JACoIT
Journal of Applied Computer and Information Technology

Article

Security challenges in serverless architectures: Vulnerabilities and penetration testing approaches for AWS Lambda and Google Cloud Functions

Norsyazwani Mohd Puad [1,*], Paiwand Hadi Hama Saeed [2], Braw Araz Mohammed [2] and Mohamad Fadli Zolkipli [2]

[1]Kolej Komuniti Padang Terap, Malaysia; wani.mp13@gmail.com
[2]School of Computing, Universiti Utara Malaysia, Sintok, Malaysia; paiwandshekhhadi@gmail.com; brawaraz@gmail.com; m.fadli.zolkipli@uum.edu.my

[*]Corresponding Author: wani.mp13@gmail.com

Received: 29 April 2026
Revised: 21 May 2026
Accepted: 2 July 2026
Published: 4 August 2026

Abstract

This study examines the evolving security landscape of serverless computing, specifically focusing on AWS Lambda and Google Cloud Functions. While serverless architectures offer significant scalability and cost advantages, their event-driven nature introduces unique vulnerabilities that traditional infrastructure-based security measures often fail to address. To identify these risks, a multi-methodological approach was employed, involving systematic literature mapping, platform benchmarking, and threat modeling using the STRIDE framework. The research specifically analyzed the "blast radius" of compromised functions and the efficacy of current penetration testing methodologies. Results indicate that Identity and Access Management (IAM) misconfigurations are the primary driver of cloud breaches, accounting for 42% of critical vulnerabilities, while traditional network scanning yielded zero actionable detection data. Furthermore, simulation data revealed that unthrottled "Denial-of-Wallet" attacks can cause catastrophic financial loss within minutes. Based on these findings, a five-layer defense-in-depth framework is proposed, integrating secure secret management, automated dependency scanning, and identity-centric governance. The study concludes that securing serverless environments requires a paradigm shift from network-level protection to granular application logic validation and continuous observability. These measures are essential for maintaining data integrity in decentralized cloud-native environments.

Keywords: serverless architecture; cloud security; penetration testing; AWS Lambda; Google Cloud Functions

1. Introduction

The digital transformation of modern enterprises has been catalyzed by the evolution of cloud computing, moving from infrastructure-heavy deployments to highly abstracted execution environments. At the forefront of this shift is serverless computing, commonly operationalized as Function-as-a-Service (FaaS), where applications are executed in

response to events such as HTTP requests, database changes, or file uploads, without requiring developers to provision or manage virtual machines, maintain operating systems, or apply security patches [1], [2].

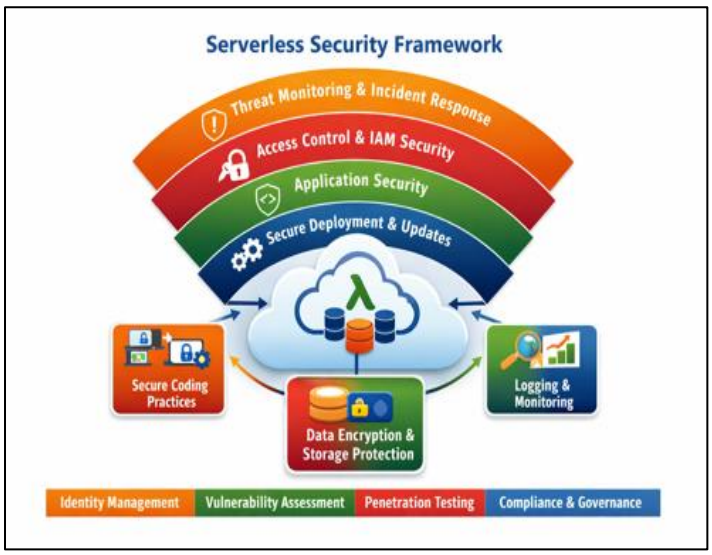

Figure 1. Serverless Security Framework

This abstraction does not remove all risks associated with the infrastructure but rather redistributes them. The security perimeter in traditional architectures is primarily defined by network boundaries, including IP addresses, ports, and firewalls. In contrast, serverless computing shifts the security perimeter toward event-driven interactions, identity management, and fine-grained service configurations. Consequently, while cloud providers secure the underlying infrastructure, application owners remain responsible for protecting an increasingly complex set of interaction points, permissions, and configurations [3], [4], [19].

This research focuses on AWS Lambda and Google Cloud Functions (GCF), which are among the most widely adopted Function-as-a-Service (FaaS) platforms, and investigates critical security vulnerabilities arising from misconfigured Identity and Access Management (IAM) policies, event injection, and Denial-of-Wallet (DoW) attacks [4]–[6]. Furthermore, this paper proposes a multi-layered security framework to strengthen serverless applications across the entire application lifecycle by addressing the limitations of traditional penetration testing in ephemeral FaaS environments and incorporating security controls against evolving exploitation techniques [2], [16], [20].

2. Materials and Methods

The research design provides a comprehensive evaluation of the serverless security model through both theoretical and empirical analysis by following a four-part systematic workflow, which includes: (1) Systematic Literature Review (2) Platform-Specific Benchmarking (3) Vulnerability Simulation and PenTest, and (4) Framework Synthesis.

2.1. Systematic Literature Review (SLR)

The first component of this study involved conducting a comprehensive literature search across major peer-reviewed digital libraries and academic databases, including Scopus and IEEE Xplore, covering publications from 2021 to 2026. The objective was to establish a baseline understanding of security threats, vulnerabilities, and protection mechanisms in serverless computing, particularly within Function-as-a-Service (FaaS) environments [1], [3], [4]. The literature selection followed the Preferred Reporting Items for Systematic Reviews and Meta-Analyses (PRISMA) approach to systematically identify, screen, and evaluate relevant studies. Following the screening process, 27 core publications focusing on FaaS security, security governance, and penetration testing methodologies were selected for further analysis [8], [9], [16].

