ADVANCED ACCESS CONTROL MECHANISMS FOR MICROSERVICES-BASED CLOUD SYSTEMS

Akshay R Gangula

akshaygangula1377@gmail.com

***Abstract*—The implementation of microservices transformed software development, but created fundamental access control problems that traditional security systems fail to solve. The literature reveals a significant gap in understanding the impact of Zero Trust Architecture (ZTA) and Policy-as-Code (PaC) on performance. This paper fills the existing knowledge gap through an extensive case study and thorough performance evaluation of a microservices system protected by contemporary security patterns. The research shows that a separate policy engine-based authorization system enforces detailed attribute-based policies without performance degradation while achieving 12.50 ms p(95) latency during intense stress testing. The research delivers an operational and tested framework that architects and engineers can use to establish high-performance, secure access control systems in contemporary distributed systems.**

***Index Terms* - Cloud Computing, Access Control, Cybersecurity, Distributed Systems, Zero Trust Architecture, Attribute-Based Access Control (ABAC), Role-Based Access Control (RBAC), API Security, Identity and Access Management (IAM)**

I. INTRODUCTION

The software landscape has been fundamentally reshaped by the trend of microservices. This architectural style enables the development of powerful, agile, and scalable distributed systems, but also dissolves the traditional security perimeter, creating significant access control complexities. While many architectural patterns for securing these systems have been proposed, there remains a notable gap in the literature regarding the quantifiable performance overhead of these solutions in a real-world setting. This paper addresses this gap by presenting a rigorous performance analysis of a system implementing a decoupled, Policy-as-Code authorization model that utilizes a standalone policy engine.

II. MICROSERVICES AND CLOUD COMPUTING BASIC CONCEPTS

The architectural and operational principles of microservices and cloud computing have caused modern-day access control problems. This will be done in a succinct manner, as it will have the key characteristics and also traditional access control models to state the limitations that they impose on distributed, cloud-native environments.

A. Microservices and Their Security Implications

Microservices let you develop applications as a set of independently deployable services that can be implemented independently. These services interact using lightweight protocols such as HTTP/REST and are frequently deployed with CI/CD pipelines [1]. Basic characteristics include decentralized governance, polyglot persistence, and a focus on resilience and evolutionary design [2].

Security implications are significant.

- Increased inter-service communication expands the attack surface.
- Not all services have the same security protection as others because they each use different technologies.
- Information is owned by multiple distributed parties; it requires enforcement of an access policy at both the service level and the data level.
- Centralized control of usage must be systematically coordinated despite localized administration.

B. Cloud Model and Security Issues

As per "NIST SP 800-145," cloud computing is a model that gives on-demand access to shared, configurable computing resources. The main features of cloud computing are on-demand self-service, broad network access, resource pooling, rapid elasticity, and measured service. [3]

Cloud services are delivered through.

- IAAS is the raw computing infrastructure.
- PaaS for application development provides a platform.
- A full-service application, SaaS.

The deployment models consist of private clouds, public clouds, community clouds, and hybrid clouds.

Security is managed through the shared responsibility model: providers protect the infrastructure; consumers protect the data, identities, and applications. Some challenges that can be encountered are low visibility, data privacy, and communication security. [4]

C. Traditional Access Control Models and Their Limitations

Common models include.

- DAC is flexible but difficult to manage at scale.
- MAC: Highly Organizational and High Standard Environments.
- The RBAC scheme simplifies administration but may lead to a role explosion.
- The ABAC is attribute-driven and is capable of offering fine-grained and context-aware policies. Limitations in cloud-native microservices.
- DAC/RBAC does not scale well enough.
- Unchanging plans that have trouble with short-term things.
- Not detailed enough at the API or function level.
- Challenges in distributed enforcement, as centralized PDPs may undermine performance.
- The unawareness of context, in particular, the dynamic attributes of the environment, such as device type and threat level.

The industry trend is shifting from RBAC to ABAC, which supports Fine-grained and Dynamic Control, along with Architectural Needs. NIST SP 800-162 standardizes this movement, which effectively positions ABAC as a core component of access control. [5]

III. ACCESS CONTROL CHALLENGES IN MICROSERVICES-BASED CLOUD SYSTEMS

Due to their distributed, dynamic, and API-driven nature, microservices deployed in cloud environments face unique access control challenges. Most traditional models do not consider the complexity that is produced, requiring the rethinking of access control mechanisms.

