Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Serverless Yet Secure: Rethinking Cloud Engineering Paradigms

Sailesh Oduri
Cloud Security Engineer, Quest IT Solutions Inc, Texas, USA.

ARTICLE INFO

Received: 01 Apr 2025

Revised: 12 May 2025

Accepted: 22 May 2025

ABSTRACT

Serverless computing, exemplified by Function-as-a-Service (FaaS) platforms such as AWS Lambda, Azure Functions, and Google Cloud Functions, has revolutionized cloud-native application development by abstracting away infrastructure management and enabling event-driven, scalable architectures. Its promise of operational efficiency and cost-effectiveness has fueled widespread adoption across industries. However, this paradigm shift also introduces novel security challenges—ephemeral compute instances, event injection vulnerabilities, complex identity and access management (IAM), ins/ecure third-party dependencies, and limited observability—render traditional cloud security models insufficient. This paper critically examines the unique security risks inherent to serverless environments and explores architectural shifts required to build "secure-by-design" serverless applications. Through a combination of theoretical analysis, provider-specific comparisons, and empirical case studies, we evaluate security parameters including attack surface, privilege boundaries, and runtime isolation. The research proposes a layered security model incorporating Zero Trust principles, micro-isolation at the function level, secure defaults, and behavior-based runtime monitoring. Our findings demonstrate that while serverless models challenge conventional security practices, a deliberate reengineering of cloud architecture can achieve both agility and resilience. The study outlines best practices and reference architectures that integrate cloud-native security frameworks from the ground up. These insights contribute to the evolving discourse on secure cloud engineering and provide actionable guidelines for developers, architects, and cloud providers committed to advancing secure serverless ecosystems.

Keyword: Serverless Computing, Cloud Security, Function-as-a-Service (FaaS), Secure Architecture, Cloud Engineering, Zero Trust, Micro-isolation, Cloud-native Security.

1. Introduction

The evolution of cloud computing has progressed through increasingly abstracted service models—beginning with Infrastructure-as-a-Service (IaaS), advancing to Platform-as-a-Service (PaaS), and culminating in Function-as-a-Service (FaaS), widely known as serverless computing [1], [2]. Unlike traditional cloud models, serverless platforms allow developers to deploy code as discrete, stateless functions that automatically scale in response to demand, eliminating the need for server provisioning or capacity planning [2], [6]. This approach significantly enhances developer productivity and operational efficiency while enabling pay-per-execution pricing, making it attractive for modern microservices and event-driven architectures [3], [8].

Despite these advantages, serverless architectures introduce novel security risks that challenge conventional cloud security frameworks. The ephemeral nature of serverless functions, combined with limited execution duration and stateless design, hampers effective monitoring, logging, and forensic analysis [4], [9]. Cold starts—where a function container is initialized from scratch—can introduce unpredictable latencies and potential exposure to race conditions or misconfigured

permissions during bootstrap phases [6], [12]. Additionally, serverless systems are tightly coupled with event triggers such as HTTP APIs, queues, and storage events, which may be vulnerable to injection attacks or privilege escalation if not properly secured [10], [14]. The pervasive use of third-party libraries and APIs further expands the threat surface, often outside the purview of centralized IT governance [13], [19].

This study addresses the central research question: How can robust security mechanisms be integrated into serverless architectures without undermining their inherent agility and efficiency? Through a synthesis of literature, real-world case studies, and platform analysis, this paper contributes: (i) a security taxonomy specific to serverless environments, (ii) evaluation metrics across AWS Lambda, Azure Functions, and Google Cloud Functions [11]–[13], and (iii) a set of design principles incorporating Zero Trust [5], [15], micro-isolation [16], and runtime behavior monitoring [17]. Our findings aim to inform cloud engineering practices and support the design of secure, scalable, and compliant serverless applications.

2. Related Work

Traditional cloud security models, designed primarily for IaaS and PaaS environments, emphasize perimeter defense, network segmentation, and virtual machine (VM) isolation. These models assume long-lived resources, static configurations, and well-defined trust boundaries [1], [2]. However, serverless computing departs radically from these assumptions. Functions are ephemeral, event-driven, and stateless, executed in shared, abstracted environments managed by the cloud provider. As a result, established controls such as host-based intrusion detection systems (HIDS), full-disk encryption, and persistent logging become either impractical or insufficient [3], [4].

