PG-TAF: Policy Governed Tool Access Framework

Sahana P S[1], Shobika S A[2], V. Karpagam[3]
[1, 2, 3]Dept of CSE
[1, 2, 3, 4] K.L.N.College of Engineering,Sivagangai, India

Abstract- *Modern enterprises rely on a growing number of internal tools, APIs, and automated agents to perform critical operations. However, direct tool access models often result in fragmented authorization logic, embedded credentials, poor auditability, and limited governance. These challenges increase security risks and operational complexity, particularly in multi-tenant environments [10].*

This paper presents the Policy Governed Tool Access Framework (PG-TAF), a centralized control framework designed to enforce secure, policy-driven access to tools and APIs. PG-TAF introduces a gateway-based architecture where all tool requests are evaluated against configurable authorization policies before execution [8]. The framework integrates identity management, role-based and attribute-based access control (RBAC and ABAC) [1], [3], centralized secret handling with runtime injection [9], lifecycle enforcement, and immutable audit logging.

Unlike traditional direct-access systems, PG-TAF enables instant secret rotation without client-side updates and provides explainable policy decisions with organizational and workspace-level precedence rules. The system follows a microservices architecture to ensure separation of concerns, scalability, and tenant isolation [7]. A controlled evaluation is conducted to analyze authorization latency overhead, operational efficiency in secret rotation, and policy enforcement correctness across multiple scenarios. Results demonstrate that PG-TAF introduces predictable and bounded performance overhead while significantly improving governance, auditability, and operational security. The proposed framework provides a practical foundation for enterprise-grade tool orchestration and secure API governance in modern distributed environments.

Keywords: Policy Enforcement, Access Control, API Governance, Enterprise Security, Multi-Tenant Systems, Secret Management, Gateway Architecture, Auditability

I. INTRODUCTION

Modern software ecosystems increasingly depend on interconnected tools, APIs, and automated services to support development, deployment, monitoring, and operational workflows [6]. As enterprises scale, the number of internal and external integrations grows significantly, leading to complex access control requirements. In many conventional architectures, services and users access tools directly using embedded credentials and decentralized authorization logic. This approach creates challenges in maintaining consistent governance, enforcing policy updates, rotating secrets, and auditing system activity [10]. Centralized access control mechanisms such as Identity and Access Management (IAM) systems have addressed user-level permissions [5]; however, fine-grained tool governance and runtime enforcement across distributed services remain complex. Organizations require mechanisms that ensure policy consistency, enforce organizational boundaries, provide operational guardrails, and maintain traceability for compliance and incident response [2]. To address these challenges, this paper proposes the Policy Governed Tool Access Framework (PG-TAF), a microservices-based control framework designed to centralize authorization, secret management, and audit logging for tool and API execution [7]. PG-TAF ensures that all tool invocations pass through a policy-enforcing gateway that validates identity context, evaluates authorization rules, injects required secrets securely at runtime, and records detailed audit trails.

The key contributions of this work are as follows:

- A centralized, gateway-based policy enforcement architecture for tool access.
- Integration of RBAC and ABAC mechanisms with organizational precedence logic.
- Secure secret handling with dynamic injection and zero-downtime rotation capability.
- Comprehensive auditability with trace-based monitoring.
- A scenario-based and performance-oriented evaluation of operational impact.

The remainder of this paper is structured as follows: Section II reviews related work in access control and API governance. Section III defines the problem statement and system requirements. Section IV presents the PG-TAF architecture and design. Section V describes implementation details. Section VI evaluates the framework through performance and governance analysis. Section VII discusses

findings and limitations, and Section VIII concludes the paper with future directions.

II. RELATED WORK

Access control, API governance, and secure system orchestration have been extensively studied in distributed and enterprise computing environments. Traditional **Identity and Access Management (IAM)** systems provide centralized user authentication and role-based authorization mechanisms [5]. Cloud providers such as Amazon Web Services (AWS IAM), Microsoft Azure Active Directory, and Google Cloud IAM offer fine-grained identity-based permission models for managing user and service access to cloud resources.

While these systems effectively handle identity-centric authorization, they primarily focus on infrastructure resources rather than runtime tool invocation governance across heterogeneous service ecosystems.

Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) models have been widely adopted to enhance flexibility in authorization decisions [1], [3]. RBAC simplifies permission management through predefined roles, whereas ABAC evaluates contextual attributes such as environment, time, and resource metadata. Research in policy-based systems has demonstrated that combining RBAC and ABAC improves adaptability in dynamic enterprise environments [3]. However, many implementations lack integrated runtime enforcement mechanisms and often depend on decentralized application-level checks, which can lead to inconsistent policy application and limited auditability. API gateways and service mesh technologies, including Kong, Envoy, and Istio, provide request routing, authentication filters, and rate limiting capabilities [8]. These systems improve traffic management and observability but typically require external integration for advanced policy lifecycle governance, secret orchestration, and cross-tenant enforcement logic. Moreover, API gateways alone do not inherently provide structured policy versioning, organizational precedence models, or explainable authorization outcomes. Secret management solutions such as HashiCorp Vault and cloud-native key management services address secure credential storage and rotation [9]. These systems focus on protecting sensitive information through encryption and access policies. However, they do not natively integrate with unified policy-driven tool execution frameworks that combine identity validation, runtime secret injection, and audit-based traceability within a single governance layer.

Recent research in secure microservices architectures emphasizes centralized control planes to improve governance, observability, and security enforcement [7], [13]. Control-plane designs decouple policy evaluation from application logic, ensuring consistency and separation of concerns. Nevertheless, many existing approaches focus on infrastructure-level access or container orchestration rather than structured tool access governance with lifecycle management, emergency override controls, and tenant-scoped policy precedence. In contrast to prior systems, the proposed **Policy Governed Tool Access Framework (PG-TAF)** integrates identity validation, RBAC/ABAC-based authorization, secret management, lifecycle enforcement, and audit logging into a unified gateway-based architecture. PG-TAF extends beyond traditional IAM and API gateway capabilities by introducing organizational and workspace-level policy precedence, immutable policy versioning, break-glass emergency overrides, and operational secret rotation without client modification. This combination positions PG-TAF as a comprehensive governance framework tailored for enterprise-scale tool orchestration.

III. PROBLEM STATEMENT AND MOTIVATION

Modern enterprise environments rely heavily on interconnected tools, APIs, automation scripts, and service-to-service integrations to support development and operational workflows [6]. As the number of tools and integrations grows, managing secure and consistent access becomes increasingly complex. In many organizations, tools are accessed directly using embedded credentials or application-level authorization checks. This decentralized approach leads to fragmented security policies, inconsistent enforcement, and limited visibility into system activity.

One significant challenge in traditional architectures is the absence of a centralized enforcement mechanism. Authorization logic is often distributed across multiple services, making it difficult to ensure uniform policy application across tenants, teams, and environments [2]. Updating or revoking permissions requires changes in multiple locations, increasing operational overhead and the risk of misconfiguration. Furthermore, embedded credentials in applications complicate secret rotation, frequently resulting in downtime or delayed updates.

Another limitation is inadequate auditability and traceability. Direct tool access models typically lack comprehensive logging of authorization decisions, policy context, and execution paths [10]. In regulated environments, organizations require detailed audit trails to support compliance, incident investigation, and governance reporting.

Without centralized logging and trace identifiers, reconstructing decision paths becomes challenging.

Existing IAM systems primarily focus on identity-to-resource access but do not fully address runtime tool orchestration across heterogeneous APIs [5]. API gateways provide routing and rate limiting, yet they often lack structured policy lifecycle governance, precedence models across organizational scopes, and integrated secret injection mechanisms [8]. Similarly, secret management systems secure credentials but do not unify authorization evaluation and execution enforcement within a single control framework [9].

These challenges highlight the need for a centralized, policy-driven framework that ensures:

- Consistent and enforceable authorization across tools and tenants.
- Secure runtime secret handling with minimal operational disruption.
- Clear separation of concerns between identity, policy evaluation, and execution.
- Comprehensive audit logging with contextual traceability.
- Operational safeguards such as rate limiting, lifecycle enforcement, and emergency overrides.

The motivation behind Policy Governed Tool Access Framework (PG-TAF) is to address these limitations through a unified gateway-based architecture that centralizes governance while maintaining scalability and tenant isolation. By introducing structured policy precedence, dynamic secret rotation, and immutable audit logging, PG-TAF aims to reduce operational risk and improve trust in enterprise tool orchestration.

