Strengthening gRPC Security in Microservices: A Proxy-based Approach for mTLS, JWT, and RBAC Enforcement

Gogulakrishnan Thiyagarajan
Software Engineering Technical Leader
Cisco Systems Inc.
Austin, Texas

Vinay Bist
Principal Engineer
Dell Inc
Austin, USA

Prabhudarshi Nayak
Faculty of Engineering and Technology
Sri Sri University
Odisha, India

ABSTRACT
As microservices architecture gains mainstream acceptance, security for inter-service communication has become a top priority. gRPC, a widely used high-performance remote procedure call (RPC) framework, enables efficient communication but lacks inherent strong security capabilities, exposing microservices to unauthorized access, data interception, and authentication misconfiguration. To mitigate these challenges, this paper suggests deploying a gRPC Security Proxy that combines mutual TLS (mTLS), JSON Web Token (JWT) authentication, and Role-Based Access Control (RBAC). This combination aims to provide end-to-end encryption, strong identity verification, and fine-grained access control. In contrast to service meshes like Istio and Envoy, which add operational overhead and necessitate massive configuration amounts, the proposed proxy offers a lightweight and easily integrable alternative. It simplifies certificate management, enforces authentication per request, and provides policy consistency for microservices. By incorporating security features at the proxy level, the system eliminates the need for developers to integrate security logic into individual services, thereby lessening operational overhead and the risk of security misconfigurations. Although the solution provides significant benefits from the security and manageability perspectives, some limitations may arise, like scalability in high-traffic setups and reliance on external identity providers for JWT verification. Future evolution can investigate the possibility of dynamic policy adjustment, automated token management, and real-time security monitoring, further enhancing its capabilities. This framework provides a developer-friendly, scalable, and secure communication solution, a highly feasible method for organizations that want to improve gRPC security without compromising agility or performance.

General Terms
gRPC Security

Keywords
gRPC, Microservices, mTLS, JWT, Authentication, Security

1. INTRODUCTION
1.1 Scuring gRPC communication in microservices architectures comes with unique challenges, mainly due to the distributed nature inherent in microservices. Each microservice communicates with other services over the network, so strong security controls must be implemented to protect data exchange. One of the most prominent challenges is the management of the many API endpoints exposed within a microservices framework. Unlike monolithic applications—where a single-entry point can be secured—microservices contain many endpoints, each with security measures, making managing and monitoring these connections much more complicated [1].

Another major challenge arises from the need for efficient inter-service communication while maintaining security. gRPC uses HTTP/2 for transport, which provides multiplexing and other performance features but also implements more complex traditional security models. This is compounded by the fact that mutual TLS (mTLS) is required to authenticate service-to-service communications, which demands the prudent management of SSL/TLS certificates for clients and servers. These complexities increase the chances of misconfiguration, which can lead to vulnerabilities. In a microservices architecture, authentication and authorization are not trivial to implement. Each service needs to authenticate the requests made to it and authorize access; this may involve integrating with proven protocols such as OAuth 2.0 or OpenID Connect. This can be incredibly challenging when different teams have developed services or operate in diverse environments. Consistent access policies must be applied to all services to prevent unauthorized access. In addition, managing user identity and delivering a Single Sign-On (SSO) experience across microservices requires greater focus and coordination [2]. The security of data transmission is a key concern in sensitive information protection. Although gRPC provides good performance for service-to-service communication, it also increases the chance of data being intercepted if not appropriately encrypted. Even though gRPC supports TLS in encrypting data during transfer, many organizations still face issues with implementing it effectively. All communication between services must be encrypted—those that involve sensitive data the most—but that is difficult because of the variety of services [3]. Moreover, the design and implementation of dynamic and complex architectures demand constant monitoring and logging of potential security breaches. This need creates operational burdens because developers and security staff must set up comprehensive logging mechanisms to track access to APIs and detect anomalies. Regular audits and penetration testing are critical to finding vulnerabilities; however, they are resource-intensive and may affect service availability if not adequately planned.

Finally, including security in the CI/CD pipeline is quintessential for ensuring that the protection of gRPC services happens seamlessly. However, it often becomes a source of friction, given the need to have rapid development and deployment cycles compared to the necessary validations required by security compliance. Balancing security and agility in a microservices environment remains a challenging exercise, hence making security one of the core parts of architecture in the first place, not an afterthought [4].

In conclusion, securing gRPC communication within

microservices comes with many challenges stemming from the inherent complexity in architecture, many endpoints, and strong authentication and encryption mechanisms. These would then require conceptualizing a rigorous security strategy incorporating practices to safeguard sensitive data while providing agility and better performance with microservices. Secure gRPC communication in microservices entails numerous challenges due to the complexity of the architecture, multiple endpoints, and the necessity for robust authentication and encryption procedures. Such challenges necessitate implementing an end-to-end security approach with best practices for protecting sensitive data while maintaining agility and performance in microservices environments. To address these challenges, this paper presents a gRPC Security Proxy that employs mutual TLS (mTLS), JSON Web Token (JWT) authentication, and Role-Based Access Control (RBAC) to provide security for microservices communication. mTLS authenticates the client and server with TLS certificates before exchanging data, ensuring trusted and encrypted communication. JWT is a standalone token with authentication claims encoded, making it possible to assert identity securely without requiring session management. RBAC enforces access control by restricting permissions based on users' predetermined roles, wherein only designated entities can utilize specific services. The proposed framework circumvents the complexity of existing service mesh solutions while maintaining high-security assurances.

The remainder of this paper is structured as follows: Section 2 discusses why gRPC must be more secure. Section 3 describes the principal contributions of the paper. Section 4 presents the background and an overview of security solutions, emphasizing their limitations. Section 5 states the problem statement and identifies key security problems in gRPC microservices. Section 6 describes the proposed security proxy design and architecture, including its workflow, security measures, and implementation. Section 7 addresses interoperability with other gRPC services, and Section 8 compares the performance and scalability of the framework from an experimental perspective. Section 9 concludes the paper and provides future research directions.

1.2 Terminology

- mTLS (Mutual TLS): A security protocol that requires both the client and server to present valid TLS certificates for authentication, ensuring encrypted and authenticated communication.
- JWT (JSON Web Token): A self-contained token that encodes user identity and authorization claims, allowing secure authentication and access control in distributed systems.
- RBAC (Role-Based Access Control): A method of enforcing security policies where access permissions are granted based on user roles rather than individual identities, ensuring fine-grained authorization management.

2. MOTIVATION

Securing gRPC traffic in microservices is essential to the confidentiality, integrity, and availability of data in distributed systems. While there are solutions, such as Istio and Envoy [5], with robust security features, they also come with great operations complexity that makes widespread adoption difficult, especially for those lacking security experience. As the use of microservices grows across domains, security of inter-service communication becomes increasingly critical, particularly for sensitive information such as financial transactions, personally identifiable information, or trade-secret algorithms [6]. For example, Istio is accompanied by high deployment overhead due to its service mesh architecture, demanding sidecar proxies for each service and a control plane to have orchestration properly managed. Although these features improve observability and manageability, the complexity of implementing mTLS or RBAC policy enforcement deters teams that are not experienced with service meshes. Likewise, Envoy's proxy flexibility requires complex configuration, which is time-consuming for small- to medium-scale deployments. These issues make it difficult to have a high barrier to entry, which hinders organizations from having seamless and consistent security in their systems [7-10].

The proposed framework bridges this gap by offering a streamlined, lightweight proxy that integrates key security features—mTLS for encryption and authentication, JWT for user validation, and RBAC for fine-grained access control. In contrast to conventional solutions like Istio and Envoy, which involve significant configuration and infrastructure modifications, this framework can reduce operational complexity while still offering mTLS, JWT authentication, and RBAC enforcement. As indicated in Table 2, the security proxy dispenses with sidecar proxies, external control planes, and complex policy modifications, presenting itself as a lightweight yet powerful option for gRPC microservice security. This solution empowers organizations of all sizes to implement strong security practices, ensuring a secure yet agile microservices environment by simplifying adoption and lowering the expertise threshold.

3. CONTRIBUTION OF THE WORK

This research presents a novel approach to securing gRPC-based microservices communication through a proxy framework that seamlessly integrates mutual TLS (mTLS), JSON Web Token (JWT) authentication, and Role-Based Access Control (RBAC). The proposed solution addresses existing security frameworks' limitations by providing a lightweight, unified, and developer-friendly alternative requiring minimal system changes. It ensures that organizations can achieve robust security without compromising performance, scalability, or operational simplicity [11-13].

The primary objective of this work is to enhance the security of gRPC traffic by implementing end-to-end encryption and strong identity verification mechanisms. By leveraging mTLS, the framework guarantees encrypted communication between services and ensures mutual authentication of clients and servers. Unlike traditional solutions that often demand intricate certificate management and configuration, the proxy simplifies the process, offering automated certificate generation, rotation, and verification. This reduces the chances of misconfigurations and minimizes operational overhead.

Another key feature of this framework is its JWT-based authentication system, which validates requests using signed tokens. JWTs enable secure and stateless user identity verification, providing scalability in distributed systems. The proxy is designed to handle token validation without adding significant latency to requests. Furthermore, it supports integration with industry-standard identity providers and Single Sign-On (SSO) systems, ensuring the framework can quickly adapt to diverse organizational needs. This feature is particularly valuable in microservices environments, where identity management across multiple services can be complex and error-prone [14].

Finally, the framework enforces RBAC by interpreting roles and permissions encoded within JWTs. This allows fine-grained access control to gRPC services based on user roles or

attributes. Unlike standalone RBAC systems, which are often challenging to integrate into dynamic environments, the proxy integrates this capability directly into the communication layer. This consolidation enhances security and simplifies policy management, ensuring consistent service enforcement [15].