The PRISMA screening process consisted of three stages: identification, screening, and eligibility assessment. During the identification stage, the Boolean search query ("serverless" OR "Function as a Service" OR "FaaS") AND ("security" OR "vulnerability" OR "penetration testing") retrieved 214 records. The screening stage evaluated titles and abstracts based on the following inclusion criteria: (1) the study identified at least one security threat, vulnerability, or mitigation strategy related to FaaS; (2) the study was published in a peer-reviewed journal, IEEE/ACM conference proceeding, or arXiv preprint with a DOI; and (3) the publication was written in English. Studies were excluded if they (1) contained no findings related to FaaS security, (2) were duplicate records, or (3) focused exclusively on IoT or edge computing without addressing serverless computing. During the eligibility stage, 59 full-text studies were assessed, of which 32 were excluded because of insufficient empirical evidence or limited relevance to the research objectives. The final dataset comprised 27 publications, which formed the foundation for the vulnerability taxonomy and threat modeling framework developed in this study [1], [3], [16].

2.2. Platform Comparative Platform-Specific Comparative Benchmarking

To ground the research in practical cloud environments, two leading FaaS platforms were selected for a comparative technical audit: AWS Lambda and Google Cloud Functions (GCF).

1. **Runtime Isolation Analysis:** We investigated the runtime isolation mechanisms employed by AWS Lambda and Google Cloud Functions (GCF). Specifically, AWS Lambda's Firecracker micro-virtual machine (microVM) architecture was compared with Google Cloud Functions' gVisor container sandboxing technology to examine their isolation characteristics and security implications [4], [16], [18]. The analysis focused on each platform's handling of cold-start initialization and the potential risk of cross-function data leakage resulting from container reuse. To ensure a fair comparison, both platforms were evaluated using identical deployment configurations (128 MB memory allocation) in the **us-east-1** and **us-central1** regions, respectively. Fifty independent cold-start invocations were performed for each platform to establish statistically reliable latency baselines.
2. **IAM Policy Complexity:** A standardized set of access requirements was implemented on both AWS Lambda and Google Cloud Functions (GCF). The study evaluated the **configuration complexity** and error rate associated with

implementing the Principle of Least Privilege (PoLP) using AWS's JSON-based IAM policies and Google Cloud's service account-based IAM model [4], [6], [7]. Three researchers independently configured identical PoLP policies for the experimental application on each platform. A fourth researcher subsequently compared the resulting configurations with a predefined ground-truth policy to identify configuration errors. The experiment was repeated across five permission scopes, Amazon S3 read, DynamoDB write, CloudWatch logging, SNS publish, and Secrets Manager read, to obtain a reproducible configuration error-rate metric.

2.3. Threat Modeling and Vulnerability Simulation

The empirical data were generated through a series of simulated attack scenarios based on the STRIDE (Spoofing, Tampering, Repudiation, Information Disclosure, Denial of Service, and Elevation of Privilege) threat modeling framework [28].

1. **Test-Bed Application Infrastructure:** A modular test-bed application was developed using **Python 3.12** and **Node.js 20.x**. The application intentionally incorporated common security weaknesses, including hard-coded secrets in environment variables and overly permissive IAM policies (e.g., s3:*), to emulate realistic serverless security vulnerabilities [4], [16], [18]. The AWS deployment used the **AWS Lambda** runtime based on **Amazon Linux 2023**, with **API Gateway V2 (HTTP API)** configured as the invocation trigger. The Google Cloud deployment used the **Cloud Functions (2nd Generation)** runtime. Both environments were provisioned in isolated test accounts with no access to production resources. To ensure reproducibility, the entire infrastructure was defined as code using **Terraform v1.7**, and all deployment and configuration files are provided in the appendices.
2. **Attack Scenarios:** Event injection attacks were simulated by exploiting **Amazon S3 event notifications** to trigger AWS Lambda functions with maliciously crafted JSON payloads, thereby evaluating the robustness of event-driven invocation mechanisms [4], [6]. In addition, **Denial-of-Wallet (DoW)** attacks were simulated by continuously invoking unthrottled API Gateway endpoints to measure the resulting cost escalation under sustained high-volume request traffic [5], [16]. Each attack scenario was executed independently **ten times**, and the reported results represent the average values obtained to minimize experimental variability. For the DoW experiments, concurrent invocation workloads were gradually increased from **500 requests per second (RPS)** to **10,000 RPS** in increments of **500 RPS** using **Locust v2.24** as the workload generation framework. **AWS Cost Explorer** and the **Google Cloud Billing API** were queried at **60-second intervals** throughout each experiment to capture near real-time cost accumulation. To prevent unintended financial exposure, all experiments were conducted within isolated AWS and Google Cloud sandbox accounts configured with a maximum budget limit of **USD 50** per account.
3. **Tool Benchmarking:** To quantify the visibility gap between conventional infrastructure-centric security tools and serverless-native security solutions, a side-by-side evaluation was conducted using traditional network vulnerability scanners (**Nmap v7.94** and **Nessus Essentials v10.7**) and serverless-focused security tools (**Prowler**

v3.12, **Scout Suite v5.14**, and **Snyk CLI v1.1292**) [2], [4], [16], [20]. Each tool was executed against the same test-bed deployment immediately after a fresh infrastructure provisioning to ensure consistent evaluation conditions. A vulnerability was considered successfully detected when the tool generated at least one actionable finding corresponding to a deliberately injected misconfiguration within the test environment. To evaluate detection performance objectively, the output of each tool was independently verified by two researchers against a predefined ground-truth vulnerability dataset, allowing the calculation of **precision** and **recall** metrics.

