

Article

SMWE: A Framework for Secure and Makespan-Oriented Workflow Execution in Serverless Computing

Hao Liang ^{1,2,*} , Shuai Zhang ^{1,2,*}, Xinlei Liu ^{1,2} , Guozhen Cheng ^{1,2}, Hailong Ma ^{1,2} and Qingfeng Wang ^{1,2}

¹ Information Technology Research Institute, Information Engineering University, Zhengzhou 450002, China; liuxl@stu.ouc.edu.cn (X.L.); chengguozhen1986@163.com (G.C.); longmanclear@163.com (H.M.); wang_qf2024@163.com (Q.W.)

² National Digital Switching System Engineering & Technological Research Center, Zhengzhou 450002, China

* Correspondence: lhmailhappy@163.com (H.L.); 2012301200229@whu.edu.cn (S.Z.)

Abstract: Serverless computing is a promising paradigm that greatly simplifies cloud programming. With serverless computing, developers simply provide event-driven functions to a serverless platform, and these functions can be orchestrated as serverless workflows to accomplish complex tasks. Due to the lightweight limitation of functions, serverless workflows not only suffer from existing vulnerability-based threats but also face new security threats from the function compiling phase. In this paper, we present SMWE, a secure and makespan-oriented workflow execution framework in serverless computing. SMWE enables all life cycle protection for functions by adopting compiler shifting and running environment replacement in the serverless workflow. Furthermore, SMWE balances the tradeoff between security and makespan by carefully scheduling functions to running environments and selectively applying the secure techniques to functions. Extensive evaluations show that SMWE significantly increases the security of serverless workflows with small makespan cost.

Keywords: serverless computing; workflow scheduling; security; makespan; proactive defense; cloud computing



Citation: Liang, H.; Zhang, S.; Liu, X.; Cheng, G.; Ma, H.; Wang, Q. SMWE:

A Framework for Secure and Makespan-Oriented Workflow Execution in Serverless Computing. *Electronics* **2024**, *13*, 3246. <https://doi.org/10.3390/electronics13163246>

Academic Editor: Aryya Gangopadhyay

Received: 9 June 2024

Revised: 7 August 2024

Accepted: 13 August 2024

Published: 15 August 2024



Copyright: © 2024 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (<https://creativecommons.org/licenses/by/4.0/>).

1. Introduction

Serverless computing, alternatively known as Function-as-a-Service (FaaS), represents a novel and burgeoning service paradigm within the realm of cloud computing [1,2]. In serverless computing, a serverless provider manages the basic environment, such as hardware devices, the operating system, and the software development environment. And developers only need to be concerned with the core logic of application or software development. Due to the prominent benefits, many popular applications have been developed in serverless computing, such as chatbots [3], data analytic [4] and scientific computing [5]. A workflow, comprising a series of interconnected tasks with data dependencies, constitutes a prevalent abstract model for applications deployed on cloud platforms [6]. In the context of serverless computing, applications can be conceptualized through serverless workflows, where multiple functions are orchestrated seamlessly into a cohesive workflow to deliver a comprehensive service. In serverless workflows, functions are equal to tasks in a traditional workflow.

However, a serverless workflow faces many security threats. First, vulnerability-based threats in traditional workflows still exist in serverless workflows. More precisely, adversaries can capitalize on the vulnerabilities within the functions to attain their objectives, including but not limited to exfiltrating sensitive data, disrupting the execution of functions, and manipulating intermediate results [7]. Second, serverless computing also brings new security threats. In serverless computing, the source code of functions is compiled at the running environment of functions, and the compiling phase could also introduce security threats, such as Return-Oriented Programming (RoP) attacks [8]. Since serverless computing adopts lightweight virtualization solutions as the isolated running environment

of functions, such as Linux containers and Unikernels [9], these solutions only achieve weak isolation and make the above security threats more severe [10]. Therefore, it is very important to enhance the security of serverless workflow.

A typical solution is to deploy reactive defense mechanisms, such as Intrusion Detection Systems. However, we need to install additional software at the functions when deploying the reactive defense mechanisms, which could change the lightweight nature of functions [11]. Taking cues from proactive defense methodologies, notably Moving Target Defense (MTD) [12], fostering dynamism and diversity within functions serves to bolster security by impeding the Cyber Kill Chain (CKC) progression for adversaries [13]. Nonetheless, these dynamic and diverse solutions may inadvertently prolong the response time of functions. Therefore, designing a dynamic and diversity-based solution to safeguard serverless workflows with low cost becomes a fundamental question in serverless computing. This design faces two main challenges. First, how to apply the idea of dynamism and diversity to serverless computing needs to be determined. Although there have been many dynamic and diversity-based solutions, these solutions cannot be applied directly to serverless functions as the protection for the compiling phase is not considered. Second, we need to realize the tradeoff between security and makespan for workflows. The workflow makespan, representing the execution time for the workflow, is a common optimization objective in workflow scheduling [14]. However, the defense techniques inevitably increase defending cost and thus increase the makespan of the serverless workflow [15].

To solve these problems, we introduce a secure and makespan-conscious workflow execution framework for serverless computing, designated as SMWE. To address the first challenge, we use the compiler shifting method and running environment replacement method to change the compiling method of source code and running environment of functions (see Section 3). To tackle the second challenge, we devised the SMAM by scrutinizing attackers' behavioral patterns and strategically deploying our defense mechanisms within serverless workflows, balancing both security and execution time (see Sections 4 and 5). We implemented our framework leveraging Kubernetes and Fission, demonstrating its efficacy and scalability. In essence, we present a framework capable of autonomously crafting serverless workflow execution strategies, encompassing function scheduling and workflow defense tactics. To our knowledge, this paper pioneers the exploration of safeguarding serverless workflows at an affordable cost. Our key contributions are encapsulated as follows:

- We adopt the idea of dynamics and diversity to design a new approach for protecting functions from attacks in serverless workflow.
- We propose the SMAM to cover the complexity attack scenarios in serverless workflows. Additionally, the security and makespan of the serverless workflow are analyzed based on the SMAM.
- We consider deploying the proposed defense techniques selectively to functions in a serverless workflow. Additionally, we propose a distribution-optimization-based algorithm to obtain the optimal workflow defense strategy and function scheduling strategy.
- We implement our framework based on Kubernetes and Fission. Rigorous evaluations underscore that SMWE significantly enhances the security of serverless workflows while incurring minimal overhead in terms of execution time.

The rest of the paper is organized as follows. In Section 2, we review the related work. Then, we present our proposed defense techniques in Section 3. In Section 4, we propose the SMAM. In Section 5, we propose a distribution-optimization-based algorithm to decide how to apply the proposed defense techniques to serverless workflow. The design of our framework and its implementation details are discussed in Section 6. The performance evaluations are shown in Section 7. Finally, we summarize this paper and discuss the future work in Section 8.

2. Related Work

2.1. Serverless Security

As a promising paradigm in cloud computing, the security of serverless computing has attracted extensive attention. On the one hand, the security risks in traditional cloud computing still exist in serverless computing. To find out the security threat clearly in serverless computing, Baldini et al. [1] analyzed several widely used serverless platform and reached the conclusion that the lack of proper isolation among functions is a major problem. Gao et al. [16] presented the information leakage channels that are accessible within containers. Vulnerabilities are also rampant in serverless computing, which can be detected and exploited by attackers [17]. To tackle these problems, Sergei et al. [18] utilized Intel's SGX to build secure containers for better isolation. Nilton et al. [19] designed an automated threat mitigation architecture to detect vulnerabilities. On the other hand, the ephemeral nature of functions in serverless computing also introduce new risks, which were analyzed in [20]. Alpernas et al. [21] proposed an approach for securing serverless systems using dynamic information flow control. However, the environment reuse optimization for a warm start is abandoned, which may degrade the performance. Considering the environment reuse optimization, Arnav et al. [10] proposed a workflow-aware access control model, which can perform authentication and role assignment at the point of ingress. In this way, the attackers are not able to exploit the environment reuse optimization and move laterally.

