LASEFlow: A Label-Aware Security Enhancement Framework for Serverless Workflows

Minghui Chen*, Keming Wang*, Chenlin Huang* ✉, Fengyuan Yu*, Renyu Yang†, Hua Cheng*, Mantun Chen§
* College of Computer Science and Technology, National University of Defense Technology, Changsha, China
† School of Software, Beihang University, Beijing, China
* Beijing Big Data Advanced Technology Research Institute, Tsinghua University, Beijing, China
§ National Defense Technology Innovation Institute, Academy of Military Science, Beijing, China
{chenminghui23@nudt.edu.cn, wkmyds@nudt.edu.cn, clhuang@nudt.edu.cn, yufengyuan@nudt.edu.cn, renyuyang@buaa.edu.cn, chenghua@ncic.ac.cn, chenmantun19@nudt.edu.cn}

***Abstract*—Serverless computing has gained widespread popularity among developers due to its low cost, fine-grained deployment, and management-free operation. However, when deploying serverless applications in practice, a single function is often insufficient to fulfill complete application requirements. This has led to the emergence of serverless workflows, which orchestrate a series of related serverless functions according to predefined logic. Through our investigation of existing serverless workflow platforms, we identify two major security limitations. First, current serverless workflows cannot guarantee execution integrity—they are unable to detect changes in the function execution order and lack mechanisms to defend against workflow-targeted denial-of-service (DoS) attacks. Second, identity management is typically coarse-grained, often resulting in over-privileged access and lacking support for function-level access control.**

To address these issues, we propose and implement the LASEFlow, a label-aware security enhancement framework for serverless workflows. A sequential function execution chain is designed based on the Platform Configuration Register (PCR) technique to guarantee the integrity of the execution of workflow in LASEFlow. In addition, a fine-grained function-level access control mechanism is designed to prevent privilege abuse in workflows. The evaluation demonstrates the effectiveness of LASEFlow against workflow attacks, with an overhead of less than 4% in performance.

***Index Terms*—serverless workflows, security label, execution integrity, access control**

I. Introduction

In the era of cloud computing, serverless computing has gained widespread popularity across the industry. Major cloud providers such as AWS, Azure, and Google Cloud have seen a year-on-year increase in serverless adoption [1]. In serverless computing, developers only need to submit implemented functions, and the cloud platform automatically deploys and schedules these functions [2]. However, as serverless applications become increasingly complex, simple function invocations can no longer meet application requirements, leading to the emergence of serverless workflows. To fulfill complex business needs, multiple serverless functions are expressed and organized in the form of workflows [3]. Serverless workflows are typically defined logically as Directed Acyclic Graph (DAG), and functions are scheduled and deployed according to the DAG during execution.

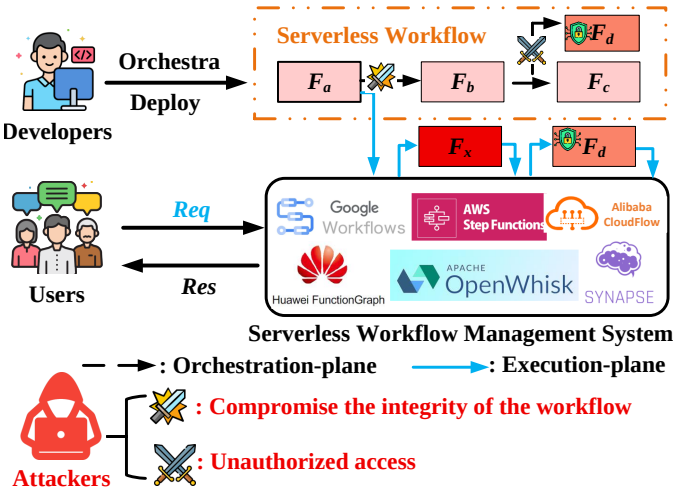

Fig. 1. Common Security Threats in Serverless Workflows.

As the adoption of serverless workflows continues to grow, security attacks targeting these workflows are also on the rise. The existing security threats in serverless workflows are illustrated in Fig 1. In such workflows, attackers can exploit vulnerabilities in third-party libraries used by function instances, tamper with unauthorized execution paths, or take advantage of user misconfigurations to compromise the system. These attacks may lead to the theft of private data [5], [12], the triggering of unintended execution paths [4], [6], or denial-of-service (or even denial-of-wallet) attacks [13], [14]. In this paper, we focus on two critical issues: the integrity of serverless workflows and the lack of function-level access control.

Existing serverless workflow platforms do not guarantee execution integrity. Execution integrity means that the workflow's execution path cannot be altered: no function should be skipped; no function should repeatedly invoke others to launch DoS attacks; no function should call another that is undefined in the workflow; and no unauthorized user should

be able to invoke functions within the workflow. Existing web application security tools—such as vulnerability scanners and log analyzers—have been adapted to serverless environments, but these tools primarily detect known vulnerabilities within individual functions or their dependencies. They are ineffective in identifying control-flow tampering that spans multiple functions. For instance, Jeremy [4] demonstrated that an attacker could craft a malicious S3 object name to inject event data, causing a function to execute unauthorized shell commands and trigger unintended workflow paths or malicious operations. These attacks [4]–[6], [12]–[14] highlight that serverless workflows security issues extend beyond individual functions—they include the integrity of the entire execution path. Ensuring that a workflow's execution strictly follows its expected sequence and logic has become a key challenge in securing serverless workflows.