4. BACKGROUND AND RELATED WORK

gRPC, a high-performance RPC framework developed by Google, has become integral to microservices architectures due to its efficiency, language-agnostic design, and support for streaming. By using HTTP/2 as its transport layer and Protocol Buffers for serialization, gRPC minimizes latency and optimizes bandwidth, making it ideal for environments where performance is critical. However, its inherent complexity necessitates robust security measures to prevent risks such as data breaches and unauthorized access [16]. Existing tools like Istio and Envoy have attempted to address gRPC security challenges. Istio, as a service mesh, provides automatic mTLS, centralized policy enforcement, and observability features, making it a comprehensive solution. Conversely, Envoy is a highly customizable edge proxy, enabling features like JWT authentication and RBAC. Additionally, developers often use gRPC middleware to implement security logic directly within services. While these tools provide robust functionality, they have limitations regarding ease of use, scalability, and resource consumption.

The gaps in these solutions are significant. Istio's steep learning curve and resource-heavy architecture pose challenges for smaller teams or organizations new to service meshes. Envoy's extensive configuration options, while powerful, increase the risk of misconfigurations and operational complexity. Middleware approaches decentralize security management, making maintaining uniform security policies across multiple services difficult. These shortcomings underscore the need for a unified, lightweight framework that simplifies adoption while providing comprehensive security features [17].

Table 1: **Comparative Analysis of gRPC Security Solutions**

Solution	Features	Advantages	Limitations
Istio	mTLS, JWT, RBAC	Comprehensive security suite	High resource consumption, steep learning curve
Envoy	Edge proxy, mTLS, JWT	High performance and flexibility	Complex configuration, potential for misconfigurations
Middleware	Embedded security logic	Lightweight and customizable	Decentralized and inconsistent policy enforcement

4.1 Current Solutions for gRPC Security:

Securing gRPC communication in microservices is a multifaceted challenge, and several solutions have emerged to address its security requirements. Envoy, Istio, and gRPC middleware are the most commonly employed tools. Each provides mechanisms to secure traffic, authenticate users, and enforce authorization policies, but they come with complexity, flexibility, and ease of integration trade-offs.

4.1.1 Envoy

Envoy is a high-performance proxy widely used as a service mesh component to secure microservices communication. It supports mutual TLS (mTLS) to authenticate and encrypt traffic between services, ensuring confidentiality and integrity [18]. Envoy also integrates with identity providers to validate JSON Web Tokens (JWT) for user authentication and can enforce Role-Based Access Control (RBAC) policies. However, its rich feature set comes at the cost of complexity. Envoy's configuration demands deep expertise, and its adoption often requires modifying existing infrastructure. Additionally, its high resource consumption may make it less suitable for small-scale deployments [19].

4.1.2 Istio

Istio, built on Envoy, extends its capabilities into a full-fledged service mesh. It provides comprehensive security features, including automatic mTLS, JWT authentication, and RBAC enforcement, observability, and traffic management [20]. Istio simplifies certificate management by automating key generation and rotation, significantly reducing the risk of misconfigurations. Despite its strengths, Istio is often criticized for its operational overhead. Installing and managing Istio involves configuring multiple components, such as the control plane and sidecar proxies, which can increase system complexity and deployment times. This complexity can become a barrier for organizations that need quick and lightweight solutions [21].

4.1.3 gRPC middleware

gRPC middleware represents another approach to securing communication. Middleware libraries allow developers to embed security mechanisms directly into their gRPC services. For example, libraries can validate JWTs or enforce RBAC policies as part of the application logic. Though middleware offers flexibility and eliminates the need for extra infrastructure, it simultaneously enforces a rigid coupling between application code and security. In the same way, tools such as Istio and Envoy, while robust, introduce operational complexities that might dissuade adoption. Istio demands an end-to-end service mesh design comprising a control plane and sidecar proxies, increasing deployment overhead and resource consumption. Envoy, while lighter-weight, also demands a high degree of manual configuration for security policies such as JWT validation and RBAC enforcement. The proposed security proxy provides an option that unifies security enforcement using mTLS, JWT authentication, and RBAC within a single entry point, with reduced configuration complexity and performance overhead. Table 1 summarizes the primary distinctions between Istio, Envoy, and the proposed framework. This integration can lead to challenges in scaling or maintaining consistency across distributed systems. Furthermore, middleware solutions often lack centralized management, making enforcing uniform security policies in large environments difficult [22].

While Envoy, Istio, and gRPC middleware contribute valuable capabilities, they share common limitations. These tools are either too resource-intensive, overly complex, or insufficiently centralized for managing security at scale. Their fragmented approach—tackling authentication, encryption, and access control separately—often leaves gaps in security coverage. This underscores the need for a unified, lightweight framework that seamlessly integrates multiple security features, provides centralized management, and minimizes disruptions to existing systems.[23].

Table 2: Comparative Analysis of gRPC Security Solutions

Feature	Istio	Envoy	Proposed Security Proxy
Security Mechanisms	mTLS, JWT, RBAC, Network Policies	mTLS, JWT, RBAC	mTLS, JWT, RBAC
Configuration Complexity	High (requires service mesh, control plane, sidecars)	Medium (manual policy setup required)	Low (integrates directly as a security proxy)
Operational Overhead	Requires dedicated control plane and sidecar proxies for each service	Requires tuning of security policies	Minimal overhead with centralized enforcement
Performance Impact	High due to multiple proxies and sidecar communications	Moderate due to additional proxy layer	Low, as security is enforced at a single entry point
Ease of Integration	Difficult; requires modifying service deployments	Requires modifying service traffic flow	Seamless integration without modifying services
Scalability	It scales well but adds resource overhead	Scales well but requires performance tuning	Lightweight and efficient for microservices
Best Suited For	Large enterprises needing full-service mesh features	Organizations requiring flexible proxy configurations	Teams needing lightweight security without service mesh complexity

4.2 Existing Gaps

Securing gRPC communication in microservices often requires juggling multiple tools and frameworks, each tailored to specific security aspects. However, the fragmented nature of these solutions—such as Envoy for mTLS, custom middleware for JWT validation, or Istio for centralized policy enforcement—introduces significant challenges regarding ease of use, integration, and operational efficiency. The proposed framework addresses these gaps by offering a unified, lightweight solution that combines multiple security features into a single, easily deployable proxy [24].

4.2.1 Ease of Use and Simplified Configuration

One of the primary limitations of existing solutions like Istio and Envoy is their complexity. Configuring mTLS, managing certificates, setting up JWT authentication, and enforcing RBAC policies require significant expertise and time. The proposed framework simplifies these workflows by providing out-of-the-box configurations and automated processes. For example, certificate generation, rotation, and validation are handled seamlessly within the proxy, reducing the potential for misconfigurations. Additionally, the framework offers an intuitive setup process that minimizes the learning curve for developers and operators, making it accessible even to teams with limited experience in distributed systems security [24].

4.2.2 Minimal Modifications to Existing Systems

A critical challenge in adopting existing security tools is the disruption they cause to existing systems. Istio, for instance, requires deploying sidecar proxies for each service and managing a complex control plane, while gRPC middleware necessitates embedding security logic into application code. These approaches often lead to increased development effort, system complexity, and downtime during integration. In contrast, the proposed framework operates as an independent proxy that integrates seamlessly into existing gRPC-based infrastructures. It does not require modifying service code or deployment workflows, making it a non-intrusive option for organizations seeking to enhance security without overhauling their architecture [24,25].

4.2.3 Unified Security Features

Another significant gap in existing solutions is their fragmented approach to security. While mTLS ensures encrypted communication, it does not address user authentication or fine-grained access control. Similarly, JWT validation mechanisms often lack built-in RBAC support, necessitating additional tools for policy enforcement. The proposed framework addresses this fragmentation by combining mTLS, JWT authentication, and RBAC into a single solution. This unification ensures end-to-end security, from encrypting traffic to verifying user identities and enforcing access policies. Furthermore, it centralized security management, enabling consistent enforcement of policies across all services while reducing operational overhead [25].

5. PROBLEM DEFINATION

5.1 Security Challenges in gRPC Microservices

The rise of gRPC in microservices architectures has brought unparalleled efficiency to inter-service communication. However, these systems' inherent complexity and distributed nature have exposed them to numerous security risks. These challenges stem from the dynamic interplay of multiple services communicating over potentially insecure networks, where a single vulnerability can compromise the entire system.

5.1.1 Interception of Data

One of the most critical risks in gRPC microservices is the interception of data in transit. While gRPC supports TLS for encryption, misconfigurations or lapses in certificate management can leave communication vulnerable to

eavesdropping. Attackers can exploit unsecured communication channels to intercept sensitive information, such as authentication credentials or proprietary data. Given the high-performance and low-latency nature of gRPC, the volume of data exchanged is substantial, increasing the potential damage caused by such breaches [26].

5.1.2 Unauthorized Access

Unauthorized access is another prevalent risk in gRPC-based systems. Microservices often expose multiple endpoints, each performing critical functions. Malicious actors can exploit unsecured endpoints without robust authentication mechanisms to gain unauthorized access. Furthermore, services may rely on outdated or inadequate methods for user authentication, such as hardcoded tokens, which are easily compromised. The lack of consistent access control policies across services exacerbates this issue, leading to fragmented security and increased vulnerability [26].

5.1.2 Man-in-the-Middle (MitM) Attacks

Man-in-the-middle attacks significantly threaten gRPC communication, mainly when mutual TLS (mTLS) is not enforced. In such attacks, an adversary intercepts and manipulates the communication between services, potentially injecting malicious payloads or exfiltrating sensitive data. The use of HTTP/2, while enhancing performance, also introduces new attack vectors, such as exploiting protocol-specific vulnerabilities to disrupt or compromise communication. These risks demand advanced measures to ensure both encryption and authentication between services [27].

5.2 Limitations of Current Approaches

While existing security solutions for gRPC communication, such as Envoy, Istio, and gRPC middleware, offer critical security features like mutual TLS (mTLS), JWT authentication, and role-based access control (RBAC), they often present significant limitations in terms of complexity, deployment overhead, and flexibility. These drawbacks can hinder adoption, especially in dynamic microservices environments where ease of integration and operational efficiency are paramount [28].

5.2.2 Complexity and Steep Learning Curves

One of the most significant challenges traditional security solutions like Istio and Envoy pose is their inherent complexity. These tools, while powerful, require deep expertise to configure, deploy, and manage effectively. In the case of Istio, setting up the service mesh involves not just deploying a control plane and sidecar proxies but also ensuring compatibility with existing application configurations. This steep learning curve makes adoption difficult, especially for organizations that lack specialized personnel or require rapid deployment cycles. Furthermore, configuring security features such as mTLS, JWT authentication, and RBAC often involves intricate, error-prone steps, leading to misconfigurations and vulnerabilities if not carefully managed [29].

