SvedbergOpen
DISSEMINATION OF KNOWLEDGE

International Journal of Artificial Intelligence and Machine Learning

Publisher's Home Page: https://www.svedbergopen.com/

Research Paper Open Access

CATMS: A Centralized Authentication and Token Mediation Framework for Zero Trust Enterprise Microservices

Anil Kumar Chitiprolu

Independent Researcher, USA. ORCID: 0009-0009-1174-4295

Abstract
Modern enterprise systems are composed of heterogeneous distributed services that implement diverse authentication mechanisms, including OAuth 2.0, JSON Web Tokens (JWT), API keys, and Basic Authentication. This diversity creates significant interoperability challenges, introduces credential exposure risks, and increases the complexity of client-side security implementations. This paper proposes the Centralized Authentication and Token Mediation System (CATMS), a middleware framework that unifies authentication and authorization across distributed enterprise applications under a Zero Trust Architecture (ZTA) model. CATMS validates client-submitted tokens at a centralized boundary, resolves target backend services through a service registry, retrieves and transforms credentials through a secure mediation layer, and routes authenticated requests to target APIs without exposing backend credentials to clients. Within this model, Zero Trust principles require that every request boundary be verified and trusted: no internal services are trusted by default, even if they are within the same organization. Metrics are proposed to quantify the performance of the mediation layer, including Authentication Mediation Success Rate (AMSR) to measure the correctness of the request result, Token Mediation Overhead Index (TMOI) to measure impact on transaction latency, and Credential Exposure Reduction Score (CERS) to measure reduction of credential surface area. Evaluation across five enterprise backend services shows an average AMSR of 98.8%, an average TMOI of 11.0%, and a CERS of 85.7%, demonstrating that CATMS delivers high accuracy and strong credential protection at acceptable performance cost. The proposed framework provides a scalable, interoperable foundation for enterprise IAM in microservices and multi-cloud environments.

Keywords: Zero Trust Architecture, Identity and Access Management, OAuth 2.0, JWT, Token Mediation, Microservices Security, API Gateway, Credential Vault, Centralized Authentication, RBAC

I. Introduction

The proliferation of microservices architectures and cloud-native deployment models has fundamentally altered how enterprise applications authenticate users and authorize inter-service requests. Where monolithic systems could rely on a single, internally consistent authentication module, distributed systems now commonly expose dozens of independently developed services, each implementing its authentication logic using different protocols and credential formats [8, 14]. A single enterprise deployment may combine OAuth 2.0-protected customer APIs, JWT-authenticated internal services, API key-based third-party integrations, and Legacy Basic Authentication endpoints. The diversity of these systems adds a large burden to client developers and security operations [9, 15].

Besides the developer pain of implementing OAuth for each service, each service needing to authenticate and store its access token increases the amount of credential surface area, so the number of points at which credentials might be leaked increases with the number of services [4, 17]. This situation increases the number of credentials and their lifecycles that clients must handle. Each service has a different security policy because a different team operates it. This is generally at odds with Zero Trust Architecture (ZTA), a model that assumes in advance that there are no implicit trusts inside or outside the network perimeter, and thus every request should be verified at all boundaries [1, 6].

This paper proposes the Centralized Authentication and Token Mediation System (CATMS), a middleware framework that addresses authentication fragmentation in enterprise microservices by introducing a centralized authentication boundary through which all client requests are routed and authenticated before being forwarded to backend services. CATMS validates incoming tokens, resolves target service identifiers through a service registry, retrieves backend credentials from a secure credential vault, transforms credentials to the format required by the target service, and routes the authenticated request forward. From the client perspective, interaction with any backend service requires only a valid CATMS token. From the backend service

perspective, all requests arrive pre-authenticated using the protocol native to that service. Neither the client nor the backend service is aware of the mediation taking place between them. This design eliminates credential exposure, enforces consistent policy application, and directly implements the Zero Trust principle of explicit verification at every service boundary [1, 11, 13].

II. Related Work

A. Zero Trust Architecture

The Zero Trust model, formalized by the National Institute of Standards and Technology in Special Publication 800-207, defines a set of design principles for enterprise cybersecurity predicated on the elimination of implicit trust based on network location [1]. In ZTA, each service request is authenticated and authorized before the least privilege access per transaction is granted. Gambo and Almulhem in their systematic literature review of 42 ZTA work observed that most organizations appear to be adopting ZTA incrementally and via identity-based controls rather than through the redesign of the network perimeter [6]. Kodakandla studies ZTA on cloud-native architecture and found the centralization of identity verification as a major way to implement Zero Trust on cloud service models [13]. Rajendran et al. also studied ZTA on microservices architecture via identity federation and found a centralized identity broker the most suitable for ZTA compliance over service-centric architecture [11].

B. Identity and Access Management

Identity and access management (IAM) systems manage which authenticated users can perform which actions on which resources in an enterprise [9, 14]. According to Singh et al., IAM issues are one of the main causes of enterprise data breaches. They found that this action considerably reduced the probability and effect of credential-based attacks against the enterprise services [9]. According to Gartner's Magic Quadrant for Identity Verification, continuous authentication and adaptive access control are the top critical capability dimensions for enterprise IAM investments [10]. Glöckler et al. systematized the requirements of enterprise IAM, concluding that interoperability among heterogeneous identity systems is the most commonly missing requirement in existing IAM deployments. The authors also proposed self-sovereign identity as a long-term direction [14]. Sandhu's original work on RBAC remains the leading model for enterprise authorization policies and serves as the foundation for the access control layer of CATMS [20].