Current serverless platforms lack function-level access control. In platforms such as AWS Lambda, Microsoft Azure Functions, and Google Cloud Functions, access control relies primarily on IAM mechanisms [7]–[9], which define static policies for controlling access to function resources. While this model is sufficient for simple, single-function scenarios, it becomes problematic in workflow-based applications. IAM typically treats the workflow as a monolithic unit, granting users access to the entire workflow. If a specific function within the workflow is intended to be accessed only by administrators, IAM cannot enforce function-level identity checks. To ensure that regular users can still access non-privileged functions in the workflow, developers often grant them access to the entire workflow, leading to over-privileged access [11].

To address the aforementioned challenges, we propose LASEFlow, a label based security enhancement framework for serverless workflows, aiming to navigate the complexity of user permission management and the assurance of execution path integrity. These goals are achieved by introducing a policy specification mechanism and a trusted execution chain tailored for the workflows.

Specifically, LASEFlow uses labels to determine the resource trustworthiness, i.e., whether a resource can be accessed or used to run a function. We introduce two types of labels: resource labels and workflow labels. Resource labels are assigned to various serverless components such as databases and compute nodes. These labels encode attributes like resource origin and underlying hardware environment. Workflow labels are dynamically propagated throughout the execution of the workflow, carrying integrity measurements of each node's execution to ensure the workflow's execution path has not been tampered with.

LASEFlow also proposes a policy specification mechanism for user permission management in serverless workflows. In our policy model, user identity and permission are tightly integrated with the workflow definition, simplifying the permission configuration process. Furthermore, users with different privilege levels are presented with different workflow views. This view-based isolation effectively prevents unauthorized access and illegal invocation of sensitive functions, enhancing both the security and controllability of the system.

Finally, LASEFlow incorporates the Platform Configuration Register (PCR) technique from trusted computing to build a verifiable execution chain for workflows. After each function executes, it generates an execution digest that is cryptographically extended with the previous state chain. This process updates the workflow label, which is then passed to the next function as an execution token. This design creates a chained state attestation between functions, ensuring that the workflow execution path cannot be skipped, forged, or replayed.

Specifically, our contributions are as follows:

- We propose a resource labeling method for serverless platforms, enabling identification and labeling of resources. We also introduce a mechanism to embed labels within workflows, allowing function execution metrics to be recorded and propagated, thus forming a verifiable execution path. In addition, we define a user permission policy specification tailored for serverless workflows.
- We implement LASEFlow, a label-based execution control framework for serverless workflows, which constructs a verifiable execution chain during the workflow execution to guarantee workflow integrity.
- We quantitatively evaluate the performance overhead of LASEFlow and analyze its effectiveness in ensuring the integrity of serverless workflows.

II. Background and threat model

A. Characteristics of Serverless Workflows

Serverless architecture has become a mainstream paradigm in modern cloud computing due to its advantages such as server management elimination, elastic scalability, and pay-per-use billing [19]. As application logic grows increasingly complex, individual serverless functions often fall short of meeting full business requirements. Developers are thus orchestrating multiple functions according to specific logic to build what is known as a Serverless Workflows [21].

In such workflows, multiple functions must execute in a defined order or based on conditional logic to accomplish specific tasks. To achieve this coordinated execution, different systems and platforms adopt a variety of approaches. Based on existing research and engineering practices, current mainstream coordination mechanisms for serverless workflows can be categorized into three types [3], [25]–[28]:

a) Event-based Orchestration: In this model, functions communicate and chain together via events. For instance, after one function completes execution, it publishes a result to a message queue or event bus (such as AWS SNS/SQS, Azure Service Bus, RabbitMQ) [20], which in turn triggers the execution of the next function. This approach offers loose coupling and high scalability but typically requires developers

to explicitly handle event listening and message-passing logic. Some platforms (e.g., AWS EventBridge) partially automate this mechanism, reducing the developer's burden.

b) Client-side Coordination: In certain scenarios, the execution order of functions is controlled directly by the client side (e.g., web pages, mobile applications, or GUI programs). These "coordinators" typically invoke cloud functions directly (e.g., via HTTP API gateways), ensuring that each step runs with the correct data and parameters. While this method is simple to implement, it lacks observability, fault tolerance, and state persistence, making it unsuitable for complex or reliability-critical workflows.

c) Workflow Engines: To address the limitations of event-driven and client-side coordination models, major serverless cloud platforms—such as AWS Step Functions, Azure Durable Functions, and Google Cloud Workflows—have introduced dedicated workflow engines. These services support the definition of control flow between tasks using either declarative syntax (e.g., JSON or YAML) or code-based formats. The platform then automatically handles key operational tasks such as function scheduling, state persistence, retry mechanisms, concurrency control, and monitoring and analytics. Compared to client-side coordinators, workflow engines offer significantly greater stability, security, and maintainability, and have become the dominant coordination method in complex serverless systems.