A. Expanded Attack Surface and Distributed Trust

Microservices significantly raise the count of independently accessible components, each having an exposed entry point [4]. Communication between services over the network may introduce more vulnerabilities. Trust should not be assumed like a perimeter-based model. Instead, every request from either the internal network or the external network requires authentication and authorization.

The choice between smaller microservices and bigger services impacts agility and access control complexity. With more services comes the need to secure multiple endpoints; fewer services will ease enforcement but may hinder scalability and flexibility.

B. Ephemeral Resources and Dynamic Environments

Cloud-native systems expand and diminish services and move workloads, which makes rules based on IPs or hostnames become invalid. The combination of Docker container runtimes and Kubernetes orchestration platforms makes this problem worse because they continuously create and destroy service instances [4]. Dynamic attributes like service identity and context, not static ones, must form the basis of effective access control.

C. Risks Related to API-Centric Gaps and Authorization

APIs drive microservices communication, thus API security is fundamental. The OWASP API Security Top 10 identifies several access control failures of high severity [6]:

- API1:2023 – BOLA: Missing object-level authorization.
- API2:2023 – Broken Authentication: Flawed token handling and user impersonation.
- API3:2023 – Property-Level Exposure: Unauthorized access to individual data fields.
- API5:2023 – Function-Level Authorization: Inadequate separation of user roles.

The problems demonstrate why organizations need detailed authorization controls to be integrated into each API endpoint instead of relying on perimeter-based security through API gateways [2].

D. Distributed Policy Management and Enforcement

Access control policies must be consistently defined, maintained, and enforced by widely deployed independent services [7]. Centralized approaches don't scale well & create bottlenecks. Complexity arises from:

- Diverse service-specific access requirements.
- Possible disagreement with policy or rules.
- Service interactions and dependencies are complex to map.
- Requirements of auditing in regulated places.

The implementation of decentralized policy management creates conflicts and inconsistent enforcement because it lacks automated resolution mechanisms. The process of managing policy versions between different teams leads to higher chances of errors while making it harder to perform rollbacks. The evolving threats of supply chain attacks and insider misuse require ongoing policy updates and immediate anomaly detection to preserve security in dynamic microservices environments.

E. Data Privacy and Multi-Tenancy Issues

Robust controls are essential in multi-tenant cloud models to ensure that strict separation between tenants is maintained to prevent access by other users or even a cloud admin. To follow laws like GDPR and HIPAA:

- Granular data access enforcement.
- Data is encrypted as it is sent and also when it is stored.
- Comprehensive audit trails.

Integration of Security Controls across the Data Lifecycle is needed to Ensure Compliance and Safeguard Sensitive Data.

F. Performance, Scalability, and Observability Limitations

Access control must not hinder system performance. As the requests increase, the latency in making decisions from the centralized PDPs or poor evaluation of policies may lead to a bad user experience. Also, the vast amounts of observability data generated by microservices logs, metrics, and traces can hinder anomaly detection if not managed [4] [7].

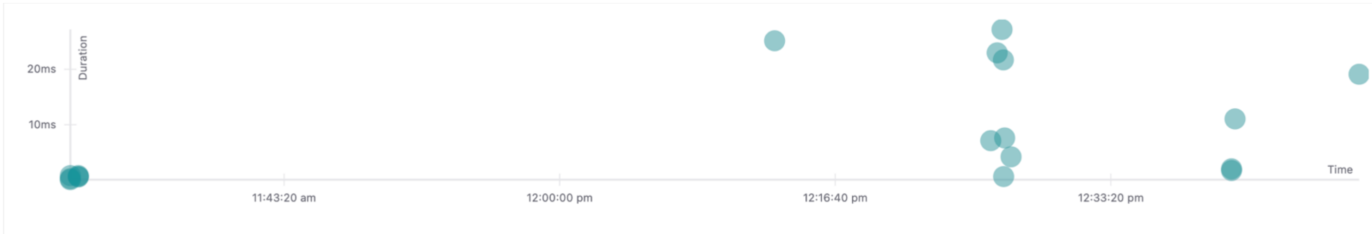

Fig. 1: Jaeger's user interface showing internal API endpoints for service discovery and trace retrieval within the Jaeger all-in-one service. This view highlights Jaeger's core capability to collect and query monitoring data, providing foundational observability for distributed systems.

