Poster: MAZU: A Zero Trust Architecture for Service Mesh Control Planes (#54)

Aashutosh Poudel
William & Mary
apoudel01@wm.edu

Pankaj Niroula
William & Mary
pniroula@wm.edu

Collin MacDonald
William & Mary
cmacdonald01@wm.edu

Lily Gloudemans
William & Mary
algloudemans@wm.edu

Stephen Herwig
William & Mary
smherwig@wm.edu

***Abstract*—Microservices are a dominant cloud computing architecture because they enable applications to be built as collections of loosely coupled services. To provide greater control over the resultant distributed system, microservices often use an overlay proxy network called a service mesh. A key advantage of service meshes is their ability to implement zero trust networking by encrypting microservice traffic with mutually authenticated TLS. However, the service mesh control plane—particularly its local certificate authority—becomes a critical point of trust. In this poster, we introduce MAZU, a system designed to eliminate trust in the service mesh control plane by replacing its certificate authority with an unprivileged principal. MAZU leverages recent advances in registration-based encryption and integrates seamlessly with Istio, a widely used service mesh. We present our preliminary implementation and highlight future work.**

Background. In cloud computing, a common software architecture is *microservices*: rather than deploy a large, monolithic application, the software developers decompose the application into a distributed system of small, loosely-coupled services, that communicate via well-defined interfaces. The primary benefits of microservices are twofold: *elasticity* (the cloud can scale each component independently), and *isolation* (a failed component does not, by itself, cause other components to fail).

The microservice architecture, being inherently distributed, presents unique challenges compared to monolithic applications in terms of reliability, observability, and security. A *service mesh* serves as middleware to address these challenges, allowing each microservice to focus exclusively on its application logic. In most service meshes, each microservice runs in an application container and is paired with a proxy container, known as a *sidecar*. The sidecar modifies the host's network routing so that all traffic to and from the application container flows through it. This design enables the sidecar to manage microservice traffic—providing features such as authentication, authorization, and logging—without requiring any modifications to the application itself.

Problem. A key feature of service meshes is zero trust networking, where the sidecars tunnel communication between microservices using mutually authenticated TLS (mTLS). To support this, the service mesh control plane acts as a certificate authority (CA), issuing certificates to the sidecar proxies. Additionally, this control plane also provisions sidecars with authorization rules that define which peer sidecars are allowed to connect. Although zero trust networking stymies an attacker's ability to laterally move in the victim's network, if an attacker compromises the service mesh's control plane, *the attacker can issue rogue certificates, impersonate applications, and redirect traffic to malicious endpoints—effectively subverting the entire system.*

In this project, we ask the following research question:

Is it possible to reduce trust in the service mesh's control plane while maintaining microservice compatibility and performance?

Threat model. For concreteness, we define our threat model in terms of the popular Istio service mesh, but note that Istio's architecture is representative of other meshes. Istio logically comprises a *control plane*—which handles traffic rules, logging, authorization, and certificate issuance—, and a *data plane*—namely, the sidecar proxies. Within the Kubernetes cluster, each node (a physical or virtual machine) runs an Istio *node agent*, which serves as an interface between the Istio control plane and the data plane.

We trust Kubernetes to manage the pods and nodes within the cluster. We assume an attacker has remote code execution on the cluster and can exploit the Istio control plane. In particular, the attacker can issue rogue certificates, spawn microservices to impersonate legitimate applications, and configure routing rules that redirect traffic to these malicious microservices. We trust the Istio node agent and sidecar, but allow for bugs in these components that leak sensitive data, such as network credentials. We do not explicitly trust the microservice applications; however, we assume that their containers are hardened to prevent a container escape.

Goals. Our primary security goal when designing our solution, MAZU,[1] is

S1 *Untrusted Service Mesh Control Plane*: An attacker that breaches the service mesh's control plane must not be able to undermine the confidentiality or integrity of the microservice application's network communications.