2.2. Workflow Scheduling

The aim of workflow scheduling is to find suitable resources for tasks in the workflow, which have been widely researched in recent years. There is a variety of criteria for the design of the scheduling algorithm, such as makespan [14], cost [22] and security [13]. Haluk et al. [23] proposed a low-complexity scheduling algorithm called Heterogeneous Earliest Finish Time (HEFT) to optimize makespan, which is a dominating objective for workflow scheduling. Cost also has become an important criteria, especially in the cloud computing scenario [14,24]. Additionally, security has also attracted more and more attention in scheduling. Wang et al. [25] proposed an approach to encrypt the sensitive intermediate data to improve security. In this approach, the idle time slots on resources are fully utilized to mitigate the impact of data encryption time. Li et al. [26] proposed a meta-heuristic optimization algorithm to minimize the cost under the security constraint, which is modeled by the risk rate.

The serverless workflow scheduling is also receiving more and more attention with the development of serverless computing. Amoghvarsha et al. [27] presented a function-level scheduler for serverless platforms, which can minimize the resource costs under the Service Level Agreement constraint. The functions will be classified into different categories, based on which the function placement is determined. Additionally, the resource allocation among collocated functions is also carefully regulated. However, the structure of serverless workflows was not considered. Considering the workflow of serverless application, Ali et al. [28] pointed out that there are many problems for the execution of serverless workflows, including inflated application run times, function drops and inefficient allocations. Additionally, a framework which allows developers to easily define the execution of serverless workflow was proposed.

2.3. MTD

The concept of MTD has emerged as a proactive defense mechanism aiming to prevent network attacks. The idea of MTD is to alter the attack surface of the system, thus increasing the difficulty of performing cyberattacks [12]. From the prospective of network attacks, MTD approaches can be categorized into two different domains: exploration surface shifting and attack surface shifting. (1) The aim of exploration surface shifting is to interfere with the exploration actions of attackers such as scanning for open-ports, uncovering system topology and searching for vulnerabilities. For example, Albanese et al. [29] presented

the IP shifting method to assign the virtual IP addresses to the host in an unpredictable way. Meier et al. [30] proposed a network topology obfuscation framework to confuse attackers, which is achieved by intercepting and modifying path tracing probes directly in the data plane. (2) The idea of attack surface shifting is to make the attack action that the attacker selects after the exportation become invalid. For example, an attack action designed for a Linux-based Operating System (OS) will be useless in Windows OS. Using this idea, Carter et al. [31] proposed a framework to switch between different OSs based on their proposed notion of diversity between two OSs. Similarly, Han et al. [32] moved the running Virtual Machine (VM) to a different physical server, thus protecting it against the host-based cyberattack action.

Although we can improve the security of the system using proactive defense mechanisms, it also adds extra cost inevitably to implement these defenses [12]. For example, the IP shifting method will decrease the throughput of the network as we need to establish TCP connection frequently. The OS rotation method will cause performance degradation as additional time is needed to change the OS. Therefore, it is necessary to balance the security and cost when applying defense techniques.

3. Security Techniques Design

In this section, we first introduce preliminary knowledge about the function in serverless computing and its security threats. Then, we propose our key security techniques to enable life cycle protection for the functions.

3.1. The Threats of Functions in Serverless Computing

A function is the basic unit in serverless computing. Generally, the developer only needs to upload the source code of the function to a serverless provider, and function instances can be created based on it. Figure 1 shows a typical process for creating a function instance. As shown in Figure 1, there are four main phases for creating a function instance. First, an isolated environment is needed as the running environment for the function, such as Linux containers. Then, the source code of the function is pulled to the running environment and compiled into executable file. Finally, we can run the executable file and thus provide services to users.

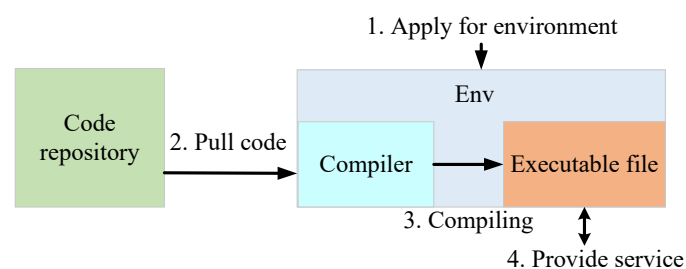


Figure 1. A typical process for creating a function instance in serverless computing.

Based on the creation process of the function, there are several weak points that are vulnerable to attackers. Both the executable file and running environment can be the attackers' target. For the executable file, attackers can decompile the executable file to the source code. In this way, the privacy information of the source code can be leaked. Additionally, attackers can analyze the source code to find vulnerabilities and launch attacks, such as an RoP attack. For the running environment, attackers can also use the vulnerabilities which can be obtained in Common Vulnerabilities and Exposures (CVE) and then turn a vulnerability into an exploit.

3.2. Key Defense Techniques

To tackle the security threats of functions, we adopt the idea of dynamism and diversity to provide protection for the executable file and running environment.

To protect the executable file, we apply the compiler shifting method to transform the same source code into executable files with different structure. Therefore, different function instances have different executable files, and the difficulty of decompiling the executable file is increased. In the compiler shifting method, we attempt to obscure the executable file by flattening the control flow [33]. Specifically, we can divide the source code of the function into multiple basic blocks, which can be seen as the basic parts in the control flow diagram of the function. In our method, the original control flow is transformed into a switch-case format, thus flattening the control flow. Figure 2 shows an example of flattening the control flow. As shown in Figure 2, all the basic blocks have the same entry block and end block after the transformation. Therefore, it can increase the difficulty for attackers in decompiling the executable file. However, flattening the control flow will also bring overhead when running the executable file [34]. Therefore, we will select a part of basic blocks randomly to be transformed into the switch-case format. As the basic blocks are selected randomly for every compiling phase, it diversifies the compiling phase and make it hard to obtain the regular pattern of the compiling and decompiling of the executable file. To safeguard the operational environment, we propose adopting a running environment replacement strategy, effectively disrupting the reconnaissance and attack phases of potential adversaries. Typically, attackers engage in extensive reconnaissance to uncover vulnerabilities. Subsequently, they exploit these vulnerabilities to launch attacks and maintain a foothold within the environment. By periodically implementing the environment replacement method, we introduce a fresh operational environment while retiring the old one. This approach ensures that attackers cannot continually build upon their reconnaissance efforts, as configurations like IP addresses and host servers may vary between iterations. Furthermore, it facilitates the eradication of any lingering threats, including activated malware, from the previous environment.

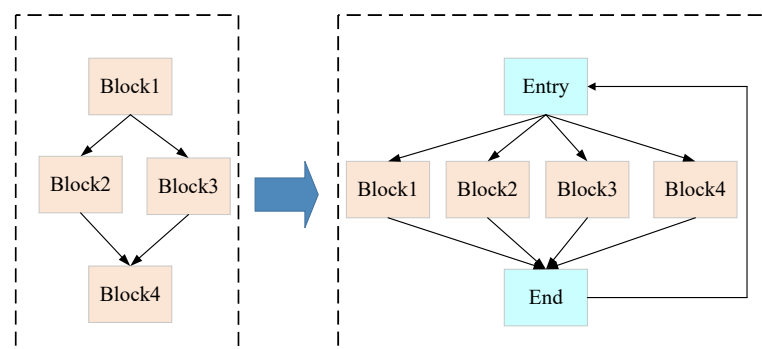


Figure 2. An example of compiler shifting method.

