
MITIGATING THE TOP 5 OWASP API SECURITY RISKS USING GATEWAY PATTERNS

Liviu MOISEI, Liviu MOCANU*

Software Engineering and Automation, SI-241M, Faculty of Computers, Informatics and Microelectronics,
Technical University of Moldova, Chișinău, Republic of Moldova

*Corresponding Author: Liviu Mocanu, liviu.mocanu@isa.utm.md

Scientific Supervisor: Dinu ȚURCANU, Associate Professor, Ph.D., Technical University of Moldova

Abstract. *This paper investigates how API gateway patterns can effectively mitigate five of the most critical risks identified in the OWASP API Security Top 10 (2023 edition). As APIs form the backbone of modern microservices and cloud-native systems, ensuring their security is vital. The research outlines a layered defense strategy, focusing on gateway-enforced controls such as rate limiting, schema validation, authentication, authorization, and access policy enforcement. The methodology involves a structured analysis of security literature, case studies, and industry documentation to evaluate how API gateways address real-world vulnerabilities – such as Broken Object Level Authorization and Unrestricted Resource Consumption – across both perimeter and contextual security. This study contributes by bridging theoretical security models with real-world gateway implementations, illustrating their effectiveness through case analysis and highlighting where such protections fall short. The research finds that gateway patterns are most effective when complemented by backend validation and secure development practices. This paper proposes a practical security framework for developers and system architects seeking to fortify API-driven infrastructures.*

Keywords: *rate limiting, access control, payload validation, microservices, web security.*

Introduction

APIs (Application Programming Interfaces) have become foundational to digital transformation, enabling organizations to build scalable, modular, and interoperable systems. With the growing adoption of microservices and cloud-native architectures, the number of publicly and privately exposed APIs has surged, bringing with it a corresponding increase in API-specific security threats.

Recognizing this, the Open Web Application Security Project (OWASP) introduced the API Security Top 10, a curated list of the most critical security issues affecting APIs. These include familiar concerns like broken authentication and injection, as well as API-specific challenges such as mass assignment and insufficient logging. One promising avenue for mitigating these risks lies in the API gateway – a centralized component that manages, routes, and secures API traffic. Beyond basic request routing, modern API gateways offer a suite of security-enhancing features such as rate limiting, access control, payload validation, and threat detection [1].

This paper explores how various API gateway patterns can be leveraged to mitigate five of the most critical and relevant OWASP API Security Top 10 threats. By synthesizing research findings and industry practices, it aims to provide a practical and strategic framework for API security, focusing on the proactive role that gateway-level controls can play in a layered defense approach.

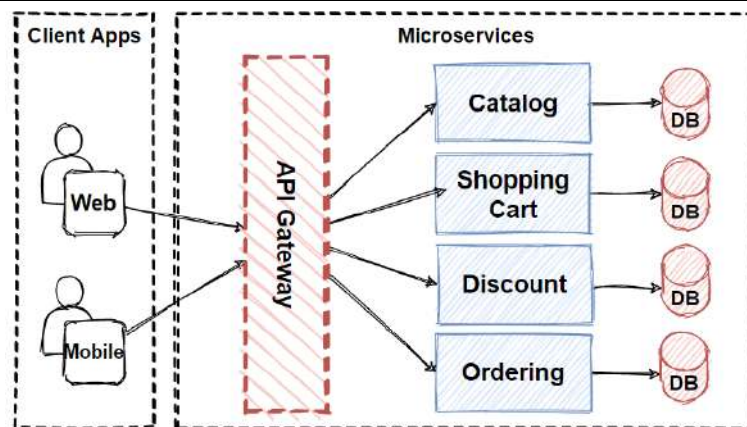


Figure 1. Gateway as an enforcement layer between client applications and microservices

Source: "API Gateway Pattern" by M. Ozkaya, 2021, Medium

Overview of OWASP API Security

The OWASP API Security Top 10 is a project developed by the Open Web Application Security Project to identify and prioritize the most critical API-related security threats. These threats are based on real-world incidents and input from the industry. In this section, we present top 5 risks that are effectively mitigated by API gateway patterns and analyze how well the pattern can help mitigate it, referencing best practices.

