NextGen Scientific Publication
American International Journal of Computer Science and Technology

Volume 5 Issue 2, Pg. No. 70-80, AIJCST-V5I2P107, 2023
https://doi.org/10.63282/3117-5481/AIJCST-V5I2P107

Original Article

OAuth2 and JWT Security Patterns for Cloud-Native Microservices

***Venkatesh Satla**
Full Stack Lead Java Developer at Innosoul, Inc, USA.

Abstract:

Cloud-native microservices architectures have become a popular approach to building scalable, resilient, and flexible applications, however the distributed nature of microservices introduces major security challenges around authentication, authorization, identity propagation, and secure inter-service communication. To solve these issues, OAuth 2.0 and JSON Web Tokens (JWTs) have become popular as they provide standardized ways to do authorization and stateless authentication. The study analyses the security patterns of OAuth2 and JWT to secure cloud-native microservices settings and evaluates their effectiveness to address typical risks such as token theft, replay attacks, privilege escalation and unauthorized access. Existing security practices are carefully analysed and a security architecture is designed based on combining an OAuth2 Authorization Server, Identity Provider, API Gateway and JWT-based access control methods. The proposed framework shows secure token issuance, validation, propagation and revocation across distributed services. The architecture is evaluated in terms of security, scalability, and performance using a case study based on a Kubernetes-based e-commerce microservices platform. Experimental results show that the proposed framework improves the authentication efficiency and the security robustness, and provides acceptable latency for distributed applications. This study contributes to highlighting the importance of short-lived tokens, mutual TLS, token introspection, role-based access control, and central identity management, and to outlining the future directions of Zero Trust Architecture, decentralized identity, confidential computing and AI-driven threat detection.

Keywords:

OAuth2, JWT, Cloud-Native Security, Microservices, Authentication, Authorization, API Security, Kubernetes, Zero Trust, Identity Management.

Article History:

Received: 24.01.2023

Revised: 26.02.2023

Accepted: 08.03.2023

Published: 14.03.2023

1. Introduction

Cloud computing and microservices architecture have altered the present software development landscape. It enables the firms to construct scalable, durable and adaptive systems. Microservices design is different from the typical monolithic systems where all the functionalities are tightly coupled in one deployable unit. It breaks the programs into smaller independent services that communicate with each other via Application Programming Interfaces or APIs. This architectural style gives agility, continuous integration and deployment, fault isolation, and scaling of individual services based on business needs. Workloads are going to native cloud techs like Kubernetes and Docker. Microservices are the architecture of choice to develop distributed applications. But microservices bring enormous advantages in operations but also new security concerns that need stronger authentication and permission mechanisms.

Cloud native security is not the same as security in traditional monolithic systems. Authentication and authorisation are typically centralised, and easy to deal with in monoliths. Microservices on the other hand are loosely connected and run independently across various containers, hosts and networks meaning communication between multiple distributed components has to be secured and

allowed. With the increasing number of services the management of user identity, permission and access control policy gets hard. They want modern security solutions that offer safe access without compromising the size and speed cloud-native apps require.

OAuth 2.0 and JSON Web Tokens (JWTs) are the de facto approach to secure cloud native services. OAuth2 is an authorization framework based on delegation. It provides a method for applications to access protected resources on behalf of a user without sharing the user's credentials. A JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties. OAuth2 Combined along with these technologies offer stateless authentication, simpler identity propagation, and less centralized session management, making them a better fit for new microservice setups.

1.1. Challenges

The move to cloud-native microservices brings a whole new set of security problems that differ greatly from previous architectures. Distributed authentication is one of the major problems. Users consume several services in the same transaction. Authentication systems should be able to operate uniformly across many separate services, while remaining secure and performant. The usage of traditional session-based authentication techniques is complicated by distributed services across several containers and instances.

Another challenge: service-to-service permission. Microservices typically execute business logic that involves inter-service communication, and each service must authenticate the identity and authorization of the entity making the request. Among the essential security requirements is the need to establish trust between services without incurring high computational expense. As apps scale, token management also becomes more complex. Organizations need to properly manage token issuance, validity, expiration, renewal, and revocation, all the while minimizing the impact on application performance.

Cloud-native environments are quite dynamic in nature, with containers and services being launched, updated, and terminated constantly. Security methods need to therefore scale automatically with the infrastructure changes. Additionally, the dispersed nature of microservices dramatically increases the attack surface. Every accessible API endpoint is an attack vector for token theft, replay assaults, credential stuffing, API abuse and privilege escalation. Attackers can exploit vulnerabilities in token processing, authentication systems, or permission controls to obtain unauthorized access to sensitive resources.

The second difficulty is identity propagation. In complex business transactions, information about user identity and authorization has to be safely passed over multiple service boundaries. The integrity, authenticity and confidentiality of the identification information during the communication process is a prerequisite to apply the same security policy across the system.