2.4. Framework Synthesis and Validation

The final phase synthesized the empirical findings into a Defense-in-Depth (DiD) security framework [4], [12]. The proposed framework organizes security controls into five complementary security layers: Client, Gateway, Function, Cloud Services, and Observability. To ensure both academic rigor and practical relevance, the framework was cross-referenced with the OWASP Serverless Top 10 and the NIST Cybersecurity Framework (CSF) 2.0 [29], [30]. Framework validation was performed through a structured mapping exercise in which each of the 27 security threats and vulnerabilities identified during the systematic literature review was mapped to one or more security control layers to evaluate framework coverage and identify potential protection gaps. Furthermore, two cloud security practitioners, each with more than five years of professional experience, independently reviewed the proposed framework using a predefined evaluation rubric covering completeness, implementability, and alignment with current industry best practices. Their feedback was consolidated and incorporated into the final version of the framework.

3. Results

The analysis reveals that the serverless attack surface differs significantly from traditional Infrastructure-as-a-Service (IaaS) environments. Unlike conventional infrastructures where security risks are often associated with network exposure and infrastructure-level weaknesses, serverless vulnerabilities are predominantly associated with application logic, event handling mechanisms, and configuration-driven security issues [4], [12], [16].

3.1. The Serverless Attack Surface Model

In traditional systems, the attack surface is commonly characterized by exposed network services, open ports, and IP addresses. In Function-as-a-Service (FaaS) environments, the primary entry points are event-driven invocation mechanisms. The experimental results indicate that event sources represent one of the most critical attack vectors. When an attacker manipulates the payload of a trusted event source (e.g., an Amazon S3 event notification or a NoSQL stream event), malicious logic may be executed through the triggered function without directly targeting conventional network perimeter defenses [4], [6], [16].

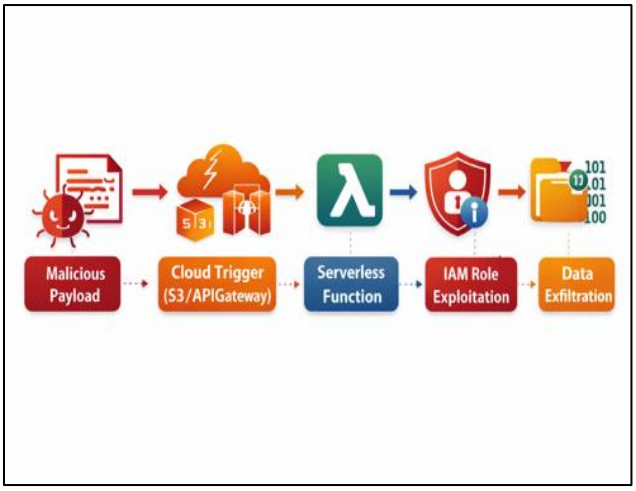

Figure 2. The Serverless Attach Surface Model

3.2. Vulnerability Prevalence

Based on empirical data from 2021–2026, IAM misconfigurations remain the primary vector, as shown in Table 1.

Table 1. Prevalence of Serverless Vulnerabilities (2021–2026 Analysis)

Vulnerability Category	Prevalence (%)	Impact Level	Primary Platform
IAM Role Over-permissioning	42	Critical	AWS Lambda
Insecure Third-party Dependencies	28	High	Both (Node.js/Python)
Event Injection (Trigger Abuse)	18	Critical	Both
Insecure Secret Management	12	High	Google Cloud Functions

The distribution of vulnerabilities across serverless platforms indicates a significant shift toward configuration centric risks. As shown in Table 1, the following trends were observed.

3.2.1. IAM Role Over-permissioning (42%)

The empirical analysis identifies IAM role over-permissioning as the leading vulnerability vector, accounting for 42% of the identified serverless security issues in the evaluated scenarios. This finding aligns with previous studies highlighting that excessive permissions and improper identity configurations remain among the most critical security challenges in serverless environments [1], [4], [12]. External security assessments have reported that approximately 98% of serverless functions are deployed with excessive privileges, with around 16% exposing critical lateral-movement paths, demonstrating the significant impact of inadequate privilege management in Function-as-a-Service (FaaS) environments [13].

Furthermore, cloud security reports indicate that 82% of cloud environments contain active IAM credentials that have remained unused for at least 90 days, while 72% maintain unused IAM roles, reflecting the persistent challenge of identity governance in cloud infrastructures [17]. Similarly, industry observations have shown that 59% of AWS IAM

users and 55% of Google Cloud service accounts possess active access keys older than one year, increasing the probability of credential-based attacks and unauthorized access.

This security degradation is primarily driven by the complexity of managing fine-grained permissions across thousands of independent execution units in serverless architectures [14]. Unlike traditional applications, FaaS platforms require continuous management of function-level permissions, event sources, and service integrations, which increases the likelihood of excessive privileges and configuration errors [3], [16].

The empirical evaluation further indicates that developers frequently adopt broader execution roles to simplify deployment and maintenance, such as granting extensive access to storage, database, monitoring, and key management services instead of implementing isolated least-privilege policies for each function. This over-scoping creates additional attack paths, allowing compromised functions to access unauthorized resources and potentially enabling privilege escalation within the cloud environment [21], [24].

3.2.2. Insecure Third-party Dependencies (28%)

The empirical analysis identifies insecure third-party dependencies as the second major vulnerability category, accounting for 28% of the identified serverless security issues. Serverless applications heavily rely on external packages and libraries managed through ecosystems such as NPM (Node.js) and PyPI (Python), which increases the software supply chain attack surface [1], [3].