Additionally, we have the following functional goals:

F1 *Application transparency*: MAZU should preserve the core property of service meshes, which is that the application itself remains unmodified.

[1]MAZU is a Chinese sea goddess and the deity of seafarers.

F2 *Compatibility with existing service meshes*: MAZU should extend current service mesh software (we use Istio).

F3 *Low performance overheads*: MAZU should impose little performance overhead on the microservice applications, both in terms of client latency and resource usage.

Registration-based encryption. At a high-level, MAZU achieves its goals by replacing the CA with a decentralized protocol based on the recently introduced concept of *Registration-Based Encryption (RBE)* [1], [2]. RBE is an alternative to identity-based encryption (IBE) that eliminates IBE's key-escrow problem. Recall that Shamir introduced IBE [3] as a public-key cryptosystem that removes the need for a complex public-key infrastructure. Instead, in an IBE scheme, a user uses a meaningful identity, such as their email address, as their public key. Alice can then encrypt a message to Bob knowing just Bob's identity, along with some additional public parameters. In turn, Bob decrypts Alice's message using an identity-specific secret key that he obtains from the *key authority*. Unfortunately, the key authority generates the secret key for each user, effectively becoming a *key escrow* and a target for undermining the entire system.

RBE replaces IBE's key authority with a weaker principal called the *key curator (KC)* that does not have knowledge of any secret key (or any secret information). A user in an RBE system locally generates their keys and then publicly registers their identity and corresponding public key with the KC. In response, the KC updates the public parameters of the system and returns to the user some supplementary, non-secret, information called the user's *opening*, which is necessary for the user to decrypt ciphertexts. As new users register and the public parameters change, existing users need to contact the KC to fetch their updated opening. Encryption and decryption work analogously as in IBE.

Design & implementation. Our prototype implementation of MAZU extends the Istio service mesh. By default, Kubernetes provisions each microservice instance with a Kubernetes-signed *admin token* containing the instance's internal URL, as well as the IDs for its cloud service account, node, and pod. In a normal Istio deployment, the node agent attests to the CA with this token as part of the certificate issuance. In contrast, MAZU uses the token to register a unique RBE ID for the instance, with the ID being the hash of the token. When registering this ID, the node agent locally generates the corresponding keypair, choosing as its private key a hash of the token and the service's IP address. Whereas RBE's original purpose is to encrypt messages, we instead use RBE purely for registration, and the non-secret choice of a private key serves as a one-time signature of registration. The node agent then generates a self-signed TLS certificate for the service that embeds the admin token.

The sidecar operates as a layer-7 (HTTP or gRPC) proxy. During the TLS handshake between two sidecars, the client sidecar verifies that the server's certificate includes a signed admin token for the intended destination URL. Additionally, the client sidecar derives the server's ID as a hash of the received token. To verify that the server registered this ID, the client locally encrypts a challenge nonce to that ID, fetches from the KC the ID's public opening, and derives the expected secret key using the token and the server's IP. If the client can decrypt the nonce, validation succeeds and the client continues with the connection. The server's validation of the client works in an analogous fashion.

Security analysis. Suppose an attacker $\mathcal{A}$ leaks B's token and tries to impersonate B. For now, we assume $\mathcal{A}$ cannot acquire the same IP as B, but that $\mathcal{A}$ can compromise Istio's control plane to route services to itself. During the TLS handshake, the peer service will derive an incorrect candidate secret key due to the mismatch in the server's IP, fail to decrypt the nonce, and thus abort the connection.

Suppose now that Kubernetes tears down B, allowing $\mathcal{A}$ to reuse B's IP to launch a malicious service. If a peer service connects to this malicious service, the peer will derive the exact same RBE private key as for the retired B, and MAZU's custom TLS validation checks will pass. To counter this threat, we note that Kubernetes automatically invalidates a service's token when tearing down that service. Thus, we amend the sidecar's certificate validation to also query Kubernetes for the validity of a token.