The distributed trace in Figure 1 demonstrates the process of a request to the `/api/products` endpoint. The visualization clearly presents security enforcement points. The `products-proxy` initiates the JWT validation process, which represents a vital security measure that blocks the **Spoofing** threat described in our threat model. The read-only `products-service` depends on JWT validation at the proxy for its security mechanism. The `users-service` requires two security steps for its requests because it first validates JWTs at the `users-proxy` before making a call to the standalone OPA service for authorization. The system implements a defense-in-depth strategy through its security controls, which match service sensitivity levels to prevent Elevation of Privilege threats for essential operations.

The solution to these complex challenges requires moving away from traditional static access control systems toward dynamic automated mechanisms that understand context. The modern authorization paradigms ZTA, ABAC, and PaC lead this transformation by providing scalable, fine-grained, adaptive authorization for microservices-based cloud environments. The following section provides detailed information about these modern approaches, which establish the basis for secure and efficient microservices access management.

IV. ADVANCED ACCESS CONTROL MECHANISMS AND PARADIGMS

Organizations working with dynamic and distributed microservices-based cloud systems must implement advanced access control paradigms. These access control paradigms must overcome static access control models. New regulatory mechanisms are emerging that focus on context sensitivity, granularity of control, and declarative sanctioning of policies, often utilizing automation and intelligence to fit into modern architectures.

A. Context-Aware and Declarative Access Control

1) ***Zero Trust Architecture (ZTA)****:* According to ZTA, one must 'never trust and always verify' for every user and service interaction. This means that both services and users must be authenticated and authorized to interact by ZTA, irrespective of their network locations [4]. It focuses on micro-segmentation, continuous validation, minimum privilege, and data-centric protection. In microservices, this translates to:

- Inter-service security with mutual TLS (mTLS).
- Enforcing policies via a service mesh.
- Dynamic trust reassessment for identifying lateral movement.

Still, using zero-trust architecture complicates things and could slow them down [8].

2) Enhanced ABAC and Policy-Based Access Control (PBAC):

ABAC is a type of access approach whereby the system keeps evaluating the request by subject, object, and action. Furthermore, ABAC allows for high granularity and dynamic policies. The PBAC system improves on the ABAC system by managing policies as formal, centrally-managed artifacts [5]. Benefits include:

- Allows fine control at the API/data level.
- Real-time policy adaptation.
- Ability to integrate environmental contexts (i.e., time, threat). Maintaining attribute consistency and runtime performance is not easy.

3) ***Evolving RBAC****:* RBAC remains useful when enhanced through:

- There are Specific positions for Delegation.
- Dynamic RBAC modifies roles according to users' context.
- Flexible conditions for parameterized roles.
- REKS is an example of an encrypted RBAC.

Although RBAC is helpful in well-defined environments, ABAC is still indispensable. Hence, both are combined to handle dynamic use cases.

B. Scalable Token-Based Authorization

The authorization system depends on token-based strategies, including OAuth 2.0 and OpenID Connect, for distributed authorization.

Advanced techniques include:

- You can embed fine-grained scopes and custom claims in JWTs.
- Temporary tokens that represent the changing privileges in real-time.

- Exchanging tokens for community service calls.
- Replay protections and secure storage to avoid token theft. [4]

By separating identity verification from access enforcement, authorization can now be stateless and resilient.

C. Policy-as-Code (PaC)

PaC integrates Policies into CI/CD pipelines, similar to version-controlled code. Using tools such as OPA allows for a declarative policy definition that can be evaluated by a **centralized or standalone policy engine**, which services can query for authorization decisions [9]. Key benefits:

- Automated testing and deployment.
- Environment consistency and auditability.
- Cross-team collaboration on security logic.

Although requiring some policy engineering skills, PaC enables policy enforcement at scale, in a repeatable manner, and aligned with DevOps.