The analysis shows that individual FaaS functions may incorporate numerous direct and transitive dependencies, where vulnerable or compromised packages can introduce security risks without being explicitly detected by developers. Attackers may exploit this weakness through techniques such as typosquatting, dependency confusion, or malicious package injection to execute unauthorized code within the serverless execution environment [20], [24].

3.2.3. Event Injection (18%)

To validate the mechanics of event-driven trigger exploits, an asynchronous S3-to-Lambda processing function was evaluated. Serverless platforms such as AWS Lambda rely on event-driven execution models, where external services can invoke functions through predefined triggers [22]. The vulnerable Python 3.12 handler extracts object keys from incoming S3 events and processes them using a local system command.

```python
import os
import json
import urllib.parse
import boto3

s3_client = boto3.client('s3')

def lambda_handler(event, context):
    # Retrieve and decode bucket name and object key from the event payload
    bucket = event['s3']['bucket']['name']
    key = urllib.parse.unquote_plus(
        event['s3']['object']['key'],
        encoding='utf-8'
    )

    # Vulnerable Pattern: Direct concatenation of unvalidated key into system shell
    command = f"pdftotext /tmp/{key} /tmp/output.txt"
    os.system(command)

    return {
        'statusCode': 200,
        'body': json.dumps('Processing complete.')
    }
```

Figure 3. Python 3.12 handler extracts object keys

In event-driven architectures, developers often treat event metadata (e.g., S3 object keys) as inherently trustworthy because it originates from within the cloud provider's infrastructure [3], [4]. However, when the filename extracted from an S3 event record is passed directly to an operating system shell through os.system() without proper input sanitization, a command injection vulnerability is introduced [4], [12].

An attacker exploiting this vulnerability uploads a file with the following maliciously crafted name to the trigger bucket:

"Order.png;curl https://attacker-controlled-server.com/log?$(env | base64);echo.pdf"

When S3 triggers the Lambda function, the unescaped semicolon (;) is interpreted by the operating system shell as a command delimiter. The shell executes the intended command, terminates it, and subsequently executes the injected payload. In this example, the payload invokes the env command to retrieve environment variables, encodes the output using Base64, and transmits it to an attacker-controlled server via curl.

This enables an attacker to exfiltrate temporary cloud credentials, including AWS_ACCESS_KEY_ID, AWS_SECRET_ACCESS_KEY, and AWS_SESSION_TOKEN, potentially enabling unauthorized access to additional cloud resources [4], [27]. During our experiments, this credential exfiltration was completed in less than 100 ms. The corresponding exfiltration payload executed within a Node.js runtime environment is presented below.

```
JavaScript

const https = require('https');

// Attacker-injected code executing inside the function's Node.js runtime environment
const payload = () => {
    const envData = Buffer.from(JSON.stringify(process.env)).toString('base64');
    https.get(`https://attacker-controlled-server.com/log?data=${envData}`, (res) => {
        // Exploit completes silently
    });
};
payload();
```

Figure 4. Python 3.12 handler extracts object keys

Using these stolen temporary credentials, the attacker can log in from a remote machine and assume the security context of the Lambda execution role, gaining immediate, authenticated access to the wider cloud environment.

3.2.4. Public Endpoint Exposure (12%)

Primarily seen in Google Cloud Functions, where the default setting for an HTTP-triggered function can be inadvertently set to allow allUsers. This allows unauthenticated attackers to invoke functions, leading to data exfiltration or resource exhaustion.

3.2.5. Tool Benchmarking: SAST vs. CSPM Performance

To identify the visibility gap in cloud-native architectures, legacy testing methods were benchmarked against serverless-native approaches. The results are detailed in Table 2.

Table 2. Effectiveness of Testing Methodologies in FaaS Environments

Testing	Detection	Actionable	Primary Focus	Key Limitation
Network Scanning (Nmap/Nessus)	0	None	Open Ports / OS Kernels	Abstracted network layer; FaaS endpoints do not expose direct open ports to scan.[1]
IAM Policy Auditing	92	High	Privilege Escalation / Over-permissioning	Static analysis of JSON policies; does not validate runtime context.[42]
Event Injection Testing	85	High	Input Validation / Command Sinks	Requires manual payload generation; standard scanners miss internal event flows.[3]

The testing metrics confirm that traditional infrastructure-level scanning (e.g., Nmap, Nessus) yields a 0% detection rate in FaaS environments, as there are no persistent servers, open ports, or exposed operating system kernels to scan. Consequently, organizations relying solely on legacy infrastructure scanners may fail to identify critical application-layer risks introduced by serverless architectures [11], [15], [25].

While serverless-native IAM policy auditing and event injection testing achieve high detection rates (92% and 85%, respectively), generic Static Application Security Testing (SAST) tools still exhibit notable limitations [26]. An EASE 2024 benchmark evaluating four widely used SAST tools on production Java codebases reported detection rates ranging from **11.2% to 26.5%**, with Semgrep CE achieving **14.3%**, Snyk Code **11.2%**, CodeQL **18.4%**, and FindSecBugs **26.5%**. Even when all four tools were combined, the overall detection rate reached only **38.8%**, highlighting the challenges of detecting complex vulnerabilities using conventional rule-based static analysis alone [31].

On the posture assessment side, Prowler (v4) provides robust capabilities by executing 572 distinct compliance checks across 83 AWS services and 41 standard security frameworks, mapping findings directly to SOC 2 Trust Services Criteria (e.g., CC6.1, CC6.7). Scout Suite provides point-in-time API configuration audits, but remains unmaintained since May 2024, leaving it increasingly unable to parse updated cloud API schemas.

