Serverless Computing: A Security Perspective

Eduard Marin[1*], Diego Perino[1] and Roberto Di Pietro[2]

Abstract

In this article we review the current serverless architectures, abstract and categorize their founding principles, and provide an in-depth security analysis. In particular, we: show the security shortcomings of the analyzed serverless architectural paradigms; point to possible countermeasures; and, highlight several research directions for practitioners, Industry, and Academia.

Keywords: Cloud computing; Serverless computing; Security; Threat models; Vulnerabilities; Architectures

1 Introduction

Virtualization technologies have played a crucial role for the wide adoption and success of cloud computing [1]. They allowed cloud providers to simultaneously share their resources with many users by placing their monolithic applications inside virtual machines (VMs), offering strong isolation guarantees while providing users with an (apparently) infinite amount of resources, readily available when their applications needed them. The cited features, together with a pay-per-use business model that has contributed to lower the total cost of ownership for cloud users, have made cloud computing the most successful computing paradigm of the last decade. However, this success also came with some drawbacks, the major one being the need for the users to directly manage the VMs [2].

In response to the above issue, we have witnessed the emergence of new programming models that drastically changed the way software developers develop and manage applications for the cloud. One such programming model relies on decomposing an application into multiple, autonomous, limited scope and loosely coupled components—also known as *microservices*—that can communicate with each other via standard APIs. Unfortunately, due to their long startup time and high resource usage, VMs were proven to be inefficient for running microservices. This led to the proposal of several container technologies (e.g., Docker) as a lighter alternative. Containers offer increased portability, lower start up time, and greater resource utilization than VMs, simplifying the development and management of large-scale applications in the cloud. These advantages have led cloud providers to adopt container technologies and to use them in combination with orchestration platforms (e.g., Kubernetes or Docker Swarm) to fully automate the deployment, scaling, and management of microservice-based applications in the cloud. However, similarly to when VMs are used, the microservices paradigm still requires users to configure and manage the underlying containers (e.g., related libraries and software dependencies), and relies on a static billing model where users pay a fixed amount for the allocated resources and not for the resources actually consumed. The cited points render microservices unsuitable for certain types of applications.

Serverless computing is emerging as a new computing paradigm for the deployment of applications in the cloud[1]. It has two important advantages over its predecessors. Firstly, it allows software developers to outsource all infrastructure management and operational tasks to cloud providers, which makes it possible for them to focus only on the business logic of their applications [3, 4]. Secondly, it follows a pure pay-per-use model, where users are only charged based on the resources they consume. Currently, serverless computing comes in two different flavors, known as backend as a service (BaaS) and function as a service (FaaS). The core idea behind BaaS is to provide software developers a set of services and tools (e.g., databases, APIs, file storage or push notifications) to ease and speed up the development of mobile and web applications. As per FaaS, it focuses on allowing software developers to deploy and execute their own functions on the cloud (note that the functions can also utilize additional services as those offered in BaaS). To date, FaaS is considered as the most dominant serverless model. In the rest of this article, we will use the term "serverless" to refer to FaaS.

Due to its simplicity and economical advantages, serverless computing is gaining significant attention in

*Correspondence: eduard.marinfabregas@telefonica.com
[1]Telefonica Research, Barcelona, Spain
Full list of author information is available at the end of the article

[1]It is worth noting that the term "serverless" does not mean that there are no servers, but rather that software developers do not need to worry about configuring and managing them.

the industry as a compelling paradigm to deploy applications and services in the cloud. Cloud providers, such as Amazon [5], Microsoft [6], Google [7], IBM [8] or Alibaba [9] are already offering serverless computing services to their customers. Similarly, many enterprises, such as Netflix, T-Mobile and Realtor, are already reaping the benefits of serverless computing [10]. According to recent market surveys, the serverless computing market is expected to grow at a CAGR of 26% during the period 2020-2029 [11]. However, with the increase in volume and diversity of attacks against the cloud, it becomes apparent that security and privacy will be a key factor which, if not addressed, could hamper the widespread adoption of serverless computing.