4. SMAM Model

Given the proposed defense strategies, further research is necessary to investigate how these techniques can be applied in serverless workflows to achieve an optimal balance between security enhancements and minimal makespan impact. For this problem, it is a general idea to apply the defense mechanism selectively to the functions. However, there lacks an analytical model to analyze the security and makespan of the serverless workflow. In this section, we present an analytical model for serverless workflow named SMAM. We first introduce the model of a serverless workflow and attackers' behavior pattern. Then, we establish the SMAM based on attack graphs [17].

4.1. Serverless Workflow and Attackers' Behavior Pattern

In serverless computing, a complicated application is provided based on the orchestration of a set of functions, which can be seen as a serverless workflow. In general, a serverless workflow can be modeled as a Directed Acyclic Graph (DAG) $G = (V, E)$, where V is the set of functions represented by vertices, and E is the set of precedent dependencies represented by edges [35]. We use v_i to represent the i -th function in the workflow and

assume that there are K functions making up the workflow; thus, V can be expressed as $V = \{v_1, v_2, \dots, v_K\}$. We use $e_{p,q} \in E$ to represent the data dependency from function v_p to function v_q , which indicates that the outputs of function v_p are inputs to function v_q . When executing a serverless workflow, limited resources will first be supplied based on capacity planning [36,37]. For example, the maximum number of containers is set in the main commercial serverless platforms, such as Amazon AWS Lambda, Google Cloud Functions and Microsoft Azure Functions [38]. Then, available running environments will be allocated to functions in the serverless workflow, which is called function scheduling.

Many workflows in serverless computing are of vital importance, such as some scientific workflows [5], which can easily become the target for attackers. Both the functions and its running environments can become the target of attacker. The attackers are assumed to launch attacks according to the CKC. To elaborate, attackers commence by conducting a thorough reconnaissance to uncover vulnerabilities within the workflow. Subsequently, they launch targeted attacks exploiting these vulnerabilities. Given the assumption that attackers operate externally to the cloud environment and all functions are potential targets, the workflow's entry functions are the initial focus of attack. Once established, attackers navigate the workflow based on function dependencies to reach their intended target. There exist two primary attack vectors: (1) data-transmission-based attack paths, shaped by the serverless workflow's architecture, as attackers can exploit legitimate communication channels; and (2) running-environment-based attack paths, stemming from the reuse of execution environments. For instance, if multiple functions share a container, attackers can hide within it, facilitating lateral movement from one function to another.

4.2. Analytical Modeling

Drawing from the aforementioned attackers' behavioral patterns, the SMAM is introduced, leveraging attack graphs to comprehensively represent the entirety of attack events. The SMAM can be seen as a directed graph $G^* = (N, C)$, where N denotes the set of function instances, and C denotes a set of attack paths between the function instances. Similarly, we use n_i to represent the i -th function instances in the workflow and assume that there are L function instances making up the graph; thus, N can be expressed as $N = \{n_1, n_2, \dots, n_L\}$. We employ the notation $c_{p,q} \in C$ to signify the attack path traversing from the function instance n_p to the function n_q . We note that the concept of function instances in the SMAM is different from functions in the DAG. A function instance n_p can be construed as the conjunction of a function v_p and its associated running environment denoted by $r_{H(p)}$, where $H(p)$ signifies the index of the allocated running environment specific to v_p .

Given an edge $c_{p,q} = (n_p \rightarrow n_q) \in C$, we define $DF(c)$ as the weight assigned to this edge, signifying the level of difficulty in launching an attack from the compromised function instance n_p to the targeted function instance n_q . To quantify this attack difficulty in a precise manner, we leverage the metrics outlined in the Common Vulnerability Scoring System (CVSS) (<https://www.first.org/cvss/>, accessed on 15 October 2022), which enable us to numerically assess the exploitation complexity associated with a specific vulnerability. It is noteworthy that the metrics employed within the CVSS framework have gained widespread adoption for evaluating and characterizing the security posture of an information system [17,39]. There are three parts of metrics in the latest CVSS v3.1: base score metrics, temporal score metrics and environmental score metrics. In base score metrics, the exploitability metric is used to reflect the exploiting difficulty of the vulnerability, which contains the following specific metrics: Attack vector (AV), Attack Complexity (AC), Privilege Required (PR) and User Interaction (UI). In our model, we define exploitability metric AD to access the attack difficulty for a specific vulnerability, which is expressed as

$$AD = (AV \times AC \times PR \times UI)^{-1}, \quad (1)$$

Given the target's operating environment, which harbors multiple vulnerabilities, an attacker can successfully compromise the target by exploiting any one of these vulnera-

bilities. Nevertheless, since we lack insight into the attackers' capabilities and behavioral preferences, it is reasonable to assume that all vulnerabilities within the target are potentially exploitable. To address this, we employ the Exploit Code Maturity (ECM) metric, a component of the Temporal Score Metrics within CVSS, to assess the likelihood of a vulnerability being exploited.

(1) Security: Using the analytical model above, we first characterize the security of the workflow. Given a compromised function instance n_p and the intended target function instance n_q , the level of attack difficulty can be categorized into the following three distinct scenarios.

case1: if n_p and n_q use the same running environment, the attackers may have the capability to remain concealed within the operating environment and proceed with their objectives, rendering the attack difficulty from n_p to n_q negligible, which can be mathematically represented as

$$DF(c_{p,q}) = 0, \quad H_p = H_q \quad (2)$$

case2: if n_p and n_q use different running environments and they have a dependency on data transmission, the attacks can access n_q and compromise n_q by exploiting vulnerabilities, which can be expressed as

$$DF(c_{p,q}) = \frac{\sum (ECM \times AD)}{\sum ECM}, e_{p,q} \in E \text{ and } H_p \neq H_q \quad (3)$$

case3: if n_p and n_q use different running environments and they do not have dependency on data transmission, the attackers cannot access n_q from n_p , which can be expressed as

$$DF(c_{p,q}) = +\infty, \quad e_{p,q} \notin E \text{ and } H_p \neq H_q \quad (4)$$

Based on the proposed weight of all edges, the security performance of the workflow can be characterized. We postulate that attackers act rationally and will initiate attacks via the shortest possible route to reach the target function instances. Consequently, the security performance of the target function instances, denoted as n_p , is defined as

$$S(n_p) = \sigma(n_{out}, n_p), \quad (5)$$

where n_{out} is the environment outside the cloud and only has access to the entry function instances of the workflow, and $\sigma(n_{out}, n_p)$ denotes the length of shortest path from n_{out} to n_p . As all the function instances may become the target of attackers, and we do not have any priori knowledge about it, we evaluate the overall security performance of the workflow by calculating the average security performance across all function instances within it, which can be calculated as follows:

$$S = \frac{1}{N} \sum_{i=1}^N \sigma(n_{out}, n_i). \quad (6)$$

(2) Makespan: The makespan represents the time from the start time of the first function to the finish time of the last function in the serverless workflow. It contains not only the execution time for functions, but also the communication time cost between functions. We define $EST(v_i, r_j)$ as the Earliest Start Time (EST) when function v_i is scheduled to running environment r_j . Similarly, $EFT(v_i, r_j)$ is defined as the earliest finish time when function v_i is scheduled to running environment r_j . For the entry function of the workflow, the EST can be derived as

$$EST(v_{entry}, r_j) = 0. \quad (7)$$

For the other function in the workflow, we first define $DF(e_{p,q})$ as the weight of edge $e_{p,q}$, which represents the size of data to be transmitted. Thus, the communication cost between v_p and v_q is defined by