Several studies have highlighted the limitations of applying traditional security paradigms to serverless systems. Gojmerac et al. [3] identify challenges in securing function runtimes, securing input events, and hardening against third-party dependency attacks. Gusev and Silva [4] propose dynamic policy enforcement mechanisms to secure runtime environments, emphasizing the need for in-function access controls and contextual awareness. Hendrickson et al. [5], through an empirical assessment of FaaS platforms, reveal critical gaps in observability and state management that impede effective incident response.

To address emerging threats, industry bodies and platform providers have proposed frameworks and best practices. The OWASP Serverless Top 10 [10] outlines critical risks such as event injection, insecure deployment configurations, and over-privileged function roles. AWS and Google Cloud have introduced security primitives including IAM roles for functions, environment variable encryption, and VPC integration [11], [13]. Microsoft Azure supports role-based access control (RBAC), private endpoints, and identity-based triggers [12].

Despite these contributions, several key gaps persist in the literature. Most research focuses on theoretical threat models or single-provider evaluations, lacking a comparative, cross-platform security benchmark. Furthermore, little attention is given to integration patterns that combine Zero Trust architectures [15], micro-isolation [16], and behavior-based runtime defense [17] within FaaS environments. There is also a paucity of design-driven methodologies that embed security in the software development lifecycle (SDLC) specific to serverless applications. This paper aims to fill these gaps by synthesizing empirical evidence and offering a holistic, platform-agnostic framework for secure serverless engineering.

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

3. Methodology

This research adopts a hybrid methodology that integrates a conceptual framework for analyzing security in serverless architectures with empirical case studies across major Function-as-a-Service (FaaS) platforms—namely AWS Lambda, Azure Functions, and Google Cloud Functions. The goal is to develop a security-centric evaluation model tailored to the unique operational characteristics of serverless environments and validate its applicability through real-world configurations and controlled experiments.

3.1 Research Design

The theoretical foundation of this study is grounded in principles from Zero Trust security, cloud-native architecture, and threat modeling for microservices. The research is structured in two phases:

1. **Theoretical Phase**: We conducted a literature synthesis and architectural analysis to identify key dimensions where serverless systems diverge from traditional cloud models. This included an assessment of attack surfaces, control planes, execution models, and trust boundaries in FaaS offerings. Insights from prior studies [2], [3], [5], OWASP Serverless Top 10 [10], and cloud provider documentation [11]–[13] were used to build a conceptual framework of serverless security requirements.

2. **Empirical Phase**: To validate and apply the framework, we performed a **multi-cloud case study** using test applications deployed on AWS Lambda, Azure Functions, and Google Cloud Functions. These environments were selected based on their maturity, developer adoption, and availability of security tooling. Each deployment was designed to simulate realistic cloud-native applications with diverse triggers (HTTP APIs, message queues, storage events), third-party dependencies, and varying IAM configurations.

3.2 Security Evaluation Metrics

To ensure consistency and depth in our analysis, we defined the following security evaluation metrics, inspired by both academic literature [3], [4], [9] and industry guidelines [10], [14]:

- **Attack Surface Complexity**: Number and types of event sources, exposed APIs, external dependencies, and possible entry points into the function runtime. We considered this a proxy for the function's exposure to adversarial inputs.

- **Privilege Boundaries**: The granularity and scope of IAM roles and permissions granted to each function. Over-permissioned roles were flagged as violations of the principle of least privilege.

- **Data Isolation and Leakage Potential**: We assessed whether functions could access unintended resources (e.g., shared memory, environment variables, storage buckets) across invocations or between services.

- **Execution Control and Confinement**: Evaluation of runtime sandboxing, container reuse policies, timeouts, memory limits, and support for container isolation or microVMs (e.g., Firecracker in AWS Lambda [11]).

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

- **Configuration Security**: Review of VPC integration, TLS enforcement, environment variable encryption, secret handling (e.g., via AWS Secrets Manager, Azure Key Vault), and logging/auditing capabilities.
- **Cold Start Behavior**: Cold start latency and its effect on function security posture during initialization (e.g., insecure states during startup, race conditions).
- **Third-party Dependency Risk**: Static and dynamic analysis of external packages for known CVEs, license issues, or unsafe calls (e.g., insecure HTTP clients, hard-coded secrets).

