Scaling Policy Assignment in Containerized Environment

Andrej Knapp
École Normale Supérieure
France
andrej.knapp@ens.fr

Madhav Linga
ColdBolt Inc.
France
madhav.l@coldbolt.io

Shuri Yen
École Normale Supérieure
France
shuri.yen@coldbolt.io

***Abstract*—In containerized environments, securing inter-service communication is paramount, and Mutual TLS (mTLS) offers a robust solution. However, traditional mTLS implementations face challenges in scaling and scalable environments. This paper introduces a novel approach to mTLS using Scaling Policy Assignment (SPA), which leverages Kubernetes' native capabilities to automate and optimize policy application based on service requirements. SPA enables flexible and efficient mTLS enforcement, scaling seamlessly with the environment's complexity. This approach enhances security, reduces manual configuration, and ensures consistent policy application across services.**

***Index Terms*—encryption, mutual-tls, containers, security policy**

I. Introduction

As organizations increasingly adopt containerized environments and microservices architectures, ensuring secure communication between services has become a critical concern. Mutual TLS (mTLS) is a security protocol that ensures both authentication and encryption between services, providing a strong defense against unauthorized access and man-in-the-middle attacks. However, the scaling and ephemeral nature of containers introduces challenges in managing and enforcing mTLS policies.

In a traditional mTLS setup, policies are often statically defined, requiring manual configuration and management. This approach is not scalable and can lead to inconsistencies [1], particularly in environments with a high rate of change. Scaling Policy Assignment (SPA) offers a solution by automating the assignment of mTLS policies based on service requirements, operational contexts, and security needs. SPA ensures that the right policies are applied to the right services at the right time, simplifying the management of mTLS and enhancing the security of containerized environments.

This paper explores the concept of Scaling Policy Assignment for mTLS, detailing its implementation in Kubernetes, integration with service meshes, and benefits in terms of security, scalability, and operational efficiency.

II. Background and Related Work

A. Mutual TLS in Containerized Environments

Mutual TLS is a widely adopted protocol for securing communication in microservices architectures [2] [3]. It involves two-way authentication, where both the client and the server present certificates to verify each other's identity. This ensures that communication occurs only between trusted parties, with encrypted data protecting against eavesdropping and tampering.

In containerized environments, where services are deployed as containers, mTLS implementation becomes challenging due to the scaling nature of containers. Containers are ephemeral, meaning they can be created, destroyed, and moved frequently. This requires mTLS policies to be flexible and adaptable to ensure that all services are consistently secured.

B. Challenges of Traditional mTLS Policy Management

Traditional mTLS implementations often rely on static configurations, where policies are manually defined and applied to services. This approach has several drawbacks in containerized environments:

- Manual Configuration: Requires significant manual effort to define and update mTLS policies, leading to potential misconfigurations.
- Inflexibility: Static policies do not adapt to changes in the environment, such as the deployment of new services or scaling of existing ones.
- Scalability Issues: As the number of services grows, managing and enforcing mTLS policies becomes increasingly complex and prone to errors.

C. Scaling Policy Assignment: A Novel Approach

Scaling Policy Assignment (SPA) addresses these challenges by automating the process of policy definition and enforcement. SPA leverages Kubernetes' native capabilities, such as labels, namespaces, and custom resource definitions (CRDs), to dynamically assign mTLS policies based on real-time service requirements and operational contexts. This approach ensures that mTLS policies are consistently applied, even as the environment changes, enhancing security and reducing the operational burden.

III. Scaling Policy Assignment: Concept and Implementation

A. Concept of Scaling Policy Assignment

Scaling Policy Assignment is built on the principle that mTLS policies should be flexible and adaptive, responding to the scaling nature of containerized environments. In SPA,

policies are not statically defined but are dynamically assigned based on various factors, including:

- Service Labels: Tags or labels associated with services in Kubernetes, such as "sensitive-data" or "external-facing," that indicate the level of security required.
- Namespaces: Kubernetes namespaces that group related services [4], allowing for policies to be applied at the namespace level for consistency.
- Operational Context: Real-time operational data, such as the current load on a service, its network configuration, or the presence of known vulnerabilities, which can trigger the application of specific mTLS policies. This scaling approach allows policies to be automatically assigned and enforced without manual intervention, ensuring that all services are secured according to their current operational needs.