$$T_{p,q} = \frac{DF(e_{p,q})}{R_{p,q}}, \quad (8)$$

where $R_{p,q}$ is the transfer rate from v_p to v_q . Then, the EST can be calculated starting from the entry function, which is shown as

$$EST(v_i, r_j) = \max \left\{ AT[r_j], \max_{v_m \in pre(v_i)} (EFT(v_m) + T_{m,i}) \right\}, \quad (9)$$

where $T_{m,i}$ can be calculated using Equation (8), $pre(v_i)$ denotes the set of immediate predecessor functions of function v_i , and $AT[r_j]$ is the earliest time at which running environment r_j is ready to execute the next function at the moment. Based on the earliest start time, the earliest finish time can be calculated as

$$EFT(v_i, r_j) = w_{i,j} + EST(v_i, r_j), \quad (10)$$

where $w_{i,j}$ is the execution time of function v_i in running environment r_j . The function execution time in different running environments can be measured by the average performance. After all the earliest finish times of functions in the workflow is derived, the makespan MS can be defined as

$$MS = \max \{ EFT(v_{exit}) \}. \quad (11)$$

where v_{exit} is the exit function of the workflow. If there are multiple exit functions in the workflow, we can select an exit function randomly.

5. Secure and Makespan-Aware Workflow Scheduling Algorithm

Based on the SMAM, the main problem is to build the strategy on how to apply the defense techniques to a serverless workflow. In this section, we first analyze the security–makespan tradeoff problem on the proposed defense techniques. Then we present a heuristic solution to this problem.

5.1. Problem Analysis

Based on our analytical model, we can analyze the security enhancement and makespan cost of the proposed defense techniques. For the compiler shifting method, the executable file can be protected from attacks. However, this method will make the logic at the compiling phase and execution phase more complex, which can lead to an increase in the execution time for the function. To balance the security–makespan tradeoff, we can select a part of basic blocks randomly to be transformed into a switch-case format. For the running environment replacement method, it can be used when the function has completed the execution. It can clean up the running environment and stop attackers from moving through the running environment. Based on the SMAM, the weights of edges in the SMAM can be influenced by this method. For example, if two function instances n_p and n_q use the same running environment, the attack difficulty from n_p to n_q is 0. If the running environment is replaced when n_p completes the execution, the attack difficulty from n_p to n_q will increase to a certain number, or increase to infinite, depending on whether there is access from n_p to n_q . Additionally, the running environment replacement method will also bring makespan cost, as it will increase $AT[r_j]$ in Equation (9), thus increasing the makespan. Therefore, the running environment replacement method also introduces a security–makespan tradeoff problem. To address this problem is very challenging, as both the function scheduling strategy and workflow defense strategy remain to be designed. On the one hand, to obtain the function scheduling strategy is a famous NP-hard problem [7]. On the other hand, there are 2^m available workflow defense strategy cases on how to apply the running environment replacement method for the workflow with m functions. Therefore, it is unrealistic to solve this problem by simply iterating through all the possible cases. Additionally, the workflow defense strategy interacts with the function scheduling strategy, which makes it more complex to design the function scheduling strategy and workflow defense strategy.

5.2. Heuristic Solution

To solve these problems, we adopt the idea of distribution optimization to schedule the function to the most suitable running environment and decide whether to apply the proposed mechanisms for each function in the serverless workflow. We use $B = \{b_{v_1}, b_{v_2}, \dots, b_{v_k}\}$ to denote the workflow defense strategy for all functions in the workflow, and the indicator $b_{v_i} = \{0, 1\}$, $b_{v_i} \in B$ is used to denote whether the running environment of function v_i will be replaced after the execution of function v_i . First, we need to decide the priorities of functions. Then, we iterate over all functions, taking into account their priorities, to formulate our workflow defense strategy and function scheduling approach. During the ranking phase, we employ the upward ranking methodology to sequence the functions within the serverless workflow. For a comprehensive understanding of the upward ranking method, please refer to [23]. For the scheduling of each function, we will first find out the relevant functions whose running environment replacement method has to be determined. For example, if function v_i is to be scheduled, there are two kinds of functions whose running environment replacement methods are necessary to determine: (1) the immediate predecessor functions of function v_i ; (2) the latest functions at every running environment. Then, we can go through all the possible cases which remain to be determined to find out the optimal workflow defense strategy and function scheduling strategy for each function.

As for the optimization goals at the scheduling phase, both security and makespan need to be taken into consideration. The structure of the DAG can determine a basic attack graph. Then, new attack paths may be introduced along with the building of the MTD deploying strategy and function scheduling strategy. We define P as the MTD deploying and function scheduling strategy that has been determined and A as a set of available strategies at the current scheduling phase, and the security loss can be calculated as

$$S_{loss,a} = S_{\{P\}} - S_{\{P,a\}}, a \in A, \quad (12)$$

where $S_{loss,a}$ denotes the security performances loss if strategy a is selected, $S_{\{G\}}$ denotes the security performance of workflow with the solution G . Similarly, we define the time loss $T_{loss,a}$ as the finish time for strategy a minus the earliest finish time, which can be expressed as

$$T_{loss,a} = T_{\{P,a\}} - \min_{i \in \{A\}} \{T_{\{P,i\}}\}, \quad (13)$$

where $T_{\{P\}}$ denotes the finish time of workflow with the solution P . Thus, the optimization goal TS at every scheduling can be defined as

$$TS = \min_{a \in A} \left(\alpha \frac{S_{loss,a}}{S_{\{P\}}} + (1 - \alpha) \frac{T_{loss,a}}{\min_{i \in \{A\}} \{T_{\{P,i\}}\}} \right). \quad (14)$$

In the optimization goal, the security loss and time loss are normalized, and $\alpha \in [0, 1]$ denotes the weight of security performance.

More details on our algorithm are shown in Algorithm 1. We note that our proposed algorithm is a low-complexity algorithm, which has an $O(e \times q)$ time complexity for e edges and q running environments in DAG. Therefore, our proposed algorithm can build the MTD deploying strategy and function scheduling strategy with low time cost, and it is practical to be applied to a serverless platform.

Algorithm 1 Proposed Algorithm

Require: Workflow DAG $G = (V, E)$, available running environments $R = \cup_{i=1}^m r_i$, security weight α .

- 1: **function** *proposedAlgorithm* (G, R, α)
- 2: $Rank \leftarrow \text{B-RANK}(G)$
- 3: $sched \leftarrow \phi$
- 4: $B \leftarrow \phi$
- 5: **for** $i = 1$ to k **do**
- 6: $B_{opt}^*, r_{opt} \leftarrow \text{optimalSche}(v_i, R, B, \alpha)$
- 7: $sched \leftarrow sched \cup r_{opt}$
- 8: $B \leftarrow B \cup B_{opt}^*$
- 9: **end for**
- 10: **return** $sched, B$
- 11: **end function**
- 12:
- 13: **function** *optimalSche* (v_i, R, B, α)
- 14: $predFuncs \leftarrow$ find the predecessor functions of v_i
- 15: **for** $j = 1$ to m **do**
- 16: $lastFunc \leftarrow$ find the last function on running environment r_j
- 17: $B^* \leftarrow$ contrasting $predFuncs, lastFunc$ with B to find the resource collection policies that have not been determined.
- 18: **for** $l = 1$ to $2^{\text{length}(B^*)}$ **do**
- 19: $eft(j, l) \leftarrow$ calculate EFT using Equations (9) and (10).
- 20: $sec(j, l) \leftarrow$ calculate security performance using Equations (5) and (6).
- 21: **end for**
- 22: **end for**
- 23: $TS(j, l) \leftarrow$ calculate the optimization goal using Equations (12)–(14) based on $eft(j, l)$ and $sec(j, l)$.
- 24: $B_{opt}^*, r_{opt} \leftarrow$ find the optimal resource policy and running environment.
- 25: **return** B_{opt}^*, r_{opt}
- 26: **end function**

6. SMWE Design and Implementation

In this section, we provide the design overview and implementation details of our framework.