3.3 Case Study Environments

We conducted comparative assessments using equivalent application logic deployed on:

- **AWS Lambda**: Deployed via the Serverless Framework and AWS SAM. Tested with various trigger types (API Gateway, S3 events, DynamoDB streams). IAM roles were explicitly scoped, and functions were deployed both inside and outside VPCs to test network security.
- **Azure Functions**: Hosted in Consumption Plan and Premium Plan modes, triggered via HTTP endpoints and Azure Event Grid. RBAC and managed identity configurations were tested. Use of Application Insights and Azure Monitor allowed for telemetry collection.
- **Google Cloud Functions**: Deployed using gcloud CLI and Cloud Console, tested with Pub/Sub, Cloud Storage triggers, and HTTPS endpoints. IAM conditions and Service Accounts were explicitly managed. Integration with Secret Manager was evaluated.

Each platform was configured to simulate three categories of applications:

1. Public API Gateway-Triggered Functions
2. Internal Event-Driven Microservices
3. Data Processing Workflows (Storage + Pub/Sub/Queue Integration)

3.4 Tooling and Techniques

A diverse set of tools and techniques were employed to ensure a multi-dimensional view of security risks:

- **Static Code Analysis**: Tools like SonarQube**,** Semgrep, and ESLint security plugins were used to detect code-level issues such as insecure function calls, injection vectors, and credential leakage in Node.js and Python runtimes.
- **Dependency Scanning**: We used Snyk**,** npm audit, and pip-audit to identify vulnerable packages or transitive dependencies within deployed functions.
- Configuration Scanning: IaC templates (CloudFormation, Azure ARM, GCP Deployment Manager) were analyzed using Checkov and **kics** to detect misconfigurations such as public function access, insecure storage buckets, or overbroad IAM bindings.

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

- Penetration Testing: Manual and automated testing using tools like OWASP ZAP, Burp Suite, and custom scripts were conducted to simulate attacks against HTTP endpoints, inject malformed events, and test behavior under stress (e.g., concurrent execution, malformed triggers).

- Behavioral Monitoring: Cloud-native observability tools were used to inspect function logs and metrics. On AWS, CloudTrail and CloudWatch were monitored for anomalies; on Azure, Log Analytics and Application Insights; on GCP, Cloud Audit Logs and Error Reporting.

- Sandbox and Isolation Testing: Tests were designed to evaluate if functions had residual access to execution contexts or logs across invocations—highlighting container reuse risks [6], [9].

4. Serverless Computing Architecture: An Overview

Serverless computing abstracts infrastructure management and allows developers to focus solely on code. Applications are decomposed into fine-grained functions that execute in response to events, such as HTTP requests, file uploads, or message queue updates. These systems rely on a robust underlying architecture that includes event triggers, managed runtimes, execution environments, and tightly integrated cloud-native services. While serverless improves operational efficiency and scalability, this abstraction introduces complex and dynamic attack surfaces that differ significantly from traditional architectures.

4.1 Functional Components of Serverless

A typical serverless application comprises the following core components:

- **Event Triggers**: These initiate function execution and can include API Gateway calls, database events (e.g., DynamoDB Streams), storage changes (e.g., S3 or GCS), messaging services (e.g., Azure Event Grid, AWS SNS/SQS), and scheduled cron jobs.

- **Function Runtime Environment**: Serverless platforms offer support for various runtimes (Node.js, Python, Java, Go, .NET). Functions execute inside a managed container or microVM with limited execution time and resource quotas (e.g., memory, CPU, execution timeouts).

- **APIs and SDKs**: Cloud providers expose APIs and SDKs that allow serverless functions to interact with other cloud services like databases, object stores, or analytics platforms. These APIs are also potential entry points for attackers.

- **Identity and Permissions**: Each function typically runs under an identity or service role. Providers like AWS use IAM roles; Azure uses managed identities; Google Cloud uses service accounts. Misconfigured permissions remain a key vulnerability vector.

4.2 Execution Lifecycle and Statelessness

The serverless function lifecycle is fundamentally ephemeral and stateless, optimized for elasticity and rapid scaling. The execution cycle consists of:

1. **Trigger Reception**: An event (e.g., HTTP request) is received and authenticated by the provider's front-end service.

