International Journal Software Engineering and Computer Science (IJSECS)
6 (1), 2026, 150-158
Published Online April 2026 in IJSECS (http://www.journal.lembagakita.org/index.php/ijsecs)
P-ISSN: 2776-4869, E-ISSN: 2776-3242. DOI: https://doi.org/10.35870/ijsecs.v6i1.6692.

RESEARCH ARTICLE Open Access

Layered Security Model for JWT-Based Authentication and Authorization in Golang Echo REST APIs

Giovanni Sebastian Ekayuda [1*], **Suprihadi** [2]

[1*,2] Department of Informatics Engineering, Faculty of Information Technology, Universitas Kristen Satya Wacana, Salatiga City, Central Java Province, Indonesia.

**Corresponding author*: giovan0909@gmail.com.

Received: February 27, 2026; Accepted: April 1, 2026; Published: April 10, 2026.

Abstract: Microservices architecture improves scalability and flexibility in modern distributed systems, yet it simultaneously widens the attack surface through decentralized service communication. Many existing implementations rely primarily on token validation without structured service-level authorization enforcement, leaving systems exposed to privilege escalation vulnerabilities. This study designed and evaluated a layered security model for a RESTful Application Programming Interface built with the Go Echo framework. The proposed approach combines JSON Web Token authentication using asymmetric cryptography with a token versioning mechanism, and pairs Role-Based Access Control with Attribute-Based Access Control within a sequential middleware pipeline. The methodology covered system architecture design, middleware implementation, structured security testing, and response time analysis. All simulated unauthorized access scenarios — including vertical and horizontal privilege escalation attempts — were successfully blocked. The average response time under the fully secured configuration measured 24.9 ms, indicating that the overhead introduced by the layered middleware remains practically acceptable. These findings suggest that separating authentication and authorization at the service level produces measurable security gains without meaningfully degrading system performance in microservices-based REST API applications.

Keywords: Microservices Security; JSON Web Token; Role-Based Access Control; Attribute-Based Access Control; Golang Echo.

1. Introduction

The adoption of microservices architecture has significantly transformed the development of distributed systems within software engineering. By decomposing monolithic applications into smaller, independent services that interact through RESTful Application Programming Interfaces (APIs), organizations can achieve greater scalability, flexibility, and maintainability. This architectural shift, however, also introduces increased security complexity — each service interaction creates additional potential vulnerability points. In distributed environments, authentication and authorization mechanisms must therefore be consistently enforced at the service level to maintain system integrity and prevent unauthorized access. Prior research has demonstrated that microservice architectures require more structured and robust security controls than traditional monolithic systems (Berardi *et al.*, 2022; de Almeida & Canedo, 2022).

Token-based authentication mechanisms such as JSON Web Tokens (JWTs) are widely adopted in modern REST API implementations. Many systems rely on JWT signature verification and token expiration checks to authenticate users. Several studies, however, highlight that authentication alone does not guarantee secure access when authorization policies are weak or inconsistently implemented. Aldea and Bocu (2025) explain that valid tokens can still be exploited when service-level authorization controls are insufficient, while Bucko *et al*. (2023) similarly demonstrate that attackers may abuse legitimate tokens to access unauthorized resources if internal validation mechanisms are poorly structured. Although JWT provides advantages such as stateless authentication and high scalability, relying solely on token validation is insufficient to prevent vertical and horizontal privilege escalation (Gbenle *et al*., 2022; Venčkauskas *et al*., 2023). As distributed systems grow in complexity, secure API implementations require stronger integration between authentication and authorization mechanisms.