API1:2023 – Broken Object Level Authorization

Broken Object Level Authorization (BOLA) is one of the most common and dangerous API vulnerabilities [2]. It occurs when an API exposes endpoints that accept user-supplied object identifiers (such as account IDs or document keys) but fails to enforce proper authorization checks on those objects. This allows attackers to manipulate IDs and gain access to resources they don't own, leading to data breaches or unauthorized actions.

API gateways can play a crucial role in defending against BOLA by acting as a first line of defense:

1. API gateways can enforce strict schemas for incoming requests, ensuring only expected formats and fields are processed. Tools like *Kong* and *AWS API Gateway* support JSON schema validation, preventing attackers from submitting malformed or unauthorized object IDs.
2. Gateways can integrate with external policy engines (e.g., *OPA*) or IAM systems to enforce object-level access policies. For example, the gateway can call out to a policy service that verifies if a user is allowed to access a given object before forwarding the request. Some platforms allow declarative access control using RBAC or ABAC models.
3. Modern gateways can inspect JWT claims or user context and apply route-level policies accordingly. For example, if a user tries to access `/orders/123`, the gateway can verify if `user_id` in the JWT matches the owner of order 123. This enables fine-grained, attribute-aware control.

Whilst gateways are effective at enforcing high-level policies, they may lack the internal business context required for true object-level access validation. Therefore, a layered approach is recommended where gateways filter obvious violations, while backend services perform deeper authorization checks.

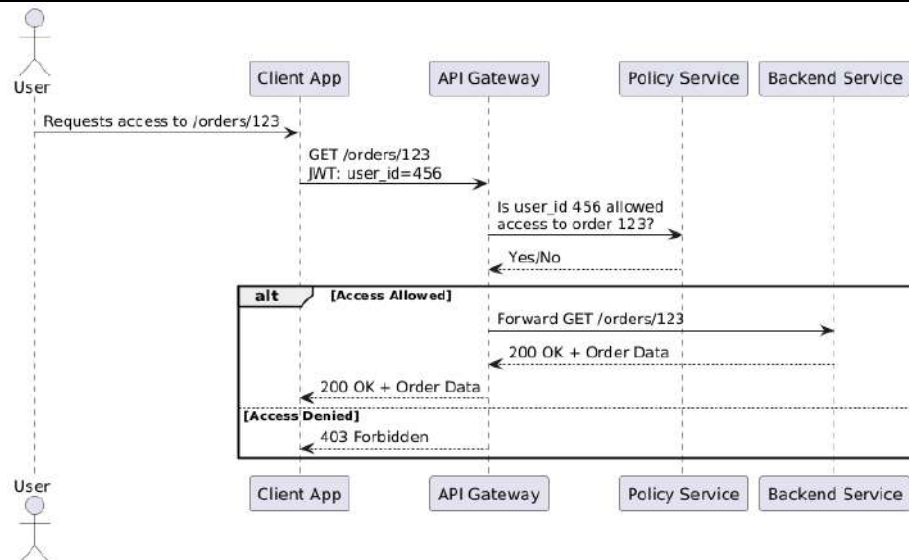


Figure 2. Gateway-mediated mitigation of Broken Object Level Authorization through policy evaluation

API2:2023 – Broken Authentication

Broken Authentication refers to failures in correctly implementing authentication mechanisms, which can lead to unauthorized access [3]. This includes insecure token storage, lack of multi-factor authentication (MFA), weak password enforcement, token replay vulnerabilities, and inconsistent session handling. APIs relying solely on session tokens or poorly implemented authentication flows are vulnerable to impersonation and account takeover attacks.

1. API gateways provide centralized enforcement of authentication using modern protocols like *OAuth2*, *OpenID Connect*, and *SAML*. This ensures uniform application of authentication policies across microservices and minimizes inconsistencies.
2. Gateways such as *Apigee*, *AWS API Gateway*, and *Kong* can inspect and validate JSON Web Tokens (JWTs) by verifying their signature, expiration, and issuer. This prevents unauthorized access using expired, forged, or tampered tokens.
3. Gateways can integrate with identity providers (IdPs) to enforce MFA requirements before forwarding requests to sensitive routes. This is particularly important for operations involving financial transactions or access to personal data.
4. Replay Attack Prevention. Advanced gateway configurations allow detection of token reuse and replay attacks by checking request timestamps, nonces, or sequence numbers – often in conjunction with backend session management.

Gateways can enforce authentication on the perimeter, but internal services must still handle fine-grained access control and session context securely. Additionally, gateway-level authentication should be complemented with transport-layer security (TLS), secure token storage on clients, and robust user verification mechanisms.