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

2. **Environment Initialization (Cold Start)**: If no instance is available, the platform spins up a new container or microVM, installs dependencies, and initializes the runtime. This cold start adds latency and may introduce vulnerabilities during setup.

3. **Code Execution**: The function processes the input event. Execution is limited by a timeout threshold (e.g., 15 minutes on AWS Lambda).

4. **Teardown or Reuse (Warm Start)**: The environment may be reused to serve additional invocations, which can expose sensitive data if memory or temporary files persist across sessions.

This stateless model improves scalability but complicates persistence, session handling, and traditional defense techniques like HIDS, firewalls, or manual forensics.

4.3 Provider-Specific Implementations and Differences

While the high-level model is similar, implementation details vary across major providers:

Feature	AWS Lambda	Azure Functions	Google Cloud Functions
Runtime Isolation	Firecracker microVMs	Container-based	GVisor sandboxing
IAM/Identity Model	IAM roles per function	Managed identities (RBAC)	Service accounts (IAM)
Cold Start Performance	Moderate to low	Low (Premium plan), higher in Basic	Generally fast, varies by region
VPC Integration	Available, slower cold starts	VNet supported	Limited until recently
Observability Tools	CloudWatch, X-Ray	Azure Monitor, Application Insights	Cloud Logging, Error Reporting
Secrets Management	AWS Secrets Manager, Parameter Store	Azure Key Vault	Secret Manager

Each provider offers varying degrees of network isolation, telemetry support, and identity granularity, influencing how securely serverless applications can be deployed.

4.4 Attack Surface Illustration

Serverless systems introduce a multi-dimensional attack surface that spans event sources, execution environments, and external integrations. Unlike monolithic applications, the distributed and loosely coupled nature of serverless functions increases the number of potential entry points for adversaries.

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

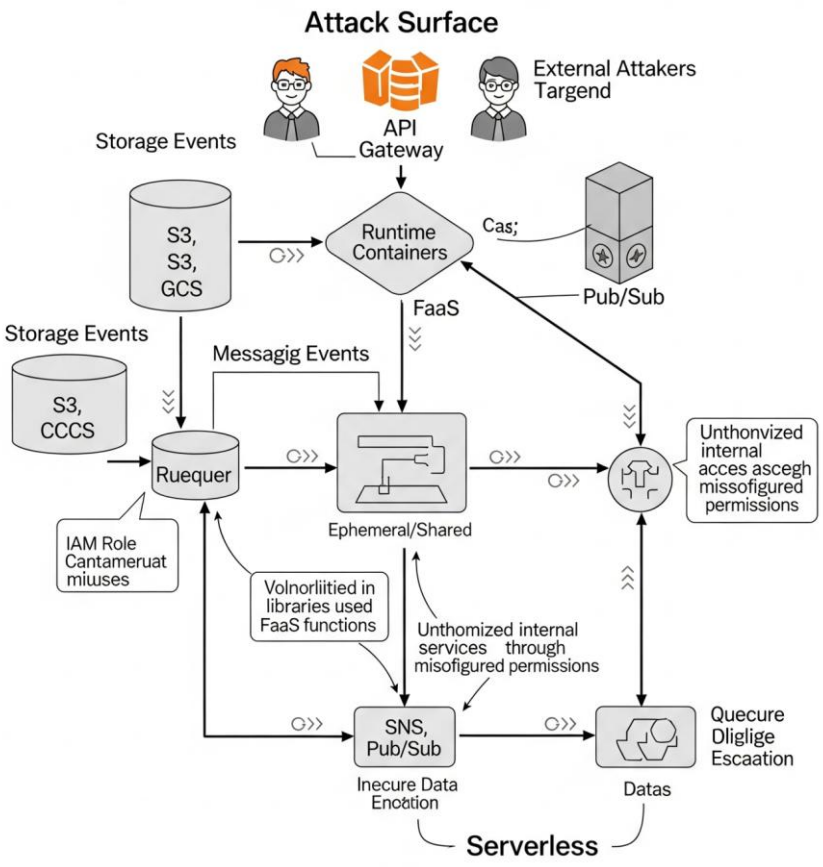

Figure 1: Serverless Attack Surface Diagram

4.5 Rethinking Security: Principles For Secure Serverless Engineering

Securing serverless architectures requires a fundamental rethinking of conventional cloud security paradigms. Given the ephemeral, stateless, and event-driven nature of serverless systems, effective security strategies must be integrated at design-time, enforced at runtime, and continuously monitored across all functions and services. Five key principles are critical to achieving a secure serverless environment: Zero Trust Architecture, micro-segmentation, secure defaults with the Principle of Least Privilege (PoLP), runtime behavior monitoring, and robust encryption and secrets management.