Access control models play an essential role in implementing authorization policies. RBAC is widely used to restrict access based on predefined user roles, yet traditional approaches often lack contextual awareness when handling dynamic authorization requirements in microservices environments (Al-Wadi & Maaita, 2023). ABAC, by contrast, enables more fine-grained authorization decisions by evaluating attributes such as user identity, resource ownership, and request context (Kovtun & Laponina, 2025). Despite the distinct advantages each model offers, both are frequently implemented separately from JWT-based authentication processes. Lee and Jeon (2024) further note that the choice of JWT signature algorithms — particularly asymmetric schemes such as RS256 — affects both security strength and computational performance. Taken together, these findings indicate that effective API protection requires not only secure token validation but also structured authorization enforcement integrated directly into the application architecture. Several broader approaches have been proposed to address distributed system security, including OAuth-based authorization, API gateway protection, and Zero Trust architecture (Phanireddy, 2023; Ramadhan Purba *et al*., 2025; Gupta, 2025). Escaleira *et al*. (2025) further emphasize that security enforcement within individual services is equally important to prevent internal misuse of valid credentials. Despite these advancements, existing research often concentrates on gateway-level security configurations or protocol-based authentication frameworks, with only limited attention given to modular, layered middleware security architectures within the Golang ecosystem that explicitly separate authentication and authorization at the service level.

To address this gap, this study proposes a layered security model implemented in a REST API developed with the Golang Echo framework. The proposed architecture separates authentication and authorization into sequential middleware components, enabling systematic validation of each incoming request. JWT authentication using asymmetric cryptography is combined with a token versioning mechanism to support secure token revocation, while authorization enforcement is carried out through the integration of RBAC and ABAC within a structured middleware pipeline. By applying multiple validation layers, the system ensures that requests pass through sequential security checks before accessing protected resources. This study was conducted with the following objectives:

1) Design a layered middleware security architecture that separates authentication and authorization processes in REST APIs built with the Golang Echo framework.
2) Implement secure JWT validation using asymmetric cryptography with a token versioning mechanism for token revocation.
3) Integrate RBAC and ABAC within a sequential middleware pipeline to prevent privilege escalation.
4) Evaluate the security effectiveness and performance impact of the proposed layered security model through structured testing and response time analysis.

Based on the identified research gap, this study addresses the following research question: How effective is a layered middleware security model in improving authentication and authorization enforcement while maintaining acceptable performance in Golang-based REST APIs? The remainder of this paper is structured as follows. Section 2 reviews related work on microservices security, token-based authentication, and access control models. Section 3 describes the proposed methodology and system architecture. Section 4 presents implementation results and performance evaluation. Section 5 concludes the paper and outlines recommendations for future research.

2. Related Work

2.1 Microservices Security Challenges

Recent research on microservice security has grown significantly, particularly in the areas of authentication and authorization. Berardi *et al*. (2022) conducted a systematic review and identified authentication inconsistency and decentralized access control as key vulnerabilities in distributed systems. De Almeida and Canedo (2022) similarly argued that consistent application of authentication and authorization

mechanisms across all services is essential to prevent unauthorized access. Hutasuhut *et al*. (2024) observed that security gaps frequently arise when individual microservices employ different validation strategies, producing uneven protection levels across the system. Collectively, these findings suggest that the distributed structure of microservices demands a structured, layered security approach that goes beyond what traditional monolithic protection methods can offer.

2.2 JWT-Based Authentication Mechanisms

JSON Web Token (JWT) has become a widely used authentication method for securing RESTful APIs due to its stateless, scalable nature. Dalimunthe *et al*. (2023) applied JWT with HMAC-SHA512 for session management and showed that it improves authentication efficiency in RESTful systems, though their work centered on authentication validation rather than comprehensive service-level authorization controls. Lee and Jeon (2024) examined how various JWT signature algorithms affect performance, finding that asymmetric schemes such as RS256 provide stronger security at the cost of higher computation compared to symmetric schemes. Bucko *et al*. (2023) proposed adding behavioral history into authorization processes, arguing that signature validation alone does not fully ensure secure resource access. These studies collectively point to JWT's advantages while also exposing its limitations when authorization is not embedded within dedicated middleware layers. Gunawan (2024) further noted that JWT-based authentication mechanisms require complementary authorization controls to ensure secure API interactions.