1.2. Problem Statement

OAuth2 and JWT are commonly used technologies for securing cloud native apps, yet many organizations are suffering from security vulnerabilities owing to poor implementation and configuration practices. Common security challenges include: poor token validation procedures, use of long-lived access tokens, weak authorization enforcement mechanisms, unsecured storing of JWTs, ineffective token revocation schemes. Badly designed API gateways and lack of monitoring further enhance the risk of unauthorized access and security breaches.

Many existing systems focus on functionality and fast deployment as opposed to complete security architecture. So vulnerabilities usually go unnoticed until they are actually exploited by attackers. Such flaws can threaten confidentiality, integrity and availability and could have a potentially serious operational and financial impact. Hence a secure and scalable OAuth2-JWT framework is needed to handle authentication, authorization, token lifecycle management, identity propagation and secure service-to-service communication in cloud-native contexts.

1.3. Motivation

The motivation behind this research is the growing prevalence of cloud-native technologies and the increasing necessity to secure dispersed applications. Use of applications on platforms such as Kubernetes and Docker is rising across all industries for better scalability and operational efficiency. However, the rising utilization of distributed services and APIs has resulted in an increase in security breaches, such as token compromise and malicious access and identity attacks.

There's also a huge drive toward Zero Trust security models, which are about constantly checking identities and access requests, rather than perimeter-based protection. This is the sort of authentication and authorization frameworks that modern enterprises need to enable, and still preserve usability and performance. Moreover, stringent access control and identity management restrictions imposed by standards and regulations such as GDPR, HIPAA, PCI DSS and ISO 27001, also increase the demand for secure authentication solutions.

Organizations want security solutions that provide low latency, excellent scalability and ease of operation, without compromising on safety. OAuth2 and JWT can fulfill those requirements, however performance will depend a lot on how well it is implemented. Therefore, this project aims to study secure OAuth2 and JWT security patterns and create a framework to improve the security, scalability and efficiency of cloud native microservices architectures.

2. Literature Review

2.1. Evolution of Authentication Mechanisms

The complexity of authentication systems has increased significantly due to the rising complexity of online applications and distributed computing environments. The early systems relied primarily on Basic Authentication, in which user credentials were transmitted with every request. This approach was simple to implement but provided only modest security and left passwords vulnerable to interception unless encrypted. The session based authentication paradigm later became the dominant model for online applications where servers could preserve user sessions via cookies and centralized session stores. This works well in case of monolithic systems but handling sessions across distributed settings was a scalability problem.

Later improvements included Security Assertion Markup Language (SAML) that enabled federated identity management and single sign-on capabilities across enterprise platforms. OAuth1 brought a number of security improvements such as delegated permission without exposing user credentials but was not generally used due to its complexity. OAuth2 simplified the authorization procedure and became the standard for protecting APIs and cloud applications. OpenID Connect (OIDC) was introduced to standardize the way of sending identity information to OAuth2 and to authenticate. OAuth2 and OIDC are more scalable, flexible, and interoperable than previous systems, and thus are better suited for cloud-native microservice architectures.

2.2. OAuth2 Framework

OAuth2 is a popular authorization framework defined in RFC 6749, which provides safe access to protected sites without the need to share user credentials. It has 4 major entities. They are: Resource Owner – the owner of the protected data, Client Application – the application requesting access on behalf of the user, Authorization Server – the entity authenticating the users and issuing access tokens and Resource Server – the entity hosting protected resources and validating the access tokens.

OAuth2 has a couple different sorts of permission grants to suit different application needs. The Authorization Code Grant is used mostly for web and mobile apps whereas the Client Credentials Grant allows computers to safely communicate with each other. Refresh Tokens allow clients to receive new access tokens without the need for the user to authenticate over and over. The Device Authorization Grant is for devices that have restricted input capability. OAuth2 is a popular authorization framework because of its convenience of use and scalability but implementation faults may lead to vulnerabilities such as token abuse, faulty validation and uncontrolled access.

Table 1. OAuth2 Grant Types and Their Usage

Grant Type	Primary Use Case	Security Level
Authorization Code + PKCE	Web & Mobile Applications	High
Client Credentials	Service-to-Service Communication	High
Refresh Token	Token Renewal	Medium
Device Authorization	IoT & Smart Devices	Medium

2.3. JSON Web Tokens (JWT)

JSON Web Tokens (JWTs) are popular for stateless authentication and authorization of distributed systems. A JWT is composed of 3 parts: header, payload and signature. The header contains information like the type of token and the cryptographic algorithm used to sign it. The payload includes claims that characterize the permitted item. Subject IDs, issuer, audience, user roles and permissions etc. The signature provides cryptographic integrity and validity for the token, so that the recipients may verify that the token was not altered.

JWTs are scalable and efficient yet they have many security problems. Research has found flaws like confusion attacks on algorithms, token tampering and cryptographic key leaks. Therefore, a safe implementation of JWT requires strong signature verification, secure handling of signing keys, and strict validation of token assertions before access to protected resources is permitted.