API3:2023 – Broken Object Property Level Authorization

Broken Object Property Level Authorization occurs when users are authorized to access an object but can manipulate fields within that object that they should not be able to read or modify. This flaw is particularly common in APIs exposing complex JSON payloads where sensitive attributes like internal roles, permissions, pricing, or user flags – are insufficiently protected [4].

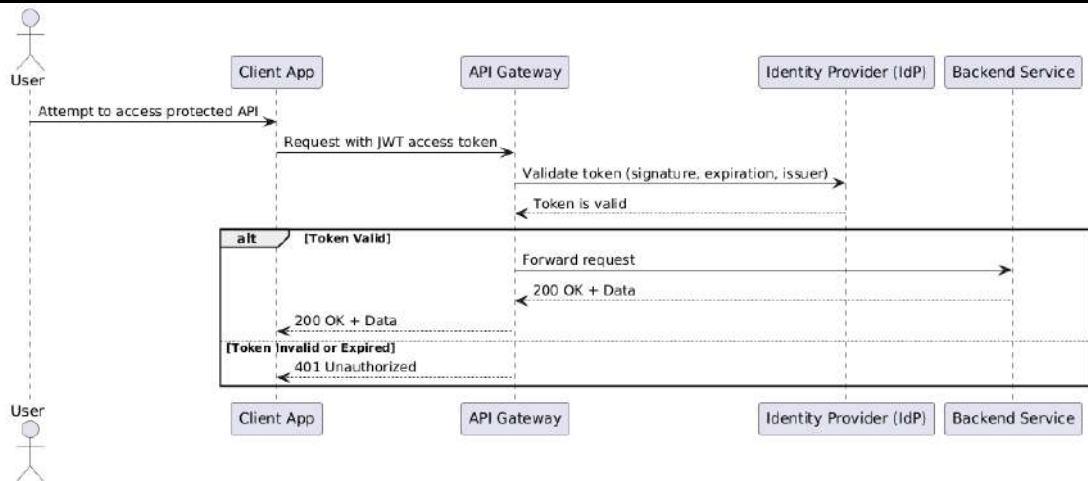


Figure 3. Authentication enforcement via Gateway using token validation and IdPs integration

1. Modern API gateways (e.g., *Apigee*, *Kong*) offer plugins and policies that dynamically modify or redact specific fields from responses based on the requesting user's identity. This reduces the likelihood of unintentionally exposing sensitive fields to unauthorized users.
2. Advanced gateways with policy-as-code support (like *OPA* or *WSO2*) allow defining granular access policies at the property level. For instance, admins might retrieve full user profiles, while customers receive filtered views excluding internal-only fields.
3. API schema enforcement can be enhanced with conditional logic, applying different schemas based on the role or group of the requester. Gateways can bind a particular schema to a JWT claim or API key scope, filtering both inbound and outbound data structures.

Even though gateways are effective for response shaping, they do not inherently understand deep domain-specific logic. Field-level policies must complement internal backend logic, and care must be taken not to rely solely on gateway masking. Additionally, runtime transformations may introduce performance costs at scale.

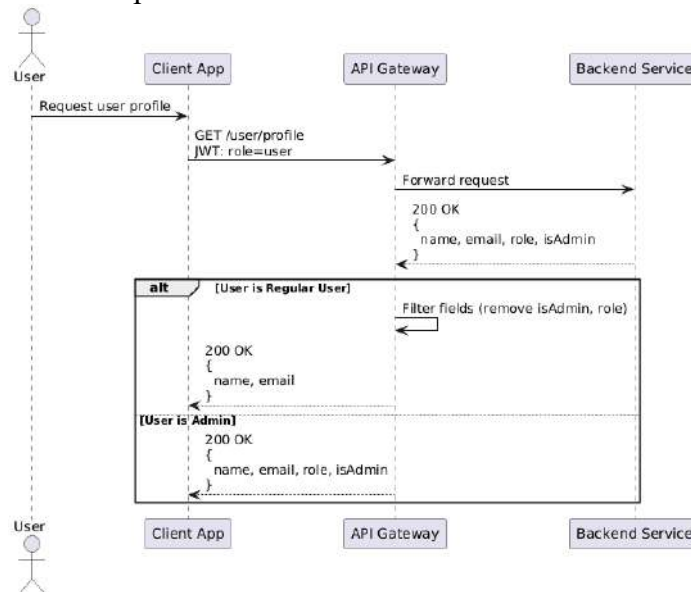


Figure 4. Gateway-mediated mitigation of Broken Object Property Level Authorization through policy evaluation

API4:2023 – Unrestricted Resource Consumption

This vulnerability arises when APIs allow clients to consume excessive resources such as CPU, memory, bandwidth, or database connections – without proper limits [5]. Unlike traditional DoS attacks that target infrastructure, API-level resource exhaustion can be subtle and exploit logical abuse like excessive pagination, large file uploads, or frequent polling.