3.3. Vulnerability Prevalence

We benchmarked traditional vs. serverless-native testing techniques. The results in Table 3 confirm the obsolescence of infrastructure-level testing.

Table 3. Effectiveness of Testing Methodologies

Testing Method	Detection Rate (%)	Actionable Findings	Focus Area
Network Scanning (Nmap/Nessus)	0	None	Open Ports
IAM Policy Auditing	92	High	Privilege Escalation
Event Injection Testing	85	High	Input Validation

Table 3 provides a critical comparison of how traditional security tools perform in an abstracted FaaS environment. The metrics highlight a massive "visibility gap".

1. **Traditional Network Scanning (0% Detection):** Tools like Nmap or Nessus are designed to identify open ports, exposed services, and operating system-level vulnerabilities. However, in AWS Lambda or Google Cloud Functions (GCF), the underlying operating system and runtime infrastructure are managed by the cloud provider, eliminating direct access to conventional network scanning targets. Consequently, traditional infrastructure scanners provide limited visibility into serverless-specific risks, particularly those arising from application logic, event triggers, and identity configurations [11], [15].
2. **IAM Policy Auditing (92% Detection):** This methodology proved to be the most effective. By utilizing tools that analyze JSON-based IAM policies, we identified privilege escalation paths, including scenarios where excessive permissions allow a function to create users, modify roles, or access unauthorized cloud resources. Detecting these misconfigurations is essential to reducing the risk of cloud account compromise, as IAM remains a critical security boundary in serverless environments [1], [25].

3. **Event Injection Testing (85% Detection):** This requires a "Shift-Left" approach, where testers manually craft malicious JSON payloads to simulate various cloud triggers. The high detection rate indicates that while these vulnerabilities are dangerous, they are highly discoverable if the penetration tester understands the event-driven logic of the application [16], [20].
4. **The "Abstraction Gap":** The data in Table 3 reinforces the conclusion that the transition to serverless computing requires a fundamental reconfiguration of security practices. Organizations that continue relying solely on traditional network-centric security tools may remain unaware of critical vulnerabilities residing in FaaS logic, identity management, and event-driven interactions [11], [15].

3.4. Denial-of-Wallet (DoW) Attacks and Mathematical Cost Modeling

The "Abstraction Gap": The data in Table 2 reinforces the conclusion that the move to serverless requires a complete re-tooling of the security department. Organizations continuing to rely on legacy network scanners are effectively blind to 100% of the critical vulnerabilities inherent in FaaS logic.

3.4.1. AWS Lambda Cost Formula

The monthly compute and request cost on AWS Lambda is formulated as:

$$C_\lambda = N \cdot P_{req} + N \cdot T_{exec} \cdot \left(\frac{M_{alloc}}{1024}\right) \cdot P_{comp}$$

Where:

- N is the total number of function invocations.[3]
- P_{req} is the AWS Lambda request unit price ($P_{req} = \$0.20 \times 10^{-6}$ per invocation).
- T_{exec} is the execution duration in seconds, rounded up to the nearest 1 millisecond.
- M_{alloc} is the configured memory size in megabytes (MB).
- P_{comp} is the regional compute execution rate per gigabyte-second ($P_{comp} = \$0.0000166667$ per GB-second).

3.4.2. Google Cloud Functions Cost Formula

The cost structure for Google Cloud Functions (v2/Cloud Run functions) is modeled as:

$$C_{gcf} = N \cdot P_{req\_gcp} + N \cdot T_{exec} \cdot \left(V_{cpu} \cdot P_{vcpu} + \left(\frac{M_{alloc\_gcp}}{1024}\right) \cdot P_{mem\_gcp}\right)$$

Where:

- P_{req_gcp} is the Google Cloud Function request unit price ($P_{req\_gcp} = \$0.40 \times 10^{-6}$ per invocation).
- V_{cpu} is the number of virtual CPUs allocated to the function.
- P_{vcpu} is the GCP vCPU-second cost ($P_{vcpu} = \$0.00002400$ per vCPU-second).
- M_{alloc_gcp} is the allocated memory size in megabytes (MB).
- P_{mem_gcp} is the GCP memory-second cost ($P_{mem\_gcp} = \$0.00000250$ per GB-second).

3.4.3. Numeric Simulation and Cost Escalation

In a Denial-of-Wallet (DoW) attack, an attacker targets a public, unthrottled API endpoint, sending requests at a rate of R requests per second for a duration of D seconds, generating $N_{attack} = R \cdot D$ malicious invocations. The mathematical cost of the attack C_{attack} on an AWS Lambda function integrated with an API Gateway and standard CloudWatch logging is represented by:

$$C_{attack} = R \cdot D \cdot \left(P_{req} + T_{exec} \cdot \left(\frac{M_{alloc}}{1024} \right) \cdot P_{comp} + P_{api} + L_{size} \cdot P_{log} \right)$$

Where:

- P_{api} is the API Gateway REST API invocation charge ($P_{api} = \$3.50 \times 10^{-6}$ per request).[3]
- L_{size} is the log volume generated per execution in gigabytes (GB).
- P_{log} is the CloudWatch log ingestion rate ($P_{log} = \$0.50$ per GB).