2.4. Microservices Security

The distributed architecture of current cloud applications has attracted great academic interest on the security of microservices systems. API Gateway is a popular security choice as it enables centralized authentication, authorization and traffic control. Service mesh systems such as Istio and Linkerd offer some extra security capabilities such as mutual TLS, service identity verification, and encrypted service-to-service communication. IAM solutions increase security by providing access control functionalities such as Role-Based Access Control (RBAC) and Attribute-Based Access Control (ABAC) that allow organizations to establish granular authorization needs for distributed services.

2.5. Recommended practices for OAuth2 Security.

In this paper we describe many best practices that may be used to strengthen the security of OAuth2. This includes: proof key for code exchange (PKCE) for securing authorization flows, short-lived access tokens to limit the risk of exposure, and refresh token rotation to prevent token replay attacks. Other proposals are token introspection for runtime validation, and mutual TLS integration for securing the client-server communication. These solutions improve security and robustness of OAuth2 implementations in cloud native systems.

2.6. Patterns of JWT Security

Using JWT, several security approaches are described to mitigate the common hazards of distributed systems. Tokens are validated using patterns to authenticate signatures and assertions and allow access. Audience restriction patterns allow tokens to be received only by the intended recipients, reducing the potential for misuse. Token exchange patterns enable secure transmission of identity across a wide number of services and revocation procedures use blacklisting and token introspection to revoke compromised tokens. This leads to improved authentication and authorization controls in microservices systems.

2.7. Gaps in the research

There are several studies on OAuth2, JWTs and microservices security, however certain holes still need to be filled. Most of the past research is focused on OAuth2 and JWT separately, not as a security architecture. Empirical research is lacking on large-scale cloud-native installations, especially for Kubernetes. Moreover, there has been limited study on combining Zero Trust concepts with OAuth2-JWT systems. Furthermore, there is relatively little comparative study for security effectiveness, scalability and performance of different implementation options which needs further investigation and confirmation.

3. Proposed Methodology

3.1. Proposed Architecture

The security architecture aims at enabling safe authentication, authorization and identity management for cloud native microservice platforms. The architecture is built on OAuth2 and JSON Web Token (JWT) technologies and a Kubernetes based infrastructure that enables scalability, security and operational efficiency. The identity provider (IdP) is the central component of the identity management system. Popular solutions such as Keycloak, Okta and Autho allow you to authenticate users and manage digital identities. The IdP interfaces with the OAuth2 Authorization Server to authenticate the user, generate an access token, generate a refresh token, and to impose authentication policies.

The API Gateway is the main entry point of all incoming external client requests. The gateway provides key security features such request routing, traffic monitoring, token validation, and rate limiting. By centralizing these tasks, the gateway may ease security for each service and provide a standard access control mechanism. Behind the gateway lies the microservices layer of independent services such as User Service, Order Service, Payment Service and Inventory Service. Each service performs some business activities and uses validated JWTs to authorize incoming requests. The whole ecosystem is based on the container orchestration architecture Kubernetes, which allows autonomous scalability, load balancing, fault tolerance. The advantage of such an architecture is the guarantee of the efficiency of security controls, regardless of the dynamic changes in the number of services and workloads.

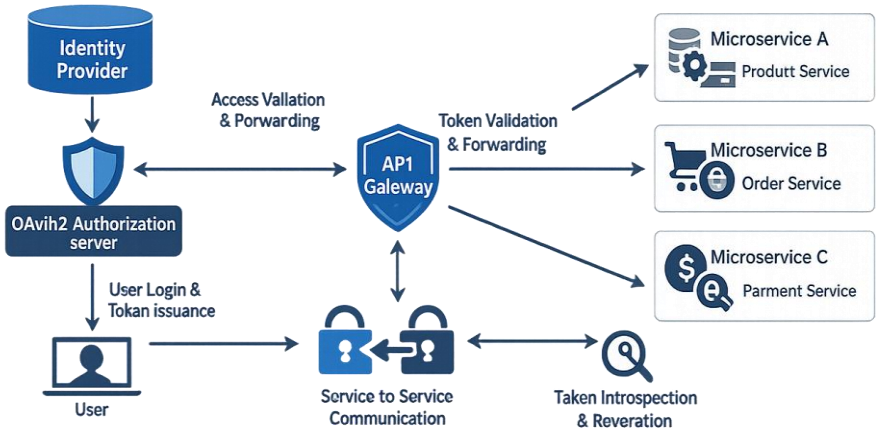

Figure 1. Proposed OAuth2–JWT Security Architecture for Cloud-Native Microservices

3.2. Flow of Security

The security workflow starts when a user tries to access a protected application resource. The user is forwarded to OAuth2 Authorization Server for authentication. The Identity Provider then validates user credentials to confirm that only authorized users are allowed access to the system. If the user authenticates properly, the Authorization Server responds with a JWT access token. This token contains information about user identification, rights, roles and other claims.

The client application then attaches the JWT in the Authorization header for every call to the API Gateway. Then the gateway does thorough token validation before forwarding the request. Verification includes checking the digital signature, token expiration, audience check and checking that the request scope matches the scope of the permissions in the token. Only requests that pass all the validation checks are allowed to continue.