IV. SYSTEM OVERVIEW AND ARCHITECTURE

1. Design Overview

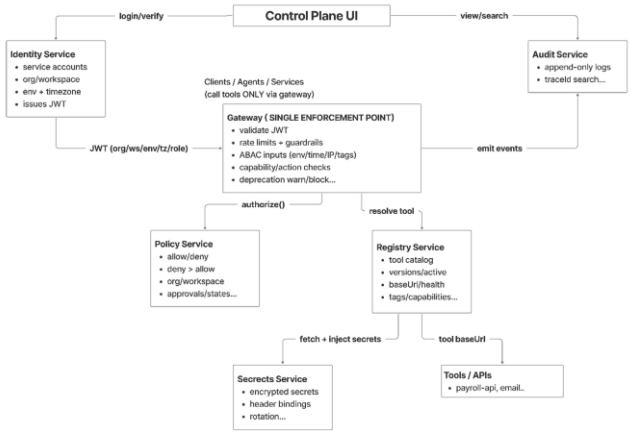

Fig. 1. PG-TAF Architecture and Request Flow

The Policy Governed Tool Access Framework (PG-TAF) is designed as a centralized control layer for governing and securing tool and API access in enterprise environments. The framework follows a microservices-based architecture [7] in which each component is responsible for a clearly defined function. This separation of concerns ensures scalability, maintainability, and strong tenant isolation.

At a high level, PG-TAF introduces a gateway-centric model where all tool invocations are routed through a single enforcement point [8]. Instead of allowing direct communication between clients and tools, the gateway validates identity context, evaluates authorization policies, injects required secrets, and records audit logs before forwarding the request to the target tool.

The architecture is composed of the following primary services:

- Identity Service
- Registry Service
- Policy Service
- Secrets Service
- Audit Service
- Gateway Service

Each service operates independently while communicating through secure internal interfaces.

2. Architectural Components

1. ***Identity Service****:* The Identity Service is responsible for authenticating principals (users or service accounts) and issuing secure tokens [5] that represent organizational context. Tokens contain attributes such as organization identifier, workspace identifier, role, and environment. These attributes form the basis

for downstream authorization decisions. The identity layer ensures strict tenant isolation by preventing clients from directly supplying organizational scope. Instead, scope information is derived from validated identity tokens.

2. ***Registry Service****:* The Registry Service acts as a tool catalog. It stores metadata describing registered tools, including:
    - Tool name and identifier
    - Base endpoint
    - Active version
    - Capabilities (e.g., read, write)
    - Lifecycle state (active, deprecated, sunset)
    - Ownership metadata

By separating tool metadata from execution logic, PG-TAF ensures controlled routing and lifecycle governance.

3. Policy Service*:* The Policy Service evaluates authorization decisions using a combination of Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) [1], [3]. Policies may include contextual attributes such as environment, time constraints, tool tags, and organizational scope.

PG-TAF implements structured precedence rules across organizational and workspace levels, ensuring deterministic decision resolution. Policies follow a lifecycle model supporting draft, approval, activation, and version rollback. Additionally, a break-glass override mechanism allows temporary emergency access with defined expiration.

4. Secrets Service*:* The Secrets Service manages encrypted credentials required for tool authentication. Secrets are stored securely and are never exposed directly to clients. When authorization succeeds, the gateway retrieves relevant secret bindings and injects them dynamically into outgoing tool requests [9].

This design enables immediate secret rotation without requiring modifications to client applications, thereby reducing operational disruption.

5. Audit Service*:* The Audit Service records all request outcomes, including allow, deny, and error events. Each audit entry contains contextual metadata such as principal identity, tool identifier, policy decision, reason, and trace identifiers [10].

6. Gateway Service*:* The Gateway Service functions as the centralized enforcement point [8]. All external requests to tools pass through the gateway. For each request, the gateway performs the following steps:

- Validates identity token.
- Extracts organizational and contextual attributes.
- Applies operational guardrails (e.g., rate limiting).
- Queries the Policy Service for authorization.
- Retrieves secret bindings if authorized.
- Enforces lifecycle constraints (e.g., deprecated or sunset tools).
- Forwards the request to the tool endpoint.
- Emits audit events.

By consolidating enforcement into a single layer, PG-TAF ensures consistent governance across all tool interactions.

3. Request Flow

The end-to-end request flow in PG-TAF can be summarized as follows:

- A principal authenticates via the Identity Service and obtains a token.
- The principal invokes a tool through the Gateway.
- The Gateway resolves tool metadata from the Registry.
- Authorization is evaluated by the Policy Service.
- If permitted, the Gateway retrieves required secrets.
- The request is proxied to the target tool.
- The Audit Service logs the outcome.