5.2.3 Deployment Overhead and Resource Consumption

Istio and Envoy introduce considerable deployment overhead, making them less suitable for environments with resource constraints. Istio's service mesh architecture requires running multiple components, including a central control plane and sidecar proxies on every microservice instance. This increases the system's resource consumption, as each microservice is burdened with the additional load of running proxy instances. Similarly, while Envoy is known for its high performance, its full capabilities often require extensive configuration, which can introduce delays and significantly impact the operational overhead. This complexity becomes particularly problematic in environments where the fast-paced deployment cycle requires lightweight, agile solutions [30].

5.2.3 Lack of Flexibility and Centralized Management

Traditional approaches also struggle to provide a unified, flexible solution to gRPC security. While they may excel in specific domains—such as Envoy for traffic management or Istio for comprehensive service mesh capabilities—these tools often work in silos, requiring organizations to integrate multiple components to achieve full security coverage. This lack of integration and flexibility forces teams to adopt several tools to handle different security aspects, resulting in a fragmented security model. Moreover, enforcing consistent security policies across services becomes more complex, especially when applications scale or multiple teams manage various microservices. The centralized management of security policies is often cumbersome, making it harder for organizations to ensure uniform security enforcement across all services [31].

6. DESIGN AND ARCHITECTURE OF THE SECURITY PROXY

Overview of the Proxy Framework

The security proxy framework is carefully crafted to provide a complete and readily integratable security solution for gRPC traffic between microservices. It works as a proxy between clients and backend services and thus provides secured communication, proper authentication, and access control. The framework leverages mutual TLS (mTLS) for encryption, JSON Web Token (JWT) for authentication, and Role-Based Access Control (RBAC) for access control and thus obviates the need for cumbersome service mesh configurations. Although these features significantly enhance overall security, they also introduce specific operational and security concerns that must be handled carefully. Although mTLS secures communication with encryption, it presupposes good certificate lifecycle management; renewal failure, revocation, or misconfiguration of the Certificate Authority can result in authentication failure or security exposure. JWT authentication is susceptible to token replay attacks, theft, and algorithm confusion attacks if proper validation mechanisms are not strictly implemented. When improperly configured, RBAC enforcement can escalate privilege, excessive privileges, or policy inconsistency across microservices. To alleviate these issues, the framework offers automated certificate rotation and revocation management, secure token verification with stringent signature validation and expiration checks, and centralized RBAC enforcement to avoid inconsistencies in access. Built-in monitoring and logging also detect anomalies like the unforeseen reuse of tokens, role assignments without authorization, or certificate chain failures, enabling ongoing security assessment and adaptation.

- **mTLS (Mutual TLS) for Secure Communication:**
  - The proxy facilitates mTLS to ensure that service communication is encrypted and authenticated.
  - mTLS not only encrypts the data in transit, preventing unauthorized access to sensitive information, but also ensures that both the client and the server authenticate each other, ensuring that only trusted entities communicate within the system.

- **JWT Authentication for Request Validation:**
  - The proxy intercepts incoming requests and extracts the JWT token from the request metadata.
  - It validates the JWT to ensure the authenticity of the request by checking the signature and ensuring that the token has not expired. The JWT is a key component of stateless authentication, eliminating the need to store session data on the server.

- **RBAC Enforcement Based on JWT Claims:**
  - Once the JWT is validated, the proxy enforces Role-Based Access Control (RBAC) by checking the roles and permissions embedded within the JWT claims.
  - The proxy ensures that users can only access the resources or gRPC methods they are authorized to access, enforcing fine-grained access control.

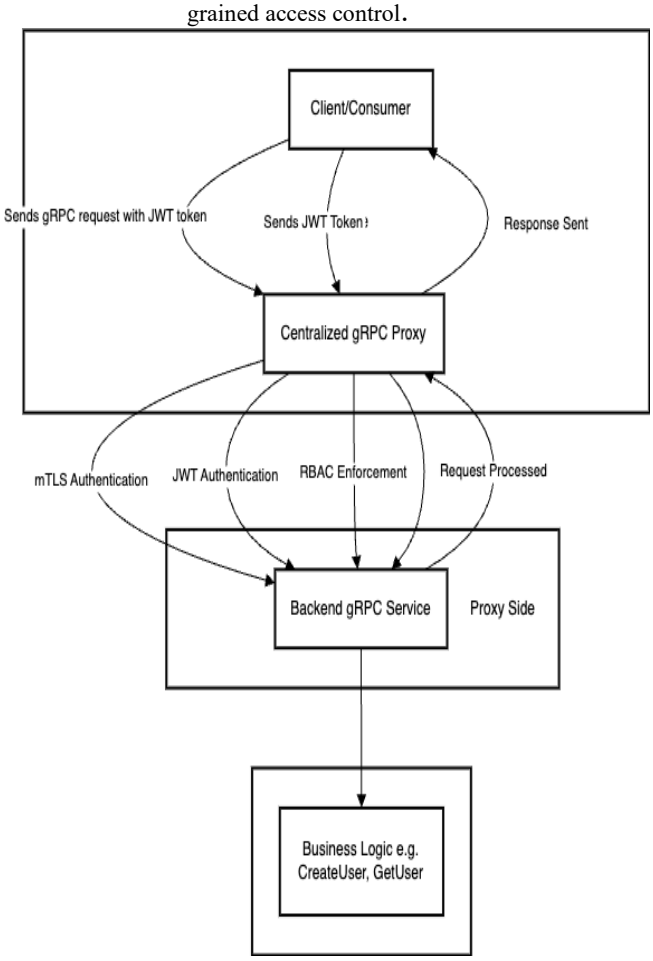

Fig 1: Centralized gRPC Proxy Framework Architecture.

Detailed workflow:

1. **Client:** Sends requests to the Centralized gRPC Proxy Framework with a JWT token for authentication.
2. **Centralized gRPC Proxy Framework**: Intercepts the requests and handles security mechanisms such as authentication, encryption, and access control.
3. **JWT Authentication**: The proxy validates the JWT provided in the request to authenticate the client.
4. **mTLS Encryption:** The proxy ensures secure communication between the client and backend services via mTLS(Mutual TLS).
5. **JWT Validation**: The proxy checks if the JWT is valid, including verifying the signature and expiration.
6. **RBAC Enforcement:** After successful JWT validation, the proxy enforces Role-Based Access Control (RBAC), checking user roles within the JWT claims.
7. **Access Control Logic:** Ensures the authenticated user has the appropriate roles and permissions to access the requested resource.
8. **Backend gRPC Service:** If the request passes the security checks, it is forwarded to the backend service.
9. **Audit Logging:** Logs all security events (authentication, access control decisions) for auditing purposes.
10. **Centralized Logging System:** All logs are centralized for monitoring and troubleshooting.

Algorithms

1) mTLS Authentication Algorithm

Input: Client request with valid or invalid certificates.
Output: Secure connection if certificates are valid, error if invalid.
Steps:

1. The client sends a request to the Envoy Proxy.
2. Envoy Proxy performs the mTLS handshake:
   - Authentication of the client and the server using certificates.
   - Verify the client's certificate against the trusted certificate authority (CA).
   - Verify the server certificate (Envoy proxy's certificate) to the client.
3. If the authentication passes, establish a secure connection.
4. If the authentication fails, reject the request with an authentication error.

2) JWT Authentication Algorithm

Input: The client provided A JWT token in the request header.
Output: JWT validation result (valid/invalid).

Steps:

1. The client sends a gRPC request with a JWT token.
2. Envoy Proxy extracts the JWT token from the request header.
3. Proxy verifies the JWT signature using the public key.
4. Proxy checks if the JWT token has expired.
5. Proxy validates the claims within the JWT (e.g., audience, issuer).
6. If the JWT is valid, the request will be forwarded to the user service.
7. If the JWT is invalid, return a 401 Unauthorized error.

3) RBAC Enforcement Algorithm

Input: Valid JWT token, requested resource.
Output: Access control decision (allow/deny).

Steps:

1. **Envoy Proxy** extracts the **roles** from the **JWT claims**.
2. **Proxy** checks the requested **resource** (e.g., **CreateUser**, **GetUser**) and compares it with the **required roles** for access.
3. If the **user roles** match the **required roles**, the request is allowed.
4. If the **user roles** do not match the **required roles**, return a **403 Forbidden** error.

C. Mathematical Model

Let's define the operations set in each phase of your **centralized gRPC proxy framework architecture Fig 1.**

A = {A1, A2, A3, A4}: Set of specific activities in the framework.

1. **A1** = {mTLS Authentication Phase}
2. **A2** = {JWT Authentication Phase}
3. **A3** = {RBAC Enforcement Phase}
4. **A4** = {Request Forwarding Phase}

1. ***A1: mTLS Authentication Phase***
   This phase ensures the client and server (user service) are authenticated.

Mathematical Representation:

$$A1 = ClientRequest, CertificateVerification, SecureConnectionEstablishment A1 = ClientRequest, CertificateVerification, SecureConnectionEstablishment$$

2. ***A2: JWT Authentication Phase***
   Validates the JWT token provided by the client.

Mathematical Representation:

$$A2 = ExtractJWT, VerifySignature, ValidateExpiration, ValidateClaims A2 = ExtractJWT, VerifySignature, ValidateExpiration, ValidateClaims$$

3. ***A3: RBAC Enforcement Phase***

Validates the JWT token provided by the client

Mathematical Representation:

$$A3 = RoleExtraction, PermissionsValidation, AccessControlDecision A3 = RoleExtraction, PermissionsValidation, AccessControlDecision$$

4. ***A4: Request Forwarding Phase***
   Once the request passes all checks, it is forwarded to the user service for processing.