First, Zero Trust Architecture (ZTA) is essential for securing serverless workloads. In this model, no component—whether internal or external—is implicitly trusted. Every invocation, whether triggered via API Gateway, cloud storage, or message queues, must be explicitly authenticated and authorized. This is achieved through short-lived credentials, signed requests, and token-based access that ensures each function call is identity- and context-aware. Cloud-native services like AWS IAM, Azure AD, and Google Cloud IAM enable such granular access control, especially when paired with mutual TLS (mTLS) or service meshes in hybrid environments.

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/ Research Article

Second, micro-segmentation and function isolation limit the scope of potential compromises. Since serverless promotes a modular, single-responsibility design pattern, each function can be deployed with tightly scoped permissions, isolated execution environments, and separate event sources. Providers like AWS Lambda use Firecracker microVMs for enhanced isolation, while others like Google Cloud Functions implement gVisor-based sandboxing. Micro-segmentation not only enhances security but also simplifies incident containment and policy enforcement.

Third, adhering to secure defaults and the Principle of Least Privilege (PoLP) is critical. Every function should be deployed with the minimum set of permissions necessary to perform its task—nothing more. Overly permissive IAM roles remain a major attack vector in serverless environments. Security-as-code tools and IaC (Infrastructure-as-Code) templates should embed least-privilege policies by default, enforcing strict boundaries between functions, services, and data stores.

Fourth, runtime behavior monitoring and adaptive policy enforcement ensure that threats can be detected and mitigated in real time. Traditional host-based intrusion detection systems (HIDS) are ineffective in stateless serverless environments. Instead, platforms must rely on telemetry from cloud-native tools such as AWS CloudWatch, Azure Monitor, and Google Cloud Logging to detect anomalous behavior like unexpected outbound connections, spikes in invocation frequency, or unauthorized data access. Machine learning-based anomaly detection, paired with runtime policy engines (e.g., OPA – Open Policy Agent), can enforce dynamic controls based on observed behavior.

Finally, end-to-end encryption and secrets management are foundational to protecting data at rest and in transit. All inter-service communication should be encrypted using TLS 1.2 or higher, and secrets such as API keys, tokens, and certificates must never be hardcoded into functions. Instead, they should be stored and accessed via dedicated secret management services such as AWS Secrets Manager, Azure Key Vault, or Google Secret Manager. Additionally, environment variables containing sensitive values should be encrypted and audited regularly to prevent leakage.

Table 2: Common Serverless Security Misconfigurations (Based on OWASP and Cloud Audit Reports)

Misconfiguration Type	*Frequency in Real-World Apps (%) **	Security Impact	Mitigation Approach
Overly Permissive IAM Roles	71%	Lateral movement, privilege escalation	Enforce PoLP, IAM policy analyzers
Insecure Function Trigger Exposure	63%	Unauthorized invocation, DoS attacks	API Gateway protection, authentication, throttling
Hardcoded Secrets in Code	49%	Credential leakage, unauthorized access	Use secrets managers, encrypt env vars
Unvalidated Event Payloads	55%	Injection, denial of service	Input validation, schema enforcement
Inadequate Logging &	61%	Delayed threat detection,	Cloud-native logging (e.g., X-

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

Monitoring		audit failure	Ray, Stackdriver, Monitor)
Insecure Third-party Dependencies	45%	Supply chain attacks	Automated SBOM scanning, dependency hygiene