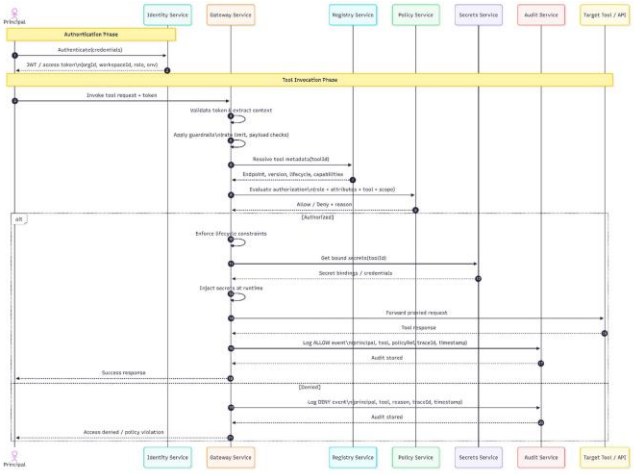

Fig. 2. Sequence Diagram of PG-TAF Request Flow

This flow guarantees that no tool can be accessed without passing through centralized policy evaluation and audit logging.

4. Design Principles

The architecture of PG-TAF is guided by the following principles:

- Centralized Enforcement: All tool access must pass through a single gateway [8].
- Separation of Concerns: Identity, policy, secrets, and audit are independent services [7].
- Tenant Isolation: Organizational context is derived strictly from validated tokens [2].
- Operational Safety: Guardrails such as rate limiting and lifecycle enforcement prevent misuse.
- Auditability: Every decision path is recorded for traceability [10].

These principles ensure that PG-TAF remains scalable, secure, and suitable for enterprise-scale environments.

V. IMPLEMENTATION DETAILS

1. *Technology Stack and Deployment Model*

The Policy Governed Tool Access Framework (PG-TAF) was implemented using a microservices-based architecture to ensure modularity and independent scalability of components [7]. The backend services were developed using a server-side framework supporting structured service-oriented design. A relational database system was used for persistent storage of identities, policies, tool metadata, and audit logs.

Each core service—Identity, Registry, Policy, Secrets, Audit, and Gateway—operates as an independent application communicating through secured HTTP-based APIs. This design enables loose coupling and allows services to evolve independently. Containerized deployment can be adopted to support scalability and distributed deployment models in enterprise environments.

The user interface component interacts exclusively with backend services through defined APIs and does not bypass gateway enforcement.

2. *Identity and Token Handling*

The Identity Service issues secure tokens to authenticated principals. Tokens include structured claims such as organization identifier, workspace identifier, role, environment, and timezone. These claims form the contextual basis for authorization decisions. The system enforces strict tenant isolation by ensuring that organizational scope is derived only from validated tokens. Clients cannot manually specify workspace or organizational identifiers, thereby preventing cross-tenant access vulnerabilities.

3. Policy Engine Design

The Policy Service implements a hybrid authorization model combining Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) [1], [3]. Policies are stored in structured form and evaluated dynamically at request time. Policy attributes may include:

- Principal role
- Tool identifier
- Organizational scope
- Environment constraints
- Time windows
- Tool tags

A deterministic precedence model is implemented to resolve conflicts across organizational and workspace-level policies. The policy engine also supports lifecycle states (e.g., draft, active, disabled) and maintains immutable versions for rollback and governance tracking.

An explain mechanism is incorporated to provide transparency into authorization decisions, enabling administrators to trace rule evaluation outcomes.

4. Gateway Enforcement Logic

The Gateway Service acts as the centralized enforcement component. It integrates identity validation, policy evaluation, lifecycle enforcement, secret resolution, and audit emission into a single execution pipeline.

Upon receiving a request, the gateway performs:

- Token validation and context extraction.
- Operational guardrail checks (e.g., rate limiting and payload size constraints).
- Policy authorization query.
- Tool metadata resolution.
- Lifecycle state verification (e.g., deprecated or sunset tools).

- Secret retrieval and runtime injection.
- Secure proxy forwarding to the target tool.
- Audit event generation with trace identifiers.

The enforcement pipeline is designed to maintain predictable latency while preserving strict governance guarantees.

5. Secret Management and Runtime Injection

Sensitive credentials required for tool access are stored in encrypted form within the Secrets Service. Secrets are bound to tools through defined injection configurations, such as HTTP header bindings [9].