Building upon workflow engines, the Serverless Workflow Management System (SWMS) has emerged as a critical layer that connects users with the underlying execution platform. SWMS is responsible for managing the entire lifecycle of a workflow—including its definition, deployment, execution, and monitoring.

Specifically, the SWMS first receives workflow definitions written in declarative formats such as YAML or JSON. It then performs syntax parsing and business logic validation to ensure the correctness of the workflow model. After validation, the SWMS sends the verified workflow configuration to the underlying workflow engine, which autonomously schedules and invokes the appropriate serverless functions based on the defined control flow. During execution, the SWMS continuously tracks the state transitions of the workflow, handles data passing, exception handling, and retry mechanisms, and collects logs and performance metrics. These capabilities enable users to monitor execution in real time and support subsequent optimization.

B. Threat Model

In this work, we focus on two core security threats in serverless workflows: integrity violations and privilege abuse. The integrity issue refers to attacks in which an adversary tampers with workflow configurations, forges state inputs, or replays previously executed paths, thereby undermining the integrity of the workflow execution path.Attackers may further exploit valid execution paths to trigger vulnerable functions, launching DoS attacks against downstream functions or using them as entry points for malicious operations. Additionally, by compromising insecure functions, attackers can extract sensitive information or alter function behavior to perform unauthorized actions.Privilege abuse stems from the prevalent issue of over-privileged access in existing access control mechanisms such as IAM. Attackers may leverage excessive permissions to indirectly or directly invoke functions that should only be accessible to privileged users, resulting in privilege escalation and security policy violations.

We assume that the serverless workflow is deployed by trusted tenants on a cloud platform, and that the platform itself—including the serverless runtime, scheduling system, and network—is trusted. Attackers do not have control over the underlying infrastructure such as the function runtime. Except for the functions and resources defined within the workflow, all other external services—including cloud storage, databases, and third-party APIs—are considered untrusted and treated as black boxes. We do not consider attack vectors involving side channels, platform-level vulnerabilities, or physical attacks.

The attacker's capabilities are limited to exploiting vulnerabilities in function code or misconfigurations to manipulate one or more functions. Attackers may also disrupt workflow execution integrity by crafting malicious inputs or triggering illegal state transitions, thereby undermining the workflow's execution path correctness.

C. Design Goals

To address the threats outlined in the threat model, LASEFlow is designed to achieve the following security goals:

a) Verifiable Execution Path Integrity: LASEFlow should ensure the integrity of function execution paths within a serverless workflow. Each function invocation must be cryptographically bound to the legitimate execution state and context of its preceding function(s), thereby preventing unauthorized path skipping, interruption, or replay attacks.

b) Protection Against Illegal State Injection and Replay Attacks: LASEFlow must validate the authenticity and trustworthiness of the state inputs received by each function. This prevents attackers from forging state data or replaying previously valid execution states to induce incorrect or malicious behavior.

c) Fine-Grained Access Control: LASEFlow should enforce access control policies based on user identity, roles, and workflow execution context. It must support the principle of least privilege to minimize over-permission risks. Users should only be able to access or perceive functions and execution paths explicitly authorized to them.

d) Minimized Attack Surface: Communication between functions and services should be strictly constrained to what is defined within the workflow. External visibility should be limited to reduce the potential impact of a compromised function on other parts of the system.

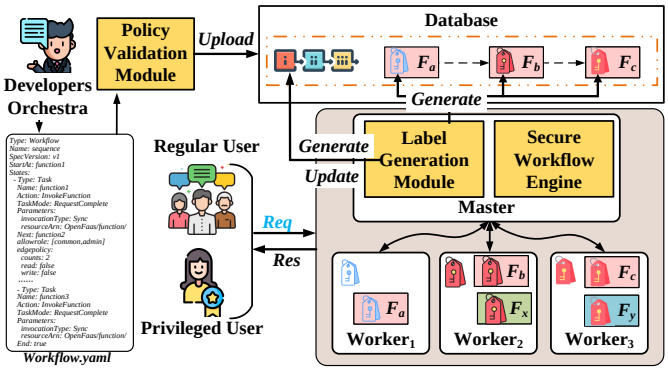

Fig. 2. The overview of LASEFlow.

e) Auditability and Post-Incident Traceability: All function invocations and state transitions must be logged in a tamper-evident manner. These logs should support forensic analysis and root cause tracing in the event of anomalous or malicious workflow behavior.