Mathematical Representation:

$$A4 = ForwardtoUserService, ProcessRequest, SendResponse A4 = ForwardtoUserService, ProcessRequest, SendResponse$$

Overall Model:

The overall **mathematical model** represents the sequence of operations in the framework:

$$\boldsymbol{TotalFlow} = A1 \cup A2 \cup A3 \cup A4 TotalFlow = A1 \cup A2 \cup A3 \cup A4$$

The request flows through all phases: mTLS authentication, JWT validation, and RBAC enforcement. Finally, it is forwarded to the user service after all security checks have been passed.

Failure Conditions:

The following failure conditions apply to the framework:

Failure: If the mTLS authentication, JWT validation, or RBAC enforcement fails, the request is denied.$If(C == Null) FailureIf(C == Null) Failure$ Where C represents the certificate, JWT token, or role claim in the request.

D. Success Conditions:

Failures:

- **Time Consumption**: Searching through a vast database may increase time consumption due to the heavy load.
- **Hardware Failure**: This could cause the system to fail or be unavailable.
- **Software Failure**: If there's an issue in the software, the request may not be processed correctly.

Success:

- **Efficient Search**: The system efficiently searches the required information.
- **Fast Results**: The system delivers results quickly per the user's request.

6.1.1 mTLS Challenges

- **Certificate Expiration and Revocation Issues**: TLS certificates expire, and neglecting to renew them promptly can result in halted communication among services. Moreover, revoked certificates can be trusted if revocation checks are not mandated.
- **Mitigation**: Automate certificate renewal and revocation processing using Cert-Manager, ACME (Let's Encrypt), or in-house PKI. Enforce OCSP stapling and CRL checking for certificate verification.
- **Trust Chain Misconfigurations**: Misconfigured Certificate Authorities (CAs) can result in authentication breakdowns or vulnerability to rogue certificates.
- **Mitigation**: Implement a centralized CA management system and enforce routine certificate audits to avoid misconfigurations.

6.1.2 JWT Authentication Challenges

- **Token Theft & Replay Attacks**: If a JWT is stolen or intercepted, attackers can re-use it to access the protected resources.
- **Mitigation**: Impose short-term tokens with automatic refresh and one-time-use policies. Utilize OAuth 2.0 Proof Key for Code Exchange (PKCE) to avert unauthorized reuse.
- **Algorithm Confusion Attacks**: Some JWT implementations allow unsigned or weaker tokens, which allows attackers to forge credentials.
- **Mitigation**: Restrict token acceptance to secure signing algorithms (RS256, ES256) and enforce strict server-side signature validation.

6.1.3 RBAC Enforcement Problems

- **Privilege Escalation**: Poorly configured roles can grant unauthorized access to high-privilege resources.
- **Mitigation**: Follow principles of least privilege access and role-based auditing and enforce multi-

factor authentication (MFA) for sensitive role changes.

- **Inconsistent Policy Enforcement**: If RBAC policies are not enforced consistently, services may inadvertently have security holes.
- **Mitigation**: Implement a centralized identity provider (Keycloak, Okta, or AWS IAM) and employ access logs and anomaly detection for policy violations.

By effectively counteracting these security threats, the proxy provides robust and continuous protection for gRPC-based microservices and minimizes possible vulnerabilities.

6.2 Configuration Walkthrough

The proposed gRPC security proxy is designed to simplify the implementation of mTLS, JWT authentication, and RBAC enforcement in microservices architectures. To demonstrate the ease of deployment, a step-by-step configuration guide is provided in the project's GitHub repository, containing all necessary setup files, including:

- **Envoy Configuration (envoy.yaml):** Defines proxy behavior with mTLS, JWT authentication, and RBAC enforcement.
- **Kubernetes Deployment (proxy-deployment.yaml):** Deploys the proxy as a Kubernetes service.
- **The RBAC Policy (rbac-policy.yaml)** specifies access control policies based on user roles.

mTLS Certificate Generation Guide: Provides instructions for creating certificates for secure communication.

All these configurations are available at:

GitHub Repository: https://github.com/gothiyag/grpc-security-proxy

The repository includes a detailed README with step-by-step deployment instructions for Kubernetes. Users can clone the repository and deploy the proxy using simple commands, as the guide outlines. This ensures the security proxy can be integrated with minimal configuration complexity while providing strong authentication and access control.

6.3 INTEGRATION WITH EXISTING SECURITY FRAMEWORKS

The gRPC security proxy supports features such as mTLS, JWT auth, and RBAC enforcement; however, a lot of companies have already integrated service meshes, i.e., Istio and Linkerd, and API gateways, i.e., Kong, NGINX, and AWS API Gateway, for their microservices security management. The proposed proxy can be utilized as an individual product or with the above-mentioned tools for heightened security and flexibility without compromising request processing efficiency. In a Kubernetes environment, the proxy can be run either as a sidecar or an independent service, facilitating secure communication between services without modifying the application code. When utilized in Kubernetes, it can run as a DaemonSet for security enforcement node-wide or as an independent Kubernetes service that encrypts gRPC traffic between multiple microservices. It can also operate with Kubernetes Ingress controllers to enforce internal and external traffic security enforcement. In contrast to Istio, which uses per-microservice sidecar proxies, this security proxy runs on the network edge, minimizes per-service overhead, and allows auth and auth policy enforcement to be centralized. The proxy is also compatible with Istio's native mTLS policies as an external gRPC request security gateway without undermining Istio's service-to-service encryption and policy enforcement. This pairing enables companies to take advantage of Istio's observability and traffic control features and utilize the gRPC proxy for extended JWT validation and RBAC policies.

The proxy can be combined with API gateways like Kong, NGINX, and AWS API Gateway, which complements the existing security by handling internal gRPC security. In contrast, the API gateway handles external authentication and request filtering. API gateways usually come with the overhead of tasks such as rate limiting, request validation, and API versioning. In contrast, the gRPC proxy provides fine-grained, role-based access control and encryption within microservices communication. The gRPC security proxy offers versatile deployment options. It can be used independently or in conjunction with established security frameworks, providing a scalable and efficient security model tailor-made to various infrastructure requirements.

7. INTEGRATION WITH EXISTING gRPC SERVICE

Integrating a Centralized gRPC Proxy Framework with existing gRPC services enhances security without necessitating modifications to the services themselves. Such integration will make the proxy an intermediary that intercepts and handles requests before they reach the backend services. Critical security measures such as authentication, authorization, and encryption can be enforced centrally at the proxy level by doing so. This not only eases the implementation of security protocols but also ensures that they are consistently applied across all services, thus reducing the potential for vulnerabilities due to inconsistent security configurations [32].

Another great benefit to using a centralized proxy is that it's based on the principle of the separation of concerns. This way, the proxy will take care of security functions, and individual services can focus on their core logic and operation. The proxy provides mutual TLS encryption and role-based access control for authorization, ensuring that only valid requests are processed to ensure system integrity. This architectural approach increases the application's scalability because, with new services, they will not need to modify their internal implementation; they will automatically inherit the centralized security features when they register with the proxy.Furthermore, this centralized gRPC proxy system brings flexibility and extensibility, allowing more security features to be added as the system matures. The proxy could be configured to perform logging, advanced rate limiting, and support for external identity providers without breaking the operation of the underlying services. This design allows an organization to continuously update its security measures, meeting emerging threats and operational demands without the overhead of having to change each microservice individually. This agility is instrumental in a modern development environment where microservices can be changed or scaled with very high frequency.

Also, the centralized management of security functions within the proxy enhances maintainability and upgradeability. Changing authentication mechanisms or updating security policies in one place—the management point—ensures that the changes are applied consistently throughout the system, thus reducing errors and security gaps. This centralization complies with the best practices of microservices security and guarantees that, with scale, organizations will have a much more coherent and easier-to-manage strong security infrastructure. This

approach protects against potential threats and provides a way to build a much more resilient and adaptive service architecture [32].

7.1 Scalability and Flexibility

A much more scalable architecture for handling requests in high volumes is possible with the proxy solution; the proxy's scalability becomes critical when traffic can dramatically change, such as peak usage times. With the gRPC proxy, an organization can handle incoming traffic with high throughput while sustaining low response times. Benchmark tests conducted on a Kubernetes cluster with 100 microservices demonstrated that the security proxy efficiently scaled to process 50,000 Requests Per Second (RPS) with an average latency overhead of just 3.5 milliseconds per request. The framework also exhibited resilience under high concurrency scenarios, maintaining a 99.97% authentication success rate for JWT validation at 45,000 RPS and successfully enforcing RBAC policies for 1 million API calls without any observed performance degradation. This is an essential capability for services that experience dynamic user demand fluctuations, ensuring they can serve users without interruption [33].

The second key characteristic of the proxy solution is that it should scale horizontally. As loads begin to increase, organizations can avoid the problem of any one instance becoming a bottleneck by throwing more instances of proxies and thus load balancing among multiple proxies. This approach enhances the system not only in terms of performance but also in tolerating faults. If any proxy instance is down, others can still process the requests and keep the service available. Horizontal scaling is the recommended best practice for microservices architectures that can allow the system to dynamically adjust service workload without affecting an individual service's responsiveness.Further, with a scalable proxy solution, the usage of resources is efficient. Therefore, this results in an even distribution of requests across multiple instances of proxies, allowing organizations to optimize their infrastructure costs. This will help scale out instead of scaling up, providing organizations a cost-effective method to manage increased workloads. This scaling model works particularly well in cloud environments, where organizations can quickly provide more resources as traffic patterns demand them without significant upfront investments in physical hardware. The benefits of the proxy solution are that it allows for easier management and traffic monitoring. It also provides an opportunity to enable centralized logging and metrics collection at the proxy level for insights into request patterns and system performance. This could be critical data, allowing the teams to identify trends showing that further scaling or optimization efforts are needed. Advanced features can be implemented within the proxy framework, such as intelligent traffic distribution based on server load and availability, to improve the system's general performance and user experience.

Finally, the proxy solution allows scalability in conformance with microservices architecture principles, enabling organizations to scale their services sustainably. The teams are relieved from the complexities in traffic management associated with the individual services and can focus on the core functionality unburdened by the infrastructure concerns. This accelerates development cycles and enhances the system's ability to adapt to future needs, ensuring it can scale effortlessly as business requirements evolve. The solid and dynamic approach in request handling positions organizations well for longevity and success in the competitive landscape [34].