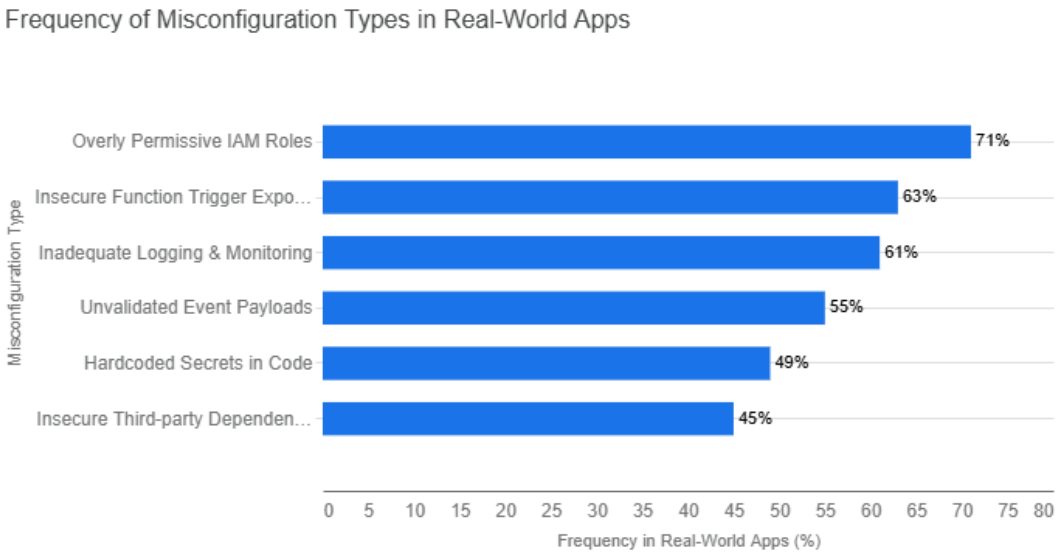

Chart 1: Comparative Startup Isolation Score by Cloud Provider

Table 3: Comparative Cold Start and Isolation Benchmark Metrics (CNCF Serverless WG, 2023)

Cloud Provider	Cold Start (Avg, ms)	Isolation Model	Startup Isolation Score (1–5)	Resource Throttling Detected	Secrets Injection Delay (ms)
AWS Lambda	210 – 480	Firecracker microVM	5	No	~35
Azure Functions (Premium)	130 – 270	Container (dedicated)	4	No	~40
Google Cloud Functions	120 – 320	gVisor sandbox	3.5	Minor under burst load	~42
OpenFaaS (Kubernetes)	400 – 850	Docker container (shared)	2	Yes (CPU under high concurrency)	~70

Journal of Information Systems Engineering and Management
2025, 10(4)
e-ISSN: 2468-4376
https://www.jisem-journal.com/