III. LASEFlow Design

LASEFlow is a label-based security framework designed for serverless workflows. It consists of three main components: the policy validation module, the label generation module, and the secure workflow engine. The policy validation module is responsible for parsing and verifying user-submitted workflow security policies. It checks for issues such as permission conflicts, omissions, or unauthorized access within the policy, ensuring its logical soundness and completeness. The label generation module produces both resource labels and workflow labels based on validated policies. Resource labels indicate the trust level and deployment requirements of individual functions, while workflow labels record the sequential dependencies between functions and serve as execution credentials. The secure workflow engine dynamically switches the workflow graph based on the user's identity and request. It schedules function executions in accordance with the policy, verifies workflow labels during runtime, and constructs a verifiable execution path chain to prevent path tampering, skipping, or replay attacks. The architecture of LASEFlow is illustrated in Fig.2.

A. Label Design for Serverless Workflows

On cloud platforms such as AWS, Azure, and Alibaba Cloud, resource security is often enforced through the combination of labels and access control policies. For instance, sensitive functions can be assigned specific labels, and access can be restricted in IAM by setting conditional policies based on those labels. Inspired by this mechanism, this paper proposes a label-based security framework for serverless workflows.To achieve the goals of preserving workflow integrity and ensuring secure function execution, we design two types of labels: resource labels and workflow labels. Resource labels specify the deployment requirements of functions and indicate the trustworthiness of external resources. Workflow labels capture the dependency and execution order between functions, enabling verification of the execution path and preventing unauthorized modification.

a) Resource Label: To ensure the security of individual functions during both deployment and runtime phases in serverless workflows, this paper introduces a resource label mechanism that enhances trustworthy resource identification and fine-grained security control.First, to guarantee the security of the function deployment environment, nodes can be assigned a trusted launch label, which indicates whether the node has undergone a trusted boot process. This ensures that functions are deployed within a secure and trusted execution environment. When users require secure deployment for functions, the presence of a trusted launch label on the hosting node serves as an indicator of deployment security. Additionally, vulnerability scanning is performed on nodes, and based on the results, nodes are classified into different security levels. Functions involving sensitive operations are preferentially scheduled on nodes with higher security levels to reduce security risks.

Second, to protect data during function execution, storage resources in the serverless platform must be assigned source labels that identify the trustworthiness of their origin (e.g., platform-owned, trusted third party, untrusted third party). The system assesses the security level of resources based on these labels. When functions access unsafe or untrusted storage resources, data encryption during transmission and storage is enforced, or access may be denied outright. Moreover, for functions handling sensitive data, the system combines the node security level labels to schedule execution on nodes with stronger security guarantees, thereby enhancing data protection.

The resource label mechanism enables trusted identification and hierarchical management of function deployment environments and data storage resources. It provides fine-grained security guarantees for serverless workflows, significantly improving the overall system's security and controllability.

b) Workflow Labels: In serverless workflows, traditional security mechanisms struggle to effectively prevent execution paths from being bypassed, tampered with, or replayed, thereby failing to guarantee the integrity and trustworthiness of the entire workflow execution process. To address this issue, this paper draws on the Platform Configuration Register (PCR) technique from trusted computing and proposes an execution path protection mechanism based on workflow labels.This mechanism measures the execution result of each function—such as the hash of output or state digest—and extends the current path hash by chaining it with the hash value recorded in the previous stage's label. Through this chain extension, the system continuously updates the current label's path hash, thereby constructing an immutable execution chain.

Workflow labels act as metadata carriers passed between functions and contain key security information related to the workflow execution path, specifically including:

1) Current PCR Value (accumulated path hash): Records the cumulative execution digest from the start node to the current function, used to verify whether the execution path has been tampered with or skipped;
2) Timestamp: Marks the generation time of the current label to prevent replay or delayed use of the label;
3) Identity: Records the caller's identity for the current function, which is used in conjunction with access control policies to verify legitimate execution permissions;
4) Workflow Instance Identifiers (Workflow ID and version): Ensures that the label is bound to a specific workflow definition and execution instance, preventing forgery or confusion across instances or versions.

Through this mechanism, workflow labels not only provide cryptographic proofs of path integrity but also serve as critical evidence for policy validation, audit tracking, and security response. Consequently, they significantly enhance the trustworthiness of serverless workflows.

B. Policy Specification Framework

To achieve identity-based least privilege control, this paper proposes a policy specification framework that is compatible with the structure of serverless workflow graphs. Building upon existing workflow definition models in cloud platforms and integrating Identity and Access Management (IAM) mechanisms, the framework binds user identity permissions to the workflow graph through a policy language. This enables the construction of differentiated workflow graph views, effectively isolating and protecting sensitive paths and critical functions.

The policy specification corresponds one-to-one with workflow state definitions and abstracts three key policy control elements: Object, Edge, and Identity, representing function resources, state transition paths, and access subjects, respectively. The policy language allows developers to define access constraints related to user identities for each state (i.e., function invocation task) and its outgoing edges, enabling access isolation control based on granular attributes such as roles, organizations, or tenants. The syntax of the policy specification is shown in Table I. The policy specification in this paper is inspired by the designs of Valve [16], Growlithe [22] and Grasp [24].