When authorization succeeds, the gateway retrieves relevant secret bindings and injects them dynamically into outgoing tool requests. This design allows instant secret rotation without requiring updates to client applications, significantly reducing operational downtime compared to traditional embedded-credential approaches.

6. Audit Logging and Traceability

The Audit Service maintains append-only records of all request outcomes. Each audit entry includes:

- Principal identity
- Tool identifier
- Authorization decision
- Policy reference
- Timestamp
- Request and trace identifiers

This design ensures end-to-end traceability and supports compliance, monitoring, and incident investigation [10]. Audit entries may also be aggregated to generate usage analytics and operational metrics.

7. Security and Operational Safeguards

PG-TAF incorporates additional safeguards to enhance reliability and prevent misuse:

- Rate limiting at workspace, tool, and principal levels
- In-flight request limits
- Payload size restrictions
- Lifecycle enforcement for deprecated or sunset tools
- Break-glass emergency overrides with expiration

These mechanisms ensure that the framework not only enforces access control but also protects system stability and governance integrity.

VI. EVALUATION

1. Evaluation Methodology

The evaluation of PG-TAF focuses on three primary objectives:

- Measuring performance overhead introduced by centralized enforcement.
- Analyzing operational efficiency of centralized secret management.
- Validating correctness of policy enforcement under representative governance scenarios.

Since PG-TAF is a systems and security framework rather than a predictive model, evaluation is conducted using controlled scenario-based analysis and performance measurement.

Performance experiments were executed using the **k6 load testing framework**, simulating tool invocation requests through the PG-TAF gateway. The system was deployed in a controlled local environment to ensure consistent and reproducible measurements.

The workload configuration included:

- Virtual Users (VUs): 1
- Test Duration: 20 seconds
- Think Time: 1000 ms between requests
- Total Requests Executed: 107

Latency measurements were collected at two levels:

- End-to-end HTTP level
- Internal gateway processing stages, including:
  - Policy evaluation
  - Registry lookup
  - Secret resolution
  - Rate limiting
  - Proxy forwarding

This setup enables fine-grained analysis of latency contributions within the centralized enforcement pipeline.

2. Performance Overhead Analysis

1. *Objective:* To quantify the latency overhead introduced by PG-TAF during authorization, validation, and request routing.
2. *Method:* Latency was measured from request initiation to response completion using k6. In

addition to overall request duration, internal gateway processing time was decomposed into individual components:

- Policy evaluation latency
- Registry lookup latency
- Secret resolution latency
- Rate limiting overhead
- Proxy forwarding latency

End-to-end HTTP request duration was also recorded.

3. *Results:* The experimental results are summarized in Table I.

   System-level observations:

   - Average gateway latency: 53.45 ms
   - Median latency: 48 ms
   - 95th percentile latency (p95): 70 ms
   - Average HTTP request duration: 85.48 ms
   - Throughput: ~5.33 requests/sec
   - Success rate: 100% (0 failed requests)

2. *Observations:* The results indicate that PG-TAF introduces a **bounded and predictable latency overhead**, with total gateway processing averaging approximately 53 ms. Policy evaluation constitutes the largest portion of processing time, followed by registry lookup and secret resolution. Proxy forwarding and rate limiting contribute minimal overhead.

Despite centralized enforcement, the system maintains low response times, with p95 latency remaining well below 100 ms. This demonstrates that centralized governance can be achieved without significant degradation in responsiveness.

3. Operational Efficiency: Secret Management

1. *Objective:* To analyze the operational benefits of centralized secret management in reducing complexity during credential updates.
2. *Approach:* In PG-TAF, secrets are managed centrally through a dedicated Secrets Service. Unlike traditional systems where credentials are embedded within client applications, this design allows:
   - Centralized secret updates
   - Runtime secret retrieval by the gateway
   - No requirement for client-side modification
3. *Observations:* Although explicit timing benchmarks for secret rotation were not conducted, observed system behavior indicates that:
   - Updated secrets are immediately applied to subsequent requests
   - No request failures were observed during testing
   - Distributed credential update overhead is eliminated

TABLE I

Performance Metrics of PG-TAF Components

Metric	**Average**	**Median**	**P95**	**Max**
Total Gateway Latency	53.45 ms	48 ms	70 ms	312 ms
HTTP Request Duration	85.48 ms	76.17 ms	134.17 ms	421.51 ms
Policy Evaluation	20.80 ms	20 ms	29 ms	60 ms
Registry Lookup	13.67 ms	12 ms	21.4 ms	48 ms
Secret Resolution	12.53 ms	11 ms	17.7 ms	53 ms
Proxy Forwarding	4.47 ms	4 ms	7 ms	30 ms
Rate Limiting	~1.37 ms	0 ms	0 ms	144 ms

This highlights the operational advantage of centralized secret management.