Once the check is successful the confirmed JWT will be sent to downstream microservices securely. Each service can independently validate token assertions and apply authorization policies without repeating authentication. This solution minimizes the latency, allows stateless communication and assures secure identity transmission over the distributed system.

3.3. Security Patterns OAuth2

The presented framework includes many OAuth2 security protocols for better authentication and authorization mechanisms. Web and mobile apps use the Authorization Code Grant with a Proof Key for Code Exchange (PKCE). PKCE adds another security layer to prevent an attacker from stealing and re-using the authorization codes during the authentication process. This is especially critical for public clients who are not well able to maintain client information.

We are using Client Credentials Grant for service-to-service communication. This grant type allows microservices to directly authenticate with the Authorization Server and obtain access tokens without the need for any user interaction. This results in secure and scalable machine to machine interactions.

The framework also supports token introspection for real-time token validation. The resource servers may query the token introspection for the state of the access tokens, either the token is revoked or invalidated. We also use refresh token rotation which limits the possibility of replay attacks. This means that every time you use a refresh token to request an access token, you will receive a new refresh token and invalidate the old refresh token. This stops an attacker from replaying the tokens.

3.4. Patterns of JWT Security

The proposed framework includes many JWT security methods to secure token-based authentication. All JWTs are digitally signed using the asymmetric cryptography algorithm RS256 to guarantee that the token is legitimate and intact. To prevent unauthorized modification of the token, and to enable secure verification between dispersed services, public and private key pairs are utilized.

Access tokens. Access tokens are short-lived (five to fifteen minutes) to reduce the impact of a stolen token. Even if a token is compromised, the limited lifetime limits the attack window. It also provides audience validation which guarantees that tokens cannot be utilized across systems that are not allowed to use them.

Scope based permission is used for fine grained access control. Certain resources or processes for a service require token scopes to be configured before you can access them. It also has a JWT blacklisting feature so you may revoke tokens fast. Invalid tokens are stored in one place and validated at the time of validation. It allows the invalidation of compromised tokens before their natural expiry.

3.5. Integration of zero trust

The proposed solution is based on the Zero Trust security concept where no person, device or service is trusted by default. Access requests are evaluated irrespective of where they come from. Least privilege Access is giving someone only the access they need to do their job. Security decisions are constantly checked and maintained up to date by monitoring the user session and token.

In practice, any request to the system is authenticated and permitted. Then the microservices check for the incoming tokens and the permissions to access the requests. The expanded continuous verification technique boosts the security of the system and reduces the chances of unlawful access in the distributed systems.

3.6. Threat Simulation

The suggested approach alleviates a number of major hazards in cloud native microservices. Access tokens are short lived and are stored securely in order to protect against token theft. Replay attacks are mitigated by refresh token rotation and strict expiration policies. Role Based Access Control (RBAC) feature allows users and services to perform only approved privilege and prevent privilege escalation. Mutual Transport Layer Security (mTLS) encrypts communication between services, and confirms that the services are who they say they are to protect against man-in-the -middle attacks. Correct validation of the oauth2 tokens and scope enforcement helps to avoid any misuse attempts. Cryptographic identity verification and mutual TLS help ensure that only trustworthy services can connect to each other, reducing service impersonation problems. “Together, these security measures represent a comprehensive approach to securing cloud native microservice architectures.

4. Case Study

4.1. System Description

A cloud-native e-commerce platform, like a typical microservice-based application, was built and deployed to verify the proposed security framework of OAuth2 and JWT. The platform is a mirror of the actual online shopping ecosystem that exists in the real world with a number of services working together to fulfill client requests, carry out transactions and maintain track of stock levels. The system architecture consists of 5 types of microservices, namely User Service, Product Service, Order Service, Payment Service and Inventory Service each working independently. Each service has a well-defined REST API that allows it to interface with other services to perform a given business activity.

The User Service is responsible for providing registration, authentication, and profile management to users. In addition to product listings, search and price information, the Product Service also provides capacity for product catalogs. This service is responsible for processing consumer orders and coordinating processes with other services. Payment processing and transaction validation is performed by the Payment Service. The Inventory Service is responsible for handling changes to the inventory and ensuring that products are available. These services are autonomous and they interact with each other through a validated and permitted service-to-service communication.

The complete application is deployed within a kubernetes cluster to use container orchestration, auto scaling, fault tolerance and workload control. The Istio Service Mesh is the part of the ecosystem that provides support for mutual TLS (mTLS), traffic management, observability and secure service communication. Keycloak is the main identity provider (IdP) that handles user authentication,

identification, token issuance, and enforcement of access control restrictions. The deployment environment is close to real cloud-native enterprise systems and hence appropriate to assess the proposed security architecture in real-world scenarios.

4.2. Installation

The implementation environment is based on well-known open source technologies, therefore guaranteeing true applicability and reproducibility. "OAuth2 is the de facto standard for authentication of users and services". This information is kept in the form of tokens using JWTs. Distributed services use JWTs to hold identification and authorization code. These tokens are signed assertions that enable secure, stateless authentication across the microservice ecosystem.