Within this framework, the system dynamically generates a workflow subgraph tailored to each user based on their identity context and matching policies, thereby creating a customized user view. Regular users, by default, only see the minimal execution path, with access permissions to all edges and state nodes denied unless explicitly granted. Privileged users can unlock hidden states, expand branching paths, or modify default policies for functions and edges by extending their permission labels, thereby obtaining a more complete workflow graph and elevated operational privileges. This design not only effectively controls unauthorized access to sensitive functions but also reduces the workflow's exposure and attack surface.

TABLE I
THE POLICY SPECIFICATION OF SERVERLESS WORKFLOWS

Term	Description
workflow	object+edge identity
object	function/storage
function	(name, result, functionpolicy)
storage	S3 bucket, DynamoDB table, etc.
edge	(preobject, nextobject, edgepolicy)
functionpolicy	key: value, requirements for node
edgepolicy	(counts, read, write)
counts	number of requests from pre to next
read	whether pre can read next
write	whether pre can write next
privilegeworkflow	object edge identity
modifyfunctionpolicy	identity functionpolicy
modifyedgepolicy	identity edgefunctionpolicy

Furthermore, the policy specification supports attaching additional control attributes to state transition paths, such as invocation frequency limits, identity consistency verification, and state dependency constraints. These features further strengthen the system's defense against denial-of-service attacks, replay attacks, and path tampering. After users write policies according to this framework, the policy validation module checks for conflicts, omissions, or privilege escalations in the permission configurations to ensure the logical correctness and completeness of the policies.

C. Secure Execution of Serverless Workflows

Serverless workflows must comply with user-defined policies during execution. The execution process is divided into two stages. First, during the static analysis stage, the policy validation module performs checks on the policy before execution. Second, during the runtime stage, policy enforcement is carried out prior to function invocation to ensure that the workflow path has not been tampered with and that execution counts comply with policy requirements. In this section, we provide a detailed description of these two mechanisms.

a) Static Analysis Stage: Before the workflow is executed, the developer must provide a permission policy file written according to the policy specification. The policy validation module constructs a workflow graph based on the user-defined policy and performs structural and semantic analysis of the policy file in the context of the graph. This analysis includes, but is not limited to, the following aspects:

1) Privilege Overlap and Escalation Analysis: The system detects whether there are workflow paths with missing permission definitions, or whether low-privileged identities attempt to access high-sensitivity resources. This analysis helps identify unauthorized paths and potential attack surfaces.
2) Unreachable Path Detection: Based on constraints defined in the policy (e.g., invocation count set to zero, or no read/write permissions), the system identifies logically unreachable workflow paths. These may result from

misconfigured policies or redundant design elements, and the system issues warnings accordingly.

3) Indirect Path Attack Detection: The system analyzes whether it is possible to bypass direct access restrictions by indirectly reaching sensitive resources through multi-hop paths, thereby preventing attackers from exploiting the workflow structure to obtain unauthorized data.

After the policy is successfully validated, the label generation module creates resource labels for each function and resource. These labels are used to guide deployment and access decisions. During deployment, each function is deployed according to the requirements specified by its resource label. When a function attempts to access a database, the secure workflow engine checks the resource label to determine whether the database is trustworthy and decides whether to allow or block the access.

b) Dynamic Execution Stage: During the actual execution of the workflow, to ensure path integrity and fine-grained access control, LASEFlow incorporates PCR technology from trusted computing to construct a verifiable execution path chain. The operational mechanism is as follows:

1) Identity Authentication: When a user initiates a workflow request, the system first performs identity authentication to ensure the legitimacy of the caller. Based on the authenticated identity, the system dynamically prunes the workflow graph, providing each user with a customized workflow view. A unique workflow instance identifier (Workflow ID and version number) is then generated for this execution, which is used for subsequent label binding and path tracking.
2) Workflow Labels Initialization and Path Chain Start: Before the workflow execution begins, the workflow engine invokes the label generation module to create the initial workflow label. This label contains a timestamp, identity information, workflow instance identifiers, and other metadata. An initial PCR value (PCR_0) is generated via hashing, serving as the anchor point for path integrity.
3) Function Call Interception and Verification: Prior to each function invocation request, the workflow engine intercepts the call and enforces the following verification procedures:
   - Label Consistency Verification: The hash of the workflow label carried in the request is computed and compared against the latest stored PCR value. If they do not match, this indicates possible tampering with the execution path or policy labels; the call is terminated and a security event is logged.
   - Policy Compliance Verification: The system checks whether the invocation complies with access control rules defined by the policy, including caller identity, invocation count, and data access permissions. If verification fails, the execution is rejected and a security error code is returned.
   - Replay Attack Prevention: Before a function begins execution, the system verifies the nonce, unique request identifier (RequestID), timestamp, and HMAC signature based on a shared key. This prevents attackers from launching unauthorized invocations by replaying historical requests or forging request contents.
   - Workflow Labels Update: If the verification succeeds, the system updates the PCR value in the workflow label. The new PCR value is computed by combining the previous PCR value, the current function's state digest, and the invocation edge information, as defined in (1). In addition to the PCR value, the workflow label also updates other fields such as the nonce and timestamp. The updated workflow label is then attached to the invocation request and forwarded to the next function.
4) Trusted Path Chain Completion: As the workflow progresses, LASEFlow constructs a complete trusted execution path chain through the PCR mechanism. This chain provides security guarantees against malicious tampering of the workflow execution path and prevents attackers from replaying historical labels to repeatedly execute sensitive paths.
5) Secure Scheduling and Node Constraint Verification: During function scheduling, the scheduler filters the available node pool to select compute nodes that satisfy all policy conditions based on resource labels. If no eligible nodes are found, the execution is rejected with a "policy deployment failure" error. This mechanism further ensures compliance of function deployment locations and data security.