2.3 Access Control Models: RBAC and ABAC

Access control models are central to defining authorization rules in distributed systems. Al-Wadi and Maaita (2023) introduced a performance-oriented RBAC model designed for microservices that improved access management efficiency, while Sänger and Abeck (2023) examined user authorization strategies in microservice applications and stressed the need for fine-grained access control to prevent privilege escalation. ABAC enables flexible authorization by evaluating contextual attributes such as user identity, resource ownership, and request parameters (Hu *et al*., 2015). Kovtun and Laponina (2025) extended this by integrating ABAC with mutual Transport Layer Security to improve contextual authorization across distributed systems. Although RBAC simplifies role management and ABAC supports contextual decision-making, most implementations treat these models separately from JWT authentication workflows — meaning authentication and authorization remain loosely connected rather than systematically integrated into a cohesive validation process.

2.4 OAuth, Zero Trust, and Gateway-Level Security

Recent studies have also emphasized the importance of structured validation mechanisms in modern distributed applications (Rahman *et al*., 2025). Gbenle *et al*. (2022) examined the integration of OAuth2 and JWT within API gateways and recommended implementing structured validation processes for distributed systems. Phanireddy (2023) proposed a threat modeling framework to secure RESTful APIs in microservice architectures, while Ramadhan Purba *et al*. (2025) showed that OAuth2-based authorization mechanisms could strengthen endpoint protection. Gupta (2025) discussed Zero Trust architecture as a contemporary model requiring ongoing verification at each interaction point in cloud environments, and Escaleira *et al*. (2025) stressed the importance of service-level security enforcement over reliance on perimeter defenses alone. While these strategies advanced distributed authentication, most research concentrated on gateway security and protocol configuration rather than on structured middleware orchestration within application services.

2.5 Security Standards and Research Positioning

Kurniawan *et al*. (2025) examined the use of role-based access control aligned with the OWASP Application Security Verification Standard (OWASP Foundation, 2023), underscoring the value of systematic validation against established controls. Standards such as ISO/IEC 27001:2022 and ISO/IEC 27002:2022 (International Organization for Standardization, 2022a, 2022b) provide detailed guidance on deploying layered security measures and access management policies in distributed systems, with particular attention to separation of duties, defense-in-depth approaches, and structured authentication protocols. Recent research has also assessed the performance implications of middleware-based validation mechanisms in microservice architectures, indicating that layered middleware can strengthen security enforcement while maintaining acceptable performance overhead (Fauziah *et al*., 2026). Although prior studies have examined authentication frameworks, access control models, and gateway-level security mechanisms, most implementations address authentication or authorization independently. Research on the integration of JWT-based authentication with layered middleware authorization mechanisms in Go-based microservice environments remains limited. This study therefore proposes a structured layered security architecture that sequentially integrates JWT authentication, token version validation, RBAC, and ABAC enforcement at the service level — an approach

aimed at strengthening internal service security while maintaining practical performance in REST API implementations.

3. Methodology

3.1 System Architecture

This study developed a layered security model for a REST API implemented using the Go Echo framework. The system architecture consists of four main components: client applications, an API server, security middleware layers, and a relational database. Client requests are transmitted through HTTP and processed by the Echo router before reaching protected endpoints. The API server hosts the application's business logic and integrates multiple middleware components that validate incoming requests, while the database stores user credentials, role assignments, resource ownership information, and token version records used for token revocation management. The experimental implementation consisted of eight REST API endpoints, including authentication, administrative, and user resource endpoints. These endpoints were used to simulate different authorization scenarios in order to evaluate how the proposed layered security model handles legitimate and unauthorized access requests.

3.2 Layered Middleware Security Model

The proposed architecture implements a layered middleware validation pipeline inspired by security filter chain concepts commonly used in enterprise frameworks. Each incoming HTTP request is processed sequentially through several validation stages. The architecture of the proposed layered security model is shown in Figure 1.