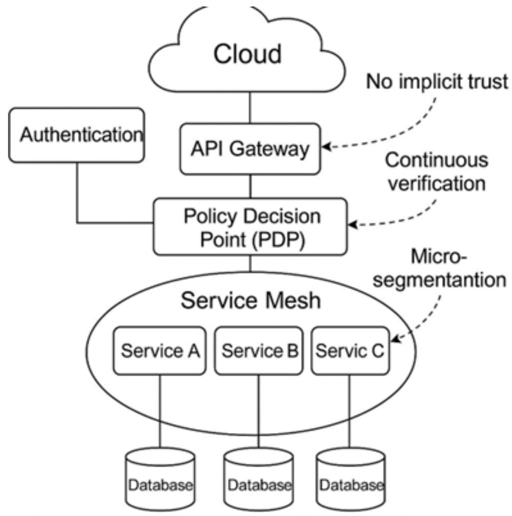

Fig. 2: Conceptual overview of Zero Trust Architecture applied to microservices, emphasizing continuous verification, micro-segmentation, and least privilege enforcement through service mesh and dynamic policy evaluation

V. Implementation Strategies And Best Practices

To secure dynamic, distributed microservices-based cloud systems, organizations must implement layered strategies that align with modern architectural demands. This section presents a practical security framework organized into six core domains, supporting both Zero Trust principles and DevSecOps practices:

A. Principle of Least Privilege (PoLP)

The system should grant users and entities the smallest set of permissions that still enable their required functions. Actions:

- Follow the Principle of Least Privilege (PoLP) to control user roles and service, container, and cloud resource access.
- Regularly review your permissions to avoid privilege creep.
- NIST SP 800171r3 should be followed for the separation of roles and restriction on privileged functions [10].

B. Robust Identity and Access Management (IAM)

Centralize identity governance and enforce secure authentication.

Actions:

- Use a combined or decentralized IAM system for `end-user/service` identity.
- Make sure all users, especially privileged accounts, use MFA verification.
- Automatic management of identity provisioning, deprovisioning, and sessions.
- Securely use APIs to access other servers' resources.

C. Access Control via API Gateways and Service Meshes

Apply access policies on all external and internal traffic**.**

Actions:

- API Gateways can be used for authentication, rate limiting, and route validation of external (north-south) traffic.
- Service meshes like Istio can be used to manage access and control over internal data.
- Both layers can be combined in a way that complies with Zero Trust.

D. Monitoring, Logging, and Auditing

Maintain visibility, traceability, and rapid incident detection.

Actions:

- Keep records of all access determinations, such as denials and privilege escalations.
- Utilize monitoring tools to detect anomalies, policy violations, and behavioral anomalies.
- The following section explains how to follow these regulations through an organization-wide audit and monitoring strategy.
  - To reduce OWASP A09:2021 risks, you need to implement strong logging systems.

E. Proactive Vulnerability Mitigation (OWASP-Aligned)

Goal: To resolve security vulnerabilities that exist throughout the access control pipeline.

Actions:

Harden access control against:

- A01:2021 (Broken Access Control)
- API1:2023 – BOLA
- API2:2023 – Broken Authentication.
- CNAS-3 – Improper Authentication/Authorization.
- K03:2022 – More permissive RBAC

Implement secure configurations, testing, and policy validation throughout the CI/CD pipeline. [6]

F. Encryption and Key Management

Goal: To safeguard data even when access-control systems are bypassed.

Actions:

- All service data must be encrypted through TLS/mTLS protocols during transit.
- All sensitive data stored at rest must be encrypted by using a FIPS 140-2 compliant algorithm.
- The system should have a centralized key lifecycle management system that handles key generation and distribution, as well as performs key rotation and revocation.
- The organization should follow NIST SP 800-53 and 800-171r3 for cryptographic control guidelines.

G. Real-World Integration Blueprint: A Case Study

To demonstrate how the proposed access control strategies operate in a real-world setting, this paper introduces "Project Sentinel," a microservices-based cloud architecture specifically designed to test and validate advanced authorization mechanisms. Project Sentinel implements a cloud-native microservices architecture that integrates PaC authorization with real-time Zero Trust enforcement and multi-layer monitoring (Prometheus, Grafana, Jaeger). The system employs OPA with static ABAC policies and service mesh with Envoy and Keycloak for authentication. The system implements robust security and observability practices.