$$\mathrm{PCR}_{i+1} = \mathrm{Hash}(\mathrm{PCR}_i \parallel \mathit{edge\_info} \parallel \mathit{function\_digest}) \quad (1)$$

IV. Implementation and Evaluation

This section evaluates the LASEFlow system from two perspectives: performance overhead and security assurance. We deployed LASEFlow on a local testbed running OpenFaaS v0.27.12 atop Kubernetes v1.32 (Docker runtime version 28.0.1), allowing us to assess its runtime performance under varying workloads. OpenFaaS is a Kubernetes-based serverless computing platform, which enables controlled experiments across the full execution lifecycle of serverless workflows.

A. Overhead of LASEFlow

We first evaluated the time overhead introduced by LASEFlow during workflow execution. Since native OpenFaaS does not provide built-in support for workflow execution control, we implemented a baseline serverless workflow framework.

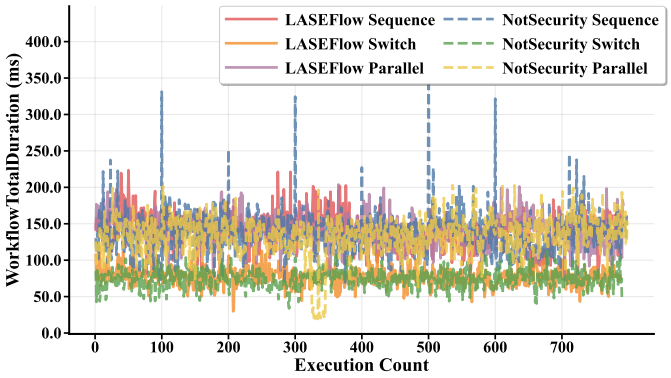

Fig. 3. Total execution time comparison for 800 workflow executions under sequential, branching, and parallel structures, using our framework versus the baseline framework.

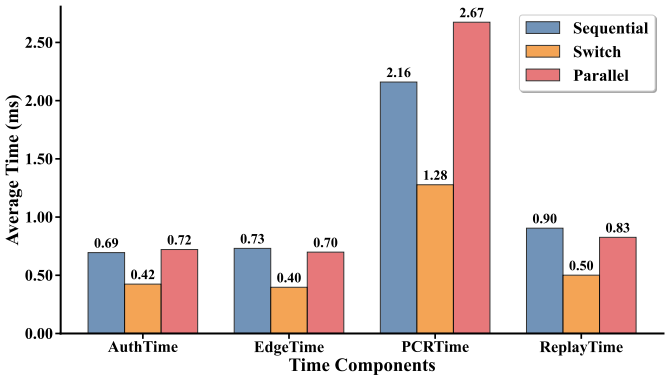

Fig. 4. Average time comparison of security mechanisms—identity authentication, policy validation, PCR verification, and replay protection—across sequential, branching, and parallel workflow structures.

This baseline framework supports sequential execution of user-defined workflows but does not include any security mechanisms. We then compared it with a security-enhanced version integrated with LASEFlow.

We tested three common workflow structures: Sequential, Branching, and Parallel. For each structure, we ensured the number of functions remained constant and pre-deployed all functions to eliminate time variations caused by cold starts or deployment delays. Each structure was executed 800 times under both the baseline and LASEFlow-enhanced frameworks. We measured the total execution time and calculated the average overhead.

As shown in Fig. 3., LASEFlow introduces approximately 3%-4% additional overhead in sequential workflows, 3.5%–4% in branching workflows, and 4%–5% in parallel workflows. It is worth noting that in branching structures, the number of executed functions may be fewer due to conditional path selection, which results in a shorter total execution time compared to sequential or parallel structures. The main sources of overhead in LASEFlow stem from four security features: identity authentication, policy validation, PCR verification, and anti-replay mechanisms. In addition, network fluctuations, CPU scheduling, and other system-level factors may contribute to minor variations in the measured timing results.

B. Performance of Security Measures

We further analyze the performance of the four security mechanisms integrated into LASEFlow across different workflow structures: identity authentication, policy validation, PCR verification, and replay attack prevention.