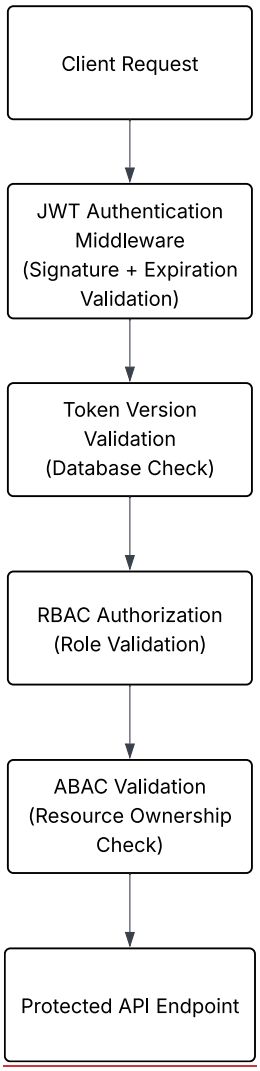

Figure 1. Layered Middleware Security Validation Flow

The authentication middleware extracts the JSON Web Token from the Authorization header and verifies the token signature using the RS256 asymmetric algorithm; requests are immediately rejected if the signature is

invalid or the token has expired. The token version validation middleware then compares the token version stored in the JWT payload with the version value stored in the database — a mechanism that enables secure token revocation without maintaining server-side session states. Following this, the Role-Based Access Control (RBAC) middleware verifies whether the authenticated user has the appropriate role required to access the requested endpoint, after which the Attribute-Based Access Control (ABAC) middleware performs contextual authorization checks by evaluating attributes such as resource ownership and request parameters. Only requests that successfully pass all validation layers are allowed to reach protected application handlers. This layered validation structure ensures that authentication and authorization processes remain logically separated while maintaining consistent security enforcement across service endpoints.

3.3 JWT Authentication Implementation

JWT authentication was implemented using asymmetric cryptography (RS256) to enhance security compared to symmetric-key approaches. During the login process, tokens are generated and signed using a private key, while verification during request validation is performed using a corresponding public key. The JWT payload contains several claims required for authorization validation, including user ID, role information, token version, and expiration timestamp. Token versioning enables secure token revocation: when user privileges change or accounts are invalidated, the system increments the token version stored in the database, and any previously issued tokens with older version numbers are automatically rejected during middleware validation. This stateless authentication mechanism allows scalable request validation in distributed systems while eliminating the need for server-side session storage.

3.4 RBAC and ABAC Authorization Mechanisms

The authorization layer integrates Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) to ensure comprehensive enforcement of authorization. RBAC manages endpoint access based on predefined roles such as administrator, lecturer, or student, with each protected endpoint associated with specific role requirements validated during middleware execution. ABAC extends this mechanism by evaluating contextual attributes associated with the request and the resource being accessed — for example, resource ownership validation ensures that users can only access resources linked to their own identity unless elevated privileges are granted. The ABAC decision logic used in this study can be expressed as follows:

$$\text{Access} = \text{allow if } (\text{user.id} = \text{resource.owner\_id}) \vee (\text{user.role} = \text{admin})$$

This rule prevents horizontal privilege escalation by ensuring that users cannot access other users' resources. By combining RBAC and ABAC within sequential middleware layers, the system enforces both role-based restrictions and contextual authorization decisions, strengthening service-level security enforcement.

3.5 Development Environment and Tools

The prototype system was implemented using Go version 1.21 and the Echo web framework. JWT processing was handled using a Go JWT library supporting RS256 signature verification, and MySQL was used as the primary database to store user credentials, roles, and token version records. The development process followed a structured workflow consisting of requirements analysis, system design, implementation, testing, and evaluation, with version control managed using Git to ensure reproducibility and traceability of code changes. All experiments were conducted in a controlled local environment to minimize interference from external network latency and background system processes, with the system deployed on a machine equipped with an Intel Core i5 processor, 16 GB RAM, and a Linux operating system.

3.6 Experimental Setup and Evaluation Design

To evaluate the effectiveness of the proposed layered security model, structured testing scenarios were designed. Security evaluation included authentication validation, role-based authorization enforcement, attribute-based resource ownership checks, and token revocation verification, with unauthorized access scenarios simulating both vertical and horizontal privilege escalation attempts. The performance evaluation measured the response-time overhead introduced by sequential middleware validation across three system configurations:

1) System without security middleware
2) System with JWT authentication middleware only
3) System with full layered middleware architecture including JWT validation, RBAC, and ABAC enforcement

Each configuration was executed 500 times to ensure statistical consistency of the experimental results. Evaluation metrics included authentication correctness, authorization enforcement accuracy, and average API response time.