We use the Kong API Gateway to route requests from external clients. The gateway first authenticates tokens, then routes requests, then enforces access control, then manages traffic to ensure requests are routed to right backend services. Istio Service Mesh provides many capabilities for microservices such as service discovery, encrypted communication channels, traffic monitoring and policy enforcement. All application services are containerized using Docker, so that they can be deployed in a consistent way and can be relocated independently between environments. Kubernetes can manage containers and also provides other services like load balancing, fault recovery and scaling of services. PostgreSQL is the backend database system which is used for user information, product data, orders, inventory records and transaction details. These tools provide a safe, scalable, and convenient cloud-native environment for testing OAuth2 and JWT security requirements.

4.3. Experimental Setup

The proposed framework is applied in several distinct experimental settings, and its performance efficacy and security are evaluated. The first one was about user login authentication which was a scenario where users were logging in to the software via OAuth2 authentication procedures. The major aim was to evaluate the performance of the authentication process and the efficiency of the token issuing procedure and user authentication experience.

The second case under consideration was the authorization of cross-service use. In this scenario, the transactions were initiated by validated personnel, and involved communication across several different platforms. The experiment was designed to verify the security of JWT-based identity propagation and authorization systems in implementing access control limitations across service boundaries.

The third case is the procedures for canceling tokens. To test the framework's capacity to detect and prevent the use of invalid or hacked credentials, access tokens and refresh tokens were intentionally revoked. This was done to assess the capabilities of the framework. This event revealed that introspection and token revocation are the responsible and trustworthy things to do.

This group also included communication between service providers. OAuth2 provided access tokens and Istio provided mutual TLS encryption. The microservices spoke with each other via OAuth2. The evaluation aimed to determine the level of communication security, the accuracy of service authentication, and the efficiency of authorization in a distributed environment.

A simulation of replay assaults was run for the sixth scenario. This meant stealing access tokens that were issued before, and trying to utilize them without authorization. The goal of this experiment was to explore the effectiveness of short-lived tokens, refresh token rotation, token validation policies and revocation procedures, with a view to lowering the chances of replay attacks.

4.4. Metrics for Data Collection

In the investigations, a lot of quantitative indicators were counted to check the effectiveness of the proposed framework. We measured the authentication latency (the amount of time taken to authenticate the user and issue the token). The JWT validation performance was measured at the API Gateway level and the microservice level by tracking the time it took to validate the token. We carried out an evaluation of the system's number of requests processed per second to establish the throughput of the system, under different workloads and security settings.

We measured the simultaneous usage of CPU and memory to analyze the overhead introduced by OAuth2 authentication, JWT validation, token inspection, and mutual TLS communication. To further investigate the capabilities of the framework to identify and prevent security breaches, such as unauthorized access attempts, token misuse, and replay attacks, we also evaluated the detection rate

of the security violation. This was done to explore the potential of the framework. The data were used to analyze the security effectiveness and performance impact of the proposed OAuth2-JWT architecture in a cloud-native microservice scenario, which provided a reasonable baseline for evaluation of the design.

Table 2. Experimental Evaluation Metrics

Metric	Description	Unit
Authentication Latency	Time required for user authentication and token issuance	Milliseconds (ms)
JWT Validation Time	Time required to validate JWT tokens	Milliseconds (ms)
Throughput	Number of requests processed per second	Requests/sec
CPU Utilization	Processing overhead introduced by security controls	Percentage (%)
Memory Consumption	Memory used during authentication and authorization	MB
Security Detection Rate	Ability to detect unauthorized access and attacks	Percentage (%)

5. Results and Discussion

5.1. Authentication Performance

The impact of the proposed OAuth2-JWT security architecture on the authentication efficiency and the overall system performance has been evaluated. Experimental results demonstrated that the proposed architecture can provide fast and reliable authentication with strong security restrictions. During testing, the average time for a user to log in was approximately 180 milliseconds, suggesting that the authentication procedure did not add any substantial latency to the end user experience. Stateless token verification performed well, with JWT validation at the API Gateway and microservice layers averaging 5 milliseconds per request. Average token refresh times including the issuing of a new access token through refresh token rotation was 120ms.

The results reveal that the overhead of authentication with JWT-based authentication is substantially smaller than the traditional session-based technique. A JWT contains the entire information required to verify and authorize a user. Then services may validate the token locally without having to hit some central authentication or session database over and over again. The lowered validation time helps to speed up processing of requests and enhances user experience notably in big scale cloud native systems that may process thousands of requests at the same time . The results indicate that the OAuth2 and the JWT technologies may offer strong security and acceptable performance characteristics for modern distributed systems.