6.1. Overview

We first introduce the overall architecture and the implementation details of SMWE. The overarching architecture of SMWE is depicted in Figure 3, encompassing four distinct modules: the Serverless Platform, Workflow Execution Engine, Semi-Global Scheduler, and Load Balancer. When there arrives an access request for the serverless workflow, the request is divided to a Semi-Global Scheduler to build the execution strategy by Loader Balancer. Then, the Workflow Execution Engine assumes the responsibility of executing the serverless workflow in accordance with the predefined execution strategy. The Serverless Platform can provide the basic running environment for a serverless workflow.

6.2. Implementation Detail

The detailed information on these modules is introduced as follows.

Loader Balancer: The Loader Balancer serves as the primary entry point to our framework, functioning as a reverse proxy that efficiently distributes workflow execution requests across a cluster of Semi-Global Schedulers. The use of Loader Balancer can increase the capacity and reliability of our framework. There are a variety of Loader Balancers in the real production environment, such as nginx, HAProxy, Linux Virtual Server and so on [40]. In our framework, we select nginx as the implementation of the Loader Balancer. Kubernetes also has excellent support for nginx by resource ingress-nginx.

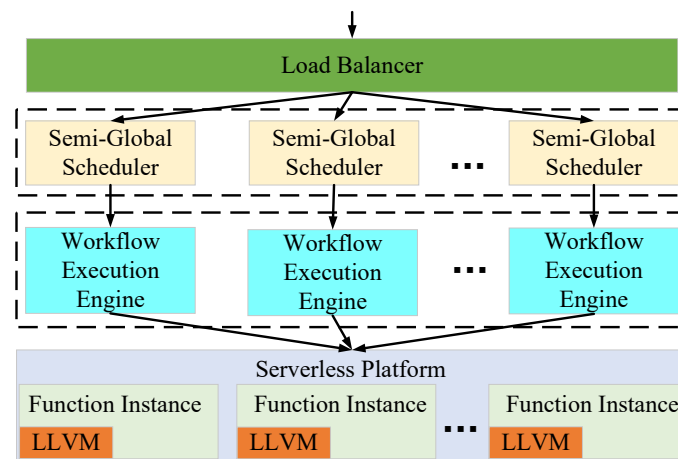


Figure 3. SMWE Framework Overview.

Semi-Global Scheduler: The Semi-Global Scheduler constitutes the pivotal module of our framework, tasked with determining the execution strategy for the workflow in accordance with the SAHEFT algorithm. To handle the multi-tenant serverless workflows, each Semi-Global Scheduler is in charge of a workflow. When a workflow execution request arrives, the Semi-Global Scheduler first applies for a number of running environments from serverless platform. The number of running environments can be decided according to the requirements of users. Then, the vulnerability database containing the CVSS score is created, which can be obtained in CVE [41]. The DAG and HAG of the workflow is built using the information of workflow and CVSS metrics. Based on the DAG, HAG and the pool of running environments, the scheduling plan can be obtained by the SAHEFT algorithm. Having completed the scheduling plan, the Semi-Global Scheduler transmits this plan to Workflow Execution Engine. We note that the resource pools among different Semi-Global Schedulers are totally isolated, thus eliminating the attacks among workflows. The Semi-Global Scheduler is implemented in a serverless way. The Semi-Global Scheduler is written in Golang, and we deploy the Semi-Global Scheduler as a function in Fission, and it will be triggered by the HTTP request from the Loader Balancer.

Workflow Execution Engine: The Workflow Execution Engine serves as the driving force behind workflow execution. Adhering to the scheduling plan, it allocates functions to designated running environments, initiates function execution, gathers function responses and assesses the need for replacing running environments post-execution. The Workflow Execution Engine operates as an HTTP server, receiving requests from the Semi-Global Scheduler, wherein the scheduling plan is embedded within the request for retrieval. In Fission, every container for function execution is equipped with a builder container called Fetcher. The scheduling plan is implemented using the interfaces exposed by Fetcher.

Serverless Platform: A serverless platform can shield the bottom information, and users can operate on just the code. In our framework, we select a Linux container as the running environment of the functions. Kubernetes and Fission are used as the serverless platform of our framework. Kubernetes is a famous open-source system for automating the deployment, scaling and management of containerized applications [42]. Fission is an open-source serverless framework for Kubernetes [43]. In order to achieve the compiler shifting method, we integrate the LLVM framework (<https://llvm.org>, accessed on 2 August 2024) into the running container for functions. In order to decrease the size of the running container image, only necessary modules in LLVM are integrated into the image.

7. Experiments and Discussion

In this section, we evaluate the performances of our framework. We first introduce the preliminary of our evaluation. Then, the usability and scalability of our framework are evaluated.

7.1. Experimental Setup

The experiments are carried out on the physical server equipped with 2.20 GHz 64-bit Intel Xeon CPU with 40-cores, 64 GB RAM, 2T disks and two network interfaces with 1 Gbps network speed. We focus on the makespan and security performance defined in our model to evaluate the effect of our framework. These two metrics can be obtained based on the MTD deploying strategy and function scheduling strategy. As the Linux container is used as our running environment, the vulnerabilities of the Linux container are collected from NVD (after 2019). The details of these vulnerabilities are depicted in Table 1.

Table 1. CVE vulnerabilities.

CVE ID	Author	Exploitability Metrics	Exploit Code Maturity
CVE-2019-13272	METASPLOIT	1.8	4.2
CVE-2019-13272	bcoles	1.0	5.6

7.1.1. Comparison Algorithms

A contrastive analysis is used to show the advantage of our framework. Three algorithms are considered as criteria to be compared with our framework, which is shown as follows:

HEFT: In this algorithm, the mapping between functions and running environments is decided, with the aim of optimizing makespan. However, the running environment replacement method is not used. This algorithm can represent the scenario where security performance is not considered.

CSHEFT (Cold-Start-based Heterogeneous Earliest Finish Time): In this algorithm, all the running environments are replaced after the execution of the function, which is also known as a “cold start” in serverless computing [10]. This algorithm can represent the scenario where the defense techniques are applied to serverless workflows without considering the cost.

RSHEFT (Random-Start-based Heterogeneous Earliest Finish Time): This algorithm is designed to compare the performance with our framework. RSHEFT is similar to our framework; the only difference is that in our framework, how to apply the defense techniques is obtained by optimizing the tradeoff between security and makespan based on our SAMA model, while in RSHEFT, how to apply the defense techniques is randomly selected.

SMWE: Our framework adopts the compiler shifting method and running environment replacement method to protect serverless functions from attacks. Furthermore, our proposed defense techniques are strategically applied to serverless workflows, ensuring a balance between security and makespan, as guided by the SMAM model.

7.1.2. Workflow Applications

In our evaluation, two kinds of workflow applications are considered: synthetic workflow with 10 functions, which has been used in [23], and real-world scientific workflow, which is called Montage. Detailed information about the synthetic workflow is shown in Figure 4. Alongside the synthetic workflow, Montage with a different number of functions is applied as the read-world workflow. Montage is a famous scientific workflow for image synthesis in astronomy [44]. As there has been a growing trend to execute scientific workflows in serverless platforms [5], Montage is used to show that our framework can also be applied to safeguard scientific workflows on serverless platforms.