Identity Authentication: Before each function invocation, the system verifies the request identity based on the user's policy. Only authenticated requests are allowed to proceed to the next function. This mechanism supports workflow graph pruning based on identity labels, enabling different users to see different execution paths and enforcing identity-based dynamic access control.

Policy Validation: During workflow deployment, developers can attach security policies to state transition edges. These policies are used to prevent DoS attacks and to enforce fine-grained access control (e.g., read/write permissions) based on user identity.

PCR Verification: To ensure execution path integrity, LASEFlow constructs a trusted execution chain during workflow execution. The system begins by computing an initial PCR_0 value from the workflow's starting state. After each function completes, the PCR value is updated and stored in the workflow label. During the verification phase, the system reconstructs the execution path from the function sequence recorded in the label, calculates the expected PCR value, and compares it with the recorded value. Any mismatch indicates that the execution path has been tampered with.

Replay Attack Prevention: Before each function request is sent to OpenFaaS, the system appends a random nonce, a unique request ID, a timestamp, and an HMAC signature to prevent malicious users from forging or resubmitting identical requests.

To evaluate the performance impact of these four mechanisms under different workflow structures, we conducted 800 executions each for sequential, branching, and parallel workflows. We recorded the average processing time for each mechanism. The results are shown in Fig. 4.

Because branching workflows typically involve fewer actual function invocations due to conditional path selection, their overall security overhead is relatively lower. In sequential and parallel workflows, the number of executed functions is the same, resulting in comparable time costs for identity authentication, policy validation, and replay protection. However, in PCR verification, the parallel structure requires merging the execution results from all concurrent paths before validation, leading to an average PCR verification time approximately 0.5 ms higher than that of the sequential structure.

In terms of overall proportion within the total workflow exe-

cution time: identity authentication accounts for approximately 0.4%, policy validation about 0.4%, PCR verification around 1.4%–2%, and replay attack protection approximately 0.5%.

C. Security Analysis

LASEFlow provides comprehensive security enhancements for the entire lifecycle of serverless workflows, systematically addressing execution path integrity, replay attack prevention, fine-grained access control, and attack surface reduction. First, LASEFlow constructs a trusted execution chain based on the PCR principle, continuously measuring and verifying the workflow path to ensure that the execution order remains unaltered. Upon detecting any path anomalies or skipped nodes, the system halts execution, effectively preventing unauthorized and covert path executions. Second, the system incorporates a uniqueness verification mechanism based on nonces and timestamps, combined with HMAC signatures to ensure request integrity and freshness, thereby defending against replay attacks. Furthermore, building on mainstream serverless platform policy models, LASEFlow enhances the expressiveness of policies on state transition edges, enabling each invocation edge to be associated with identity labels and access control policies. This facilitates function-level permission management and mitigates the problem of over-privileged access. Finally, to minimize the attack surface, LASEFlow defaults to denying all service requests beyond the defined workflow scope, blocking potential unauthorized access paths at the platform level. Overall, by integrating trusted computing primitives with enhanced policy enforcement and access control mechanisms, LASEFlow systematically improves the controllability and security robustness of serverless workflows.

V. Related Work

Existing research on protecting serverless workflow execution paths primarily falls into two categories: Information Flow Control (IFC) and Control Flow Integrity (CFI) approaches. IFC is particularly suited for serverless applications composed of multiple functions. Trapeze [15] implements a dynamic IFC model that enforces security through static program annotations and dynamic data labeling. However, its policy specification and decryption mechanisms rely heavily on developer implementation and are constrained by the supported programming languages and a set of predefined key-value storage functions. In contrast, Valve [16] achieves default policy enforcement without code modification by proxying network requests and propagating taints, while also allowing developer customization. Unlike Trapeze, Valve requires no decryption and can infer function behavior at the network layer. Addressing the limitations of Trapeze, will.iam [17] introduces a language- and platform-agnostic approach capable of automatically inspecting access control policies and making proactive decisions on incoming requests. Valve and will.iam are complementary: the former provides fine-grained data flow insights, while the latter strengthens policy enforcement mechanisms.

In terms of control flow integrity, Kalium [18] enforces control-flow correctness by combining local function state with the global application state. It constructs both local and global control-flow graphs (CFGs) by monitoring interactions between functions and external services, validating them during system calls. Compared to IFC, CFI approaches ensure the correct order and frequency of function executions, but face challenges such as lack of support for concurrent requests and incomplete CFG coverage.Moreover, many of these methods assume access to function source code, which is often unavailable in practice—especially when functions are sourced from public marketplaces. This lack of visibility makes it difficult to defend against path tampering attacks, and the protection offered for critical workflow steps remains insufficient.