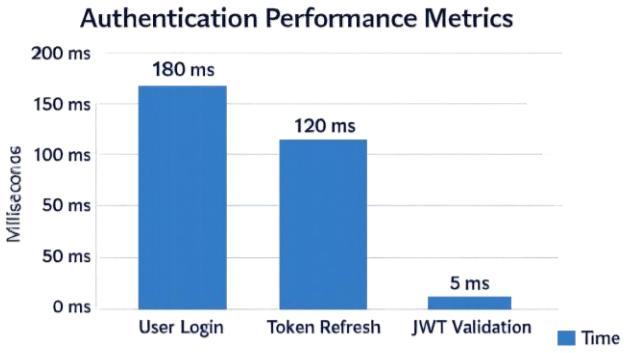

Figure 2. Authentication Performance Metrics of the Proposed OAuth2–JWT Framework

5.2. Scalability Results

Scalability tests were performed with three different workload scenarios of 1,000 users, 10,000 users and 100,000 users. The objective was to test the suggested framework's capacity to maintain performance and security as the system demand increases. The framework showed steady behavior in all test scenarios without a significant decrease in authentication performance and service responsiveness.

Another crucial element of this scale-ability is the statelessness of JWT based authentication. This is different from traditional session-based systems that require to save and synchronize the session in a centralized manner. In the case of JWTs we don't need to save the session information on server side. This allows many instances of the service to handle authentication requests independently and without bottlenecks in session management. The number of service replicas was automatically scaled based on workload requirements and incoming traffic was smartly routed to the available instances by the API Gateway.

The testing also proved that the resource usage increases progressively and predictably with the growth of users. We did not detect any major bottlenecks in authentication processing, token validation, or service communication. The results show that the proposed architecture is able to enable large scale cloud-native deployments with security and performance constraints.

5.3. Safety evaluation

To test the resilience of the proposed architecture against the most frequent assaults, a series of attack simulations were performed to evaluate the security efficacy. Replay attack simulations involved acquiring valid access tokens and trying to reuse them after expiration or revocation. The framework was able to prevent all the attempts of replay attacks with an attacker success rate of 0%. This was achieved primarily by using short-lived access tokens, refresh token rotation and continual token validation.

We also constructed modified and forged JWTs and utilized these to evaluate attempts to forge tokens. The RS256 cryptographic signature validation stopped these vulnerabilities. However, the private signing key was effectively protected in the Authorization Server and the attackers were not able to generate tokens with a valid signature. Any modifications to the contents of the token would make the signature verification fail and the token would be rejected immediately.

We evaluated attempts to access services without authorization by allowing services to request resources beyond the rights assigned for them. The Role Based Access Control (RBAC) system prevented privilege escalation and limited services to permitted operations only. These rules, OAuth2 validation and mutual TLS authentication, enhanced the overall security posture of the microservices environment.

5.4. Contrast

We have done comparative analysis by comparing traditional session based authentication and the proposed OAuth2-JWT approach. The results show that OAuth2 with JWT provides better scalability, flexibility and cloud compatibility. Session-based authentication requires a central location to store sessions, which makes it more difficult to scale in a distributed environment. On the other hand, The JWT based authentication is stateless and thus easy to scale across many service instances.

The proposed framework also performed better with less dependency on the backend session management services. There was also a big gain for cloud-native compatibility, because distributed services can independently validate JWTs. However, it is more difficult to revoke tokens than in session-based systems, since JWTs are designed to be self-contained and stateless. However, this deficiency is remedied by features like token introspection and token blacklisting which allow it to cope successfully with revocation challenges.

5.5. Discussion

The results show that the proposed OAuth2-JWT security architecture provides some notable advantages. The architecture does not depend on session storage, therefore it is scalable and can deliver stateless authentication to remote services. This reduces the operational effort and also results in faster applications due to the reduction in server-side computations. Moreover, the use of standard protocols and token formats allows the interaction among a wide range of services and platforms, making the framework the best ideal for modern cloud native environments.

But there are some boundaries. Access tokens are valid in general until they expire or other revocation processes are in place. Which raises the complexity of token revocation. Good cryptographic key management is also vital as compromising of the signature key could affect the security of the entire system. Moreover, the architectural complexity added by OAuth2, JWTs, API Gateways, service meshes and Kubernetes might also result in further deployment and operating challenges. However, the results in general show that the suggested framework provides a decent trade-off between security, scalability and performance, thus becoming a feasible solution for the protection of cloud-native microservices applications.

6. Conclusion and Future Scope

6.1. Conclusion

In this work, we study the applicability of OAuth2 and JSON Web Token (JWT) security protocols in cloud-native microservice systems. We show that combining OAuth2 authorization procedures with JWT based authentication provides a safe, scalable and efficient framework for distributed applications. The proposed architecture has been able to address the important security requirements such as user authentication, authorization enforcement, token life cycle management, identity propagation and secure service-to-service communication. The framework improves the overall system security by using API gateways, identity providers, RBAC and mTLS, while maintaining the authentication latency as low as possible and the throughput as high as possible. The Kubernetes-based case study also verified the feasibility of the proposed approach and its high resilience to replay attacks, token forging and unauthorized access tries. The results show that used according to known security best practices, OAuth2 and JWT still are the key technology for securing modern cloud native ecosystems.