1) Practical Orchestration and Configuration-as-Code: A real-world implementation requires a concrete orchestration plan. A tangible artifact, such as a `docker-compose.yml` file, defines the end-to-end environment, orchestrating over ten services and managing their networking, volumes, and dependencies.

This "as-code" paradigm extends to security components. Rather than relying on manual setup, an automated script like `setup-keycloak.sh` is used to configure the IAM system, including realms, clients, and user roles. This approach ensures a consistent, repeatable, and auditable security baseline, directly supporting the principles of a robust IAM strategy.

2) Integrating Frontend Security Considerations: While the primary focus of this paper is backend, service-to-service access control, a complete security architecture must also address frontend integration. In the Project Sentinel case study, the `frontend-service` is built with **Spring Boot** and **Thymeleaf** for server-side rendering. This setup directly interacts with core

access control components such as central IAM (Keycloak) and authorization policies. Therefore, frontend-specific security concerns—such as server-side session management and the handling of form-based authentication—play a necessary role in ensuring end-to-end access control coverage:

- **Server-side Session Management**: Securely maintaining user sessions within the Spring Boot application to prevent session hijacking, complementing backend access control flows.
- **Form-based Authentication**: Ensuring form-based login flows required by the central IAM (Keycloak) are securely managed, which aligns with backend authentication and authorization mechanisms.

3) Comprehensive, Automated Validation (DevSecOps):

A modern security posture requires continuous, automated testing integrated directly into the development lifecycle. This involves a suite of executable scripts that validate different facets of the system's security and performance.

- **Automated Security and ABAC Testing**: To mitigate risks like Broken Object Level Authorization (BOLA) and validate fine-grained access policies, a dedicated test suite is essential. A script like `test-all-tc-abac.sh` automates the execution of dozens of test cases that confirm ABAC policies are enforced correctly for different user attributes and contexts. This provides a continuous feedback loop on the correctness of the Policy-as-Code implementation.
- **Automated Performance Validation**: Access control mechanisms must not introduce prohibitive latency. A dedicated `performance-tests/` directory containing specific scripts for **baseline, stress, and spike testing** offers a concrete methodology for validating the performance impact of the security architecture. Documenting these findings in a results file e.g., `PERFORMANCE_TEST_FINAL_RESULTS.md`) provides a clear case study for performance validation under load.
- **End-to-End Functional Verification**: Finally, a top-level utility script such as `verify-functionality.sh` can be used to run a holistic check of the entire system, ensuring that all components, from the frontend to the database and security services, are integrated and functioning correctly.

TABLE I
COMPARATIVE PERFORMANCE METRICS UNDER BASELINE, STRESS, AND SPIKE LOAD SCENARIOS

Test Scenario	Max VUs	Total HTTP Requests	Avg. Response Time	p(95) Latency	Checks Passed
Baseline	10	754	4.73 ms	9.17 ms	100%
Stress	25	1,511	6.67 ms	12.50 ms	100%
Spike	30	700	8.69 ms	41.14 ms	100%

H. Threat Model and Mitigations

To ensure the robust security posture of the Project Sentinel architecture, a comprehensive threat modeling approach was adopted using the widely accepted STRIDE framework. STRIDE categorizes threats as Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege. Each threat category was carefully analyzed to identify potential vulnerabilities and associated mitigation strategies.

TABLE II
STRIDE THREAT MODEL AND IMPLEMENTED MITIGATIONS

STRIDE Threat	**Mitigation Strategy and Implemented Component**
Spoofing	User and service identity is verified using signed JWTs issued by Keycloak and validated by each microservice via the Spring Security framework (`SecurityConfig.java`).
Tampering	Data integrity is protected by using TLS for all external communication and signed JWTs, which are tamper-proof.
Repudiation	All access decisions and service interactions are logged and traced using Prometheus and Jaeger, providing non-repudiable audit trails.
Information Disclosure	Data is encrypted in transit using TLS. Fine-grained authorization policies (`abac.rego`) prevent unauthorized services or users from accessing sensitive data endpoints.
Denial of Service	The Envoy Service Mesh provides distributed DoS protection through circuit breakers and outlier detection, with each service proxy implementing connection limits and automatic failure isolation to protect backend services
Elevation of Privilege	Fine-grained, attribute-based access control is enforced by an OPA sidecar on every API request, preventing users or services from performing actions beyond their designated permissions as defined in the `abac.rego` policy.