In particular, as per serverless security, at first glance one could argue that serverless computing is intrinsically more secure than its predecessors because of its characteristics (e.g., the short duration of functions), or due to the fact that it could inherit security features already developed for other virtualization solutions. Yet, as we will show in the following, serverless brings many new, idiosyncratic security challenges that open the door for new types of security attacks. Further, implementing serverless applications requires a major change in mindset from software developers [12], not only in the way applications are written but also in the way they are protected from security attacks [13]. These latter requirements are rarely met, hence introducing new vulnerabilities.

Contribution. This work is, to the best of our knowledge, the first structured and principled attempt to shed light on the security of serverless computing. In particular, in this paper, we first review and categorize state of the art serverless solutions; later, we analyze pros and cons of the introduced architectural categories; further, we assess, from a security perspective, the fundamental principles of the main revised architectural choices. Finally, starting from the highlighted weaknesses, we sketch a few solutions and provide several research directions, appealing to practitioners, Industry, and Academia, to further enhance the security of the serverless ecosystem as a whole.

Roadmap. The sequel of this paper is structured as follows: In Section 2 we provide the necessary background for the reader to understand the main motivation behind serverless computing, its advantages, the components any serverless platform is composed of and the security mechanisms that are commonly employed to thwart attacks. Section 3 defines the threat model. Section 4 discusses some inherent properties of serverless that are beneficial security-wise while in Section 5 we cover aspects of serverless that could negatively affect security. In Section 6 we devise a list of possible application-level and infrastructure-level attacks which we believe deserve further attention. This is followed by possible countermeasures to alleviate them. Finally, Section 7 provides concluding remarks.

2 Background

In this section, we revise the current serverless ecosystem. More concretely, we first briefly introduce serverless computing, then analyze the five main elements any serverless platform is composed of, and finally discuss the currently available security solutions.

2.1 Serverless computing

In serverless computing, the application logic is divided into a set of small, short-lived and stateless functions, each one running within a separate execution environment (e.g., a container), that communicate with each other and with various cloud services (e.g., storage services) to carry out their tasks. By using stateless functions, serverless computing decouples storage from computation, making it possible to provision, manage, and price these two elements separately. Furthermore, in this context the cloud provider is now responsible for automatically and transparently spawning and managing function instances in worker nodes as well as performing all operational tasks (e.g., server and OS maintenance, patching, logging, load balancing or auto-scaling). Finally, serverless computing also significantly reduces application deployment cost via a pure pay-per-use model where users are only billed based on the resources (e.g., CPU, network or memory) they consume.

Besides the clear advantages serverless offers to software developers (in terms of flexibility, scalability, performance and costs), it is worth noting that cloud providers can benefit from using it too. As functions are invoked only occasionally and are executed for a very short period of time, cloud providers can achieve a higher degree of co-location in their servers and further optimize the usage of their resources. These two latter points, when carefully planned and orchestrated, can result in an even more profitable model for cloud providers.

2.2 Serverless ecosystem

As shown in Figure 1, a serverless platform is comprised of (at least) five elements: i. functions; ii. API gateways; iii. (shared) cloud services; iv. security tools; and, v. control plane.

1) Functions. Functions are the core component of serverless platforms. They can be written in many different programming languages (e.g., JavaScript, Python, and Go). Software developers can either write them by themselves, rely on open-source third-party functions (e.g., [14]), or use proprietary functions for