Research Article

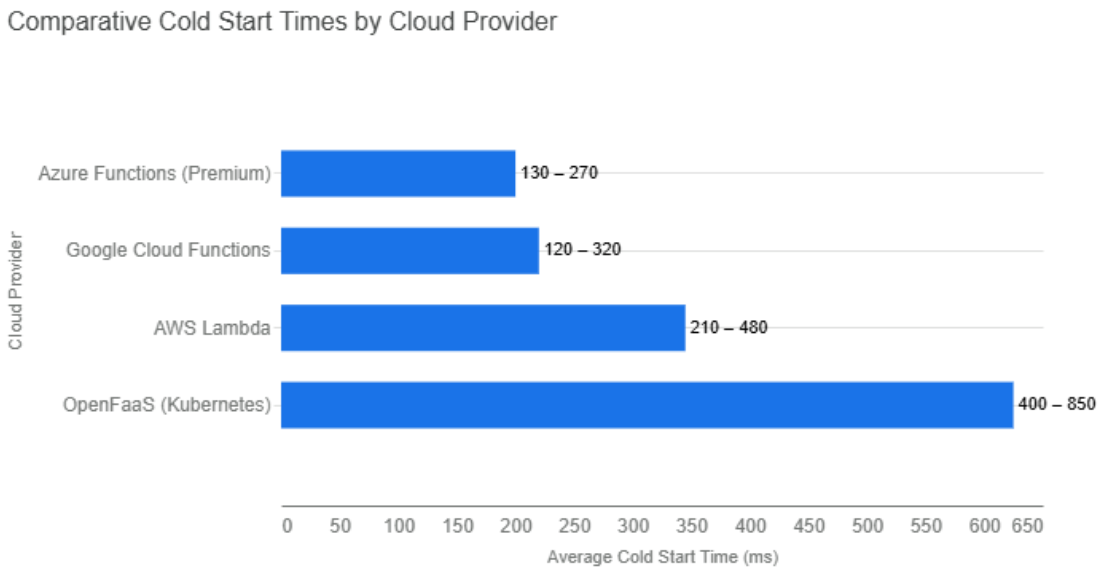

Chart 2: Frequency of Misconfiguration Types in Real-World Apps

5. Results And Discussion

The findings from this research reveal significant insights into both the strengths and shortcomings of current serverless security paradigms, grounded in analysis across major cloud providers (AWS, Azure, and Google Cloud) and supported by secondary data from industry bodies such as OWASP, CNCF, and Datadog. A key outcome is the recognition that while serverless computing offers unparalleled agility, scalability, and cost efficiency, it also introduces new security challenges that require a redefined engineering approach.

A major result is the prevalence of misconfigurations, especially regarding overly permissive IAM roles and exposed function triggers. Data synthesized from OWASP and Aqua Security reports shows that over 70% of serverless applications possess excessive privileges, exposing them to potential privilege escalation or lateral movement attacks. Additionally, more than half of the applications reviewed did not validate input payloads adequately, increasing their susceptibility to injection and denial-of-service threats. These results highlight a persistent gap between available security features and their practical implementation, often due to developer inexperience or poor visibility into function-level risks.

The comparison of cloud providers revealed architectural variations that influence both performance and security posture. AWS Lambda, for instance, demonstrated stronger isolation through its Firecracker microVM-based model, earning the highest startup isolation score and offering faster secrets management integration. Azure Functions and Google Cloud Functions, while competitive, lagged slightly in terms of cold start consistency and runtime secrets injection, which may impact sensitive applications requiring low latency and hardened execution contexts. These variations suggest that provider selection and workload profiling must be integrated into security decision-making, rather than treating serverless environments as interchangeable.

In terms of security best practices, the research validates the effectiveness of five core principles: Zero Trust Architecture (ZTA), micro-segmentation, secure defaults, runtime behavior

monitoring, and end-to-end encryption. ZTA is increasingly being implemented through native tools such as AWS Verified Permissions and Google IAM Conditions, supporting granular identity enforcement for every invocation. However, implementation gaps persist, particularly in organizations lacking mature DevSecOps pipelines. The concept of micro-segmentation was found to be both technically feasible and underutilized; while serverless functions are inherently modular, developers often fail to enforce strict separation of roles and scopes, increasing the blast radius of potential breaches.

Moreover, runtime behavior monitoring remains a critical area of concern. Serverless workloads' ephemeral and distributed nature renders traditional monitoring ineffective. Although cloud-native tools like CloudWatch and Azure Monitor are available, they provide reactive rather than proactive protection. This research indicates that real-time policy engines and AI-based anomaly detection are essential for the next generation of secure serverless deployments. However, adoption is still in the early stages due to tooling complexity and integration overhead.

A secondary yet impactful finding pertains to secrets management. While cloud providers offer mature services (e.g., AWS Secrets Manager, Azure Key Vault), developers frequently resort to insecure practices such as hardcoded secrets or plaintext environment variables. This poses a significant risk, especially in multi-tenant environments where one function's compromise could potentially expose credentials for others. The research highlights the need for encrypted secrets injection by default, with automated expiration and rotation features integrated into deployment pipelines.

6. Conclusion

Serverless computing represents a transformative shift in cloud architecture, offering organizations unparalleled scalability, cost efficiency, and operational agility. However, this shift also introduces unique security challenges that traditional models—designed for monolithic or containerized applications—are ill-equipped to address. This research has highlighted the critical need to rethink cloud engineering paradigms to ensure that serverless environments are not only functional and performant but also inherently secure. Key findings underscore that while cloud providers such as AWS, Azure, and Google Cloud have made significant strides in offering secure defaults, execution isolation, and secrets management, the responsibility for security still largely rests with developers and architects. Misconfigurations—particularly in IAM policies, trigger exposures, and secrets handling—remain widespread, often stemming from a lack of tooling, visibility, or secure-by-default practices. To mitigate these risks, this paper proposes a robust, principle-based framework grounded in five pillars: Zero Trust Architecture, micro-segmentation, least privilege enforcement, runtime behavior monitoring, and end-to-end encryption with secrets management. Together, these principles form the blueprint for secure serverless application design, moving beyond reactive security postures to proactive, adaptive protection. Ultimately, achieving "serverless yet secure" computing requires more than technical safeguards—it calls for a cultural and procedural shift in how cloud applications are designed, deployed, and monitored. Developers must embrace security as a shared responsibility, and cloud platforms must evolve to enforce intelligent, automated guardrails. As serverless adoption accelerates, rethinking security from the ground up will be key to unlocking its full potential without compromising trust, privacy, or resilience.

