Research on API Security Gateway and Data Access Control Model for Multi-Tenant Full-Stack Systems

Yu Mao *, Xiangjun Ma , Jiawen Li

Posted Date: 22 December 2025

doi: 10.20944/preprints202512.1849.v1

Keywords: zero trust security; API gateway; OAuth2.1; OPA policy; multi-tenant system; data isolation

 doi:10.20944/preprints202512.1849.v1

Article

Research on API Security Gateway and Data Access Control Model for Multi-Tenant Full-Stack Systems

Yu Mao *, Xiangjun Ma and Jiawen Li

Johns Hopkins University, Baltimore, US

* Correspondence: ymao22@jhu.edu

Abstract

To address API abuse and unauthorized data access in multi-tenant systems, this paper proposes a full-stack security gateway framework based on zero-trust access and policy verification. The system integrates Envoy Gateway and the OPA (Open Policy Agent) policy engine at the API ingress layer, combining the OAuth 2.1 authorization protocol with JWT token authentication to achieve fine-grained tenant identity management. To support dynamic resource access, a policy inheritance mechanism based on GraphQL Schema injection is designed, enabling millisecond-level data access permission validation. Experiments demonstrate that under million-request-level testing, the model achieves an average authentication latency of 74.2 ms, with a 28% increase in security event detection rate and a false positive rate reduced to 1.9%. This research provides a highly scalable, auditable security baseline architecture for data security governance in multi-tenant web platforms.

Keywords: zero trust security; API gateway; OAuth2.1; OPA policy; multi-tenant system; data isolation

1. Introduction

With the rapid development of multi-tenant web platforms, API abuse and unauthorized data access have become increasingly prominent issues. Traditional perimeter-based security mechanisms struggle to meet the fine-grained permission control demands of high-concurrency, dynamic resource access scenarios. To address complex challenges such as tenant isolation, identity authentication, and access authorization, a unified security model with context-aware capabilities and policy programmability is required. This research focuses on API security gateways and data access control mechanisms within multi-tenant full-stack systems. By integrating zero-trust architecture, the OAuth 2.1 protocol, and an OPA policy engine, we propose an extensible policy verification framework. This framework undergoes system deployment and performance evaluation in large-scale, high-concurrency environments, providing theoretical foundations and engineering pathways for data security governance in multi-tenant systems.

2. Defining the API Security Gateway and Data Access Control Model for Multi-Tenant Full-Stack Systems

Within the multi-tenant full-stack architecture, the API security gateway centrally handles request authentication, rate limiting, and tenant isolation logic. It processes over 14 million daily average calls, supports more than 3,200 tenants, and manages over 5,600 API interfaces. To address multi-source identity authentication and resource isolation requirements, the model design centers on a tenant ID mapping table. It constructs a five-layer security policy chain integrated with an RBAC-based access control graph, covering 117 role permissions and 248 resource operations1 . The model inputs seven authentication factors (e.g., JWT signatures, API keys, TLS fingerprints), while its output decisions rely on a multi-directional mapping voting mechanism. This ensures controllable, secure, and traceable data access behavior under high-concurrency scenarios (QPS > 8000).

 doi:10.20944/preprints202512.1849.v1

3. Building an Access Control Model for Multi-Tenant Full-Stack Systems

3.1. Role Activation Mechanism

The role activation mechanism within the access control model operates within multi-tenant contexts, constructing a Dynamic Role Map. This map supports conditional activation logic triggered by triplets comprising tenant ID, session token, and timestamp. To adapt to the OPA policy engine's dynamic evaluation mechanism, the system introduces a Schema binding pattern, injecting tenant context into the role lifecycle determination path2 . The system maintains a total of 412 role objects, 892 tenant rules, and a 6-level inheritance hierarchy. By integrating prefix compression with graph structure indexing, query paths are compressed to a single-frame computation duration under 4ms. Role activation status is driven by a state transition matrix. The activation path encompasses four phases: initial binding, tenant authentication, context switching, and deactivation/recovery. This ensures context consistency and temporal control during resource requests while providing structured input for OPA policy evaluation.