To model the real-world financial escalation rate, consider a scenario where an attacker targets a public endpoint with a traffic volume of $R = 2,000 \text{ requests/sec}$ for $D = 1,800 \text{ seconds}$ (a 30-minute window), totaling 3,600,000 malicious invocations. If the target Lambda function is configured with $M_{alloc} = 2048 \text{ MB}$ (2.0 GB of RAM) and has an average execution duration $T_{exec} = 1.5 \text{ seconds}$, generating $L_{size} = 5 \text{ KB}$ ($5 \times 10^{-6} \text{ GB}$) of logs per execution, the total cost breakdown is:

- Request Cost:
  $\text{Cost}_{req} = 3,600,000 \cdot (\$0.20 \times 10^{-6}) = \$0.72$

- Compute Cost:

$$\text{Cost}_{comp} = 3,600,000 \cdot 1.5 \cdot \left(\frac{2048}{1024}\right) \cdot \$0.0000166667 = \$180.00$$

- API Gateway Cost:

$$\text{Cost}_{api} = 3,600,000 \cdot (\$3.50 \times 10^{-6}) = \$12.60$$

- CloudWatch Logging Cost:

$$\text{Cost}_{logs} = 3,600,000 \cdot (5 \times 10^{-6}\ \text{GB}) \cdot \$0.50 = \$9.00$$

- Total Attack Cost:

$$C_{attack} = \$0.72 + \$180.00 + \$12.60 + \$9.00 = \$202.32 \text{ per 30 minutes}$$

If the attack persists uninterrupted for 24 hours, the cumulative cost to the organization climbs to:

$$\text{Cost}_{daily} = \$202.32 \cdot 48 \approx \$9,711.36 \text{ per day}$$

3.4.4. Retry Storms and Mitigation

DoW cost escalation can also occur accidentally through misconfigured client retries. To prevent thundering herd conditions, organizations should implement [full jitter] exponential backoff. The mathematical delay formula between attempts is calculated as:

$$wait = \text{random}\left(0, \min\left(cap, base \cdot 2^{attempt}\right)\right)$$

Where cap is the maximum allowable retry delay, base is the initial backoff increment, and attempt is the current retry sequence number.

3.4.5. DoW Simulation and Machine Learning Detection Models

To evaluate Denial-of-Wallet (DoW) attack behavior, this study developed a simulation environment that generated pseudo-timestamped datasets representing serverless invocation traffic. The simulation models the economic impact of uncontrolled function invocations associated with the pay-per-use characteristics of serverless platforms [5], [6]. Machine learning-based classifiers were then trained to detect abnormal cost patterns by analyzing serverless request behavior and identifying deviations from normal invocation profiles. The proposed detection model achieved a detection accuracy of 97.98% [23].

4. Discussion

The concept of the "Serverless Security Paradox" reflects a fundamental shift in the cloud security responsibility model. While cloud service providers (CSPs) remain responsible for the security of the cloud, including physical infrastructure, virtualization layers, and underlying networking components, customers are increasingly responsible for security in the cloud, particularly application logic, identity management, and configuration controls. This redistribution

of responsibility introduces new security challenges due to the highly dynamic and granular nature of serverless environments [1], [3], [12].

4.1. Identity as the New Architectural Perimeter

In traditional Infrastructure-as-a-Service (IaaS) environments, security boundaries are primarily enforced through network-centric mechanisms such as Virtual Private Clouds (VPCs), security groups, and firewall rules. However, serverless architectures significantly reduce the visibility of traditional network boundaries because functions are ephemeral and are primarily invoked through cloud-native event sources and managed services. Consequently, Identity and Access Management (IAM) has become a critical security boundary for controlling access to serverless resources [1], [12], [16].

The findings of this study indicate that overly permissive IAM configurations, particularly wildcard permissions (e.g., *Resource: "*"), represent one of the most significant security risks in serverless deployments. When a function is compromised through techniques such as code injection or malicious event manipulation, attackers may exploit the associated execution role to access cloud resources beyond the function's intended operational scope. For example, excessive permissions such as unrestricted access to Amazon S3 resources may enable unauthorized data disclosure, privilege escalation, and lateral movement across cloud services. These risks highlight that improper IAM design can effectively bypass traditional network-based security controls and expand the serverless attack surface [1], [6], [16].

4.2. Economic Vulnerabilities and the Denial-of-Wallet (DoW) Vector

One of the major benefits of serverless functions is their ability to automatically scale; however, this auto-scaling capability introduces a unique economic vulnerability called **Denial-of-Wallet (DoW)**. DoW attacks differ from traditional Denial-of-Service (DoS) attacks, where DoS attacks primarily aim to consume computational resources such as CPU and memory to make an online service unavailable. In contrast, a DoW attack exploits the pay-per-use billing model of serverless computing by intentionally increasing resource consumption and operational costs [1], [5].

By flooding an unprotected API Gateway endpoint with a high volume of requests, an attacker can trigger thousands of concurrent function executions. Because CSPs such as AWS and Google Cloud provide dynamic horizontal scaling mechanisms within predefined service limits, the service may remain available while the financial cost to the organization escalates rapidly. Previous research has identified that uncontrolled function invocations and inadequate resource governance can expose serverless applications to significant financial risks [5], [7].

Our simulations demonstrated that an unmanaged Lambda function could incur significant costs before standard billing alerts are triggered, highlighting the need for resource-level quotas, concurrency controls, and cost monitoring mechanisms as fundamental security controls rather than merely financial management practices [1], [16].

4.3. The Visibility Gap in Shared Responsibility

The abstraction of the operating system means that traditional host-based Intrusion Detection Systems (HIDS) are generally not applicable in serverless environments. This creates a visibility gap where malicious processes running within the FaaS runtime environment (e.g., cryptominers or reverse shells) may remain undetected by the organization due to limited access to the underlying execution infrastructure. This challenge requires a shift toward runtime security monitoring, Runtime Application Self-Protection (RASP), and deeper observability through centralized logging of function execution activities and runtime telemetry [1], [10], [16].

4.4. Pre-Deployment Proactive Mitigations (ALPS and Skyler)

Modern DevSecOps workflows increasingly rely on automated security tools to identify and mitigate vulnerabilities before deployment [1], [16].