1. One of the most effective defenses against resource consumption is enforcing request limits. API gateways like *Kong*, *Apigee*, and *AWS API Gateway* support IP-based, user-based, or key-based rate limiting and quota enforcement. These policies throttle abusive clients before requests hit the backend.
2. Gateways can enforce maximum payload sizes for both requests and responses. This helps prevent attackers from sending massive JSON payloads or uploading large files that could overwhelm the system.
3. Some gateways allow validation of query parameters like *limit*, *page*, or *offset*. You can define constraints (e.g., *limit* $\leq$ 100) to prevent inefficient queries or data scraping attacks.
4. Certain gateway platforms support concurrency control policies that restrict how many simultaneous requests a client can make. This prevents flooding scenarios in multi-threaded or async environments.
5. By enabling response caching at the gateway level for non-sensitive GET requests, platforms can offload frequently accessed data and reduce backend strain – particularly useful for APIs that return static or slow-changing data.

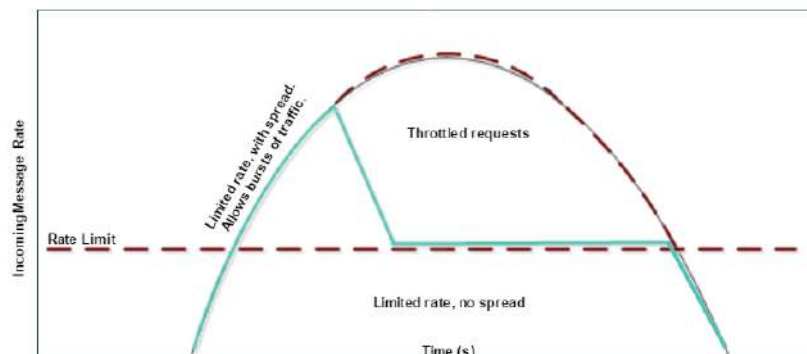


Figure 5. Rate limiting enforced by the Gateway showing throttled requests exceeding the limit

Source: “Apply Rate Limit Assertion” by Broadcom, 2024

Whilst API gateways can mitigate many forms of overuse, they cannot detect business logic abuse without context. For instance, repeated use of a costly but valid function (e.g., dynamic PDF generation) may bypass technical rate limits but still cause load issues. Backend services should monitor usage patterns and integrate behavioural analytics or usage thresholds to complement gateway-level protections.

API5:2023 – Broken Function Level Authorization

This vulnerability occurs when an API fails to restrict access to sensitive functions or operations based on user roles or privileges [6]. While object-level authorization (API1) protects specific data, function-level authorization ensures that only appropriate users can invoke specific operations, such as administrative actions or data deletions

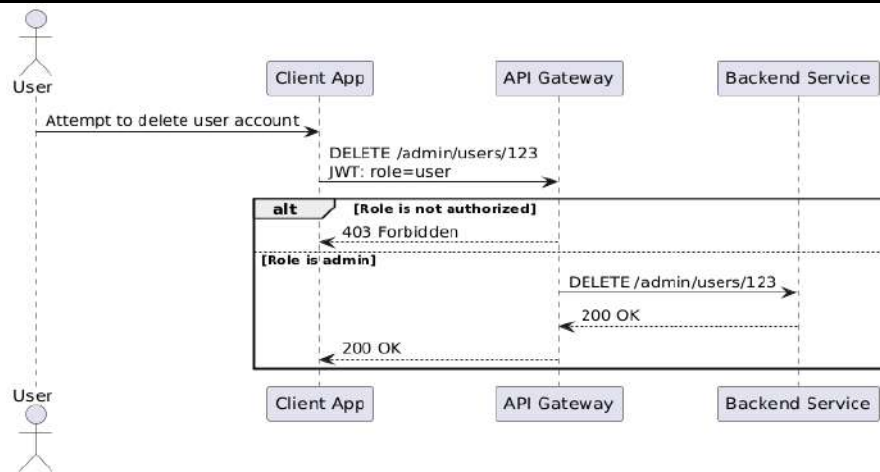


Figure 6. Role-based route and method authorization enforced by API Gateway to block unauthorized access