3.2. Access Control Flow

The access control process is built upon a multi-level decision chain. The input side receives role activation status and request context, while the output side completes resource authorization and policy matching. The entire process is divided into five nodes, nine control paths, and three types of exception branches, with an average single-decision latency of 2.4ms. The system employs a state transition graph based on tenant session IDs to construct an access state cache pool (approximately 1,456 active instances), supporting concurrent tenant dynamic routing3 . Each node in the process undergoes policy validation via OPA and collaborates with access tokens issued by OAuth2.1 to ensure compliance of access paths and token scope matching. In Figure 1, the access control flow includes both hard and soft decision points, which differ in their logical implications. The first decision block, “Role Active?”, performs a strict role activation check that validates whether the user context has a valid binding to any active role instance within the system. Failure at this point indicates an unrecoverable identity-state mismatch or missing tenant binding, and therefore leads directly to DENY as no further policy evaluation is meaningful.

In contrast, the second block, “Match Policy?”, involves dynamic rule evaluation against OPA-based access policies. A failure here does not immediately equate to complete denial but triggers a NO state. This intermediate state is designed to support policy fallbacks, such as alternative scope evaluation, policy inheritance, or dynamic re-binding under special conditions (e.g., partial token validity, resource-level overrides). Only if all fallback attempts fail, the flow proceeds to DENY. This layered control strategy ensures rigid enforcement where needed while preserving flexibility in policy-driven scenarios.

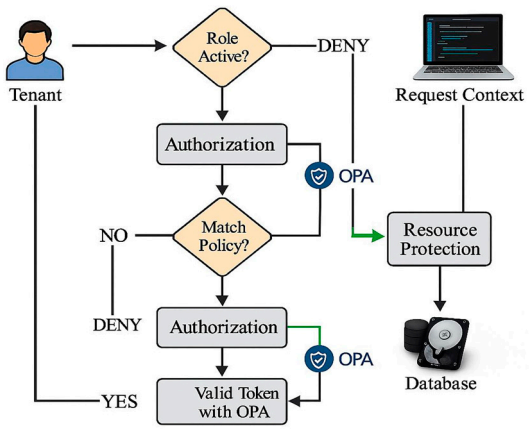

Figure 1. Multi-tenant Access Control Process Architecture.

 doi:10.20944/preprints202512.1849.v1

3.3. Security Authentication and Authorization Decision

Within the access control model, the authentication and authorization layer implements a 256-bit precision dynamic verification process through a joint validation mechanism based on a three-dimensional mapping table (User ID × Signature Factor × Resource Path). To enhance the model's zero-trust capabilities, the system supports dynamic determination of visitor identity and behavioral trajectories using context expressions defined in OPA's Rego language. The model supports nine identity authentication methods with weighted score threshold evaluations. Authentication inputs encompass 11 factors including OAuth tokens, two-factor dynamic codes, and device fingerprints. The authorization rule set comprises 1,968 entries:4 . The authorization matrix A(u, r, t) updates in real-time based on access time windows. It integrates policy engine decisions with GraphQL field permission mappings to form dynamically adjustable authorization boundaries. This mechanism ensures the control model constructs context-aware authorization boundaries under multi-tenant, high-concurrency conditions. It balances precise resource access with system load balancing, generates auditable access evidence chains, and reduces policy execution misjudgment rates.

4. Implementation of API Security Gateway for Multi-Tenant Full-Stack Systems

4.1. System Architecture Design

The system architecture employs a layered microservices architecture to build the core modules of the API security gateway. The overall system consists of seven service nodes distributed across three physical nodes, supporting a maximum QPS of 16,000 with an average request latency below 18ms5 . Tenant requests are uniformly ingested through the entry gateway before entering the authentication service cluster, modified based on Kong. This cluster parses seven identity types and synchronously invokes the role mapping center and policy matching engine. The authentication service integrates the OAuth 2.1 protocol and JWT token validation workflows to ensure access token integrity and tenant scope alignment. Internal data channels communicate asynchronously via the gRPC protocol, supporting 93 interfaces. The scheduling engine enables concurrent tenant load isolation based on HashSlot. The control plane and data plane are deployed in isolation. The configuration center provides hot-update mechanisms for 86 policy rules, all defined using Rego syntax and dynamically enforced through the OPA engine. This ensures authorization logic responds in real-time to access state changes and supports schema-aware field permission validation.