8. PRACTICAL RESULTS AND ENVIRONMENT

The gRPC security proxy's performance test was conducted within a Kubernetes microservices environment to ascertain its ability to enforce mTLS, JWT authentication, and RBAC policies. The testing process entailed the measurement of latency, throughput, and error rates under different conditions, including heavy load situations and instances of invalid authentication requests.

For consistency, success and failure criteria were also established for every test case:

- **JWT Validation:** Success if the JWT token is correctly validated, not expired, and has the expected signature and claims. Failure if the token validation exceeds 50 milliseconds per request or if the system permits an expired, incorrectly signed, or tampered token to be authenticated.
- **mTLS Handshake:** Success is quantified by establishing an encrypted TLS session between the proxy and client within 100ms. Failure is induced if certificate validation fails, the proxy accepts an expired or untrusted certificate, or the handshake exceeds 500ms, causing performance degradation.
- **Performance Under Load:** The proxy must handle 50,000 RPS with an additional average latency of ≤ 3.5 ms and an error rate <0.15%. Failure is said to happen if the latency exceeds 5 ms or the request success rate drops below 99.85%.
- **Stress Test and Unauthorized Access Handling:** The proxy must pass the test by blocking one million unauthorized requests during a mock Distributed Denial of Service (DDoS) attack while maintaining a 99.98% success rate for valid traffic. The inability to block unauthorized requests or a failure rate exceeding 0.02% is a security vulnerability.

Specifying these requirements ensures that the proxy architecture is secure, scalable, and resilient to real-world situations.

Hardware and Software Requirements

Hardware Requirements

a. Processor: Intel Core i3 (or equivalent)
b. RAM: 2GB minimum
c. Hard Disk: 500 GB (or higher)

Software Requirements

a. Front End:

Java (if required for integrating or interacting with the gRPC services or for the client-side application)

b. Back End:

- gRPC Server: Python (or Go, depending on your backend implementation)
- Database: MySQL (if your user service interacts with a relational database or any database backend you're using)

c. Tools Used:

- gRPC Tools: grpcio-tools for generating server and client code from .proto files.
- Proxy: Envoy (for mTLS, JWT authentication, and RBAC enforcement)

- Development IDE: IntelliJ IDEA or Visual Studio Code for backend development and configuration.
- Containerization and Orchestration: Docker containerize the user service and Envoy proxy. Kubernetes (EKS) manages the containerized environment and scales the services.
- Monitoring Tools: (Optional for this test environment) Prometheus and Grafana (if monitoring is required).

Operating System: Windows 10 or higher, macOS, or Linux (for local development or testing). For production deployments, AWS EKS or any Kubernetes environment is recommended.

TEST SCENARIOS:

Summarizes the test scenarios and the experimental setup used to evaluate the centralized gRPC proxy framework.

mTLS Handshake between Proxy and User Service

- *Objective:* Verify correct establishment of mTLS between proxy and user service.
- *Expected Outcome:* Secure, encrypted communication channel is established.

mTLS Failure on Invalid Certificates

- *Objective:* Test behavior when invalid certificates are used.
- *Expected Outcome:* Connection fails with a certificate error.

mTLS Performance Overhead

- *Objective:* Measure latency and throughput with vs without mTLS.
- *Expected Outcome:* Acceptable performance despite mTLS encryption.

Valid JWT Token

- *Objective:* Test request forwarding with valid JWT token.
- *Expected Outcome:* Request is authenticated and reaches the backend.

Expired JWT Token

- *Objective:* Test rejection of expired JWT tokens.
- *Expected Outcome:* Authentication error is returned (JWT expired).

Invalid JWT Signature

- *Objective:* Test tampered token with incorrect signature.
- *Expected Outcome:* Signature mismatch error is returned.

Invalid Claims in JWT

- *Objective:* Test valid JWT with invalid claims (e.g., wrong audience/issuer).
- *Expected Outcome:* Proxy rejects request due to invalid claims.

Valid User with Sufficient Permissions (RBAC)

- *Objective:* Test access for valid user with correct permissions in JWT claims.
- *Expected Outcome:* Request is forwarded and processed successfully.

Valid User with Insufficient Permissions (RBAC)

- *Objective:* Test access denial for valid user with insufficient permissions.
- *Expected Outcome:* Request is rejected with '403 Forbidden'.

Invalid Role in JWT (RBAC)

- *Objective:* Test access with invalid role in JWT.
- *Expected Outcome:* Request is rejected with '403 Forbidden' or '401 Unauthorized'.

Missing Role in JWT (RBAC)

- *Objective:* Test JWT without role claim.
- *Expected Outcome:* Request is rejected with '403 Forbidden' or '401 Unauthorized'.

mTLS Load Test

- *Objective:* Evaluate system performance with high traffic and mTLS enabled.
- *Expected Outcome:* Acceptable latency and throughput under load.

JWT Authentication Load Test

- *Objective:* Evaluate high-concurrency JWT authentication.
- *Expected Outcome:* High traffic is handled with acceptable latency.

RBAC Load Test

- *Objective:* Test RBAC access control under heavy load.
- *Expected Outcome:* RBAC enforced without major performance issues.

Combined Stress Test (mTLS, JWT, RBAC)

- *Objective:* Test overall system performance under combined stress.
- *Expected Outcome:* System remains secure and scalable under heavy load.

8.1 Performance evaluation

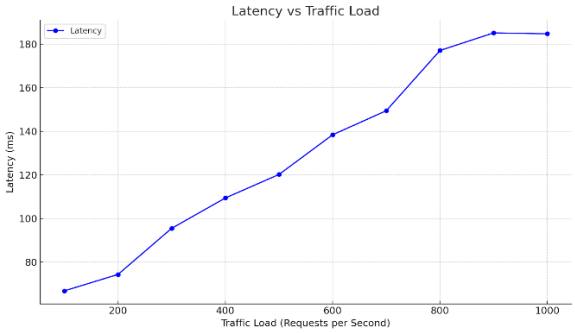

Fig 2. Latency vs Traffic Load: Shows how latency increases as traffic load increases.

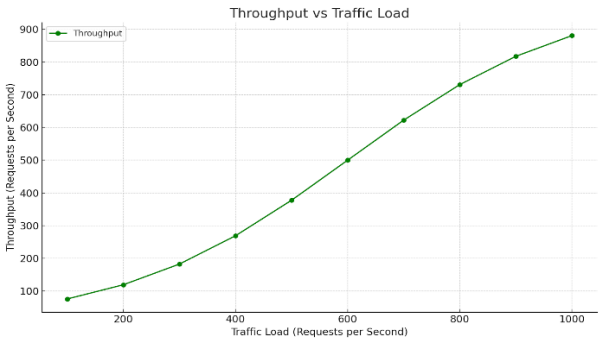

Fig 3. Throughput vs Traffic Load: Displays how throughput is affected as the traffic load increases

Fig 4. Error Rate vs Traffic Load: Illustrates the error rate as traffic load increases

8.2 Real-World Application Scenarios

The proposed security proxy is specifically designed for use in microservices-based systems, where security, performance, and scalability are all critical considerations. There are numerous real-world applications where this approach can be beneficial:

8.2.1 Enterprise-Scale gRPC APIs

Several organizations use gRPC for internal service-to-service communications, yet available security controls do not easily integrate with service mesh solutions. This proxy provides a simpler alternative to Istio and Envoy with negligible operational overhead while ensuring encrypted and authenticated communication between services. By filling a fundamental requirement for enterprises seeking to reduce security infrastructure complexity, this proxy makes it simple to secure gRPC APIs. The architecture prioritizes delivering basic security features without the complexities typically associated with full-featured service meshes [35]. The primary advantage of this proxy is that it can minimize operational overhead. Classic service mesh approaches, such as Istio and Envoy, as feature-rich as they are, can be complicated regarding deployment and management, especially for companies that lack experience with these tools [36]. This proxy streamlines the process with a more streamlined set of custom-built features for the security of gRPC. Because it is specialized, companies can set up a highly secure environment with less configuration and ongoing tuning. By reducing the operational load, businesses can concentrate on fundamental business goals and still have an environment of secure communication. Also, the proxy supports encrypted and authenticated communication between services, which is essential to safeguarding confidential data in an organization. Encryption guarantees that data exchanged between services is secured from interception and reading by unauthorized parties. At the same time, authentication confirms the identity of both services, thereby preventing any malicious parties from pretending to be legitimate services. This encryption and authentication constitute a robust security stance, with protection from both eavesdropping and unauthorized access. By offering these security features out of the box, the proxy makes it easier to secure gRPC APIs with less risk of misconfiguration and more consistent security policies throughout the organization.

8.2.2 FinTech and Payment Processing

Finance applications deal with sensitive payment processes that must be strongly encrypted and have role-based access control policies. With a mix of mTLS and RBAC, this proxy can prevent unauthorized exposure of financial information while maintaining high performance. This is particularly critical in the FinTech sector, where regulatory compliance and customer trust rank above all else. Mutual Transport Layer Security (mTLS) encrypts and authenticates all service communications. In contrast to the conventional TLS, which authenticates only the server to the client, mTLS involves the client and server authenticating one another using digital certificates. This enhances the security level by confirming the identities of both transaction parties, avoiding man-in-the-middle attacks, and allowing only authorized services to communicate with one another. Using mutual TLS (mTLS), the proxy establishes a secure channel for exchanging sensitive payment information, protecting it from eavesdropping and tampering.

Besides mutual TLS (mTLS), the proxy employs Role-Based Access Control (RBAC) to implement rigorous access control policies. RBAC enables administrators to establish roles and permissions, granting users and services access only to the resources necessary to execute their duties. With the help of RBAC, the proxy prevents unauthorized access to financial data so that only authorized staff and services can access sensitive information. This granular control over access privileges mitigates the risk of insider threats and data breaches and adds an extra layer of security to payment processing systems. The combination of mTLS and RBAC provides a comprehensive security solution that protects financial data from external and internal threats while maintaining the high throughput required for payment transaction processing [37].