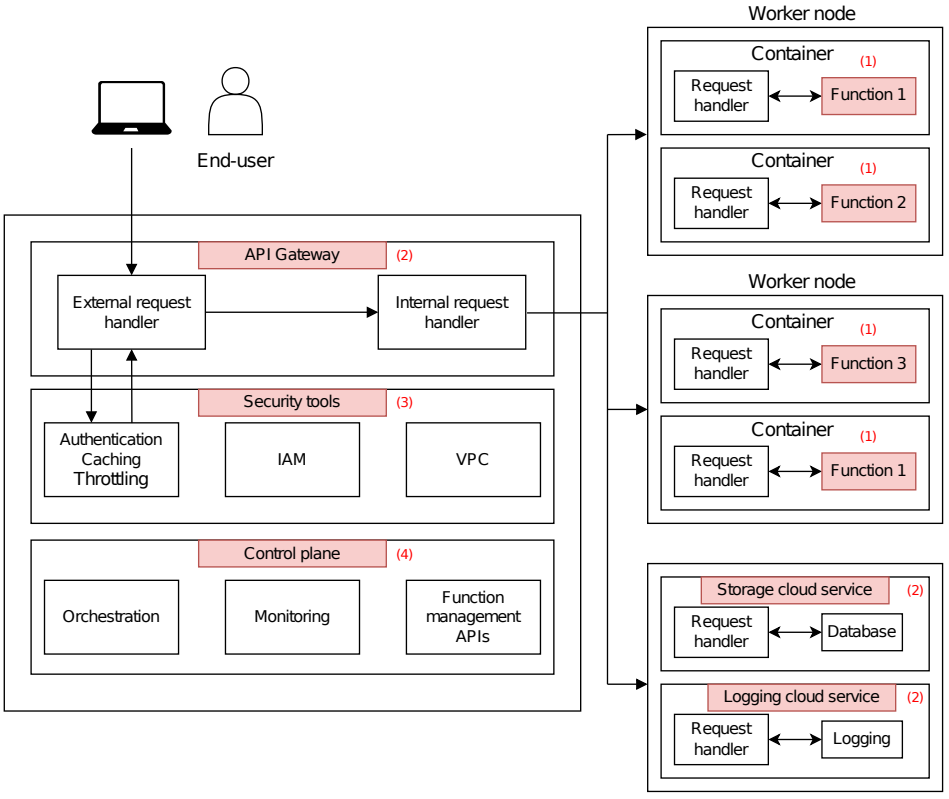

Figure 1 Serverless platform

which they must pay licensing fees. Functions are typically run inside a newly-generated, isolated execution environment (e.g., a container) within a worker node. The cited functions are executed in response to external and/or internal events specified by application owners (e.g., HTTP requests, modification to objects in storage, table updates, or function transitions). It is worth noting that not all defined functions must necessarily communicate directly with the outside world (it may be the case that there are functions that can only communicate with other functions and cloud services and that are not directly accessible from the outside).

2) Cloud services. Current serverless platforms integrate a wide range of cloud services used to extend the functions' capabilities, e.g., to collect various types of data (e.g., using Amazon Kinesis), to quickly react to events (e.g., using Google cloud pub/sub message bus system or API Gateways), to manage the entire application lifecycle and enable DevOps capabilities (e.g., using Microsoft Azure DevOps) or to achieve long- and short-term storage (e.g., using Amazon S3 and DynamoDB).

3) Security tools. Cloud providers make available to software developers a set of tools and services to ease workflows security management. Some of these tools and services are also used in the context of microservices; however, the task of configuring them correctly becomes much more challenging in serverless. For example, the Identity and Access Management (IAM) service, which allows the configuration of fine-grained access controls to authenticate and restrict the resources functions have access to. Another widely used security service is the so called Virtual Private Cloud (VPC), which allows the creation of private, isolated networks for secure communications between applications that belong to the same organization. In addition to the cited services, we believe that other services and tools, such as those used for Runtime Application Self-Protection (RASP), Infrastructure as Code (IaC) scanning, and source code composition analysis, can play an important role in protecting serverless applications and serverless platforms against attacks.

4) Control plane functionalities. Serverless platforms typically comprise multiple control plane functionalities for cloud providers to operate, manage, and monitor their infrastructures. For example, there is an orchestrator component that handles the process of assigning functions to worker nodes. Similarly, a monitoring component is used to periodically check the status of worker nodes, the software they execute,

as well as the execution environments that run on them. To this end, the monitoring component gathers metering data, logs, and a few metrics emitted by the worker nodes. This way, if a failure is detected, the affected functions can be quickly instantiated in other worker nodes. While the functionalities can vary slightly across several serverless platforms, all of them have in common the fact that the data plane needs to receive periodic configuration updates from the control plane, and the control plane needs to frequently receive (or collect) operational state from the data plane. Thus, it is fundamental for the control plane to stay in sync with the data plane.