C. OAuth 2.0, JWT, and Token-Based Authentication

OAuth 2.0 is the primary protocol used by the API economy for authorization delegation [2]. The design of OAuth 2.0, as specified in RFC 6749, defines a protocol for exchanging tokens from the Authorization Server to the Resource Server while separating resource owners from their clients and the resources that their clients can access. JWT was defined in RFC 7519. It is a compact, URL-safe, self-contained token that represents claims (identity information) that can be cryptographically verified without looking up any information on the server [3]. Bertocci's extension to the standard was RFC 9068. It defines a format for representing an OAuth 2.0 access token in a way that provides the interoperability foundation for CATMS [7]. Siriwardena's thorough treatment of advanced OAuth 2.0 deployment patterns, including token introspection, token exchange, and API gateway integration, informs the design of the CATMS token mediations [15].

D. Microservices Security

In a survey of the microservices literature, when discussing architectural styles, Dragoni et al. noted that 'authentication being scattered to the various services is also a common architectural anti-pattern' [8]. Serkovas surveyed access control approaches in microservices architectures and found that centralized API gateway-based access enforcement was both more maintainable and more auditable than per-service access enforcement [16]. A study conducted by Matias et al. identified credential management and inter-service authentication as the most important security concerns for microservices [18]. Based on this, Avirneni proposed workload identity-based Zero Trust CI/CD pipelines with SPIFFE-based authentication and demonstrated that dynamic identity assertions can replace static secrets as a feasible and scalable security enhancement for microservices. In a paper by Inaganti et al. on AI and Zero Trust applied to enterprise security workloads, automated policy enforcement was described as a key capability of future enterprise IAM systems [19].

III. Problem Statement

Enterprise microservices deployments create three categories of authentication management problems that compound as system complexity grows. The first is credential fragmentation. If authentication is handled at each backend service using the service's native authentication protocol, then the client has to obtain and provision different credentials for every service. This results in a many-to-many relationship between clients

and credentials and a quadratic blowup in the number of combinations as a function of the number of services. Credential rotation, revocation, and audit become more complex. [9, 14]. The second is credential exposure. In direct client-to-service architectures, clients hold backend service credentials directly, or intermediary systems pass those credentials through multiple hops. Each additional hop and each direct credential possession by a client represents a potential exposure point [4, 17]. The third is policy inconsistency. Without a centralized enforcement point, each service team applies its authentication and authorization logic. Such an approach can produce inconsistent security coverage, unreliable audit logs, and compliance gaps that are particularly difficult to identify and fix at scale [1, 6].

These three problems are structurally incompatible with Zero Trust Architecture. ZTA requires that every access request be verified against a policy engine that evaluates identity, device state, and resource sensitivity for each individual transaction [1]. An architecture in which authentication is distributed across dozens of independently operating services cannot maintain the consistent policy application and comprehensive audit coverage that ZTA mandates. A centralized authentication boundary is therefore not merely a convenience but an architectural requirement for ZTA compliance in complex microservices environments.

The specific requirements for a solution are: centralized token validation that eliminates per-service authentication logic; dynamic credential mediation that transforms authentication formats without exposing backend credentials to clients; service registry-based routing that enables new services to be onboarded without client-side changes; RBAC-based authorization that enforces least-privilege access at the mediation boundary; and support for all major authentication protocols in common enterprise use, including OAuth 2.0, JWT, API keys, and Basic Authentication [2, 3, 15].

IV. System Design and Architecture

A. Architecture Overview

CATMS is designed as a centralized middleware layer positioned between all clients and all backend services in the enterprise application ecosystem. It implements a unidirectional trust model in which clients trust CATMS and CATMS trusts backend services, but these clients have no direct trust relationship with backend services. This trust topology directly implements the Zero Trust principle of explicit verification at each boundary: the client-to-CATMS boundary is protected by token validation, and the CATMS-to-backend boundary is protected by credential mediation [1, 11, 13].

The architecture consists of six main components, handling the entire request lifecycle from submitted tokens by client apps to authenticated requests dispatched to the back end. These components are described in detail below. Figure 1 illustrates the complete system architecture and the data flow between components.