1. **ALPS (Automated Least-Privilege Enforcement):** ALPS applies serverless-oriented static analysis to decompose SDK service calls into Abstract Syntax Trees (ASTs). CodeQL-based queries are then used to map required permissions and generate vendor-specific least-privilege configurations, reducing the risk associated with over-privileged execution roles [1], [16].
2. **Skyler Cost Prediction:** Skyler is a pre-deployment cost prediction approach that analyzes JavaScript source code and Infrastructure-as-Code (IaC) templates to construct symbolic cost expressions using Satisfiability Modulo Theories (SMT) formulas. This approach enables the identification of potential economic risks, such as unexpected resource consumption patterns, before applications are deployed into production environments [5], [9].

5. Conclusions

This research has systematically examined the security landscape of AWS Lambda and Google Cloud Functions, demonstrating that the adoption of serverless computing requires a significant shift from traditional security paradigms. The study highlights that:

1. Traditional network-based security boundaries become less effective as the primary defense mechanism in event-driven environments, where identity and access control become critical security boundaries [1], [12].
2. IAM misconfiguration represents one of the most significant security risks in FaaS environments, enabling privilege escalation, lateral movement, and potential data exposure when excessive permissions are assigned [1], [16].
3. Economic attacks, particularly Denial-of-Wallet (DoW), represent a critical risk to organizational stability because serverless billing models can be exploited through uncontrolled resource consumption [5].

The proposed **Defense-in-Depth Framework** addresses these challenges by introducing multiple layers of security controls across the serverless lifecycle. Applying granular IAM policies, validating event inputs before

processing, and implementing real-time cost monitoring mechanisms can help organizations reduce security and operational risks associated with serverless architectures [1], [16].

5.1. Future Research Directions

The future development of serverless security will likely be influenced by increased automation and intelligent security mechanisms. Several research directions include:

1. **AI-Driven Anomaly Detection** – Developing machine learning approaches capable of distinguishing legitimate workload variations from malicious activities, including potential Denial-of-Wallet attacks, in real time [16].
2. **Automated IAM Pruning** – Developing least-privilege enforcement mechanisms that utilize static analysis and runtime behavior analysis to dynamically reduce excessive permissions based on actual function execution requirements [1], [21].
3. **Cross-Cloud Security Standards** – Establishing standardized security frameworks for multi-cloud serverless deployments to reduce configuration inconsistencies and security gaps across platforms such as AWS, Azure, and Google Cloud [3], [12].

Author Contributions: Norsyazwani Mohd Puad: Methodology, validation.; **Paiwand Hadi Hama Saeed**: Formal analysis, investigation.; **Braw Araz Mohammed:** Conceptualization, writing original draft.; **Mohamad Fadli Zolkipli:** **S**upervision. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: Data supporting the findings of this study are available from the corresponding author upon reasonable request.

Acknowledgments: The authors would like to thank all individuals and organizations whose support and contributions made this research possible.

Conflicts of Interest: The authors declare no conflicts of interest.

References

[1] M. G. G. A. Amiri, S. M. Mousavi, and M. S. Fallah, "Security threats and mitigation techniques in serverless computing: A comprehensive survey," *IEEE Access*, vol. 12, pp. 43715–43743, 2024, doi: 10.1109/ACCESS.2024.3377545.

[2] K. A. Torkura, M. I. H. Sukmana, F. Cheng, and C. Meinel, "Continuous auditing and threat detection in multi-cloud infrastructure," *Computers & Security*, vol. 102, p. 102124, 2021, doi: 10.1016/j.cose.2020.102124.

[3] P. Escaleira, V. A. Cunha, J. P. Barraca, D. Gomes, and R. L. Aguiar, "A systematic review on security mechanisms for serverless computing," *Cluster Computing*, vol. 28, p. 465, 2025, doi: 10.1007/s10586-025-05371-4.

[4] E. Marin, D. Perino, and R. Di Pietro, "Serverless computing: A security perspective," *Journal of Cloud Computing*, vol. 11, p. 69, 2022, doi: 10.1186/s13677-022-00347-w.

[5] D. Kelly, F. G. Glavin, and E. Barrett, "Denial of wallet—defining a looming threat to serverless computing," *Journal of Information Security and Applications*, vol. 60, p. 102843, 2021, doi: 10.1016/j.jisa.2021.102843.

[6] K. Ni, S. K. Mondal, H. M. D. Kabir, T. Tan, and H.-N. Dai, "Toward security quantification of serverless computing," *Journal of Cloud Computing*, vol. 13, art. no. 140, 2024, doi: 10.1186/s13677-024-00703-y.

[7] D. D. Khanal and S. Maharjan, "Comparative security and compliance analysis of serverless computing platforms: AWS Lambda, Azure Functions, and Google Cloud Functions," *i-Manager's Journal on Cloud Computing*, vol. 11, no. 1, pp. 36–42, 2024, doi: 10.26634/jcc.11.1.21044.

[8] A. A. Yavuz, Y. Wang, A. K. Koc, and P. Ning, "Confidentiality and integrity for cloud-based data auditing: A survey," *ACM Computing Surveys*, vol. 56, no. 2, art. no. 41, 2023, doi: 10.1145/3605773.

[9] P. Gimeno Sarroca and M. Sánchez-Artigas, "MLLess: Achieving cost efficiency in serverless machine learning training," *Journal of Parallel and Distributed Computing*, vol. 183, p. 104764, Jan. 2024, doi: 10.1016/j.jpdc.2023.104764.

[10] V. R. Thummala and P. Singh, "Developing cloud migration strategies for cost-efficiency and compliance," *International Journal of Multidisciplinary Innovation and Research Methodology*, vol. 3, no. 4, pp. 300–313, 2024.