Future research on enhanced Zero Trust security models that focus on continuous identity verification and least-privilege access may result in more advancements in cloud-native security. New technologies such as AI-driven threat detection could improve anomaly detection and monitoring of token abuse, while Decentralized Identity (DID) frameworks could provide a more secure and user-centric approach to identity management. Other research topics include confidential computing for hardware-assisted credential protection, adoption of OAuth 2.1 security enhancements, fine-grained authorization using Open Policy Agent (OPA), Rego policies and Attribute-Based Access Control (ABAC), and multi-cloud identity federation and trust management. Such innovations can significantly improve security, privacy and resiliency of next-generation cloud-native microservices applications.

References

[1] Tran Florén, S. (2021). Implementation and Analysis of Authentication and Authorization Methods in a Microservice Architecture: A Comparison Between Microservice Security Design Patterns for Authentication and Authorization Flows.

[2] Parakala, A. (2022). Integrating Salesforce and UiPath: Cross-System Intelligent Automation. International Journal of Emerging Trends in Computer Science and Information Technology, 3(4), 88-99. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I4P109

[3] Torkura, K. A., Sukmana, M. I., & Meinel, C. (2017, December). Integrating continuous security assessments in microservices and cloud native applications. In *Proceedings of the 10th IEEE/ACM International Conference on Utility and Cloud Computing* (pp. 171-180).

[4] Shiramalla, R., & Guntupalli, B. . (2021). Cost-Effective Softphone Integration in CRM Platforms Using RESTful APIs: A Salesforce Case Study for Voice-to-Text Sales Enablement. International Journal of Emerging Trends in Computer Science and Information Technology, 2(1), 101-114. https://doi.org/10.63282/3050-9246.IJETCSIT-V2I1P112

[5] Srigadde, B. R. (2021). Future Methods, Most Underrated Apex Features. *American International Journal of Computer Science and Technology*, *3*(1), 35-45. https://doi.org/10.63282/3117-5481/AIJCST-V3I1P104

[6] de Almeida, M. G., & Canedo, E. D. (2022). Authentication and authorization in microservices architecture: A systematic literature review. *Applied sciences*, *12*(6), 3023.

[7] Katangoori, S., & Deore, S. (2022). Predictive Drift Detection and Adaptive Reconciliation in Multi-Cloud Data Environments. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(4), 184-194. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I4P119

[8] Kumar Doodala, A. N., & Thatraju, S. (2022). NLP-Driven Benefits Interpretation Engine for Personalized Member Communication. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *3*(1), 173-183. https://doi.org/10.63282/3050-9262.IJAIDSML-V3I1P118

[9] Srivastava, R. (2021). *Cloud Native Microservices with Spring and Kubernetes: Design and Build Modern Cloud Native Applications using Spring and Kubernetes (English Edition)*. BPB Publications.

[10] Allenki, S. S., & Korutla, R. (2021). Agile Development in Practice: From Intern to Contributor. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(3), 91-103. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I3P110

[11] Ciarla, F. (2022). *Analisi e miglioramento dell'architettura di un'applicazione cloud native esistente sfruttando microservizi e service mesh= Analyze and improve the architecture of an existing cloud-native application exploiting microservices and service meshes* (Doctoral dissertation, Politecnico di Torino).

[12] Muppaneni, K. (2022). Optimizing React Hooks for Efficient State and Side-Effect Management. *American International Journal of Computer Science and Technology*, *4*(6), 44-55. https://doi.org/10.63282/3117-5481/AIJCST-V4I6P105

[13] Ranchal, R., Bastide, P., Wang, X., Gkoulalas-Divanis, A., Mehra, M., Bakthavachalam, S., ... & Mohindra, A. (2020). Disrupting healthcare silos: Addressing data volume, velocity and variety with a cloud-native healthcare data ingestion service. *IEEE Journal of Biomedical and Health Informatics*, *24*(11), 3182-3188.

[14] Cerny, T. (2022). Microservices Security Challenges and Approaches. *Information Systems Development: Artificial Intelligence for Information Systems Development and Operations (ISD2022 Proceedings)*.

[15] Vppalapati, M., & Talasila, P. K. (2022). Correlated Independence: Why Redundant Storage Systems Share the Same Fate. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(1), 169-179. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P119