Fig. 3: Real-time Grafana dashboard monitoring the QuickCart microservices, showcasing critical KPIs including 95th percentile request latency and throughput across API endpoints. These metrics provide observability to ensure security enforcement does not adversely impact system performance.

TABLE III
KEY ACCESS CONTROL CHALLENGES IN MICROSERVICES AND CLOUD SYSTEMS WITH MITIGATION APPROACHES

Challenge	Brief Description of Challenge	Impact on Access Control	Relevant Advanced Mitigation Mechanism(s)	Key Best Practices
Expanded Attack Surface	Increased number of services and APIs creating more potential entry points [4]	More points to secure; higher risk if any service has weak AC	ZTA, Micro-segmentation, Service Mesh	PoLP, API Security (OWASP API Top 10), Strong Authentication [10]
Dynamic & Ephemeral Resources	Services/instances scale rapidly; IPs and locations change frequently	Static rules fail; hard to track and authorize dynamic services	ABAC/PBAC, Policy-as-Code, AI/ML-driven AC	Dynamic attribute-based policies, Infrastructure Automation [1], Continuous Monitoring
API Security Risks	Heavy reliance on APIs; vulnerabilities like BOLA, Broken Authentication [6]	Unauthorized access/modification; compromised service integrity	Advanced Token Strategies, ZTA, Fine-grained ABAC/PBAC	OWASP API Top 10 Mitigation, API Gateway, Input Validation, Secure Coding Practices
Distributed Policy Management	Difficulty in defining and enforcing consistent policies across services	Policy conflicts; inconsistent enforcement; weak audit trails	Policy-as-Code, Centralized/Federated PAPs, Service Mesh policy enforcement	Standardized policy languages, Version control, Automated validation/deployment
Data Security & Privacy	Multi-tenant risks; data regulation compliance (e.g., GDPR, HIPAA) [4]	Risk of data leakage; regulatory penalties; trust erosion	ABAC (with data tagging), ZTA, Encryption-aware AC (e.g., REKS)	Data encryption (at rest/in transit), IAM governance, Regular audits, Data-centric policies
Scalability & Visibility	AC systems must scale efficiently; visibility into service interactions is often lacking [4]	Performance bottlenecks; difficulty detecting/responding to anomalies	Optimized PDPs (ABAC), Efficient token validation, AI/ML for anomaly detection	Logging & monitoring, Distributed tracing, Observability platforms, AC component performance testing

VI. FUTURE RESEARCH DIRECTIONS

The growing complexity of microservices and cloud-native environments has made access control problems more severe. The following promising research areas have been identified:

A. *Quantum-Resistant Access Control:*

The cryptographic basis of access control faces a threat from quantum computing.

1) Standardize and develop quantum-resistant signature schemes for policy and token approval.
2) Examine the key exchange protocol that is compatible with post-quantum security.
3) Study the impact of post-quantum primitives on access control process efficiency.

B. *Privacy-Preserving Authorization:*

User and system sensitive attributes must be protected with context-aware granular systems.

1) Use Zero-Knowledge proof (ZKPs) to validate attributes without disclosing them. [11]
2) Look into homomorphic encryption for a secure policy evaluation.
3) Credentials that allow users to prove their identity without revealing it.
4) Find the equilibrium between great data and minimizing data.

C. *Standardization and Interoperability:*

Microservices are decentralized, which adds difficulty to the uniform policy enforcement [2].

1) Identify and define cross-platform policy languages and schemas for use in tools such as Rego.
2) The exchange protocols and attribute formats of identity providers shall be standardized.
3) The system should enable token interoperability across security solutions and ecosystems.

D. *AI-Driven Adaptive Control and Explainability:*

A promising area for future research lies in leveraging AI and ML to create more dynamic and intelligent access control systems. Future work could focus on developing models that enhance security by:

1) ***Detecting Anomalies****:* Using ML to analyze behavioral baselines of services and APIs to identify suspicious deviations in real-time. [12]

2) ***Automating Policy Refinement****:* Exploring how ML could learn from historical access patterns to recommend or automatically generate optimized, context-aware security policies.