4. Policy Enforcement Validation

1. *Objective:* To verify correctness and consistency of policy enforcement under representative governance conditions.
2. *Method:* A set of controlled policy scenarios was validated, including:
   - Conflicting policies across organizational and workspace levels
   - Policy lifecycle states (draft and active)
   - Break-glass emergency access
   - Tool lifecycle states (deprecated and sunset)
3. *Results:* The outcomes are summarized in Table II.
4. *Observations:* All validated scenarios produced expected outcomes, confirming that:
   - Policy precedence rules are deterministic
   - Organizational policies override workspace-level policies
   - Lifecycle constraints are correctly enforced
   - Break-glass access operates with proper expiration

These results validate the correctness and reliability of the policy engine.

5. Auditability and Trace Coverage

All executed requests generated audit logs containing:

- Principal identity
- Tool identifier
- Authorization decision
- Policy reference
- Timestamp
- Trace identifier Across 107 requests:
- 100% audit coverage was observed
- No inconsistencies or missing entries were detected

This confirms that PG-TAF provides complete traceability for monitoring, compliance, and incident analysis.

6.

Discussion

The evaluation demonstrates that PG-TAF achieves centralized governance with **low and predictable performance overhead** and **high operational reliability**.

Key findings:

- Centralized enforcement introduces approximately 53 ms average overhead
- High reliability with 0% failure rate
- Deterministic policy enforcement behavior
- Simplified secret management through centralization
- Complete audit trace coverage

Although the evaluation was conducted under a single-user workload in a controlled environment, it establishes a baseline for system performance and correctness.

Future work will focus on:

- High-concurrency workload evaluation
- Distributed deployment scenarios
- Scalability and fault-tolerance analysis

VII. DISCUSSION AND LIMITATIONS

Discussion

The evaluation of PG-TAF demonstrates that centralized policy enforcement and governance can be achieved with predictable performance overhead and significant operational advantages. By consolidating identity validation, policy evaluation, secret injection, lifecycle enforcement, and audit logging into a unified gateway architecture, the framework ensures consistent access control across tools and tenants [7]. One of the most impactful outcomes observed is the operational improvement in secret rotation. Traditional distributed credential management often results in service interruptions and manual update overhead. PG-TAF eliminates this complexity by enabling centralized secret updates that propagate instantly without client-side modifications. This strengthens both security posture and operational agility [9].

Furthermore, the deterministic policy precedence model ensures reliable enforcement across organizational scopes. The integration of RBAC and ABAC mechanisms enhances flexibility while maintaining structured governance. Audit traceability across all request paths reinforces accountability and supports compliance requirements. Overall, the framework demonstrates that governance and operational efficiency can coexist within a scalable enterprise control architecture.

TABLE II
Scenario Outcomes in PG-TAF Evaluation

Scenario	**Expected Outcome**	**Observed Outcome**
Workspace allow + Org deny	Deny	Deny
Policy in draft state	Not enforced	Not enforced
Break-glass active	Temporary allow	Allow
Break-glass expired	Deny	Deny
Deprecated tool	Warning	Warning
Tool after sunset	Deny	Deny

Limitations

While PG-TAF provides strong governance capabilities, several limitations should be acknowledged.

First, the evaluation was conducted in a controlled deployment environment. Large-scale distributed or cloud-native deployments may introduce additional network latency or concurrency considerations that require further benchmarking.

Second, the current implementation relies on a centralized gateway model. Although this simplifies governance, high-availability deployments would require replication and load balancing mechanisms to prevent single points of failure. Third, while the policy engine supports RBAC and ABAC, it does not currently integrate with external policy description languages such as Rego or standardized policy-as-code ecosystems. Future extensions could explore interoperability with broader enterprise governance platforms.

Finally, the framework focuses on runtime tool access governance and does not address infrastructure-level orchestration or container-level policy enforcement, which may be complementary in full-scale enterprise deployments.