[16] Patchamatla, P. S. S. (2021). Design and implementation of zero-trust microservice architectures for securing cloud-native telecom systems. *International Journal of Research and Applied Innovations*, *4*(6), 6169-6177.
[17] Muppaneni, R. K. (2021). Securing the Enterprise: How Dynamics 365 Meets Global Compliance Standards. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 133-143. https://doi.org/10.63282/3050-922X.IJERET-V2I1P114
[18] Abdelfattah, A. S., & Cerny, T. (2022). Microservices security challenges and approaches.
[19] Suryadevara, S. S. K., & Polinati, A. K. (2022). Cross-Cloud Governance Engine Using Policy-as-Code for CMS Platforms. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 165-175. https://doi.org/10.63282/3050-922X.IJERET-V3I4P118
[20] Gaddam, R. R. (2022). Advanced Data & Model Drift Detection at Scale. *International Journal of AI, BigData, Computational and Management Studies*, *3*(2), 124-136. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V3I2P113
[21] Lewis, A., & Roberts, E. (2022). Secure API Gateway Configurations for Java Workloads.
[22] Allenki, S. S. (2022). Securing Databases in the Cloud with RBAC and Encryption Best Practices. *International Journal of Emerging Research in Engineering and Technology*, *3*(3), 173-182. https://doi.org/10.63282/3050-922X.IJERET-V3I3P117
[23] Shiramalla, R. (2022). Design of a Unified API Interface Using Workato for Cross-Platform Data Orchestration Between Salesforce and Oracle ERP. *International Journal of Emerging Trends in Computer Science and Information Technology*, *3*(1), 157-168. https://doi.org/10.63282/3050-9246.IJETCSIT-V3I1P118
[24] Vanapalli, Satyendra Kumar. "Bridging Business and Technology: The Role of CRM in Digital Transformation." *International Journal of Artificial Intelligence & Digital Transformation* 5.2 (2022): 01-17.
[25] Fugaro, L., & Vocale, M. (2019). *Hands-On Cloud-Native Microservices with Jakarta EE: Build Scalable and Reactive Microservices with Docker, Kubernetes, and OpenShift*. Packt Publishing Ltd.
[26] Vppalapati, M. (2022). The Storage Stack Nobody Draws: Cabling, Panels, and the Illusion of Isolation. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 211-220. https://doi.org/10.63282/3050-922X.IJERET-V3I2P121
[27] Knutson, M., Winch, R., & Mularien, P. (2017). *Spring Security: Secure your web applications, RESTful services, and microservice architectures*. Packt Publishing Ltd.
[28] Muppaneni, K. (2021). HTTP/3 & REST Latency Improvement. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 122-132. https://doi.org/10.63282/3050-922X.IJERET-V2I1P113
[29] Gaddam, R. R. (2021). Vertex AI as a Unified Control Plane for MLOps. *International Journal of Artificial Intelligence, Data Science, and Machine Learning*, *2*(2), 92-102. https://doi.org/10.63282/3050-9262.IJAIDSML-V2I2P110
[30] Kamadi, S. (2022). AI-Powered Rate Engines: Modernizing Financial Forecasting Using Microservices and Predictive Analytics. *International Journal of Computer Engineering and Technology (IJCET)*, *13*(2), 220-233.
[31] Kumar Doodala, A. N. (2021). Intelligent EOB/ERA Generation and Validation Framework on Legacy Systems like Mainframes. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 111-121. https://doi.org/10.63282/3050-922X.IJERET-V2I1P112
[32] Kommuru, Madhurima. "Concurrency is hard: Java vs Python pitfalls you can't ignore." *International Journal of Modern Research in Science & Engineering* 5.2 (2022): 01-18.
[33] Parakala, A. (2021). Building Analytics-Driven Bots: RPA Meets Business Intelligence. *International Journal of Emerging Research in Engineering and Technology*, *2*(1), 77-87. https://doi.org/10.63282/3050-922X.IJERET-V2I1P109
[34] Katangoori, S., & Deore, S. (2022). Edge-Cloud Hybrid Data Pipelines: Architectures for Federated Analytics and Learning. *American International Journal of Computer Science and Technology*, *4*(3), 20-34. https://doi.org/10.63282/3117-5481/AIJCST-V4I3P103
[35] Jani, Y. (2020). Spring boot for microservices: Patterns, challenges, and best practices. *European Journal of Advances in Engineering and Technology*, *7*(7), 73-78.
[36] Vanapalli, Satyendra Kumar. "AI-Driven Fraud Detection in Financial Systems: A CRM-Centric Approach." *International Journal of Machine Learning and Predictive Analytics* 5.2 (2022): 01-14.
[37] Suryadevara, S. S. K., & Polinati, A. K. (2022). Cross-Cloud Governance Engine Using Policy-as-Code for CMS Platforms. *International Journal of Emerging Research in Engineering and Technology*, *3*(4), 165-175. https://doi.org/10.63282/3050-922X.IJERET-V3I4P118
[38] Srigadde, B. R., & Devaraju, J. M. (2022). The Wrath of Limitations: Lightning Fields and Their Constraints. *International Journal of Emerging Research in Engineering and Technology*, *3*(2), 201-210. https://doi.org/10.63282/3050-922X.IJERET-V3I2P120
[39] Muppaneni, R. K. (2020). Retail Reimagined: How Dynamics 365 Commerce Is Driving Omnichannel Experiences. *International Journal of AI, BigData, Computational and Management Studies*, *1*(1), 49-59. https://doi.org/10.63282/3050-9416.IJAIBDCMS-V1I1P106
[40] Kasaram, C. R. (2018). Spring Boot and Microservices: Engineering Modular and Scalable Back-End Architectures. *Development (IJCSERD)*, *9*(1), 1-10.