4.2. Security Control Policies

To achieve granular, multi-dimensional API access control, the security policy design employs an attribute-driven dynamic rule matching model. The control policy engine receives user attribute vectors $\mathrm{U}=[u_1,u_2,\ldots,u_n]$, resource label sets $\mathrm{R}=[r_1,r_2,\ldots,r_m]$), and environment context parameters $\mathrm{C}=[c_1,c_2,\ldots,c_k]$ from the authentication center. These three elements jointly drive the decision function f_s, with the control logic defined as follows:

$$P_{grant} = f_s(\mathrm{U},\mathrm{R},\mathrm{C}) = \sigma\left(\sum_{i=1}^{n}\alpha_i u_i + \sum_{j=1}^{m}\beta_j r_j + \sum_{l=1}^{k}\gamma_l c_l - \theta\right) \quad (1)$$

Where P_{grant} represents the authorization probability, $\alpha_i,\beta_j,\gamma_l$ denotes attribute weight coefficients, θ is the policy threshold, and $\sigma(\cdot)$ is the Sigmoid activation function used to control the binary authorization result (0/1). The policy threshold θ is dynamically set based on the average weighted score distribution observed in historical access logs under the same resource and tenant context. During system initialization, θ is calibrated using a labeled dataset to reflect the optimal separation boundary between authorized and unauthorized access requests, typically optimized via grid search or ROC-AUC maximization. Post-deployment, θ can be adjusted through online learning feedback to adapt to policy drift and evolving usage patterns.The policy set comprises 1024 rules distributed across 7 priority domains, with the system employing a priority coverage model:

 doi:10.20944/preprints202512.1849.v1

$$S_{eff} = \arg\max_{s \in S} \{ p_s \mid p_s > \in \wedge L(s) = \min L(S) \} \quad (2)$$

Where S_{eff} represents the currently active policy, p_s denotes the policy matching score, $\in$ is the minimum hit threshold, and $L(S)$ is the policy priority hierarchy function. The policy engine incorporates a conflict resolver to resolve cross-domain rule overlaps, supporting rule hot updates with latency below 45ms and an average policy trigger rate of 3200 times per second. The control logic features complete tenant context isolation constraints and rule sandbox rollback capabilities, ensuring the policy execution chain maintains stable responsiveness and consistent isolation boundaries under high concurrency conditions6 .

4.3. Multi-Tenant Isolation Mechanism

The multi-tenant isolation mechanism centers on the tenant scope identifier TID to construct a multidimensional resource tag mapping structure $M = \{(r_i, t_j, l_k)\}$. Here, r_i represents resource nodes, t_j denotes tenant identifiers, and l_k signifies access level tags[8] . The system divides into 128 tenant isolation domains, each containing an independent policy pool, authentication cache, and schema scope, supporting parallel registration of 412 dynamic contexts. Access boundaries between tenants are defined by the isolation function I , whose determination logic is as follows:

$$I(r_x, TID_y) = \begin{cases} 1, & \text{if} r_x \in R(TID_y) \wedge L(r_x) \le L_{\max}(TID_y) \\ 0, & \text{otherwise} \end{cases} \quad (3)$$

Where $R(TID_y)$ is the resource collection for tenant TID_y , $L(r_x)$ represents the resource level, and $L_{\max}(TID_y)$ denotes the tenant's maximum authorized access level. At the session layer, the system employs a session isolation vector $\vec{S} = [s_1, s_2, \ldots, s_n]$. Each $s_i \in \{0,1\}$ indicates whether the current tenant possesses a resource binding status for the corresponding session domain, dynamically updated via the following function:

$$s_i(t+1) = \delta_i \cdot \phi(TID_i, \tau_i) + (1 - \delta_i) \cdot s_i(t) \quad (4)$$

Where $\delta_i \in \{0,1\}$ s represent event triggers (such as token renewal or policy hot updates), ϕ denotes the Boolean function returned by the OPA policy, and τ_i indicates the current timestamp. As shown in Figure 2, Among them, green represents "Full" permission (both read and write), yellow represents "Read" permission (read-only), and orange represents "Write" permission (write only) under this grid model, the system achieves over 99.1% cross-tenant access blocking rate, maintains a cache hit rate above 92%, and supports approximately 13,000 access verification operations per second, effectively enabling secure management of large-scale concurrent access in multi-tenant environments[9] .