VIII. CONCLUSION AND FUTURE WORK

This paper presented PG-TAF (Policy Governed Tool Access Framework), a centralized governance framework designed to secure and audit tool and API access in enterprise environments [7], [10]. The framework integrates identity validation, policy-driven authorization, secure secret management, lifecycle enforcement, and comprehensive audit logging into a unified gateway-based architecture.

Through controlled evaluation, PG-TAF was shown to introduce bounded and predictable performance overhead while significantly improving operational efficiency, particularly in secret rotation and policy consistency. Scenario-based validation confirmed deterministic enforcement of precedence rules, lifecycle states, and emergency override mechanisms.

The results indicate that centralized policy-governed access control is a practical and scalable approach for enterprise tool orchestration. By separating identity, policy evaluation, secret handling, and audit into distinct services, PG-TAF promotes modularity, tenant isolation, and operational transparency.

Future work will focus on scalability benchmarking under high-concurrency workloads, distributed deployment across multiple regions, integration with standardized policy-as-code [11] engines, and advanced analytics for anomaly detection in audit logs. Additionally, exploring automated policy verification and formal validation methods may further strengthen governance reliability.

PG-TAF establishes a foundation for secure, auditable, and operationally efficient tool access governance in modern distributed systems.

REFERENCES

[1] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, "Role-Based Access Control Models," *IEEE Computer*, vol. 29, no. 2, pp. 38–47, Feb. 1996, doi: 10.1109/2.485845.

[2] V. C. Hu, D. F. Ferraiolo, and D. R. Kuhn, "Assessment of Access Control Systems," NIST Interagency Report (NISTIR) 7316, 2006. [Online]. Available: https://csrc.nist.gov/publications/detail/nistir/7316/ final

[3] V. C. Hu, D. R. Kuhn, and D. F. Ferraiolo, "Attribute-Based Access Control," *IEEE Computer*, vol. 48, no. 2, pp. 85–88, Feb. 2015, doi: 10.1109/MC.2015.33.

[4] A. J. Ferrer, F. Herna´ndez, J. Tordsson, E. Elmroth, A. Ali-Eldin, C. Zsigri, R. Sirvent, J. Guitart, R. M. Badia, K. Djemame, and W. Ziegler, "OPTIMIS: A Holistic Approach to Cloud Service Provisioning," *Future Generation Computer Systems*, vol. 28, no. 1, pp. 66–77, Jan. 2012, doi: 10.1016/j.future.2011.05.022.

[5] Amazon Web Services, "AWS Identity and Access Management (IAM)." [Online]. Available: https://docs.aws.amazon.com/IAM/latest/ UserGuide/introduction.html

[6] C. Pautasso, O. Zimmermann, and F. Leymann, "RESTful Web Services vs. 'Big' Web Services: Making the Right Architectural Decision," in *Proc. 17th Int. World Wide Web Conf. (WWW)*, Beijing, China, 2008, pp. 805–814, doi: 10.1145/1367497.1367606.

[7] W. Hasselbring and G. Steinacker, "Microservice Architectures for Scalability, Agility and Reliability in E-Commerce," *IEEE Software*, vol. 34, no. 3, pp. 38–45, May/Jun. 2017, doi: 10.1109/MS.2017.63.

[8] NGINX, "API Gateway Pattern." [Online]. Available: https://microservices.io/patterns/apigateway.html

[9] HashiCorp, "Vault Documentation." [Online]. Available: https:// developer.hashicorp.com/vault/docs

[10] S. Pearson and A. Benameur, "Privacy, Security and Trust Issues Arising from Cloud Computing," in *Proc. IEEE 2nd Int. Conf. Cloud Computing Technology and Science (CloudCom)*, 2010, pp. 693–702, doi: 10.1109/CloudCom.2010.66.

[11] Open Policy Agent (OPA), "Policy-as-Code for Cloud Native Environments." [Online]. Available: https://www.openpolicyagent.org/ docs/latest/

[12] L. Bass, I. Weber, and L. Zhu, *DevOps: A Software Architect's Perspective*. Boston, MA, USA: Addison-Wesley, 2015.

[13] B. Burns, B. Grant, D. Oppenheimer, E. Brewer, and J. Wilkes, "Borg, Omega, and Kubernetes," *ACM Queue*, vol. 14, no. 1, pp. 70–93, Jan.–Feb. 2016, doi: 10.1145/2898442.2898444.

[14] M. Fowler, "Microservices." [Online]. Available: https://martinfowler. com/articles/microservices.html