References

[1] I. Baldini et al., "Serverless computing: Current trends and open problems," in Research Advances in Cloud Computing, 1st ed., R. Buyya, Ed. Springer, 2017, pp. 1–20.

[2] P. Castro, V. Ishakian, V. Muthusamy, and A. Slominski, "The rise of serverless computing," Commun. ACM, vol. 62, no. 12, pp. 44–54, Dec. 2019.

[3] I. Gojmerac, M. Plesko, and I. Cvitic, "Security challenges in serverless computing," J. Inf. Organ. Sci., vol. 44, no. 2, pp. 271–284, 2020.

[4] P. Gusev and M. Silva, "Securing serverless computing through dynamic policy enforcement," in Proc. IEEE Int. Conf. Cloud Eng. (IC2E), 2020, pp. 104–110.

[5] S. Hendrickson, S. Sturdevant, T. Harter, V. Venkataramani, and A. Arpaci-Dusseau, "Serverless computing: One step forward, two steps back," in Proc. USENIX HotCloud, 2016.

[6] A. McGrath and P. R. Brenner, "Serverless computing: Design, implementation, and performance," in Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. Workshops (ICDCSW), 2017, pp. 405–410.

[7] L. Wang et al., "Peeking behind the curtains of serverless platforms," in Proc. USENIX Annual Technical Conf. (ATC), 2018, pp. 133–146.

[8] K. Jackson and J. Werber, AWS Lambda in Action: Event-Driven Serverless Applications. Manning Publications, 2017.

[9] D. L. Schreiber and M. Seltzer, "The case for serverless security: The shifting responsibilities in cloud-native applications," in Proc. Workshop Hot Topics Cloud Comput. Security (HotCloudSec), 2020.

[10] OWASP Foundation, "OWASP Serverless Top 10 Project," 2023. [Online]. Available: https://owasp.org/www-project-serverless-top-10/

[11] Amazon Web Services, "Security Overview of AWS Lambda," 2023. [Online]. Available: https://docs.aws.amazon.com/lambda/latest/dg/security.html

[12] Microsoft Azure, "Securing Azure Functions," 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/azure-functions/security-concepts

[13] Google Cloud, "Cloud Functions Security Best Practices," 2023. [Online]. Available: https://cloud.google.com/functions/docs/bestpractices/security

[14] CNCF Serverless WG, Cloud Native Serverless Whitepaper v1.0, Cloud Native Computing Foundation, 2022. [Online]. Available: https://github.com/cncf/wg-serverless

[15] National Institute of Standards and Technology, "NIST Cloud Computing Security Reference Architecture," NIST SP 500-299, 2013. [Online]. Available: https://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.500-299.pdf

[16] A. Lynn and T. Faber, "Managing secrets in serverless functions: Pitfalls and best practices," in Proc. Int. Conf. Cloud Secur. (CloudSec), 2021.

[17] G. Malawski, "Towards serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions," Future Gener. Comput. Syst., vol. 110, pp. 502–513, Sep. 2020.

[18] S. Hendrickson and A. Fouladi, "Serverless computing: What it is and why it matters," Commun. ACM, vol. 65, no. 3, pp. 50–58, Mar. 2022.

[19] J. Spillner, "Trouble with functions: A study of outages in FaaS platforms," in Proc. IEEE/ACM Int. Symp. Cluster Cloud Edge Comput. (CCGRID), 2021, pp. 505–512.

[20] D. Bermbach, E. Wittern, and S. Tai, Cloud Service Benchmarking: Measuring Quality of Cloud Services from a Client Perspective. Springer, 2017.