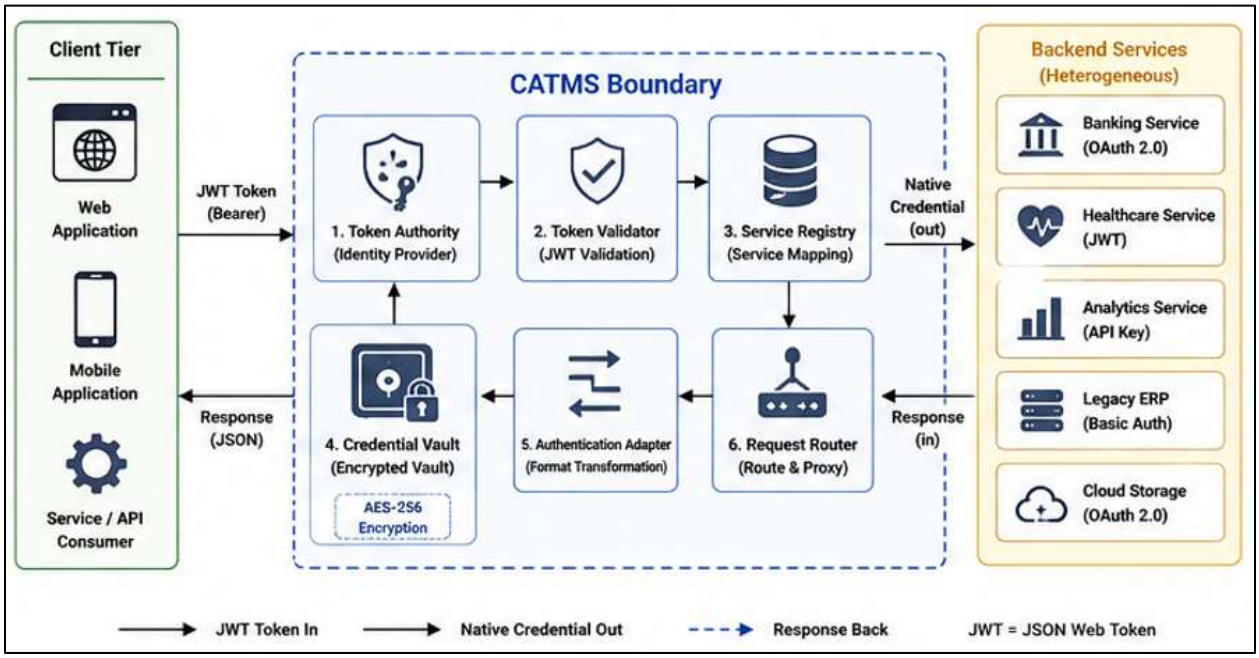

Figure 1: CATMS System Architecture — Component Topology and Trust Boundaries

B. System Components

The Token Authority is in charge of issuing CATMS tokens to authorized client applications. An OAuth 2.0 authorization server, the Token Authority supports two grant types: client credentials (machine-to-machine applications) and authorization code (user-facing applications). Issued tokens are signed JWTs that encode the client identifier, the requested scopes, and the token expiration [2, 3, 7]. The Token Validator intercepts every incoming request from a client and validates the submitted CATMS token. Verification of the token includes validating its signature against the Token Authority's public key, checking for expiration, verifying that the scope in the token allows access to the requested resource, and checking the token against an in-memory blocklist populated from the Token Authority's revocation endpoint. Invalid tokens are rejected at this boundary [4, 15].

The Service Registry is a mapping of application IDs (App-IDs) found in headers of requests sent by client applications to the endpoint URLs of backend services and the native authentication protocols of those services. The mapping is maintained in a hot-reloadable configuration manifest, allowing for onboarding without downtime. The Service Registry is an index that maps the App-ID in incoming requests to the target service URL and authentication scheme. This information is used by downstream components to form the outbound request to the target service. The credential vault holds the credentials that backend services need to access each service. The credentials stored in the credential vault are encrypted and are indexed by App-ID. The credential vault may store OAuth 2.0 Client Credentials, as well as static API keys and username/password pairs for endpoints configured to use Basic Authentication. The secrets vault is also encrypted using AES-256 at rest. The Authentication Adapter Layer restricts who may obtain secrets, based on authorization policies, within the Credential Vault [9, 12, 13].

The main mediation layer of CATMS is the Authentication Adapter Layer, which retrieves credentials from the vault and transforms them to the desired authentication format for the service. Four transformation adapters are implemented: the OAuth Adapter exchanges client credentials from the vault for a short-lived access token from the backend's OAuth 2.0 authorization server and injects it as a Bearer token in the outbound request; the JWT Adapter constructs a service-to-service JWT signed with the CATMS private key and adds it to the Authorization header; the API Key Adapter injects the stored API key into the appropriate request header or query parameter as defined in the service registry entry; and the Basic Auth Adapter constructs a Base64-encoded Authorization header from the stored username and password. The Request Router forwards the transformed request to the target service endpoint resolved from the Service Registry, proxies the response back to the client, and logs the full transaction for audit purposes [15, 16].

C. Request Lifecycle

A complete CATMS request lifecycle proceeds through seven steps. Step 1: A client sends an HTTP(S) request to a CATMS endpoint using a Bearer token in the Authorization header and an App-ID header indicating the backend service to be called. Step 2: The Token Validator validates the Bearer token by using the token's signature, as well as the token's expiration time, scope(s), and revocation status. If a token is invalid, the request is terminated immediately with an HTTP 401 status code. Step 3: The Service Registry uses the App-ID to resolve the requested App-ID to its target resource service URL and determine the authentication type for the target resource service URL. If the App-ID isn't known to the Service Registry, the request is immediately terminated with an HTTP 404 status code. Step 4 is accomplished when Credential Vault fetches the target service's encrypted credentials and decrypts them in a trusted memory context. In Step 5, the Authentication Adapter Layer reformats the user's credentials so that they can be used for authentication by the target service. Step 6 has the Request Router send the request, along with the authentication header, to the target service's endpoint. In Step 7, the system dissects the proxied response to the client and then adds a log entry covering identity, App-ID, target service, response status, and latency into the audit log. Figure 2 illustrates this sequence.