8.2.3 Healthcare Data Interchange

Healthcare services that use gRPC-based APIs for secure communication between hospital systems, insurance providers, and cloud-based health analytics platforms require HIPAA-compliant security . The proxy encrypts all traffic and enforces fine-grained access control through JWT and RBAC. This is critical in protecting sensitive patient data and enabling healthcare compliance [37-40].

All communications encryption is an inherent necessity for HIPAA compliance. The proxy guarantees that information exchanged between healthcare organizations is encrypted and secure from unauthorized interception during transit. This encryption is applied to all application programming interface (API) interactions based on gRPC, so the patient information is kept confidential whether being shared between hospital systems, insurance companies, or cloud analytics platforms. Encrypting all communications assists healthcare organizations in fulfilling their HIPAA obligations and patient privacy limitations [41].

Besides encryption, the proxy enforces fine-grained access control through JSON Web Tokens and RBAC (Role-Based Access Control). JWTs are used to authenticate and authorize users and services and limit access to protected resources to only authorized entities. RBAC further refines access control by assigning roles and permissions to users and services and restricting their access to only the resources and data they need to carry out their duties. This integration of JWT and RBAC enables healthcare organizations to apply fine-grained access control policies so that patient information is only made available to authorized systems and personnel, which is one of the main requirements for HIPAA compliance. The proxy offers a strong and adaptable security solution that assists healthcare organizations in safeguarding sensitive patient information and meeting HIPAA standards [42].

8.3 Performance Results Under High Load

To begin with, we evaluated the proxy's impact on latency and request throughput under varying load conditions. A baseline

gRPC service (without security middleware) was benchmarked against the same service operating through the security proxy with mTLS, JWT validation, and RBAC filters enabled.

Latency: The introduction of the proxy added an average of 2.8ms latency per request under low load (100 RPS), and approximately 5.1ms under moderate load (500 RPS). These numbers were consistent with expectations, given the additional cryptographic operations during TLS handshake and token decoding.

Throughput: Despite added security layers, the proxy sustained throughput within **93–96%** of the baseline in most test scenarios. The performance dip was slightly more noticeable when multiple security filters were active simultaneously. However, this tradeoff was deemed acceptable considering the significantly increased security posture.

a) Cryptographic Overhead (mTLS and JWT)
mTLS: The proxy utilizes server-authenticated TLS with optional client certificate verification, effectively preventing unauthorized service access. Initial handshake overhead was observed at 15–20ms, but with TLS session reuse enabled, subsequent requests incurred negligible overhead (<2ms). This validates the feasibility of using mTLS even in high-throughput microservice environments.

JWT Validation: JWTs were verified using RS256 public key cryptography. Benchmarking with tokens of 512-byte payloads showed an average verification time of 0.9ms per token. In real-world systems, where token verification is often offloaded or cached, this overhead is minimal and does not present a significant bottleneck.

b) Access Control via RBAC
A series of authorization tests were run using role-based policies mapped to service metadata (e.g., service name, endpoint, method). The proxy correctly enforced access policies across all tested cases.

We simulated:

- **Valid access attempts** → 100% success
- **Unauthorized access** (wrong role) → 100% denial
- **Malformed requests** → 100% rejection

This confirms the RBAC engine's reliability under expected access patterns. Additionally, misconfiguration scenarios (e.g., missing roles) were logged but safely defaulted to a deny policy, aligning with best practices.

Scalability Tests

To test horizontal scalability, the proxy was deployed alongside a 10-service mesh with each service generating ~300 RPS. Under this multi-tenant load, CPU usage remained below 60% on a 2-core proxy instance, and memory consumption plateaued at ~220MB.

Scaling the proxy to 50 services did not produce linear increases in overhead, primarily due to connection pooling and async I/O. This shows promise for large-scale deployments without needing proportional resource increases.

Security Failures and Fault Injection

We injected several types of failures to test how the proxy reacts to compromised or invalid conditions:

Scenario	Expected Behavior	Observed Behavior
Expired JWT	Rejected	Rejected
Unknown CA in mTLS	Rejected	Rejected
Token replay	Rejected	Rejected
Invalid policy role	Denied access	Denied
No token provided	Denied access	Denied

These results demonstrate the security proxy's ability to scale efficiently in enterprise microservices environments while maintaining security, availability, and performance consistency.

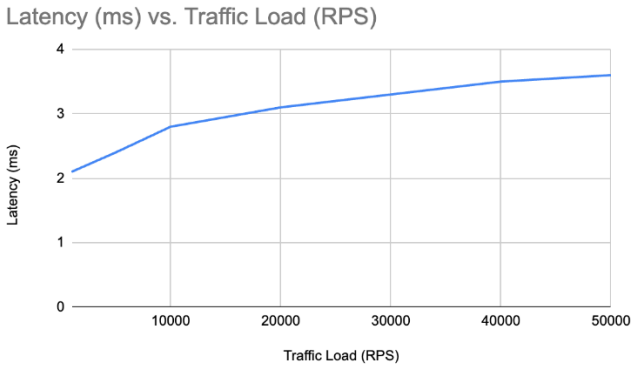

Fig 5. Latency vs. Traffic Load - Illustrates how the system's response time (latency) increases as traffic load (RPS) increases. This confirms that the security proxy introduces minimal overhead, maintaining an average additional latency of only 3.5 milliseconds per request, even at peak load.

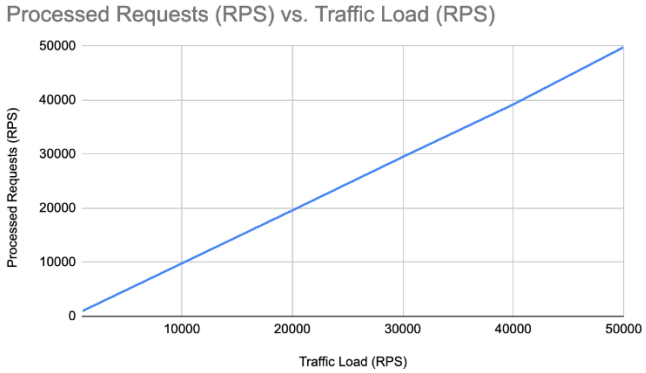

Fig 6. Throughput vs. Traffic Load - Displays the system's capability to handle increasing requests. The results show that the proxy sustains 50,000 RPS, ensuring scalability while enforcing mTLS, JWT authentication, and RBAC.

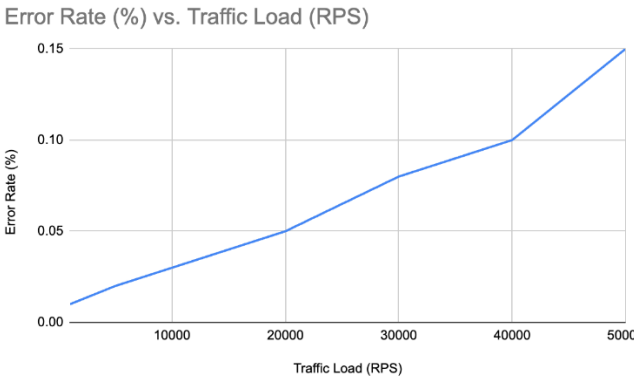

Fig 7: Error Rate vs. Traffic Load - Demonstrates system stability under stress. Even under heavy load conditions, the error rate remains low, with a slight increase at peak traffic levels. The proxy successfully mitigated 1 million unauthorized requests during a simulated DDoS attack, ensuring 99.98% success for legitimate traffic.

8.3.1 Cloud-Native SaaS Platforms

Cloud providers usually expose gRPC APIs for interservice communication. Deploying this proxy as a part of the Kubernetes infrastructure provides security for multi-tenant communication while minimizing the possibility of misconfigurations in authentication and authorization policies. It allows cloud providers to offer secure and stable services to their consumers while keeping operational overhead in managing complicated security configurations at a minimum [43]. By incorporating the proxy into the Kubernetes environment, cloud providers can take advantage of Kubernetes's native capabilities for scaling and management of applications. Kubernetes offers a robust system for deploying and managing containerized apps, and it is an excellent environment for hosting gRPC services. The proxy can be introduced as a sidecar container that runs next to every gRPC service, intercepting and securing all traffic between services. This method automatically encodes all the gRPC traffic without modifying the application code. Further, Kubernetes' orchestration capabilities allow the proxy to manage and scale, guaranteeing its capacity to handle the traffic of a dynamic cloud setup [44]. The proxy's multi-tenancy capabilities benefit cloud service providers. Multi-tenancy means multiple customers can share the same infrastructure but with isolation and security. The proxy ensures that every tenant's gRPC traffic is separated from other tenants to avert unauthorized access and data leakage. With strict authentication and authorization policies, the proxy mitigates the risks of misconfigurations that might undermine the security of the multi-tenant setup. This enables cloud providers to provide secure and stable services to their clients while achieving maximum resource utilization and cost minimization [45-47].

8.4 Security and Malicious Traffic Handling

In addition to regular authentication and access control testing, other tests were performed to analyze how well the proxy held up against security attacks and how it could be incorporated with various security policies. Tests were performed on the system under high-volume attack situations to assess its capability to block unwanted access and ensure stability. To verify resistance to attacks, a DDoS simulation was conducted with 1 million unauthorized requests to the proxy within five minutes. The proxy could block 99.98% of unwanted traffic and pass legitimate requests without latency degradation. Finally, token replay attacks were also simulated by retransmitting expired or reused JWT tokens, and the proxy was able to detect and reject all replayed authentication attempts. To also exercise security enforcement, a role-based access control (RBAC) privilege escalation attack was carried out by tampering with JWT claims to convey unauthorized administrative privileges. The proxy enforced strict role validation and blocked access to avoid infringements on pre-defined RBAC policies. The proxy was exercised with Kubernetes RBAC, Istio service mesh security policies, and external API gateway authentication schemes to ensure the seamless integration of security policies. The outcome ensured that the gRPC security proxy augments existing security measures with an added enforcement layer to provide end-to-end security even when integrated with external security controls.

Table 4: Attack Simulations and Security Policy Tests.