4. Result and Discussion

4.1 Results

4.1.1 Implementation Results

The layered security model was successfully integrated into a REST API developed with the Go Echo framework, utilizing a sequence of middleware components — JWT authentication, token version verification, Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC). All protected routes were grouped to ensure that each request completed the entire security pipeline before reaching business-logic handlers. JWT authentication using the RS256 asymmetric algorithm was incorporated into the login procedure, with tokens signed using a private key and validated with a public key on each subsequent request. The token payload included user ID, role information, token version, and expiration timestamp, while token versioning enabled secure revocation by invalidating previous tokens when user permissions changed. The middleware execution order was carefully configured to avoid bypass vulnerabilities, ensuring that only validated identities could proceed to role and attribute checks. This implementation confirmed that separating authentication and authorization within the Echo middleware framework is both feasible and effective.

4.1.2 Security Testing Results

Security validation was conducted using structured scenario-based testing to evaluate the correctness and robustness of the proposed layered security model. Unit testing verified the functionality of each middleware component, including JWT signature verification, token expiration validation, token version comparison, role validation, and attribute-based resource ownership checks. Integration testing was then performed to ensure that all middleware layers operated sequentially and consistently across protected API endpoints. The testing process was carried out iteratively during system development, with several implementation refinements applied — particularly to the ordering of authentication and authorization middleware and to token version validation synchronization — to ensure that authentication checks were enforced before authorization decisions were evaluated, preventing potential bypasses. Privilege escalation scenarios were simulated to assess the system's resilience against unauthorized access attempts. Vertical privilege escalation was tested by attempting to access administrator endpoints using a standard user role token, while horizontal privilege escalation was simulated by attempting to access resources belonging to other users. The final implementation successfully blocked all simulated unauthorized access attempts, as summarized in Table 1.

Table 1. Security Testing Results

Test Scenario	Expected Result	Actual Result	Status
Access without a token	Access denied	Access denied	Pass
Access with an expired token	Access denied	Access denied	Pass
Access with an invalid signature	Access denied	Access denied	Pass
Vertical privilege escalation attempt	Access denied	Access denied	Pass
Horizontal privilege escalation attempt	Access denied	Access denied	Pass
Valid access with correct role and ownership	Access granted	Access granted	Pass

4.1.3 Performance Evaluation

A performance evaluation was conducted to quantify the response-time overhead introduced by the layered middleware validation. Three system configurations were tested: (1) without security middleware, (2) with JWT authentication middleware only, and (3) with the full layered security architecture including JWT validation, RBAC authorization, and ABAC validation. Each configuration was executed 500 times to ensure statistical consistency of the results. The average response time was calculated using Equation (1), while response time variability was evaluated using the standard deviation formula shown in Equation (2).

Equation (1) — Average Response Time:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n}$$

Where $\bar{x}$ is the average response time, x_i represents the response time recorded for each request, and n denotes the total number of experimental runs.

Equation (2) — Standard Deviation:

$$\sigma = \sqrt{\frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}}$$

Where σ represents the standard deviation, x_i represents each measured response time, and $\bar{x}$ denotes the average response time.

Table 2. Response Time Statistics Across Security Configurations

Configuration	Avg (ms)	Min (ms)	Max (ms)	Std Dev (ms)
Without security middleware	12.4	11.1	14.0	0.73
JWT authentication only	18.7	17.0	21.3	1.08
Full-layered authentication and authorization	24.9	23.2	28.1	1.23

The baseline configuration without security middleware recorded an average response time of 12.4 ms, with values ranging between 11.1 ms and 14.0 ms and a standard deviation of 0.73 ms. When JWT authentication middleware was applied, the average response time increased to 18.7 ms, ranging from 17.0 ms to 21.3 ms, with a standard deviation of 1.08 ms. The full layered security configuration — including JWT validation, RBAC, and ABAC authorization checks — produced an average response time of 24.9 ms with values between 23.2 ms and 28.1 ms and a standard deviation of 1.23 ms. The relatively small standard deviation values across all configurations indicate that the measured response times remained consistent across repeated testing iterations.