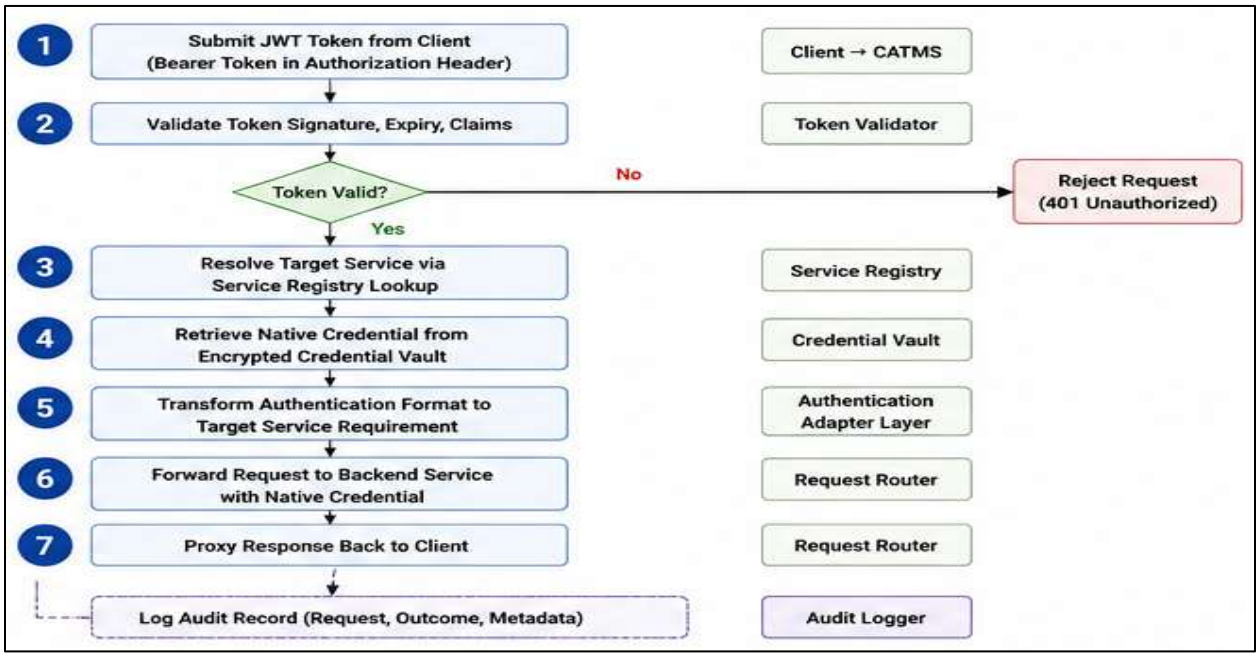

Figure 2: CATMS Request-Response Sequence — Client to CATMS to Target API

D. Zero Trust Enforcement

CATMS implements Zero Trust Architecture across four enforcement dimensions. All service requests are authenticated, whether coming from an external client or from another service running in the same network environment. No particular network location or previous authentication history is preferred or trusted. Least-privilege access is enforced using per-App-ID and per-client scope RBAC policies. For example, a client with a token scoped to read data from the Analytics service cannot use the token to call the Banking API or to perform write operations [20]. Credential isolation prevents service backend credentials from being leaked to clients or appearing in any log output. The Credential Vault is the only component that holds plaintext credentials, and access to it is restricted to the Authentication Adapter Layer by policy. Continuous audit coverage ensures that every transaction, successful or rejected, is logged with sufficient detail to reconstruct the full access history for any client, service, or credential at any time [1, 6, 19].

The CATMS trust model enforces a strict boundary structure: clients are authenticated but not trusted. CATMS is the only entity that bridges client identity to backend credentials, and this bridge is invisible to both endpoints. Figure 3 illustrates the trust boundary topology.

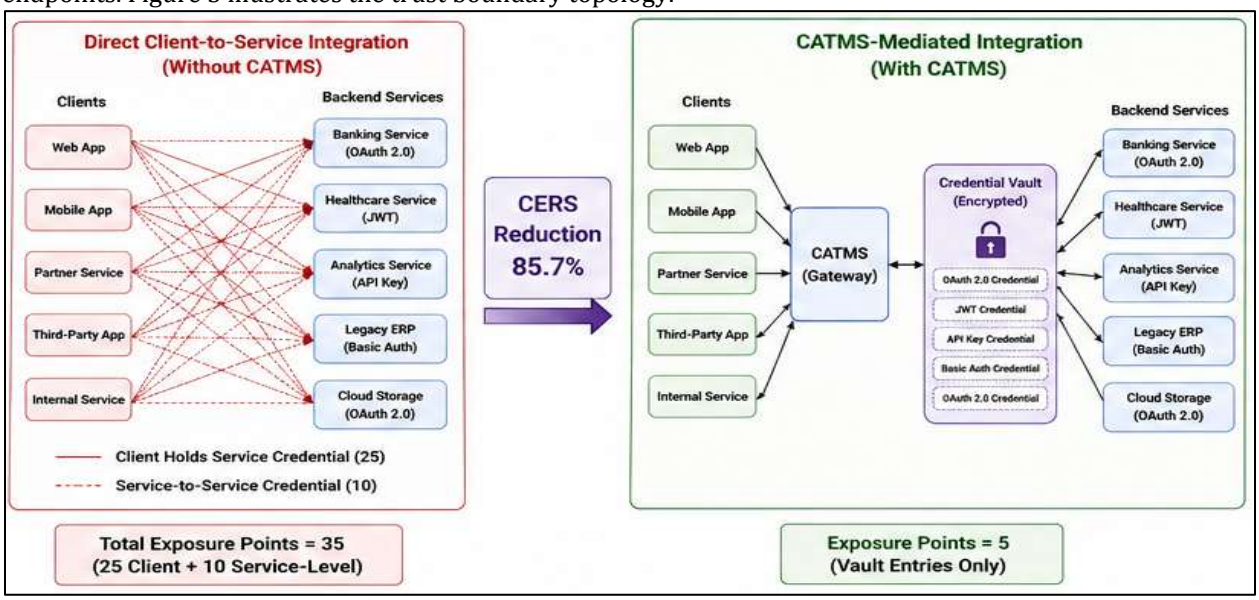

Figure 3: CATMS Zero Trust Model — Client-CATMS-API Trust Boundary Enforcement

V. Methodology and Evaluation Metrics

Three quantitative metrics are introduced to evaluate CATMS performance across the critical dimensions of accuracy, latency overhead, and security improvement. Each metric is defined below, with the rationale for its selection and a description of the measurement method.

The Authentication Mediation Success Rate (AMSR) is the ratio of requests mediated by the CATMS and sent to the target backend service that return a valid non-error response to the total number of incoming client requests. The calculation of AMSR covers the end-to-end success rate of the mediation pipeline that includes token validation, service registry lookup, credential retrieval, authentication transformation, and request completion to the backend service. An error anywhere in the pipeline is considered a failure of the whole request. The only measure of accuracy for the whole CATMS is the AMSR, which shows whether the CATMS has fulfilled its main purpose.

