dec. 13, 2023).

[11] "CNI benchmark: Understanding cilium network performance," Cilium, https://cilium.io/blog/2021/05/11/cni-benchmark/ (accessed Dec. 13, 2023).

[12] "Github.com/cilium/cilium vulnerabilities: Snyk," Find detailed information and remediation guidance for vulnerabilities and misconfigurations., https://security.snyk.io/package/golang/gi

[13] G. D. Pietro, "What is continuous profiling, and what is Pyroscope?," Is It Observable, https://isitobservable.io/open-telemetry/what-is-continuous-profiling-and-what-is-pyroscope (accessed Dec. 13, 2023).

[14] S. Pinkerton, "Datadog," Automatically discover, map, and monitor all your services in seconds with Universal Service Monitoring, https://www.datadoghq.com/blog/universal-service-monitoring-datadog/ (accessed Dec. 13, 2023).