B. Implementation in Kubernetes

Implementing Scaling Policy Assignment in Kubernetes involves several key components:

- Policy Definitions: mTLS policies are defined as custom resources in Kubernetes. These policies specify the authentication and encryption requirements for services, including certificate authorities (CAs), key lengths, and encryption algorithms.
- Label-Based Assignment: Services are labeled with metadata that indicates their security requirements. For example, a service handling sensitive data might be labeled as "sensitive," triggering the assignment of a stricter mTLS policy.
- Namespace-Based Policies: Policies can also be applied at the namespace level [5], ensuring that all services within a namespace adhere to the same security standards. This is particularly useful for environments where services are grouped based on their function or sensitivity.
- Automated Policy Enforcement: Kubernetes operators and controllers are used to monitor services and enforce policies. When a new service is deployed or an existing service is updated, the controller checks the service's labels and assigns the appropriate mTLS policy dynamically.
- Policy Updates and Rotation: SPA also supports the automatic rotation of certificates and updating of policies in response to changing security requirements. This ensures that services always use up-to-date security configurations without manual intervention.

C. Continuous Policy Adaptation

The final component of the SPA framework is continuous policy adaptation. This involves monitoring the environment for changes and adjusting mTLS policies in response to these changes. The SPA framework continuously collects data on the state of the environment, including service deployments, traffic patterns, and security events.

Based on this data, the SPA framework can make real-time adjustments to mTLS policies [3]. For example, if a new service is deployed in a sensitive environment, the framework might automatically tighten the mTLS policies for that service. Similarly, if the traffic between two services increases significantly, the framework might optimize the mTLS policies to maintain performance [6] while ensuring security.

Continuous policy adaptation is facilitated by the integration of the SPA framework with observability tools and security monitoring systems. These tools provide the necessary data to make informed decisions about mTLS policies, ensuring that the security of the environment is maintained without compromising performance or operational efficiency.

IV. Implementation and Case Study

To demonstrate the effectiveness of the Scaling Policy Assignment framework, we implemented it in a Kubernetes environment using Istio [3], [7] as the service mesh. We applied the framework to a microservices application composed of several services with varying levels of security requirements.

A. Implementation Details

Service Metadata Labeling: Each service in the application was labeled with metadata that described its security requirements [8]. For example, the payment service was labeled as "sensitive" and "production," while the logging service was labeled as "non-sensitive" and "development."

- Policy Engine: We developed a policy engine [9] that monitored the service metadata and generated mTLS policies based on predefined rules. The engine was implemented as a Kubernetes controller that continuously watched for changes in service labels [7].
- Service Mesh Integration: The policy engine communicated with Istio to apply the generated mTLS policies. Istio's sidecar proxies were configured to enforce these policies dynamically, ensuring that all communication between services was secure.
- Continuous Adaptation: The SPA framework was integrated with Prometheus and Grafana for observability. These tools provided real-time metrics on service performance and security events, allowing the policy engine to make informed decisions about mTLS policy adjustments.

B. Case Study: Securing a Financial Application

We applied the SPA framework to a financial application composed of several microservices, including payment processing, user authentication, and transaction logging. The application was deployed in a Kubernetes cluster, with Istio managing the service-to-service communication.

Initial Policy Assignment: Upon deployment, the SPA framework automatically assigned mTLS policies [10] based on the service metadata. The payment processing service, labeled as "sensitive," was assigned a strict mTLS policy with strong encryption and frequent certificate rotations. The transaction logging service, labeled as "non-sensitive," was assigned a more relaxed mTLS policy.

Real-Time Enforcement: As the application began processing transactions, the service mesh enforced the assigned mTLS policies. All communication between the payment processing service and other services was encrypted using mTLS, ensuring that sensitive data was protected.

Continuous Adaptation: During a high-traffic period, the SPA framework detected an increase in traffic between the user authentication service and the payment processing service [9]. In response, the framework optimized the mTLS policies to maintain performance while ensuring security. This included adjusting the frequency of certificate rotations [11] and optimizing traffic management rules in the service mesh.

V. Results and Evaluation

The Scaling Policy Assignment framework was evaluated on several key metrics, including security, performance, and scalability.

- Security: The SPA framework successfully enforced mTLS policies across all services, ensuring that sensitive data was protected. The framework's ability to adapt policies in real-time prevented potential security vulnerabilities that could arise from outdated or misaligned policies.
- Performance: The integration with the service mesh ensured that mTLS policies were enforced with minimal impact on service performance [10]. The continuous policy adaptation mechanism allowed the framework to maintain a balance between security and performance, even during periods of high traffic.
- Scalability: The SPA framework demonstrated the ability to scale with the environment, managing mTLS policies across a large number of services without introducing significant overhead. The use of Kubernetes' labeling system and the policy engine's automation capabilities allowed the framework to handle complex environments efficiently.

VI. Discussion

The Scaling Policy Assignment framework addresses the primary challenges of implementing mTLS in containerized environments. By automating policy generation, enforcing policies in real-time, and continuously adapting to changes, the framework provides a robust solution for securing service-to-service communication in microservices architectures.