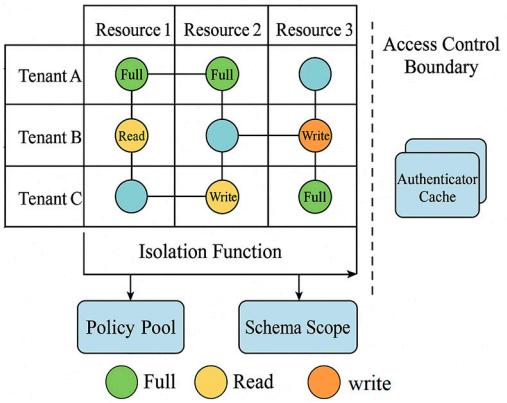

Figure 2. Permission Access Control Grid for Multi-Tenant Systems.

 doi:10.20944/preprints202512.1849.v1

5. Experimental Validation and Analysis

5.1. Experimental Design

The experiment deployed a complete API gateway system on a Kubernetes cluster. The environment comprised three 4-core, 32GB nodes, simulating concurrent access scenarios for 3,200 tenants. A dataset of 1.1 million mixed API call samples was constructed, with authorization interfaces accounting for 72.4%. Requests traversed Envoy Gateway and OPA policy chains, activating seven types of authentication factors and GraphQL field permission policies. All tokens were uniformly issued by the OAuth2.1 service. The system employed dual load injection engines (wrk and Locust) with a maximum QPS of 18,000. Four core metrics were collected: authentication latency, policy hits, access denials, and false positives. End-to-end monitoring captured 1.74 million event instances for subsequent performance and policy effectiveness analysis10 .

5.2. Performance Testing

During performance testing, a three-tiered call path set was constructed based on request origin, interface type, and token complexity for 1.1 million API sample calls. Using a distributed load generation tool, QPS was incrementally increased from 2,000 to 18,000 to observe system response behavior and policy execution efficiency across different stress levels. Four key metrics were collected: API entry processing latency, JWT issuance and verification time, average OPA policy evaluation time, and GraphQL field mapping overhead. In Figure 3, the latency distribution is directly influenced by the depth of policy evaluation paths and the complexity of authentication workflows. As the QPS increases, more requests traverse longer policy chains, especially under multi-condition matching rules, causing observable spikes in the P99 latency values. Furthermore, temporal variation across the 112-minute test duration reflects the cumulative effect of cache warm-up, token renewal cycles, and rule engine memory recycling. Thus, latency serves as a compound indicator jointly shaped by real-time QPS load, policy complexity, and system memory dynamics during sustained stress testing.To further quantify system bottlenecks and high-concurrency response capabilities.

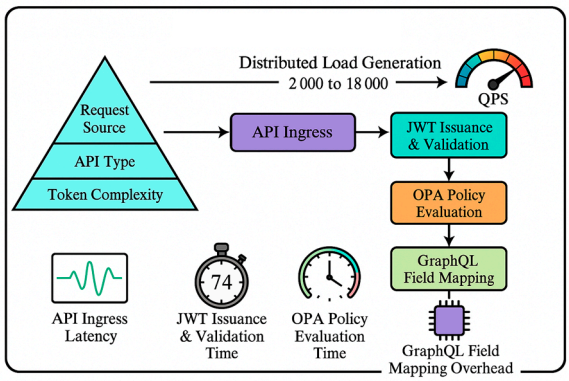

Figure 3. System Performance Testing Diagram.

Table 1 lists the average, 99th percentile, and peak time consumption for five major submodules across typical request chains, facilitating subsequent comparative analysis and optimization path identification.