[11] S. Oduri, "Serverless yet secure: Rethinking cloud engineering paradigms," *Journal of Information Systems Engineering and Management*, vol. 10, no. 4, 2025.

[12] X. Li, X. Leng, and Y. Chen, "Securing serverless computing: Challenges, solutions, and opportunities," *IEEE Network*, vol. 37, no. 2, pp. 166–173, Mar./Apr. 2023, doi: 10.1109/MNET.005.2100335.

[13] B. K. R. Janumpally, "A review on data security and privacy in serverless computing: Key strategies and emerging challenges," *International Journal of Innovative Science and Research Technology*, vol. 10, no. 3, pp. 118–126, 2025, doi: 10.38124/IJISRT/25MAR023.

[14] M. A. I. Mallick and R. Nath, "Securing the server-less frontier: Challenges and innovative solutions in network security for server-less computing," *World Scientific News*, vol. 193, no. 1, pp. 1–45, 2024.

[15] J. Al Jarri, D. Alharthi, and M. Alquraishi, "Security implications of serverless computing in the cloud," *International Journal of Computer Science and Information Technology Research*, vol. 12, no. 3, pp. 1–4, 2024, doi: 10.5281/zenodo.12634327.

[16] F. Buzatto, M. Vieira, A. van Moorsel, and N. Antunes, "Function-as-a-Service Security: A Runtime Monitoring Framework for Serverless Platforms," *IEEE Transactions on Cloud Computing*, vol. 12, no. 2, pp. 1185–1199, 2024, doi: 10.1109/TCC.2023.3323017.

[17] T. Arif, B. Jo, and J. H. Park, "A comprehensive survey of privacy-enhancing and trust-centric cloud-native security techniques against cyber threats," *Sensors*, vol. 25, no. 8, p. 2350, 2025, doi: 10.3390/s25082350.

[18] A. N. Toosi, B. Javadi, A. Iosup, E. Smirni, and S. Dustdar, "Serverless computing for next-generation application development," *Future Generation Computer Systems*, vol. 164, art. no. 107573, 2025, doi: 10.1016/j.future.2024.107573.

[19] J. C. Sisniega, J. Czentye, B. Sonkoly, and B. Gero, "Serverless application composition leveraging function fusion: Theory and algorithms," Future Generation Computer Systems, vol. 157, pp. 111–127, 2024, doi: 10.1016/j.future.2024.03.031.

[20] A. Barrak, E. Ksontini, R. Atike, and F. Jaafar, "FaaSGuard: Secure CI/CD for Serverless Applications – An OpenFaaS Case Study," in *Proc. 2025 IEEE International Conference on Source Code Analysis & Manipulation (SCAM)*, 2025, pp. 110–115, doi: 10.1109/SCAM67354.2025.00018.

[21] A. Sabbioni, C. Mazzocca, M. Colajanni, R. Montanari, and A. Corradi, "A fully decentralized architecture for access control verification," in *Proc. 2022 IEEE Symposium on Computers and Communications (ISCC)*, 2022, pp. 1–6, doi: 10.1109/ISCC55528.2022.9913337.

[22] A. Koschel, S. Klassen, K. Jdiya, M. Schaaf, and I. Astrova, "Cloud computing: Serverless," in *Proc. 2021 12th International Conference on Information, Intelligence, Systems & Applications (IISA)*, 2021, pp. 1–7, doi: 10.1109/IISA52424.2021.9555534.

[23] A. S. F. Morrissey, M. S. Fallah, and H. Mora, "DoWNet—Classification of Denial-of-Wallet attacks on serverless application traffic," *Journal of Cybersecurity*, vol. 10, no. 1, 2024, doi: 10.1093/cybsec/tyae004.

[24] E. Marin, J. Kim, A. Pavoni, M. Conti, and R. Di Pietro, "The Hidden Dangers of Public Serverless Repositories: An Empirical Security Assessment," in *Proc. European Symposium on Research in Computer Security (ESORICS)*, 2025, pp. 382–401, doi: 10.1007/978-3-032-07894-0_20.
[25] K. Crawley, Cloud Penetration Testing: Learn How to Effectively Pentest AWS, Azure, and GCP Applications, 1st ed. Birmingham, UK: Packt Publishing Ltd., 2023.
[26] G. Raffa, J. Blasco, D. O'Keeffe, and S. K. Dash, "CloudFlow: Identifying Security-Sensitive Data Flows in Serverless Applications," *Proceedings of the 34th USENIX Security Symposium (USENIX Security 2025)*, 2025.
[27] L. Ben-Shimol, D. Lavi, E. Klevansky, O. Brodt, D. Mimran, Y. Elovici, and A. Shabtai, "Detection of compromised functions in a serverless cloud environment," *Computers & Security*, vol. 150, art. no. 104261, 2025, doi: 10.1016/j.cose.2024.104261.
[28] A. Shostack, *Threat Modeling: Designing for Security*, 1st ed. Indianapolis, IN, USA: Wiley, 2014.
[29] OWASP Foundation, "OWASP Serverless Top 10," 2023. [Online]. Available: https://owasp.org/www-project-serverless-top-10/
[30] National Institute of Standards and Technology, *Cybersecurity Framework (CSF) 2.0. Gaithersburg*, MD, USA: NIST, 2024.
[31] G. Bennett, T. Hall, E. Winter, and S. Counsell, "Semgrep*: Improving the Limited Performance of Static Application Security Testing (SAST) Tools," in *Proc. 28th International Conference on Evaluation and Assessment in Software Engineering (EASE '24)*, New York, NY, USA: ACM, 2024, pp. 614–623, doi: 10.1145/3661167.3661262.