4.2 Discussion

The experimental results showed that separating authentication and authorization into distinct middleware layers effectively prevented privilege escalation attacks. The sequential validation process ensured that requests could not bypass security measures once routed to protected endpoints, which aligns with prior research highlighting the importance of service-level enforcement in distributed systems (Berardi *et al.*, 2022; Escaleira *et al.*, 2025). Compared with earlier studies that relied solely on gateway validation (Gbenle *et al.*, 2022; Phanireddy, 2023), the proposed model provided stronger internal service protection by enforcing validation within the application middleware itself. The addition of token versioning addressed JWT revocation limitations noted by Bucko *et al.* (2023), while combining RBAC and ABAC produced more precise authorization control than traditional role-based methods alone (Al-Wadi & Maaita, 2023; Kovtun & Laponina, 2025). Although the layered approach introduced some response time overhead, overall performance remained within acceptable bounds, and the security gains justify the slight increase in computational cost. These findings confirm that a structured middleware architecture can effectively balance secure authentication and authorization enforcement with practical system performance.

5. Conclusion and Recommendations

This study proposed a layered security model for REST APIs implemented using the Golang Echo framework. The model separates authentication and authorization processes through sequential middleware layers that integrate JWT authentication, token version validation, Role-Based Access Control (RBAC), and Attribute-Based Access Control (ABAC). The experimental results demonstrated that the proposed architecture effectively prevents unauthorized access scenarios, including both vertical and horizontal privilege escalation attempts. Performance evaluation results showed that the layered security mechanism introduces additional computational overhead compared to an unsecured configuration. However, the measured response times remained relatively low, with the fully secured configuration producing an average response time of 24.9 ms. The token versioning mechanism introduced a lightweight database validation step during request processing, but the experimental results indicate that this additional check did not significantly affect overall system performance.

From a practical perspective, the proposed layered middleware architecture provides a flexible approach for implementing service-level security in microservice-based systems. By separating authentication and authorization responsibilities into distinct middleware components, the model improves security enforcement while maintaining manageable performance overhead. This approach can be particularly beneficial for applications that require fine-grained access control and scalable API security mechanisms. Despite these promising results, this study has several limitations. The performance evaluation was conducted in a controlled experimental environment and did not include high-concurrency workload testing. Additionally, the implementation was developed specifically for the Golang Echo framework, which may limit direct applicability

to other frameworks without adaptation. Future work may extend this research by evaluating system performance under concurrent workloads, exploring caching mechanisms for authorization checks, and implementing the layered security architecture in other frameworks to assess its portability and scalability.

References

Aldea, C. L., & Bocu, R. (2025). Authentication challenges and solutions in microservice architectures. *Applied Sciences*, *15*(22). https://doi.org/10.3390/app152212088

Al-Wadi, R. A., & Maaita, A. A. (2023). Authentication and role-based authorization in microservice architecture: A generic performance-centric design. *Journal of Advances in Information Technology*, *14*(4), 758–768.

Berardi, D., Giallorenzo, S., Melis, A., Prandini, M., Mauro, J., & Montesi, F. (2022). Microservice security: A systematic literature review. *PeerJ Computer Science*, *7*. https://doi.org/10.7717/peerj-cs.779

Bucko, A., Vishi, K., Krasniqi, B., & Rexha, B. (2023). Enhancing JWT authentication and authorization in web applications based on user behavior history. *Computers*, *12*(4). https://doi.org/10.3390/computers12040078

Dalimunthe, S., Hasri Putra, E., & Fadhly Ridha, M. A. (2023). RESTful API security using JSON Web Token (JWT) with HMAC-SHA512 algorithm in session management. *IT Journal Research and Development*, *8*(1), 81–94.

de Almeida, M. G., & Canedo, E. D. (2022). Authentication and authorization in microservices architecture: A systematic literature review. *Applied Sciences*, *12*(6). https://doi.org/10.3390/app12063023

Escaleira, P., Cunha, V. A., Barraca, J. P., Gomes, D., & Aguiar, R. L. (2025). A systematic review on security mechanisms for serverless computing. *Cluster Computing*, *28*(7). https://doi.org/10.1007/s10586-025-05371-4

Fauziah, R., *et al*. (2026). Evaluating middleware performance in microservices architectures. *Electronics*, *15*(1), 221.