7.2. Effectiveness Evaluation

To evaluate the effectiveness of our framework, makespan and security performance are measured to compare our framework with HEFT and CSHEFT, which is mostly used in the real-world serverless platform. The weight of security performance is set as 0.2 to satisfy the makespan constraint. The average evaluation results of synthetic workflow

and Montage workflow are shown in Figures 5 and 6. Comparing with the HEFT algorithm, we observe that the makespan of our framework is a little longer, but the security performance is greatly enhanced, especially for the scenario with little containers. When there are enough containers for the workflow functions, the effect of our framework is weakened. This is because HEFT tends to schedule functions to different containers with enough containers, which also reduces the risk of reusing containers. As for the CSHEFT algorithm, no new security threats are introduced as the replacement of each running environment; thus, CSHEFT can obtain the best security performance in these algorithms. However, the makespan is much longer. If there is a constraint for makespan, the only way for CSHEFT is to increase the number of available running environments. Additionally, inferring from Figure 5, increasing the number of containers cannot always decrease makespan, because of the data transmission time between functions. These factors limit the adoption of CSHEFT. Comparing the results for different workflows, we observe that workflows with more functions also need more running environments. As our framework can selectively apply the defense techniques carefully, a better tradeoff between security and makespan can be obtained with limited resources. Upon configuring an optimal number of containers (3 containers for synthetic workflows and 5 containers for Montage workflows), we observe a significant improvement in security performance by approximately 15 times, while incurring a negligible makespan cost of around 0.44% compared to the conventional HEFT algorithm.

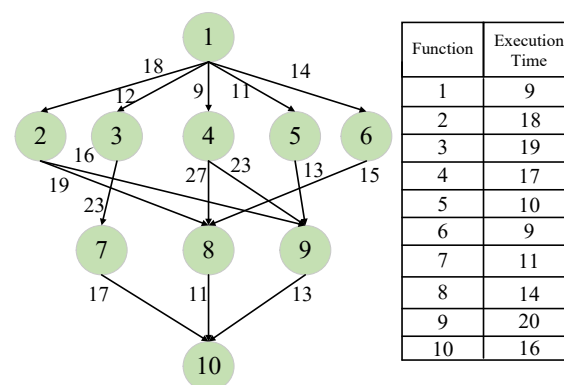


Figure 4. A synthetic workflow with 10 functions.

Then, we compare our framework with RSHEFT. The number of available containers is set as 3, and the different weight (ranging from 0 to 1) of security performance is set to obtain different scheduling results. The RSHEFT is repeated for 1000 times to obtain different results, which is depicted in Figure 7. It can be inferred that our framework has a better security performance than that of RSHEFT with the same makespan, which demonstrates the effectiveness of our framework. In the practical scenario, we can adjust the weight of the security performance to meet the makespan constraint and optimize the security performance using our framework.

7.3. Scalability Evaluation

Generally, some real-world workflows have a larger number of functions, such as some scientific workflows (e.g., Epigenomics, Inspiral, CyberShake and Montage) [45,46]. Accordingly, we conduct a set of simulations to evaluate the scalability of our algorithm. In our simulation, Montage workflows with 1000 functions are applied to evaluate the time cost for generating the scheduling solution. The ratio of scheduling cost to makespan is selected as the scalable property of our framework, which is shown in Figure 8. It can be inferred that the ratio of scheduling cost to makespan is less than 3% for Montage with 1000 functions. Thus, the cost of scheduling delay can be seen as negligible. Based on these

results, we can determine that our framework can support the workflow scheduling with thousands of functions.

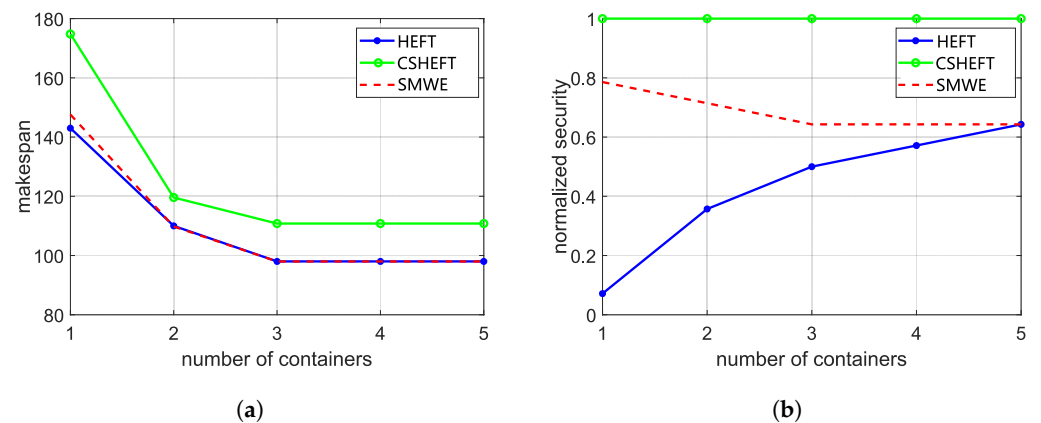


Figure 5. The average evaluation results of synthetic workflow. (a) The makespan of synthetic workflow; (b) The security of synthetic workflow.

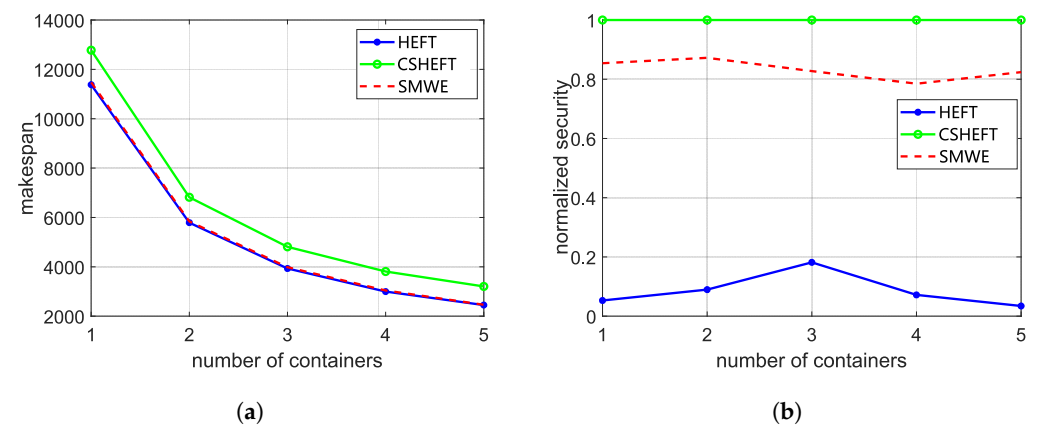


Figure 6. The average evaluation results of Montage workflow. (a) The makespan of Montage workflow with 1000 functions; (b) The security of Montage workflow with 1000 functions.

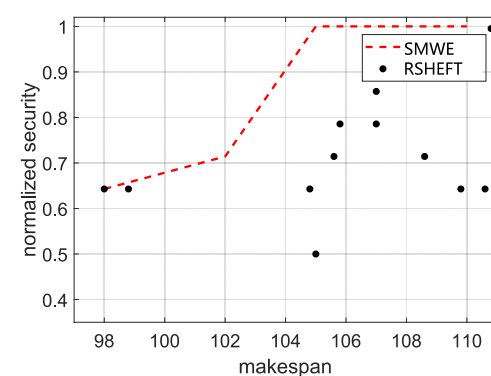


Figure 7. The makespan and security of synthetic workflow for different weight.

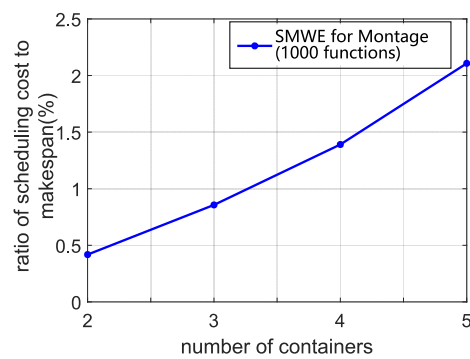


Figure 8. The ratio of scheduling cost to makespan for different number of containers.