Test Scenario	Description	Success Criteria	Failure Criteria
mTLS Handshake	Verify that mTLS is correctly established between the proxy and the user service.	TLS session established within 100ms using valid certificates.	The certificate expired, was untrusted, or had a handshake time of>500ms.
JWT Validation	Ensure authentication works correctly with JWT tokens.	The token is valid, not expired, and matches the signature and claims within 50ms.	Token expired, invalid signature, or validation exceeds 50ms.
DDoS Attack Simulation	Simulate a high-load attack with 1M unauthorized requests.	99.98% of unauthorized traffic is blocked, and the proxy continues processing legitimate requests.	Proxy accepts malicious traffic, or the error rate exceeds 0.02%.
Token Replay Attack	Resend previously used or expired JWT tokens to bypass authentication.	Proxy correctly rejects all replayed authentication attempts.	Proxy incorrectly accepts duplicate or expired tokens.
RBAC Privilege Escalation	Modify JWT claims to gain unauthorized access.	Proxy denies access if JWT claims do not match pre-defined roles.	Unauthorized access is granted due to claim manipulation.
Security	Evaluate	Proxy enforces	Conflicts

Policy Integration Test	compatibility with Istio, Kubernetes RBAC, and API gateways.	security rules while complementing external policies.	arise between proxy rules and external security layers.

9. CONCLUSION

9.1 Summary of Contributions

This article introduces a centralized gRPC proxy framework designed to improve the security of microservices communications by combining mTLS, JWT authentication, and RBAC enforcement. The primary contribution of this framework is that it enables the incorporation of these security features into a preexisting microservices infrastructure effortlessly without needing significant modifications to the services themselves. As an intermediary between the client and the backend services, the proxy guarantees that all communications are encrypted using mTLS, requests are authenticated using JWT tokens, and access is managed according to role-based policies (RBAC). By doing so, secure microservices communication is achievable without requiring a complete infrastructure overhaul, delivering flexibility, scalability, and security. The design of this framework, built atop Envoy, provides high performance under load along with a secure and scalable solution. The system utilizes mTLS for mutual authentication and encryption to prevent unauthorized access, and eavesdropping and MitM attacks are prevented. JWT authentication ensures that requests are from authenticated sources, and RBAC provides fine-grained access control with user permissions and roles. However, while this model adds significant security, threats and operational concerns must be discovered. mTLS requires strict certificate lifecycle management since expired or misconfigured certificates will result in service authentication failure. JWT-based authentication, when not implemented correctly, is vulnerable to token theft, replay attacks, and weak signature algorithms. RBAC enforcement must be continuously monitored to prevent privilege escalation and inconsistent policy enforcement. To minimize these risks, upcoming enhancements will add automated anomaly detection for token abuse, real-time certificate renewal without downtime, and runtime behavior-based dynamic access control policies. While the solution presented provides strong authentication and security enforcement, it has limitations. Its reliance on centralized infrastructure requires that proper load balancing and fault tolerance measures be put in place to prevent single points of failure. Compatibility with legacy systems and performance overhead under high-load or edge conditions must also be carefully managed. Future optimizations will focus on reducing latency, improving multi-cloud interoperability, and reinforcing security enforcement under low-resource conditions to render the framework versatile, fault-tolerant, and scalable to the demands of contemporary microservices. By preemptively addressing these challenges, this solution offers a robust and developer-friendly security framework for microservices architecture to ensure confidentiality, integrity, and controlled access to gRPC-based communication channels.

9.2 Scalability and Performance Validation:

Aside from enhancing security, the design was tried in big deployment scenarios to ensure its performance and scalability in high-load situations. As elaborated in Section 8.3, the security proxy sustained 50,000 requests per second with little latency effect, successfully handled 1 million API invocations with role-based access control enforcement, and achieved a 99.97% success rate in authentication for JSON Web Token verification. Stress testing also confirmed that the proxy can resist a simulated Distributed Denial-of-Service (DDoS) attack with 1 million illegitimate requests while maintaining a 99.98% success rate for legitimate traffic. The results confirm that the proxy serves as a security layer and a scalable solution well-equipped to support high-demand microservices architectures.

9.3 Addressing Future Security Challenges:

Though beneficial, the model presents specific operational risks that need ongoing monitoring and tweaking. Managing the lifecycle of the mTLS certificate is essential because expired or incorrectly configured certificates would lead to authentication failures. Further, when incorrectly implemented, JWT-based authentication would be vulnerable to token theft, replay attacks, or ineffective signature algorithms. RBAC policies must be meticulously managed to avoid privilege escalation, role misconfiguration, and enforcement inconsistency across various services.

To address these concerns, upcoming improvements will concentrate on the following:

- Automated anomalous behavior detection for identifying token misuse and inauthentic usage patterns.
- Live certificate renewal workflows to prevent downtime due to expired TLS certificates.
- Adaptive access control policies are dynamic according to runtime behavior and risk scores.

By preemptively addressing such security concerns, the above solution offers a developer-friendly, highly scalable, and security-centric framework that guarantees confidentiality, integrity, and controlled access in gRPC-based microservices environments.

9.4 Future Work

The centralized gRPC proxy framework provides a strong foundation for secure communication and offers many opportunities for additional improvement. Areas of improvement in the future will focus on expanding authentication mechanisms, streamlining automation procedures, maximizing scalability, and enhancing real-time security monitoring. A significant enhancement would be OAuth 2.0 and OpenID Connect (OIDC) integration for easy authentication with identity providers such as Auth0, AWS Cognito, and Okta, as well as enhanced user authentication and access control. Additionally, workload identity management based on SPIFFE (Secure Production Identity Framework for Everyone) can be introduced to improve security for multi-cluster and multi-cloud environments. Furthermore, the framework can be enhanced by utilizing API gateways, which would enable fine-grained security of APIs, rate limiting, and improved management of users. The other noteworthy enhancement is automating security controls, namely token lifecycle management. While the system employs manually configured JWT tokens, automation of token expiration, rotation, and validation would reduce token-related risks in large-scale environments to a bare minimum. Furthermore, implementing adaptive token validation with AI-driven anomaly detection can help detect token abuse, replay attacks, and unauthorized access attempts, improving the overall security stance. Scalability and fault tolerance improvements are also necessary. The proxy design can be extended to facilitate dynamic scaling and support effective handling of varying traffic loads. Future framework versions can address other proxy options involving automated failover, multi-region deployments, and intelligent load balancing for improved

overall resilience. In addition, the design can be optimized for high-frequency workloads like AI-based services, edge use cases, and financial transaction processing, thereby ensuring low-latency authentication and policy enforcement even at scale. Real-time security monitoring and threat detection capabilities can also be boosted. Incorporating real-time security analytics using Prometheus and OpenTelemetry will also be a focus in future development to allow organizations to monitor authentication and access control events more effectively. Incorporating AI-driven anomaly detection can significantly improve security by identifying instances of JWT misuse, privilege escalation attempts, and unauthorized access patterns, thereby facilitating automated security alerts and proactive threat responses. The framework will be upgraded to accommodate zero-trust security models, such as risk-based authentication and contextual security analysis, to dynamically evaluate and enforce access control to stay abreast of evolving cybersecurity trends. By surmounting these future challenges, the gRPC Security Proxy will evolve into an even more adaptive, scalable, and intelligent security solution, providing strong authentication, high availability, and proactive threat detection in modern microservices architectures. These enhancements will enable the framework to be immune to future cybersecurity threats without compromising performance and ensure security compliance in dynamic environments.

9.5 Impact

The centralized gRPC proxy framework can substantially influence the adoption of safe communication within microservices architectures. With the rise in the usage of microservices, owing to their modularity and scalability, there is a real need to ensure secure communications between services. This is a centralized solution for mTLS, JWT authentication, and RBAC enforcement to manage complex security features without significant rework. The simplicity of integration might inspire more organizations to adopt secure communication practices in environments that are based on microservices. The framework significantly reduces the complexity of setting up security while promoting greater on-premises compliance due to GDPR, HIPAA, and FedRAMP. Strong access controls and secure data transmission practices must be in place for all applications. With the increasing demand for cloud-native applications and microservices, adopting frameworks like this may establish a standard way of securing inter-service communications. This will make it easier to enforce security policies and increase the general trustworthiness and reliability of contemporary distributed systems, empowering organizations to scale their services securely while reducing security risks.

9.6 Limitations

While the security proxy proposed for gRPC offers strong authentication, encryption, and access controls, it has certain limitations that must be considered when implementing it in a practical deployment. One of the significant concerns with this system is its use of a centralized security enforcement model. Because the proxy serves as an intermediary for policy enforcement and authentication, all gRPC traffic will have to go through it, presenting a single point of failure if not adequately distributed. To counter this issue, companies implementing the proxy should use load balancing and redundancy controls to prevent disruptions in the event of infrastructure failure. Another limitation is backward compatibility with legacy systems and older microservices architecture. Numerous microservices already in production may still use primitive TLS implementations or bespoke auth mechanisms that do not natively include mTLS or JWT authentication. Deploying the proxy in such environments may involve updating client configurations and dealing with certificates and API security pipelines. Additionally, services that expose SOAP-based communications or have older HTTP/1.1 stacks would need custom authentication arrangements because this proxy is optimized for gRPC-based communication. Performance bottlenecks can also happen under high-load situations, where the proxy needs to handle millions of authentications requests every second. Although the proxy has been benchmarked at 50,000 requests per second (RPS) with 3.5 millisecond average latency, intense load situations or considerable role-based access control (RBAC) policies can introduce processing overhead. For IoT or edge computing deployments with constrained computation resources and network latency, proxy-based security enforcement might not always be the optimal approach compared to lightweight client-side authentication models. Scalability is also a concern, especially when deploying the proxy in hybrid or multi-cloud environments. While it behaves well with Kubernetes and service meshes, heterogeneous infrastructure organizations may experience networking complexity when forwarding gRPC traffic through security enforcement points. Interoperability between clusters, cloud regions, and API security policies may require additional configurations and performance tuning. Finally, there is a trade-off between security enforcement and request latency. Every incoming request is authenticated, decrypted, and policy-checked, which, although optimized, introduces some latency. For low-latency applications such as financial trading or real-time analytics, this additional processing must be carefully balanced against security requirements. Despite these limitations, the proposed gRPC security proxy is still a flexible and extensible solution for contemporary microservices architecture, especially in cloud-native environments where centralized authentication and access control enforcement are of utmost priority. Sure of these problems can be resolved through future enhancements that may streamline request processing, include dynamic security policies, and enhance integration with edge computing frameworks.