Gbenle, T. P., Abayomi, A. A., Uzoka, A. C., Ogeawuchi, J. C., Adanigbo, O. S., & Odofin, O. T. (2022). Applying OAuth2 and JWT protocols in securing distributed API gateways: Best practices and case review. *International Journal of Multidisciplinary Research and Growth Evaluation*, *3*(5), 628–634.

Gunawan, R. (2024). Enhancing data security using JSON Web Token (JWT) and HMAC-SHA256 algorithm. *International Journal on Recent and Innovation Trends in Computing and Communication*.

Gupta, R. K. (2025). Beyond the perimeter: Zero-trust architecture as a framework for cloud API security. *World Journal of Advanced Research and Reviews*, *26*(1), 3389–3398.

Hu, V. C., Kuhn, D. R., & Ferraiolo, D. F. (2015). Attribute-based access control. *Computer*, *48*(2), 85–88. https://doi.org/10.1109/MC.2015.33

Hutasuhut, N. R. P., Amri, M. G., & Aji, R. F. (2024). Security gap in microservices: A systematic literature review. *International Journal of Advanced Computer Science and Applications*, *15*(12).

Ibnu Muakhori, & Syamsiah, N. (2025). Pengamanan arsitektur microservices pada aplikasi perusahaan: Strategi dan implementasi. *Info Kripto*, *19*(1), 29–37. https://doi.org/10.56706/ik.v19i1.116

International Organization for Standardization. (2022a). *ISO/IEC 27001:2022 information security, cybersecurity and privacy protection — Information security management systems — Requirements*. ISO.

International Organization for Standardization. (2022b). *ISO/IEC 27002:2022 information security, cybersecurity and privacy protection — Information security controls*. ISO.

Kovtun, D. P., & Laponina, O. R. (2025). Using attribute-based access control and mTLS in microservice architecture. *International Journal of Open Information Technologies*, *13*(6).

Kurniawan, M. C., Gamayanto, I., Sanyoto, G. K., & Widjajanto, B. (2025). Security evaluation of keycloak-based role-based access control in microservice architectures using the OWASP ASVS framework. *Journal of Applied Informatics and Computing*, *9*(6).

Lee, C., & Jeon, S. (2024). A study on the security performance of JWT token signature algorithms. *Journal of Information and Security*, *24*(4), 3–10. https://doi.org/10.33778/kcsa.2024.24.4.003

OWASP Foundation. (2023). *OWASP Application Security Verification Standard (ASVS) version 4.0.3*. https://owasp.org/www-project-application-security-verification-standard/

Phanireddy, S. (2023). Securing RESTful APIs in microservices architectures: A comprehensive threat model and mitigation framework. *International Journal of Emerging Research in Engineering and Technology*, *4*, 64–73. https://doi.org/10.63282/3050-922x.ijeret-v4i2p107

Rahman, M., *et al*. (2025). Trends and best practices in API-based web development security. *Journal of Information Technology and Digital Systems*.

Ramadhan Purba, G., Rizal, & Afrillia, Y. (2025). Keamanan endpoint API menggunakan OAuth2 pada unit layanan terpadu universitas Malikussaleh. *Rabit: Jurnal Teknologi dan Sistem Informasi Univrab*, *10*(2), 1424–1434.

Sänger, N., & Abeck, S. (2023). User authorization in microservice-based applications. *Software*, *2*(3), 400–426. https://doi.org/10.3390/software2030019

Venčkauskas, A., Kukta, D., Grigaliūnas, Š., & Brūzgienė, R. (2023). Enhancing microservices security with token-based access control method. *Sensors*, *23*(6). https://doi.org/10.3390/s23063363.