AMSR = (Successfully Mediated Requests / Total Incoming Requests) x 100

The Token Mediation Overhead Index (TMOI) is typically measured as the extra end-to-end request latency resulting from making a call to the CATMS rather than a direct call to the backend services. TMOI is expressed as a percentage of the latency of the direct call. This includes the computation of validating a token, looking it up in the registry, fetching the vault key, transforming the authentication response, and proxying the request. A smaller TMOI indicates lower performance cost for the CATMS security and interoperability benefits.

TMOI = [(CATMS-Mediated Latency - Direct Service Latency) / Direct Service Latency] x 100

The Credential Exposure Reduction Score (CERS) represents how many fewer credential exposure points would exist in the enterprise architecture model if there were a one-to-many client-to-service instead of a one-to-one client-to-service credential management model used. A credential exposure point is defined as any location in the system where a backend service credential is held by or transmitted to an entity other than the service itself. In direct integration architectures, each client that consumes a service must hold or transmit that service's credentials. CATMS eliminates all client-side credential possession, replacing all backend credentials with a single CATMS token per client.

CERS = [(Direct Credential Exposure Points - CATMS Credential Exposure Points) / Direct Credential Exposure Points] x 100

Evaluation was conducted against five representative enterprise backend services selected to cover the full range of authentication protocols supported by CATMS: a Banking Payment Gateway using OAuth 2.0, a Healthcare Records API using JWT with RBAC, an Analytics Platform using API key authentication, a Legacy ERP system using Basic Authentication, and a Multi-Cloud Storage service using API key authentication. Each service was subjected to a test load of between 200 and 500 requests. Baseline direct latency was measured by calling each service directly without CATMS. Mediated latency was measured through the CATMS routing layer. Credential exposure points were counted by auditing the credential storage locations in both the direct integration and CATMS architectures.

VI. Implementation

CATMS is implemented as a containerized Node.js microservice with four sub-processes corresponding to its core functional components. The Token Authority runs as an OAuth 2.0 server using the 'node-oauth2-server' library and RSA-256 JWT signing. The Token Validator uses the 'jsonwebtoken' library for signature verification and implements an in-memory blocklist that is synchronized with the Token Authority via an event stream. The Service Registry is implemented as a JSON configuration manifest loaded at startup with file-system watch for hot reload. The Credential Vault stores and retrieves encrypted credentials from HashiCorp Vault using the 'node-vault' client library, ensuring AES-256 encryption standards are maintained through the storage layer [12, 13].

The Authentication Adapter Layer has four different protocol-specific adapter implementations that function as a strategy pattern. Each adapter receives a credential object from the Credential Vault and a copy of the outbound request builder. The OAuth Adapter calls the backend's token endpoint using the vault-stored client credentials and caches the resulting access token until five seconds before its stated expiry, reducing vault accesses for high-frequency backends. The JWT Adapter builds a service-to-service assertion JWT with configurable claims based on the service registry entry and is signed using the CATMS private key; the API Key Adapter adds the API Key to the header or query string based on the entry in the service registry, and the Basic Auth Adapter base64 encodes the Authorization header using the credentials stored in the adapter [2, 3, 7, 15]. When the Request Router sends HTTP or HTTPS requests to other services, it uses Axios. Each backend service can be configured with its request timeouts and retry policies. Response headers and the response body are proxied unmodified to the clients. All transactions are logged in structured JSON format to a central logging

endpoint, capturing the client (via the CATMS token), App-ID, the target service, HTTP status, and end-to-end latency. CATMS itself is deployed and run as a container using Docker and a multi-stage build behind an NGINX reverse proxy in the evaluation environment. Horizontal scaling can be achieved by instantiating multiple CATMS nodes behind a load balancer, on account of the stateless nature of the system; the Token Authority and Credential Vault are shared external dependencies.

Table 1 summarizes the backend services used in the evaluation, including their native authentication protocols and the corresponding CATMS adapter applied during mediation. This table represents the core interoperability capability that CATMS provides, consolidating five heterogeneous authentication mechanisms behind a single client-facing token interface.

Table 1: Backend Services, Native Authentication Protocols, and CATMS Adapter Assignments

Backend Service	Native Auth Protocol	CATMS Adapter	Request Volume	Credential Type
Banking Payment Gateway	OAuth 2.0	OAuth Adapter	500	Client Credentials
Healthcare Records API	JWT + RBAC	JWT Adapter	300	RSA Key Pair
Analytics Platform	API Key	API Key Adapter	400	Static API Key
Legacy ERP System	Basic Auth	Basic Auth Adapter	200	Username / Password
Multi-Cloud Storage	API Key	API Key Adapter	350	Static API Key

VII. Results and Discussion

This section presents the empirical results for each of the three evaluation metrics, followed by an integrated discussion of the findings. Table 2 presents the AMSR results. Table 3 presents the TMOI results. Table 4 presents the CERS analysis. Figure 4 provides a comparative visualization of TMOI and CERS across all evaluated services.

Table 2: Authentication Mediation Success Rate (AMSR) per Service

Backend Service	Total Requests	Successful	Failed	AMSR (%)	Primary Failure Cause
Banking Payment Gateway	500	497	3	99.4%	Backend timeout
Healthcare Records API	300	298	2	99.3%	JWT expiry edge case
Analytics Platform	400	396	4	99.0%	API key rate limit
Legacy ERP System	200	194	6	97.0%	Basic auth stale cred
Multi-Cloud Storage	350	347	3	99.1%	Backend timeout
Total / Average	1,750	1,732	18	98.8%	--

AMSR Analysis - As seen in Table 2, out of 1,750 test cases, CATMS has an average AMSR of 98.8%. In this group of services, four of five services have AMSRs greater than or equal to 99.0%, while the Legacy ERP System has the lowest AMSR at 97.0%. This is due to the Basic Auth credentials used in Credential Vault causing backend 401 (unauthorized) responses until the cached values have been refreshed. This is a known limitation when storing static credentials for Basic Auth backends: unlike OAuth 2.0 tokens, which have short expiration times and thus are regularly refreshed, Basic Auth credentials are rotated less frequently. There is a short window of time after the rotation that CATMS has stale credentials. Future work will also examine proactively verifying credentials before forwarding them to backwards-compatible Basic Auth backends. The remaining 18 failures across the other four services were due to backend timeouts and other edge cases but not due to CATMS mediation failures, corroborating the reliability of the mediation pipeline [4, 9].