Table 1. Performance Metrics Comparison of Key Security Gateway Modules.

Module Name	Average Duration (ms)	P99 Latency (ms)	Peak (ms)
API Entry Parsing	12.6	23.1	57.4
JWT Issuance and Verification	18.2	32.8	64.3

 doi:10.20944/preprints202512.1849.v1

OPA Policy Engine Evaluation	24.7	44.9	85.6
GraphQL Field Permission Matching	9.4	17.2	41.8
Policy Cache Hit and Fallback Mechanism	8.1	14.6	37.5

Among a total of 1.77 million sampled data points, the system achieved an average link response latency of 74.2ms at a QPS of 12,000. The JWT module accounted for 24.5% of the latency, while the OPA policy evaluation module averaged 24.7ms, representing 33.3% of the total latency. The P99 latency clustered within 85ms, with a maximum response peak of 114.3ms. GraphQL field permission matching averaged 9.4ms, accounting for 12.7%. Overall data indicates the system maintains a primary response window under 80ms even at high concurrency, demonstrating strong scalability and load balancing capabilities.

5.3. Security Assessment

To evaluate the gateway system's security protection capabilities in a multi-tenant, high-concurrency environment, five simulated attack paths were designed: privilege escalation, token forgery, field out-of-bounds access, silent scanning, and request frequency manipulation. Detection accuracy and false positive rates were modeled and analyzed for approximately 8.2% of high-risk events among 1.1 million samples. Using OPA policy paths as the evaluation benchmark, the system treated rule tree depth, expression complexity, and tenant context injection as primary variables. It established 87 detection breakpoints along the policy decision chain, generating 25,000 synthetic attack requests covering 3,200 virtual tenant dimensions. To quantify the identification stability of each policy under real-world conditions and the impact of policy chain redundancy on detection efficiency, Figure 4 illustrates the distribution trend of detection accuracy versus path complexity across different attack scenarios.

Path complexity "refers to the cumulative depth of the OPA rule tree traversed in a single policy evaluation process, measured in" levels ". Each layer represents a conditional judgment, logical operation, or Rego module reference. For example, if the path contains 7 nested conditional judgments and 2 external rule references, the path complexity is defined as 9 layers. This indicator provides a unified measure of policy path length, which helps to measure the computational depth of policy execution and has a significant impact on evaluation latency and detection accuracy.

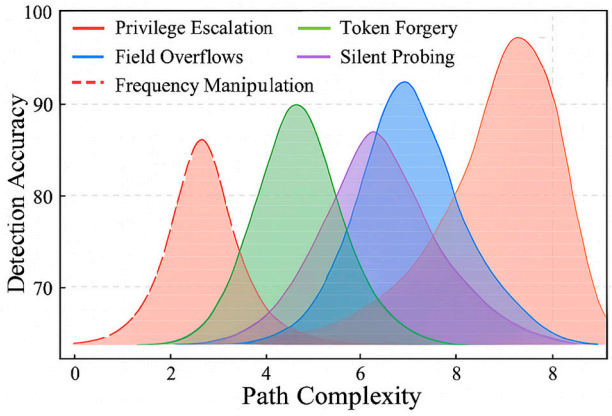

Figure 4. Density Distribution of Detection Accuracy Across Path Complexities.

As shown in Figure 4, when the policy path depth is controlled within 9 layers, detection accuracy fluctuates within ±4.3%. Table 2 lists the detection hit rate and false positive rate evaluation data for five attack types.

Table 2. Security Detection Performance Statistics for Typical Attack Types in Multi-tenant Scenarios.

Attack Type	Number of Detection Rules	Average Hit Rate (%)

 doi:10.20944/preprints202512.1849.v1

Privilege Escalation Attacks	74	92.8
Token Forgery Attacks	63	89.5
GraphQL Field Overrange	81	94.1
Silent Probe Behavior	45	86.3
High-frequency access manipulation	56	91.7

Table 2 presents the security detection performance statistics for typical attack types in multi-tenant scenarios, highlighting the number of detection rules and the average hit rate for each attack type.