Finally, a quirk of RBE is that anyone can unregister an ID. While MAZU does not prevent re-registration attacks, it does make such an attack detectable. For each registration, the (untrusted) KC updates each ID's opening; a node agent periodically polls the KC for the updated openings for its resident services. We modify this operation so that the KC returns to the node agent a history of updates, where each update includes the token, the registered public key, and an RBE proof that the corresponding ID was registered. In this way, ID re-registration is auditable.

Preliminary evaluation. We evaluate MAZU using the Fortio load testing tool. Our results show that MAZU introduces minimal overhead, adding just 0.17 ms of latency for typical workloads compared to Istio with mTLS. We also conduct a Prometheus-aided micro-benchmark to measure the latency of update queries as the number of registered services increases. The results indicate that update latency grows approximately linearly with the number of services.

Future work. As future work, we will evaluate the performance of MAZU using additional popular benchmarks, such as the DeathStarBench suite. We also plan to integrate a lightweight consensus protocol that mitigates a Byzantine-faulty KC that presents different registration histories to different microservices.

REFERENCES

[1] S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi, "Registration-based encryption: Removing private-key generator from IBE," in *Theory of Cryptography Conference (TCC)*, 2018.

[2] N. Glaeser, D. Kolonelos, G. Malavolta, and A. Rahimi, "Efficient registration-based encryption," in *ACM Conference on Computer and Communications Security (CCS)*, 2023.

[3] A. Shamir, "Identity-based cryptosystems and signature schemes," in *International Cryptology Conference (CRYPTO)*, 1984.

MAZU: A Zero Trust Architecture for Service Mesh Control Planes

WILLIAM & MARY
CHARTERED 1693

Aashutosh Poudel — apoudel01@wm.edu
Pankaj Niroula — pniroula@wm.edu
Collin MacDonald — cmacdonald01@wm.edu
Lily Gloudemans — algloudemans@wm.edu
Stephen Herwig — smherwig@wm.edu

/etc/lab
Extending Trust in Computing Lab

Background

Microservices: Modern software architecture where applications are split into small, independent services that communicate through APIs - enabling elastic scaling and fault isolation

Service Mesh: Infrastructure layer that manages communication between microservices, handling security and reliability without modifying the services themselves

Zero Trust Networking: Security model that requires authentication and authorization for all service-to-service communication, treating every request as potentially hostile regardless of its origin

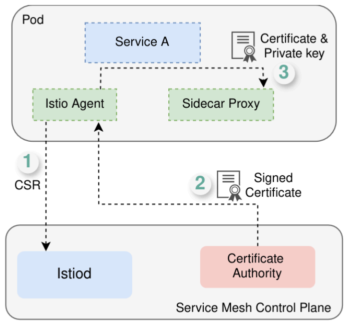


Figure 1: Identity and certificate management in Istio Service Mesh

Problems

- Secure service-to-service communication in a Service Mesh is based on TLS certificates issued by mesh-local CA
- Cloud providers offering managed Service Mesh platforms have full control over security infrastructure, including the CA
- A compromised CA (resulting from misconfigured, vulnerable software, or insider threats in cloud provider) allows attackers to impersonate services and ex-filtrate unauthorized information

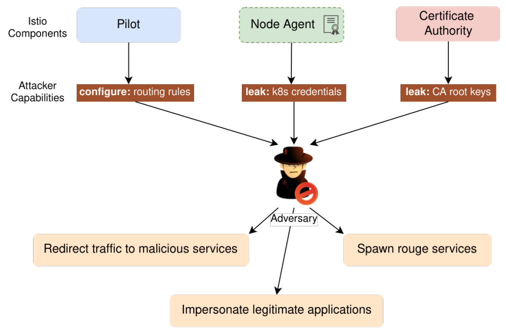
Research Questions

Is it possible to reduce trust in the service mesh's control plane while maintaining microservice compatibility and performance?