One of the key strengths of the SPA framework is its flexibility. The use of service metadata labels allows for fine-grained control over mTLS policies, ensuring that each service is protected according to its specific security requirements. This flexibility is particularly important in environments where services have varying levels of sensitivity and where the security landscape can change rapidly.

Another important aspect of the SPA framework is its integration with existing tools and platforms. By leveraging Kubernetes' native capabilities and integrating with service meshes like Istio, the framework can be easily adopted in existing environments without requiring significant changes to the underlying infrastructure.

However, there are also some limitations to the SPA framework. One of the challenges is ensuring that the policy engine's rules are comprehensive and up-to-date. As the environment evolves, new services and security requirements may emerge, requiring updates to the policy generation rules. This highlights the need for ongoing maintenance and monitoring of the SPA framework to ensure that it continues to meet the security needs of the environment.

VII. Conclusion

Scaling Policy Assignment offers a novel and effective approach to implementing mTLS in containerized environments. By automating the generation and enforcement of mTLS policies and adapting these policies in real-time, the SPA framework provides a scalable and flexible solution for securing microservices communication.

As containerized environments continue to grow in complexity, the need for scaling and adaptable security solutions will only increase. The SPA framework addresses this need by providing a robust and scalable approach to mTLS policy management, ensuring that organizations can maintain the security of their microservices applications without sacrificing performance or operational efficiency.

VIII. Future Work

Future research on Scaling Policy Assignment could explore several areas:

Advanced Policy Rules: Developing more sophisticated rules for policy generation that take into account additional factors such as service dependencies, historical traffic patterns, and threat intelligence data.

Extended Observability: Enhancing the observability capabilities of the SPA framework to provide deeper insights into the effectiveness of mTLS policies and to detect potential security threats in real-time.

Broader Compatibility: Extending the SPA framework to support a wider range of container orchestration platforms and service meshes, ensuring that it can be adopted in diverse environments.

User-Centric Policies: Investigating ways to incorporate user and developer feedback into the policy generation process, ensuring that mTLS policies are aligned with the needs and expectations of the organization.

References

[1] M. P. Andersen, J. Kolb, K. Chen, G. Fierro, D. E. Culler, and R. Katz, "Democratizing authority in the built environment," *ACM Transactions on Sensor Networks (TOSN)*, vol. 14, no. 3-4, pp. 1–26, 2018.

[2] R. Bahmani, F. Brasser, G. Dessouky, P. Jauernig, M. Klimmek, A.-R. Sadeghi, and E. Stapf, "{CURE}: A security architecture with {CUstomizable} and resilient enclaves," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021, pp. 1073–1090.

[3] S. Ashe and H. Ramachandra, "The effect of continuous encryption of data in cloud native architecture," in *2024 IEEE Cloud Summit*. IEEE, 2024, pp. 163–169.

[4] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner, "S-faas: Trustworthy and accountable function-as-a-service using intel sgx," in *Proceedings of the 2019 ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 185–199.

[5] Y. Ren, G. Liu, V. Nitu, W. Shao, R. Kennedy, G. Parmer, T. Wood, and A. Tchana, "{Fine-Grained} isolation for scalable, dynamic, multi-tenant edge clouds," in *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, 2020, pp. 927–942.

[6] T. Jager, F. Kohlar, S. Schäge, and J. Schwenk, "Authenticated confidential channel establishment and the security of tls-dhe," *Journal of Cryptology*, vol. 30, pp. 1276–1324, 2017.

[7] C. Cremers, M. Horvat, J. Hoyland, S. Scott, and T. Van Der Merwe, "A comprehensive symbolic analysis of tls 1.3," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 1773–1788.

[8] K. A. Bailey and S. W. Smith, "Trusted virtual containers on demand," in *Proceedings of the fifth ACM workshop on Scalable trusted computing*, 2010, pp. 63–72.

[9] J. Huang, C. Xiao, and W. Wu, "Rlsk: A job scheduler for federated kubernetes clusters based on reinforcement learning," in *2020 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2020, pp. 116–123.

[10] M. Zhang, D. Marino, and P. Efstathopoulos, "Harbormaster: Policy enforcement for containers," in *2015 IEEE 7th International Conference on Cloud Computing Technology and Science (CloudCom)*. IEEE, 2015, pp. 355–362.

[11] T. Polk, K. McKay, S. Chokhani *et al.*, "Guidelines for the selection, configuration, and use of transport layer security (tls) implementations," *NIST special publication*, vol. 800, no. 52, p. 32, 2014.