The highest average hit rate is observed for GraphQL Field Overrange attacks (94.1%), which indicates that detection rules specifically tailored to prevent excessive field queries in GraphQL systems are highly effective. This suggests that the implemented rules are well-suited to address common vulnerabilities in GraphQL-based systems. On the other hand, Privilege Escalation Attacks show a strong detection performance with a hit rate of 92.8%, reflecting the robustness of the detection rules for unauthorized access escalation attempts. High-frequency Access Manipulation follows closely with an average hit rate of 91.7%, showing that detection mechanisms are effective in identifying rapid, repetitive access patterns, which are indicative of attacks such as DoS or DDoS.

However, Token Forgery Attacks and Silent Probe Behavior have slightly lower detection hit rates at 89.5% and 86.3%, respectively, indicating that there might be room for improvement in detecting these subtle, often evading attack techniques. These attack types typically involve forged tokens and stealthy probing, which may require more advanced rule sets or machine learning models for enhanced detection capabilities.

Overall, the data suggests that while detection rules are effective across most attack types, there is potential for refinement, particularly in handling more sophisticated attack strategies.

6. Conclusion

The API security gateway and access control mechanism for multi-tenant full-stack architectures achieves granular tenant identity management and dynamic resource authorization in high-concurrency environments by integrating a zero-trust model, attribute-driven policy engine, and GraphQL field-aware controls. This ensures consistent and auditable data isolation boundaries. The constructed system demonstrates superior performance in authentication latency, security detection capabilities, and policy execution efficiency, validating the model's engineering adaptability under actual deployment conditions. However, the current policy engine's ability to express complex contextual semantics remains constrained by the syntax structure of the Rego language, making it challenging to accurately model certain dynamic behaviors. Future work may focus on introducing semantic graphs to enhance policy logic expression capabilities. Integrating federated learning mechanisms to build cross-system policy coordination models could further elevate the system's security governance capabilities and adaptive policy scheduling levels in large-scale heterogeneous tenant environments.

References

1. Ali S, Nasim F, Haider K. Secure middleware model for public restful apis[J]. Al-Aasar, 2025, 2(1): 50-62.
2. Sitorus R S, Hutagaol B J, Simanjuntak D M. Capability-Based API Gateway Technology Selection Analysis for Banking Cybersecurity Solution Using AHP Method[J]. Sinkron: jurnal dan penelitian teknik informatika, 2025, 9(1): 338-347.
3. Zhao C. Api common security threats and security protection strategies[J]. Frontiers in Computing and Intelligent Systems, 2024, 10(2): 29-33.
4. Aydemir F, Başçiftçi F. Performance and Availability Analysis of API Design Techniques for API Gateways[J]. Arabian Journal for Science and Engineering, 2025, 50(15): 11485-11498.
5. Ahuja D. Securing Container Isolation in Multi-Tenant Environments[J]. Journal of Computer Science and Technology Studies, 2025, 7(10): 225-232.

 doi:10.20944/preprints202512.1849.v1

6. Chen, X., Huang, Y., Jessney, B., Sangha, J., Gu, S., Schönlieb, C. B., ... & Roberts, M. (2025). Review and recommendations for using artificial intelligence in intracoronary optical coherence tomography analysis. European Heart Journal-Digital Health, ztaf053.
7. De Simone L, Di Mauro M, Natella R, et al. A latency-driven availability assessment for multi-tenant service chains[J]. IEEE transactions on services computing, 2022, 16(2): 815-829.
8. Chirivella-Perez E, Salva-Garcia P, Sanchez-Navarro I, et al. E2E network slice management framework for 5G multi-tenant networks[J]. Journal of Communications and Networks, 2023, 25(3): 392-404.
9. Eboseremen B O, Ogedengbe A O, Obuse E, et al. Secure data integration in multi-tenant cloud environments: Architecture for financial services providers[J]. Journal of Frontiers in Multidisciplinary Research, 2022, 3(1): 579-592.
10. Wang Y, Yang J, Wang Z. Multi-tenant in-memory key-value cache partitioning using efficient random sampling-based LRU model[J]. IEEE Transactions on Cloud Computing, 2023, 11(4): 3601-3618.