Threat Model

Istio Control Plane: Pilot, Mixer, CA	Untrusted: can be exploited by attacker
Node agent + Sidecar	Trusted: can leak credentials due to bugs
Microservice Application	Untrusted: cannot escape container boundary
Kubernetes Control Plane	Trusted

Example Attack Paths



The compromise of a CA gives attackers significant advantages since CAs hold extensive trust in the security system. To reduce this risk, we can limit our reliance on CAs by replacing them with a decentralized public trust system.

Building blocks: RBE

Registration Based Encryption (RBE) helps achieve the notion of decentralized trust using a public key curator.

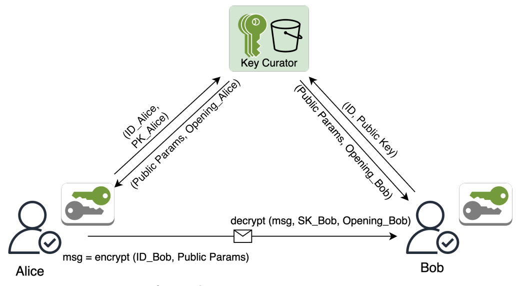

Key Curator
(ID_Alice, PK_Alice)
(Public Params, Opening_Alice)
(ID, Public Key)
(Public Params, Opening_Bob)
decrypt (msg, SK_Bob, Opening_Bob)
msg = encrypt (ID_Bob, Public Params)
Alice
Bob

Figure 2: How RBE works

Mazu Design

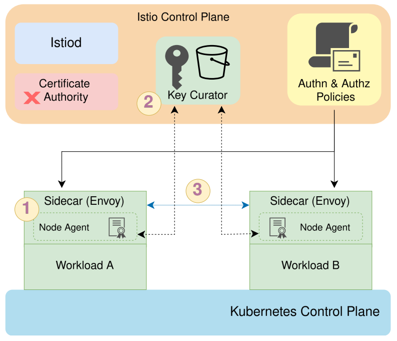


Figure 3: Mazu's Implementation on Istio Service Mesh

1. **Workload Identity and Certificate:** Node agents register <u>*workload IDs*</u> (hash of Kubernetes-signed <u>*admin token*</u>) with the key curator and generate self-signed certificates containing both the *ID* and <u>*admin token*</u>.
2. **Key Curator:** Key Curator accepts node agent registration requests and responds to queries for updated public parameters.
3. **mTLS Certificate Validation:** During mTLS setup, the Envoy sidecar accepts the connection:
   a. if <u>*admin token*</u> is valid confirming authorized node deployment
   b. if the <u>*workload ID*</u> is registered with the Key Curator

Preliminary Evaluations

Our preliminary results show that Mazu introduces minimal latency overhead, adding 0.17ms latency at both 16 and 64 concurrent connections compared to mTLS-enabled Istio.

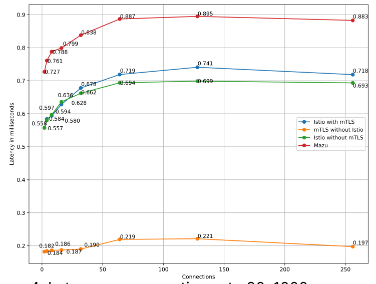


Figure 4: Latency vs connections at p90, 1000 qps over 120 seconds

References

- S. Garg, M. Hajiabadi, M. Mahmoody, and A. Rahimi, "Registration-Based Encryption: Removing Private-Key Generator from IBE," Lecture Notes in Computer Science, pp. 689–718, 2018, doi: https://doi.org/10.1007/978-3-030-03807-6_25.
- N. Glaeser, Dimitris Kolonelos, Giulio Malavolta, and A. Rahimi, "Efficient Registration-Based Encryption," Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security, pp. 1065–1079, Nov. 2023, doi: https://doi.org/10.1145/3576915.3616596.