1. API gateways allow defining fine-grained access policies at the route or method level. For instance, requests to `/admin/*` can be restricted to users whose JWT includes a `role=admin` claim. This prevents unauthorized access before the request reaches the backend.
2. Gateways can enforce maximum payload sizes for both requests and responses. This helps prevent attackers from sending massive JSON payloads or uploading large files that could overwhelm the system.
3. Some gateways allow validation of query parameters like `limit`, `page`, or `offset`. You can define constraints (e.g., `limit <= 100`) to prevent inefficient queries or data scraping attacks.

Despite the fact that gateways can prevent unauthorized function calls on the perimeter, internal services must still validate user roles and permissions to prevent privilege escalation or bypass through internal APIs or service chaining. Moreover, defining and maintaining access rules for every function or route at the gateway level can become complex at scale.

Conclusions

API gateways play a crucial role in mitigating the top five OWASP API Security risks by enabling centralized enforcement of access control, rate limiting, schema validation, and token-based authentication. As APIs become central to modern applications, these gateway patterns provide consistent, scalable protection across services.

The analysis reveals that API gateways are particularly effective for addressing perimeter security concerns like broken authentication (API2) and unrestricted resource consumption (API4), where centralized policy enforcement can block malicious requests before they reach backend services. For object-level authorization issues (API1 and API3), gateways provide a valuable first line of defense but must be complemented by service-level validation that understands the business context of data relationships. Function-level authorization (API5) benefits significantly from gateway-enforced route and method restrictions tied to user roles and scopes.

While gateways offer strong perimeter defenses, they are not sufficient alone. True resilience requires layering gateway protections with secure coding practices, thorough backend validation, and proper identity management. This multi-layered strategy is essential because certain vulnerabilities, particularly those involving complex business logic or data relationships, may bypass gateway-level controls.

Future research should explore how emerging technologies like artificial intelligence and machine learning can enhance API gateway security through anomaly detection and adaptive policy enforcement [7–9]. Additionally, as API ecosystems grow more complex, further investigation into scalable, automatable security patterns will be essential for maintaining effective protection across thousands of endpoints and services.

References

1. JOSEPH, W.; JONES, A. Modern API Security Frameworks for Enterprise Systems. 2025.
2. DIAZ-ROJAS, J.A.; OCHARAN-HERNANDEZ, J.O.; PEREZ-ARRIAGA, J.C.; LIMON, X. Web API Security Vulnerabilities and Mitigation Mechanisms: A Systematic Mapping Study. In *2021 9th International Conference in Software Engineering Research and Innovation (CONISOFT)*; IEEE: San Diego, CA, USA, 2021; pp 207–218. <https://doi.org/10.1109/CONISOFT52520.2021.00036>.
3. ADEBAYO, H. Digital Banking and API Security: Best Practices for Secure Financial Transactions. 2025.
4. ZHAO, C. API Common Security Threats and Security Protection Strategies. *FCIS* 2024, 10 (2), pp. 29–33. <https://doi.org/10.54097/k5djs164>.
5. DIXIT, P. How to Secure APIs to Defend Against Emerging Cyber Threats to Digital Web Assets. *IJCTT* 2024, 72 (9), pp. 157–164. <https://doi.org/10.14445/22312803/IJCTT-V72I9P125>.
6. ARAVINDA A KUMAR; DIVYA TL. Security Measures Implemented in RESTful API Development. *Open Access Res. J. Eng. Technol.* 2024, 7 (1), pp. 105–112. <https://doi.org/10.53022/oarjet.2024.7.1.0042>.
7. PAUL, J. Building a Robust API Security Framework with Machine Learning. 2024.
8. PAUL, J. Integrating Machine Learning with API Gateway Security Solutions. 2024.
9. KAUL, D.; KHURANA, R. AI to Detect and Mitigate Security Vulnerabilities in APIs: Encryption, Authentication, and Anomaly Detection in Enterprise-Level Distributed Systems. 2021.