VI. Conclusion

This paper addresses key security challenges in serverless workflows, including the difficulty of ensuring execution path integrity and the risk of excessive permissions in IAM. We propose LASEFlow, a label-based security framework inspired by trusted computing's PCR technology. By leveraging resource labels and workflow labels, LASEFlow constructs a verifiable execution chain that guarantees the immutability of function execution order and the trustworthiness of execution paths. Additionally, combining identity labels with policy specifications enables fine-grained access control and dynamic pruning of workflow views, reducing the complexity of permission configurations and mitigating privilege escalation risks. Experimental results demonstrate that LASEFlow enhances security with minimal performance overhead. In summary, LASEFlow effectively strengthens the security guarantees of serverless workflows, resolves critical issues related to execution path integrity and permission management, and enriches both the theoretical and practical foundations of serverless workflow security mechanisms.

References

[1] Datadog. 2024. The State of Serverless. https://www.datadoghq.com/state-of-serverless/. Accessed: 2024-01-28.

[2] Jonas, Eric, et al. "Cloud programming simplified: A berkeley view on serverless computing." arXiv preprint arXiv:1902.03383 (2019).

[3] Eismann, Simon, et al. "A review of serverless use cases and their characteristics." arXiv preprint arXiv:2008.11110 (2020).

[4] Jeremy Daly. Event injection: Protecting your serverless applications. https://tinyurl.com/4udrdhmu, 2019.

[5] Ory Segal. Securing serverless: Attacking an aws account via a lambda function. https://ubm.io/2FIrKq2, 2018.

[6] Puresec. New attack vector: Serverless crypto mining. https://www.puresec.io/serverless-crypto-mining-resource-download,2018.

[7] "AWS Identity and Access Management," https://aws.amazon.com/iam/.

[8] "Azure Identity and Access Management," https://azure.microsoft.com/en-ca/products/category/identity.

[9] "Google Cloud Identity and Access Management," https://cloud.google.com/security/products/iam.

[10] "Serverless on IBM Cloud," https://www.ibm.com/products/code-engine.

[11] Jonathan Greig. 2020. 2020 Cloud Misconfigurations Report. https://divvycloud.com/misconfigurations-report-2020/.

[12] Frederik Willaert. 2019. AWS Lambda Container Lifetime and Config Re fresh. https://www.linkedin.com/pulse/aws-lambda-container-lifetime-config-refresh-frederik-willaert/.
[13] 2019. New Attack Vector- Serverless Crypto Mining. https://www.puresec.io/blog/new-attack-vector-serverless-crypto-mining.
[14] Yan Cui. 2021. Many-faced threats to Serverless security. https://hackernoon.com/many-faced-threats-to-serverless-security-519e94d19dba.
[15] Alpernas, Kalev, et al. "Secure serverless computing using dynamic information flow control." Proceedings of the ACM on Programming Languages 2.OOPSLA (2018): 1-26.
[16] Datta, Pubali, et al. "Valve: Securing function workflows on serverless computing platforms." Proceedings of The Web Conference 2020. 2020.
[17] Sankaran, Arnav, Pubali Datta, and Adam Bates. "Workflow integration alleviates identity and access management in serverless computing." Proceedings of the 36th Annual Computer Security Applications Conference. 2020.
[18] Jegan, Deepak Sirone, et al. "Guarding serverless applications with Kalium." 32nd USENIX Security Symposium (USENIX Security 23). 2023.
[19] Li, Zijun, et al. "The serverless computing survey: A technical primer for design architecture." ACM Computing Surveys (CSUR) 54.10s (2022): 1-34.
[20] Figueira, João, and Carlos Coutinho. "Developing self-adaptive microservices." Procedia Computer Science 232 (2024): 264-273.
[21] Wen, Jinfeng, and Yi Liu. "A measurement study on serverless workflow services." 2021 IEEE International Conference on Web Services (ICWS). IEEE, 2021.
[22] Gupta, Praveen, et al. "Growlithe: A Developer-Centric Compliance Tool for Serverless Applications." 2025 IEEE Symposium on Security and Privacy (SP). IEEE, 2025.
[23] Shafiei, Hossein, Ahmad Khonsari, and Payam Mousavi. "Serverless computing: a survey of opportunities, challenges, and applications." ACM Computing Surveys 54.11s (2022): 1-32.
[24] Polinsky, Isaac, et al. "GRASP: Hardening serverless applications through graph reachability analysis of security policies." Proceedings of the ACM Web Conference 2024. 2024.
[25] Li, Yongkang, et al. "Serverless computing: state-of-the-art, challenges and opportunities." IEEE Transactions on Services Computing 16.2 (2022): 1522-1539.
[26] Wang, Keming, et al. "Octopus: decentralized workflow-granular scheduling for serverless workflow." Proceedings of 45th IEEE International Conference on Distributed Computing Systems (IEEE ICDCS 2025). IEEE Computer Society, 2025.
[27] Li, Yaojie, et al. "DCSA: The Deployment Mechanism of Chained Serverless Applications in JointCloud Environment." 2024 IEEE International Conference on Joint Cloud Computing (JCC). IEEE, 2024.
[28] Liu, Jianfei, et al. "Fcloudless: A performance-aware collaborative mechanism for jointcloud serverless." 2023 IEEE International Conference on Joint Cloud Computing (JCC). IEEE, 2023.