3) ***Risk-Based Permissioning****:* Creating systems that could dynamically adjust permissions based on real-time threat intelligence and environmental signals.

As these AI-driven systems make more autonomous decisions, **explainability (XAI)** will become critical. Future research must also focus on creating interpretable models and developing tools to audit and visualize AI-driven authorization processes, ensuring that the decisions are fair, unbiased, and transparent.

VII. CONCLUSION

The research demonstrates how cloud-native microservices architectures transform software security through the analysis of legacy access control system limitations when facing modern security challenges. The research used theoretical foundations together with practical case study evidence to show how ZTA with PaC enables the implementation of detailed and adaptive access policies.

Project Sentinel proved that a standalone policy engine with attribute-based policies maintains system performance by demonstrating a consistent 12.50 ms p(95) latency under heavy stress conditions without security compromises. The evidence indicates that distributed microservices environments can maintain both high security and scalability.

Organizations need to establish adaptive, automated, data-driven authorization frameworks as essential building blocks for developing trustworthy, resilient systems in fast-evolving cloud environments. Future research should concentrate on developing AI-based policy enhancement methods alongside privacy-protecting approaches to improve these access control systems.

References

[1] C. Richardson, "What Good Looks Like - July 2024 Edition - Microservices Rules," *Microservices.io*, Jul. 10, 2024. [Online]. Available: https://microservices.io/post/architecture/2024/07/10/microservices-rules-what-good-looks-like.html

[2] M. Fowler and J. Lewis, "Microservices," *martinfowler.com*. [Online]. Available: https://martinfowler.cofm/articles/microservices.html

[3] P. Mell and T. Grance, *The NIST Definition of Cloud Computing*, NIST Special Publication 800-145, Sep. 2011. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-145.

[4] G. D. Maayan, "Understanding Cloud Native Security," *IEEE Computer Society Tech News*, Apr. 17, 2024. [Online]. Available: https://www.computer.org/publications/tech-news/trends/cloud-native-security

[5] V. C. Hu, D. F. Ferraiolo, D. R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, *Guide to Attribute Based Access Control (ABAC) Definition and Considerations*, NIST Special Publication 800-162, Jan. 2014. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-162.

[6] "OWASP API Security Top 10 Vulnerabilities: 2023," *APIsecurity.io*. [Online]. Available: https://apisecurity.io/owasp-api-security-top-10/

[7] "Why Observability Was Key to Citigroup's Cloud-Native Transition," *The New Stack*, Apr 19, 2024. [Online]. Available: https://thenewstack.io/why-observability-was-key-to-citigroups-cloud-native-transition/

[8] B. Christudas, "Advanced High Availability and Scalability," in *Practical Microservices Architectural Patterns: Event-Based Java Microservices with Spring Boot and Spring Cloud*, Berkeley, CA: Apress, 2019, pp. 589–637. doi: 10.1007/978-1-4842-4501-9_16.

[9] F. Cao, C. Fang, X. Qin, X. Jin, and F. Shu, "Towards a Cloud-Controlled Heterogeneous Robot System Based on Microservices," in *Proceedings of the 2023 9th International Conference on Control Science and Systems Engineering (ICCSSE)*, Shenzhen, China, 2023, pp. 202–207. doi: 10.1109/ICCSSE59359.2023.10245035.

[10] R. Ross and V. Pillitteri, *Protecting Controlled Unclassified Information in Nonfederal Systems and Organizations*, NIST Special Publication 800-171 Rev. 3, May 2024. [Online]. Available: https://doi.org/10.6028/NIST.SP.800-171r3.

[11] Z. Ma and J. Zhang, "Efficient, Traceable and Privacy-Aware Data Access Control in Distributed Cloud-Based IoD Systems," *IEEE Access*, vol. 11, pp. 45206–45221, 2023. doi: 10.1109/ACCESS.2023.3272484.

[12] C. Rajasekharaiah, "Securing Microservices on Cloud," in *Cloud-Based Microservices: Techniques, Challenges, and Solutions*, Berkeley, CA: Apress, 2021, pp. 179–202. doi: 10.1007/978-1-4842-6564-2_9.