8. Conclusions

The serverless workflow faces severe security threats, while using the idea of dynamism and diversity can provide an affordable and useful way to safeguard serverless workflows from attacks. In this paper, we present a framework called SMWE to manage the execution of serverless workflows. First, the compiler shifting method and running environment replacement method are proposed to provide an all-life-cycle defense for functions in serverless workflows. Then, the SMAM is proposed to model the attack events for a serverless workflow and characterize the security and makespan. Based on the SMAM, we study how to apply the proposed defense techniques to serverless workflows and propose a distribution optimization algorithm to build the MTD deploying strategy and function scheduling strategy, taking both security and makespan into consideration. Further, we introduce the framework design and implementation details and show how our framework can be embedded in a serverless platform. Our evaluation results show that SMWE can obtain the optimized tradeoff between security and makespan. When comparing our framework against the traditional HEFT algorithm, we witness a remarkable enhancement in security performance by approximately 15 times, accompanied by a marginal increase in makespan cost of roughly 0.44%. The proposed method not only protects functions from attacks in serverless workflows but also contributes to a deeper understanding of security measures in cloud computing environments. Furthermore, it can be practically implemented, managed and integrated into real-world serverless architectures, thereby enhancing the overall security posture while maintaining operational efficiency.

Author Contributions: Conceptualization, H.L. and S.Z.; methodology, H.L.; validation, H.L., S.Z. and X.L.; formal analysis, H.L. and X.L.; investigation, H.L. and S.Z.; resources, S.Z., G.C. and H.M.; data curation, H.L.; writing—original draft preparation, H.L., S.Z. and X.L.; writing—review and editing, H.L., Q.W.; supervision, S.Z. and G.C.; project administration, S.Z., H.M. and Q.W. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by National Natural Science Foundation of China (Grant No. 62002383), National Key Research and Development Program (Grant No. 2021YFB1006200, 2021YFB1006201).

Data Availability Statement: Dataset available on request from the authors.

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

Abbreviations

The following abbreviations are used in this manuscript:

SMWE	Secure and Makespan-oriented Workflow Execution
SMAM	Secure and Makespan-oriented Analytical Model
MTD	Moving Target Defense
CKC	Cyber Kill Chain
HEFT	Heterogeneous Earliest Finish Time

OS	Operating System
VM	Virtual Machine
CVE	Common Vulnerabilities and Exposures
CVSS	Common Vulnerability Scoring System
DAG	Directed Acyclic Graph
CSHEFT	Cold-Start-based Heterogeneous Earliest Finish Time
RSHEFT	Random-Start-based Heterogeneous Earliest Finish Time

References

- Baldini, I.; Castro, P.; Chang, K.; Cheng, P.; Fink, S.; Ishakian, V.; Mitchell, N.; Muthusamy, V.; Rabbah, R.; Slominski, A. Serverless Computing: Current Trends and Open Problems. In *Research Advances in Cloud Computing*; Springer: Singapore, 2017.
- Aditya, P.; Akkus, I.E.; Beck, A.; Chen, R.; Hilt, V.; Rimal, I.; Satzke, K.; Stein, M. Will serverless computing revolutionize NFV? *Proc. IEEE* **2019**, *107*, 667–678. [\[CrossRef\]](#)
- Yan, M.; Castro, P.; Cheng, P.; Isahagian, V. Building a chatbot with serverless computing. In Proceedings of the 1st International Workshop on Mashups of Things and APIs, Trento, Italy, 12–13 December 2016.
- Gimenez-Alventosa, V.; Molto, G.; Caballer, M. A framework and a performance assessment for serverless MapReduce on AWS Lambda. *Future Gener. Comput. Syst.* **2019**, *97*, 259–274. [\[CrossRef\]](#)
- Malawski, M.; Gajek, A.; Zima, A.; Balis, B.; Figiela, K. Serverless execution of scientific workflows: Experiments with HyperFlow, AWS Lambda and Google Cloud Functions. *Future Gener. Comput. Syst.* **2017**, *110*, 502–514. [\[CrossRef\]](#)
- Versluis, L.; Matha, R.; Talluri, S.; Hegeman, T.; Prodan, R.; Deelman, E.; Iosup, A. The workflow trace archive: Open-access data from public and private computing infrastructures. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 2170–2184. [\[CrossRef\]](#)
- Wang, Y.; Guo, Y.; Wang, W.; Liang, H.; Huo, S. INHIBITOR: An intrusion tolerant scheduling algorithm in cloud-based scientific workflow system. *Future Gener. Comput. Syst.* **2021**, *114*, 272–284. [\[CrossRef\]](#)
- Prandini, M.; Ramilli, M. Return-oriented programming. *IEEE Secur. Priv.* **2012**, *10*, 84–87. [\[CrossRef\]](#)
- Lin, C.; Khazaei, H. Modeling and optimization of performance and cost of serverless applications. *IEEE Trans. Parallel Distrib. Syst.* **2021**, *32*, 615–632. [\[CrossRef\]](#)
- Sankaran, A.; Datta, P.; Bates, A. Workflow integration alleviates identity and access management in serverless computing. In Proceedings of the Annual Computer Security Applications Conference, Austin, TX, USA, 7–11 December 2020.
- Jonas, E.; Schleier-Smith, J.; Sreekanti, V.; Tsai, C.C.; Khandelwal, A.; Pu, Q.; Shankar, V.; Carreira, J.; Krauth, K.; Yadwadkar, N. Cloud programming simplified: A Berkeley view on serverless computing. *arXiv* **2019**, arXiv:1902.03383.
- Cho, J.H.; Sharma, D.P.; Alavizadeh, H.; Yoon, S.; Ben-Asher, N.; Moore, T.J.; Kim, D.S.; Lim, H.; Nelson, F.F. Toward proactive, adaptive defense: A survey on moving target defense. *IEEE Commun. Surv. Tutor.* **2020**, *22*, 709–745. [\[CrossRef\]](#)
- Wang, Y.; Guo, Y.; Guo, Z.; Liu, W.; Yang, C. Protecting scientific workflows in clouds with an intrusion tolerant system. *IET Inf. Secur.* **2019**, *14*, 157–165. [\[CrossRef\]](#)
- Chen, Z.; Zhan, Z.; Lin, Y.; Gong, Y.; Gu, T.; Zhao, F.; Yuan, H.; Chen, X.; Li, Q.; Zhang, J. Multiobjective cloud workflow scheduling: A multiple populations ant colony system approach. *IEEE Trans. Cybern.* **2019**, *49*, 2912–2926. [\[CrossRef\]](#) [\[PubMed\]](#)
- Bardas, A.G.; Sundaramurthy, S.C.; Ou, X.; Deloach, S.A. MTD CBITS: Moving target defense for cloud-based IT systems. In Proceedings of the European Symposium on Research in Computer Security, Oslo, Norway, 11–15 September 2017.
- Gao, X.; Steenkamer, B.; Gu, Z.; Kayaalp, M.; Pendarakis, D.; Wang, H. A study on the security implications of information leakages in container clouds. *IEEE Trans. Dependable Secur. Comput.* **2021**, *18*, 174–191. [\[CrossRef\]](#)
- Jin, H.; Li, Z.; Zou, D.; Yuan, B. DSEOM: A framework for dynamic security evaluation and optimization of MTD in container-based cloud. *IEEE Trans. Dependable Secur. Comput.* **2019**, *18*, 1. [\[CrossRef\]](#)
- Arnautov, S.; Trach, B.; Gregor, F.; Knauth, T.; Martin, A.; Priebe, C.; Lind, J.; Muthukumaran, D.; O’Keeffe, D.; Stillwell, M.; et al. SCONE: Secure Linux containers with intel SGX. In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation, Savannah, GA, USA, 2–4 November 2016.
- Bila, N.; Dettori, P.; Kanso, A.; Watanabe, Y.; Youssef, A. Leveraging the serverless architecture for securing linux containers. In Proceedings of the 37th IEEE International Conference on Distributed Computing Systems Workshops, Atlanta, GA, USA, 5–8 June 2017.
- Li, X.; Leng, X.; Chen, Y. Securing serverless computing: Challenges, solutions, and opportunities. *IEEE Netw.* **2023**, *37*, 166–173. [\[CrossRef\]](#)
- Alpernas, K.; Flanagan, C.; Fouladi, S.; Ryzhyk, L.; Sagiv, M.; Schmitz, T.; Winstein, K. Secure serverless computing using dynamic information flow control. *Proc. ACM Program. Lang.* **2018**, *2*, 118. [\[CrossRef\]](#)
- Elgamal, T. Costless: Optimizing cost of serverless computing through function fusion and placement. In Proceedings of the 2018 IEEE/ACM Symposium on Edge Computing, Seattle, WA, USA, 25–27 October 2018.
- Topcuoglu, H.; Hariri, S.; Wu, M.Y. Performance-effective and low-complexity task scheduling for heterogeneous computing. *IEEE Trans. Parallel Distrib. Syst.* **2002**, *13*, 260–274. [\[CrossRef\]](#)
- Wang, Z.J.; Zhan, Z.H.; Yu, W.J.; Lin, Y.; Zhang, J.; Gu, T.L.; Zhang, J. Dynamic group learning distributed particle swarm optimization for large-scale optimization and its application in cloud workflow scheduling. *IEEE Trans. Cybern.* **2020**, *50*, 2715–2729. [\[CrossRef\]](#) [\[PubMed\]](#)