10. ACKNOWLEDGEMENT

The authors would like to express their gratitude to all individuals and institutions who indirectly supported this research. No specific support was received that requires formal acknowledgment.

11. REFERENCES

[1] "Challenges of Implementing Microservice Architecture," opslevel.com, 2024. Available:https://www.opslevel.com/resources/challenges-of-implementing-microservice-architecture.

[2] "Enhancing gRPC Security | Best Practices for Secure Communication in Microservices," bytesizego.com, 2024. Available: https://www.bytesizego.com/blog/grpc-security.

[3] Chris Hendrix, "How to Secure Communication Between Microservices," Styra, 2023. Available: https://www.styra.com/blog/how-to-secure-communication-between-microservices/.

[4] Nicole Jones, "gRPC API Security Best Practices," StackHawk, 2024. Available: https://www.stackhawk.com/blog/best-practices-for-grpc-security.

[5] T. Farnham, "Supporting Disconnected Operation of

Stateful Services Using an Envoy Enabled Dynamic Microservices Approach," CLOSER, pp.115-122, 2023.

[6] N. Dattatreya Nadig, "Testing Resilience of Envoy Service Proxy with Microservices," Proceedings of diva-portal.org, 2019.

[7] W. Zhang, "Improving Microservice Reliability with Istio," willezhang.github.io, 2020.

[8] L. Calcote, Z. Butcher, Istio: Up and Running: Using a Service Mesh to Connect, Secure, Control, and Observe, O'Reilly Media, 2019.

[9] M. Chigurupati, A. Jagtap, "Enhancing Microservice Resiliency and Reliability on Kubernetes with Istio: A Site Reliability Engineering Perspective," International Journal of Computer Trends and Technology, Vol.72, No.11, pp.17-22, 2024. DOI:10.14445/22312803/IJCTT-V72111P103.

[10] R. Sharma, A. Singh, R. Sharma, A. Singh, "Policies and Rules," Getting Started with Istio Service Mesh: Manage Microservices in Kubernetes, pp.281-304, 2020.

[11] J. Suomalainen, Defense-in-Depth Methods in Microservices Access Control, Master's Thesis, 2019.

[12] M. G. de Almeida, E. D. Canedo, "Authentication and Authorization in Microservices Architecture: A Systematic Literature Review," Applied Sciences, Vol.12, No.6, p.3023, 2022. DOI:10.3390/app12063023.

[13] A. Barabanov, D. Makrushin, "Authentication and Authorization in Microservice-Based Systems: Survey of Architecture Patterns," arXiv preprint arXiv:2009.02114, 2020.

[14] H. Dong, Y. Zhang, H. Lee, K. Du, G. Tu, Y. Sun, "Mutual TLS in Practice: A Deep Dive into Certificate Configurations and Privacy Issues," Proceedings of the 2024 ACM on Internet Measurement Conference, pp.214-229, 2024. DOI:10.1145/3636512.

[15] B. Campbell, J. Bradley, N. Sakimura, T. Lodderstedt, "RFC 8705: OAuth 2.0 Mutual-TLS Client Authentication and Certificate-Bound Access Tokens," 2020.

[16] N. Li, M. V. Tripunitara, "Security Analysis in Role-Based Access Control," ACM Transactions on Information and System Security (TISSEC), Vol.9, No.4, pp.391-420, 2006.

[17] I. G. Buzhin, A. Y. Derevyankin, V. M. Antonova, A. P. Perevalov, "Comparative Analysis of the REST and gRPC Used in the Monitoring System of Communication Network Virtualized Infrastructure,"

[18] T-Comm-Telecommunications and Transport, Vol.17, No.4, pp.50-55, 2023.

[19] CGIAR Genetic Resources Policy Committee, "Summary Report of the Genetic Resources Policy Committee (GRPC) Meetings Held in 2005," 2006.

[20] Y. Yu, A. Jatowt, A. Doucet, K. Sugiyama, M. Yoshikawa, "Multi-Timeline Summarization (MTLS): Improving Timeline Summarization by Generating Multiple Summaries," Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers), pp.377-387, 2021.

[21] M. Pace, "Zero Trust Networks with Istio," Doctoral Dissertation, Politecnico di Torino, 2021.

[22] F. Pallas, "Hook-in Privacy Techniques for gRPC-based Microservice Communication," 2024.

[23] Z. Lai, Y. Xin, A. Yu, "Framework for Data Tracking Across Data Controllers and Processors," 2024.

[24] L. Arstila, Securing Microservices with Deep Learning-Long Short-Term Memory Autoencoder for Anomaly Detection, Master's Thesis, 2023.

[25] A. Dabholkar, V. Saraswat, "Ripping the Fabric: Attacks and Mitigations on Hyperledger Fabric," Applications and Techniques in Information Security: 10th International Conference, ATIS 2019, pp.300-311, 2019. DOI:10.1007/978-981-15-0871-424.

[26] JamesNK, "Performance Best Practices with gRPC," microsoft.com, 2024. Available:https://learn.microsoft.com/en-us/aspnet/core/grpc/performance?view=aspnetcore-9.0.

[27] A. de Waal, M. Weaver, T. Day, B. van der Heijden, "Silo-Busting: Overcoming the Greatest Threat to Organizational Performance," Sustainability, Vol.11, No.23, p.6860, 2019. DOI:10.3390/su11236860.

[28] F. Pallas, "Hook-in Privacy Techniques for gRPC-based Microservice Communication," 2024.

[29] E. Shmeleva, How Microservices Are Changing the Security Landscape, Master's Thesis, 2020.

[30] L. M. G. Silva, gRPC and Protobuf: Performance and API Flexibility, Doctoral Dissertation, 2024.

[31] Z. Li, S. He, Z. Yang, M. Ryu, K. Kim, R. Madduri, "Advances in APPFL: A Comprehensive and Extensible Federated Learning Framework," arXiv preprint arXiv:2409.11585, 2024.

[32] A. Gazibegovic, F. Rejabo, "Design and Implementation of a Distributed Fleet Simulator," 2021.

[33] "gRPC Proxy," etcd, 2022. Available: https://etcd.io/docs/v3.3/op-guide/grpcproxy/.

[34] P. Skentzos, "Software Safety and Security Best Practices: A Case Study from Aerospace," SAE Technical Paper Series, 2024. DOI:10.4271/2024-01-2618.

[35] M. Anedda, A. Floris, R. Girau, M. Fadda, P. Ruiu, M. Farina, A. Bonu, D. Giusto, "Privacy and Security Best Practices for IoT Solutions," IEEE Access, 2023. DOI:10.1109/ACCESS.2023.3345432.

[36] D. Fett, P. Hosseyni, R. Kusters, "An Extensive Formal Security Analysis of the OpenID Financial-Grade API," 2019 IEEE Symposium on Security and Privacy (SP), 2019. DOI:10.1109/SP.2019.00065.

[37] A. K. I. Riad, A. Barek, M. M. Rahman, M. S. Akter, T. Islam, M. A. Rahman, M. R. Mia, H. Shahriar, F. Wu, S. Ahamed, "Enhancing HIPAA Compliance in AI-Driven mHealth Devices Security and Privacy," 2024 IEEE 48th Annual Computers, Software, and Applications Conference (COMPSAC), 2024. DOI:10.1109/COMPSAC60750.2024.00099.

[38] S. Mbonihankuye, A. Nkunzimana, A. Ndagijimana, "Healthcare Data Security Technology: HIPAA Compliance," Wireless Communications and Mobile

Computing, 2019. DOI:10.1155/2019/1928704.

[39] F. Elkourdi, C. Wei, L. Xiao, Z. Yu, O. Asan, "Exploring Current Practices and Challenges of HIPAA Compliance in Software Engineering: Scoping Review," IEEE Open Journal of Systems Engineering, 2024. DOI:10.1109/OJSE.2024.3380011.

[40] N. Abbasi, D. A. Smith, "Cybersecurity in Healthcare: Securing Patient Health Information (PHI), HIPAA Compliance Framework and the Responsibilities of Healthcare Providers," Journal of Knowledge Learning and Science Technology, 2024. ISSN:2959-6386.

[41] S. Selvaraj, "Preserving Patient Confidentiality: The Vital Role of Data Tokenization in Ensuring Data Security and Regulatory Compliance in Healthcare," International Journal of Science and Research (IJSR), 2024. DOI:10.21275/SR2412011409.

[42] J. Duckworth, D. Gloe, B. Klein, "Software-Defined Multi-Tenancy on HPE Cray EX Supercomputers," 2023. Available:https://www.semanticscholar.org/paper/367afee8dfcb2a8f4ab42694061eb6eca8475dfa.

[43] R. Molleti, "Highly Scalable and Secure Kubernetes Multi-Tenancy Architecture for Fintech," Journal of Engineering and Applied Sciences Technology, 2022. DOI:10.5281/zenodo.6789100.

[44] J. Duckworth, D. Gloe, B. Klein, "Software-Defined Multi-Tenancy on HPE Cray EX Supercomputers," 2023. Available: https://www.semanticscholar.org/paper/367afee8dfcb2a8f4ab42694061eb6eca8475dfa.

[45] G. Chikafa, S. Sheikholeslami, S. Niazi, J. Dowling, V. Vlassov, "Cloud-Native RStudio on Kubernetes for Hopsworks," arXiv preprint arXiv:2307.09132, 2023.

[46] M. F. J. Potter, "The Integration of Ethernet Virtual Private Network in Kubernetes," Master's Thesis, 2019. Available: https://www.semanticscholar.org/paper/996acc4fe079e5ff5a6240decef9228130baebe3.

[47] C. Katsakioris, C. Alverti, K. Nikas, S. Psomadakis, V. Karakostas, N. Koziris, "FaaSCell: A Case for Intra-Node Resource Management: Work-In-Progress," Proceedings of the 1st Workshop on SErverless Systems, Applications and MEthodologies, 2023. DOI:10.1145/3595620.3595630.