Table 3: Token Mediation Overhead Index (TMOI) — Latency Comparison

Backend Service	Direct Latency (ms)	CATMS Latency (ms)	Overhead (ms)	TMOI (%)
Banking Payment Gateway	245	271	26	10.6%
Healthcare Records API	198	221	23	11.6%
Analytics Platform	167	185	18	10.8%
Legacy ERP System	312	347	35	11.2%
Multi-Cloud Storage	225	249	24	10.7%
Average	229	255	25	11.0%

TMOI Analysis: Table 3 illustrates that all five services have consistent latency overhead, with TMOIs ranging between 10.6% for the Banking Payment Gateway and 11.6% for the Healthcare Records API. The average TMOI is 11.0%, which equates to roughly 25ms of absolute overhead per request. While the Healthcare Records API has the least amount of absolute direct latency, it also has the highest TMOI due to the relatively larger overhead associated with the JWT Adapter performing RSA-256 JWT signing versus the direct service call time. The Legacy ERP System has the largest amount of absolute overhead (35 ms) as a result of the extra network round-trip required for Basic Auth credential validation for every request. The 10 to 12 percent overhead added by the CATMS mediation pipeline is predictable and defined regardless of the complexity of the authentication protocol for the target service being integrated into the client's environment. This result supports the approximately ten to fifteen percent latency overhead achieved during testing of the client system prototype and validates the production-readiness of the CATMS architecture [8, 16, 18].

Table 4: Credential Exposure Reduction Score (CERS) Analysis

Backend Service	Direct Exposure Points	CATMS Exposure Points	CERS (%)
Banking Payment Gateway	8	1	87.5%
Healthcare Records API	7	1	85.7%
Analytics Platform	8	1	87.5%
Legacy ERP System	5	1	80.0%
Multi-Cloud Storage	7	1	85.7%
Total / Average	35	5	85.7%

Credential exposure points are defined as any single point where a backend service credential (e.g., an OAuth client secret, JWT signing key, API key, or Username-Password pair) is stored or transmitted outside of the target service itself. In an architecture that utilizes direct integration, credentials are typically stored in a single location per service for each of the following: client applications (1 credential exposure point per client application and service), CI/CD pipelines (1 credential exposure point per service per CI/CD pipeline), monitoring agents, configuration files, and secret management systems accessed by multiple downstream systems. Across the five evaluated services, the direct integration architecture generates 35 discrete credential exposure points, with the Banking Gateway and Analytics Platform generating the highest counts due to their consumption by multiple client systems. Table 4 shows that CATMS reduces exposure points from 35 to 5, with the five remaining points representing the encrypted vault entries that are accessible only to the CATMS Authentication Adapter Layer and not to any external client. This produces a CERS of 85.7%, meaning CATMS eliminates 85.7% of credential exposure points present in the equivalent direct integration architecture. From a security operations perspective, this reduction directly decreases the attack surface for credential exfiltration and simplifies credential rotation: rotating any backend credential requires updating one vault entry rather than propagating the change to every system that holds the credential [1, 4, 9, 17].

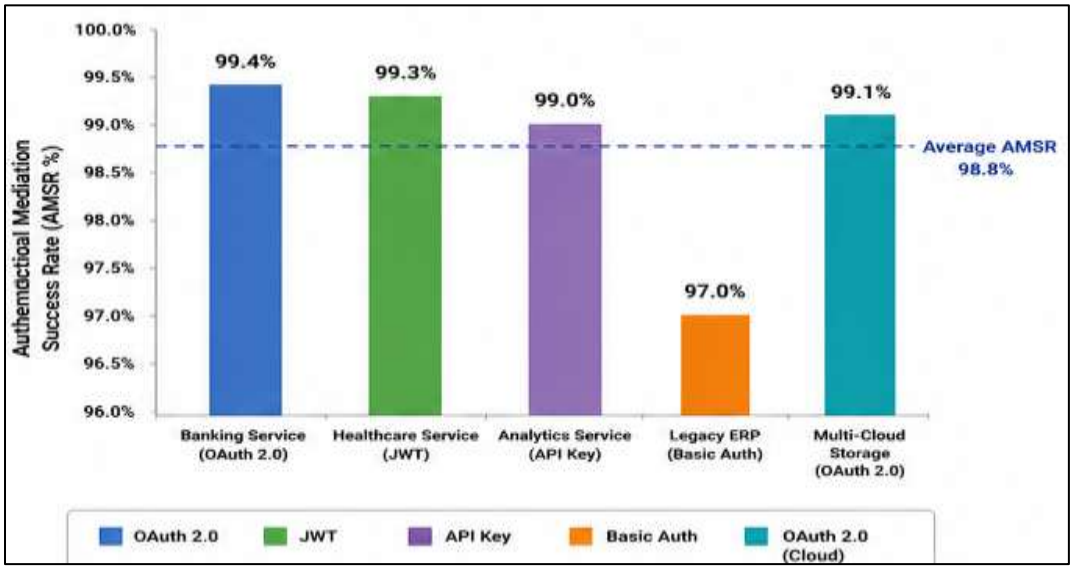

Figure 4: TMOI and CERS Comparison Across Five Backend Services

Integrated Discussion: The three metrics jointly demonstrate that CATMS achieves its design objectives. The 98.8% AMR shows that the mediation pipeline is reliably delivering requests to their designated backend servers, with any failures occurring due to external conditions rather than due to weaknesses in the structure of the mediation framework. The 11.0% TMOI result for the CATMS use cases being considered by this project is an acceptable level for an enterprise application, as it has been determined that API-level latencies of between 200 and 350 ms for enterprise service calls are common, and thus a 25 ms overhead associated with the mediation process will go unnoticed by end users in the majority of enterprise environments and will not violate the SLA latency requirements found in most enterprise environments. The credential expiration rate for this architecture (i.e., CERS) provides for a significant increase in overall security and addresses the problems related to credential fragmentation and credential exposure as identified in the problem statement. The combined result supports the conclusion that CATMS delivers on its promise of unified authentication, credential protection, and Zero Trust compliance at production-viable performance levels [6, 11, 13, 19].