25. Chen, H.; Zhu, X.; Qiu, D.; Liu, L.; Du, Z. Scheduling for workflows with security-sensitive intermediate data by selective tasks duplication in clouds. *IEEE Trans. Parallel Distrib. Syst.* **2017**, *28*, 2674–2688. [\[CrossRef\]](#)
26. Li, Z.; Ge, J.; Yang, H.; Huang, L.; Hu, H.; Hu, H.; Luo, B. A security and cost aware scheduling algorithm for heterogeneous tasks of scientific workflow in clouds. *Future Gener. Comput. Syst.* **2016**, *65*, 140–152. [\[CrossRef\]](#)
27. Suresh, A.; Gandhi, A. FnSched: An efficient scheduler for serverless functions. In Proceedings of the 5th International Workshop on Serverless Computing, Davis, CA, USA, 9–13 December 2019.
28. Tariq, A.; Pahl, A.; Nimmagadda, S.; Rozner, E.; Lanka, S. Sequoia: Enabling quality-of-service in serverless computing. In Proceedings of the SoCC '20: ACM Symposium on Cloud Computing, Virtual Event, USA, 19–21 October 2020.
29. Albanese, M.; Benedictis, A.D.; Jajodia, S.; Sun, K. A moving target defense mechanism for MANETs based on identity virtualization. In Proceedings of the IEEE Conference on Communications and Network Security, National Harbor, MD, USA, 14–16 October 2013.
30. Meier, R.; Tsankov, P.; Lenders, V.; Vanbever, L.; Vechev, M.T. NetHide: Secure and practical network topology obfuscation. In Proceedings of the 27th USENIX Security Symposium, Baltimore, MD, USA, 15–17 August 2018.
31. Carter, K.M.; Riordan, J.; Okhravi, H. A game theoretic approach to strategy determination for dynamic platform defenses. In Proceedings of the First ACM Workshop on Moving Target Defense, Scottsdale, AZ, USA, 7 November 2014.
32. Han, Y.; Alpcan, T.; Chan, J.; Leckie, C.; Rubinstein, B.I.P. A game theoretical approach to defend against co-resident attacks in cloud computing: Preventing co-residence using semi-supervised learning. *IEEE Trans. Inf. Forensics Secur.* **2016**, *11*, 556–570. [\[CrossRef\]](#)
33. Jackson, T.; Salamat, B.; Homescu, A.; Manivannan, K.; Franz, M. *Compiler-Generated Software Diversity*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 77–98.
34. Boehm, B.W. Software cost estimation meets software diversity. In Proceedings of the 39th International Conference on Software Engineering, Buenos Aires, Argentina, 20–28 May 2017.
35. Li, X.; Zhang, L.; Wu, Y.; Liu, X.; Zhu, E.; Yi, H.; Wang, F.; Zhang, C.; Yang, Y. A novel workflow-level data placement strategy for data-sharing scientific cloud workflows. *IEEE Trans. Serv. Comput.* **2019**, *12*, 370–383. [\[CrossRef\]](#)
36. Li, Z.; Jin, H.; Zou, D.; Yuan, B. Exploring new opportunities to defeat low-rate DDoS attack in container-based cloud environment. *IEEE Trans. Parallel Distrib. Syst.* **2020**, *31*, 695–706. [\[CrossRef\]](#)
37. Yuan, B.; Zhao, H.; Lin, C.; Zou, D.; Yang, L.T.; Jin, H.; He, L.; Yu, S. Minimizing financial cost of DDoS attack defense in clouds with fine-grained resource Management. *IEEE Trans. Netw. Sci. Eng.* **2020**, *7*, 2541–2554. [\[CrossRef\]](#)
38. Wang, L.; Li, M.; Zhang, Y.; Ristenpart, T.; Swift, M.M. Peeking behind the curtains of serverless platforms. In Proceedings of the 2018 USENIX Annual Technical Conference, Boston, MA, USA, 11–13 July 2018.
39. Hong, J.B.; Kim, D.S. Assessing the effectiveness of moving target defenses using security models. *IEEE Trans. Dependable Secur. Comput.* **2016**, *13*, 163–177. [\[CrossRef\]](#)
40. Cui, J.; Lu, Q.; Zhong, H.; Tian, M.; Liu, L. A load-balancing mechanism for distributed SDN control plane using response time. *IEEE Trans. Netw. Serv. Manag.* **2018**, *15*, 1197–1206. [\[CrossRef\]](#)
41. National Vulnerability Database. 2021. Available online: <https://nvd.nist.gov/vuln> (accessed on 5 January 2021).
42. Kubernetes. 2021. Available online: <https://kubernetes.io> (accessed on 3 September 2021).
43. Fission. 2021. Available online: <https://fission.io> (accessed on 10 August 2021).
44. Wang, Y.W.; Jiang-Xing, W.U.; Guo, Y.F.; Hong-Chao, H.U.; Liu, W.Y.; Cheng, G.Z. Scientific workflow execution system based on mimic defense in the cloud environment. *Front. Inf. Technol. Electron. Eng.* **2018**, *19*, 1522–1536. [\[CrossRef\]](#)
45. Wang, Y.; Guo, Y.; Guo, Z.; Liu, W.; Yang, C. Securing the intermediate data of scientific workflows in clouds with ACISO. *IEEE Access* **2019**, *7*, 126603–126617. [\[CrossRef\]](#)
46. Niu, M.; Cheng, B.; Feng, Y.; Chen, J. GMTA: A geo-aware multi-agent task allocation approach for scientific workflows in container-based cloud. *IEEE Trans. Netw. Serv. Manag.* **2020**, *17*, 1568–1581. [\[CrossRef\]](#)

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.