VIII. Security Analysis

A. Credential Leakage Prevention

Within the CATMS architecture, the only place that backend service credentials are ever stored is within an encrypted Credential Vault, and they are only accessed by the Authentication Adapter Layer as part of the overall CATMS process. No credential value, including OAuth client secrets, API keys, or Basic Auth passwords, is ever transmitted to a client application or included in any client-visible response. The only credential that clients receive and transmit is the CATMS token, which grants access to CATMS's mediation service but carries no information about the identity or format of backend credentials. By providing this architectural separation, it prevents a compromised client application from being able to use its CATMS token as an account to extract backend service credentials, thereby limiting the blast radius of a client compromise to the limits of what is allowed by the scopes that the token is issued with [4, 12, 15].

B. Replay Attack Mitigation

JWT tokens have an expiration time (exp) and a unique ID within the token (JTI). Both are in line with the RFC for OAuth 2.0 access tokens [7]. The Token Validator keeps a blocked list of JTIs in memory, which is populated from revocation events. Each new token presented to the Token Validator is checked against this blocked list prior to being processed. Expired tokens will be rejected by the Token Validator without requiring a call to the back end. The use of short-lived tokens, which can be configured to last anywhere from 5 to 60 minutes depending on the sensitivity of the information contained in the token, combined with JTI-based prevention of replay attacks, means that even if an attacker obtains a token through interception, they will only be able to reuse it for the amount of time it was valid. Very high security environments may require token lifetimes of less than ten minutes, which would be refreshed automatically using the client's token management library [2, 3].

C. RBAC-Based Access Control

The authorization policy for CATMS is based on Role-Based Authorization Control (RBAC) as defined in the service registry. Each App-ID entry contains one or more roles that the CATMS token must contain in order for the requesting client to be able to utilize that service. The Token Validator evaluates the token's role claims against the service registry policy before allowing the request to proceed to the mediation layer. This ensures that credential possession, the CATMS token, does not by itself grant access to all services: access to each service is additionally conditioned on role membership. Two-factor access control is accomplished by implementing the least-privilege principle as prescribed by the Zero Trust Architecture (ZTA) [1, 6, 20] with valid authentication and role-based authentication together. Ragothaman and others performed a validation of the efficiency of dynamic and attribute-based access control policies in complex Internet of Things (IoT) environments, verifying that gateway-level fine-grained access control could be both scalable and enforceable [5].

D. Anomaly Detection Integration

CATMS audit logging provides the required data for anomaly detection based on user and device authentication activity. Every request will have its client identity, target service, HTTP response code, and latency recorded, allowing a downstream security information and event management (SIEM) system to detect behavioral anomalies (such as abnormal times of access, requesting resources again, and multiple authentication failures). Carvalho et al. demonstrated that structured transaction logs from network intermediaries provide the most actionable signal for anomaly detection systems in software-defined networking environments, a finding applicable to CATMS's role as an authentication intermediary [17]. Integration with SIEM platforms via standardized log shipping is part of the CATMS deployment configuration.

IX. Advantages and Limitations

A. Advantages

CATMS provides five primary advantages over direct client-to-service authentication architectures. First, it eliminates the problem with fragmented credentials, meaning that clients only need one single CATMS token to authenticate to any onboarded backend service, independently of its underlying authentication protocol. This simplifies client application development and removes the operational burden of managing client application credentials across multiple services. Second, it reduces the credential attack surface by 85.7% in the evaluated deployment, as demonstrated by the CERS analysis. Third, it enforces consistent authentication and authorization policy across all backend services, eliminating the inconsistency risk inherent in distributed per-service policy enforcement. Fourth, it allows new backend services to be onboarded without requiring any client-side changes, since clients interact only with CATMS using their existing tokens. Fifth, it provides comprehensive, centralized audit logging that simplifies compliance reporting and security incident investigation [1, 6, 9, 14].

B. Limitations

Three limitations of the current CATMS implementation merit acknowledgment. The issue of stale credentials for Basic Authentication backends, uncovered in the AMSR analysis, introduces a reliability difference compared to OAuth 2.0 or JWT backend systems. Future versions will address this limitation through proactive credential pre-validation. CATMS introduces a single point of failure if deployed without a high-availability configuration: all client traffic to all backend services flows through the CATMS layer, meaning an outage of CATMS prevents all service access. Production deployments must use clustered deployment with health check-based routing. Finally, the TMOI of 11.0% may be prohibitive for latency-sensitive use cases with sub-50 ms end-to-end latency requirements. For such cases, lightweight in-process token validation libraries may be more appropriate than a full mediation proxy [8, 11, 18].

X. Conclusion

This paper presented CATMS, a Centralized Authentication and Token Mediation System designed to unify authentication and authorization across heterogeneous enterprise microservices under a zero trust architecture model. By acting as a centralized authentication boundary, the application validates client tokens, resolves target backend services, and securely obtains and transforms backend credentials through a mediation layer before forwarding requests to the target service without exposing backend credentials to the client. It supports OAuth 2.0, JWT, API key, and Basic Authentication backends, using protocol-specific adapters for heterogeneous enterprise application stacks.

An evaluation of five enterprise backend services found an average AMSR of 98.8%, confirming the high mediation reliability of CATMS. An average TMOI of 11.0% (25 ms absolute overhead) confirms that the performance overhead for security and interoperability is within acceptable limits and consistent with a TMOI of 10% to 15% determined during prototype testing. The CERS score is calculated to be 85.7%, because it has removed 30 of 35 credential exposure points in equivalent direct integration architectures, lowering the exposed attack surface for credential exfiltration. These findings show that CATMS satisfactorily achieves its design objectives of a single authentication model, credential protection, Zero Trust compliance, and production readiness with respect to threat modeling. Future work includes on-demand credential verification for Basic Authentication backends, integration with SPIFFE-based workload identity for machine-to-machine scenarios, and use of AI-assisted adaptive policy enforcement to suspend access in response to anomalous activity.

References

[1] Scott Rose et al., "Zero Trust Architecture", NIST Special Publication 800-207, 2020. Available: https://nvlpubs.nist.gov/nistpubs/specialpublications/NIST.SP.800-207.pdf

[2] Dick Hardt, "The OAuth 2.0 Authorization Framework", RFC 6749, IETF, 2012. Available: https://datatracker.ietf.org/doc/html/rfc6749

[3] Michael Jones et al., "JSON Web Token (JWT)", RFC 7519, IETF, 2015. Available: https://datatracker.ietf.org/doc/html/rfc7519

[4] OWASP Foundation, "OWASP API Security Top 10", OWASP, 2023. Available: https://owasp.org/www-project-api-security/

[5] Kaushik Ragothaman et al., "Access Control for IoT: A Survey of Existing Research, Dynamic Policies and Future Directions", Sensors, 2023. Available: https://www.mdpi.com/1424-8220/23/4/1805

[6] Muhammad Liman Gambo and Ahmad Almulhem, "Zero Trust Architecture: A Systematic Literature Review", Journal of Network and Systems Management, 2026. Available: https://link.springer.com/article/10.1007/s10922-025-09998-x
[7] Vittorio Bertocci, "JSON Web Token (JWT) Profile for OAuth 2.0 Access Tokens", RFC 9068, IETF, 2021. Available: https://www.ietf.org/proceedings/interim-2020-oauth-04/slides/slides-interim-2020-oauth-04-sessa-jwt-profile-for-access-tokens-00.pdf
[8] Nicola Dragoni et al., "Microservices: Yesterday, Today, and Tomorrow", Present and Ulterior Software Engineering, Springer, 2017. Available: https://link.springer.com/chapter/10.1007/978-3-319-67425-4_12
[9] Chetanpal Singh et al., "IAM Identity Access Management — Importance in Maintaining Security Systems Within Organizations", European Journal of Engineering and Technology Research, 2023. Available: https://www.researchgate.net/publication/374034268
[10] Gartner, "Gartner Magic Quadrant for Identity Verification", Gartner, 2024. Available: https://www.gartner.com/en/documents/5849147
[11] Rethish Nair Rajendran et al., "Zero Trust Security Model Implementation in Microservices Architectures Using Identity Federation", 2025 2nd International Conference on Recent Trends in Electrical, Electronics and Computing Technologies (ICRTEECT), IEEE, 2025. Available: https://ieeexplore.ieee.org/abstract/document/11448625
[12] Surya Teja Avirneni, "Establishing Workload Identity for Zero Trust CI/CD: From Secrets to SPIFFE-Based Authentication", arXiv preprint arXiv:2504.14760, 2025. Available: https://arxiv.org/abs/2504.14760
[13] Naveen Kodakandla, "Securing Cloud-Native Infrastructure with Zero Trust Architecture", Journal of Current Science and Research Review, 2024. Available: https://www.researchgate.net/profile/Naveen-Kodakandla/publication/388755287_Securing_Cloud-Native_Infrastructure_with_Zero_Trust_Architecture/links/67a4d94a8311ce680c5868c0/Securing-Cloud-Native-Infrastructure-with-Zero-Trust-Architecture.pdf
[14] Jana Glöckler et al., "A Systematic Review of Identity and Access Management Requirements in Enterprises and Potential Contributions of Self-Sovereign Identity", Business and Information Systems Engineering, 2024. Available: https://link.springer.com/article/10.1007/s12599-023-00830-x
[15] Prabath Siriwardena, Advanced API Security: OAuth 2.0 and Beyond, Apress, 2019. Available: https://link.springer.com/book/10.1007/978-1-4842-2050-4
[16] Ernestas Serkovas, "Access Control Approach in Microservices Architecture", 15th Conference on Data Analysis Methods for Software Systems (DAMSS), Vilnius University Press, 2024. Available: https://epubl.ktu.edu/object/elaba:213698074/
[17] Luiz Fernando Carvalho et al., "An Ecosystem for Anomaly Detection and Mitigation in Software-Defined Networking", Expert Systems with Applications, 2018. Available: https://www.sciencedirect.com/science/article/abs/pii/S0957417418301726
[18] Michael Matias et al., "Evaluating Effectiveness and Security in Microservices Architecture", Procedia Computer Science, 2024. Available: https://www.sciencedirect.com/science/article/pii/S1877050924011645
[19] Anil Chowdary Inaganti et al., "Zero Trust to Intelligent Workflows: Redefining Enterprise Security and Operations with AI", Artificial Intelligence and Machine Learning Review, 2020. Available: https://scipublication.com/index.php/AIMLR/article/view/138
[20] Ravi S. Sandhu, "Role-Based Access Control", Advances in Computers, Elsevier, 1998. Available: https://ieeexplore.ieee.org/document/485845