2.3 Existing infrastructure-level security controls

Today's serverless platforms typically run functions inside containers (or similar execution environments) that are protected by various open-source security mechanisms and services (some of which are built into the Linux kernel) in combination with security mechanisms developed by the cloud providers themselves. In the following, we focus solely on open-source, widely used security mechanisms (as the mechanisms developed by cloud providers are typically adhoc, and are often not public or well documented). Note that these security mechanisms play an important role in the security of execution environments used in today's production environments like g-Visor and Firecracker; g-Visor is essentially an additional security layer that is developed atop the Linux security mechanisms, whereas the Firecracker sandboxes (run in user space) are also restricted by Linux security mechanisms like seccomp, cgroup, and namespaces.

These security mechanisms can be clustered into the following four categories: (i) host hardening; (ii) isolation of processes; (iii) network security; and, (iv) access control. For an overview of security mechanisms in the first three categories, we refer the reader to [15] (as these mechanisms are generally applicable to containers regardless of what is executed inside them). As for access control, cloud providers typically offer several mechanisms built-in in the API gateway to throttle, cache, authenticate, and authorize external API calls before passing the requests to the corresponding functions, e.g., relying on external identity providers or specifying a range of IP addresses from which legitimate requests can originate.

Security of current mechanisms. Over the last few years, researchers have investigated in depth the real security guarantees provided by the existing mechanisms used to protect container-based infrastructures. This resulted in the identification of serious weaknesses in the security mechanisms used for process isolation [16] and network security [17]. In addition, previous work pointed out that host hardening mechanisms, such as seccomp, AppArmor and SELinux, require cloud operators to manually configure them, which is a laborious task that is prone to errors.

3 Threat Model

Serverless platforms are complex and dynamic ecosystems with many distinct components. To design a secure serverless ecosystem, one must consider the security provided by each of its components and their interplay. Further, to properly frame the serverless security ecosystem, as it will be done in the sequel, we first need to define the corresponding threat model. To this aim, we mainly distinguish between two types of adversaries: i. *external*; and, ii. *internal*, discussed in the following.

External adversaries typically carry out their attacks from outside the cloud by leveraging user-controlled input fields in any of the existing APIs that are offered to handle events. The same is true for serverless platforms. These attacks can enable adversaries to run arbitrary commands inside the function in order to retrieve sensitive data (e.g., session tokens stored in environment tables) or tamper with the execution of any function (or cloud service that receives maliciously-crafted input data and does not apply proper input data sanitation techniques). While some injection attacks are well-known because they are applicable to standard web applications (e.g., those ones that exploit cross-site scripting or the ones based on code/command injection), serverless functions can instead be triggered from many different event sources—this latter feature broadening significantly their attack surface [18, 19, 20].

Internal adversaries refer to adversaries who have full control of one (or more) functions and conduct attacks from inside the cloud. In the case of public clouds, it is relatively easy for such adversaries to deploy malicious functions in order to attempt to perform attacks from the inside. These adversaries can attempt to: i. create covert channels [21, 22]; ii. conduct privilege escalation attacks (e.g., to compromise other co-resident functions or worker nodes) [23]; iii. retrieve or tamper with sensitive data (e.g., data in storage services) [24]; iv. gather knowledge about runtime environments and infrastructure [22]; or, v. conduct various types of Denial-of-Service (DoS) attacks [25] (including so called Denial-of-Wallet attacks) [22, 26, 27, 28]. In a separate line of work, researchers have also shown that if registry services exist where serverless functions developed by other software developers can be found, adversaries with access to the registry can carry out so called typosquatting

attacks [29]. The goal of such attacks is to distribute malicious container images by exploiting the potential typos made by container users. Similarly, there exist attacks whereby the goal of adversaries is to influence the scheduler to co-locate the attacker's application with a targeted victim applications [30, 31]. It is worth mentioning that co-location is an important prerequisite to perform certain attacks like Rowhammer [32], Spectre [33] or Meltdown [34].

Though privacy concerns are out of the scope of this paper, it is also worth mentioning that from a privacy standpoint there is an increasing concern that cloud providers can inadvertently or deliberately reveal sensitive data to third-parties (e.g., through malicious insiders). Due to this latter threat, within the research community it is common to model cloud providers as *honest-but-curious* entities. Under this model, cloud providers are assumed to run their customers functions as intended but, at the same time, they may try to learn as much information as possible about the ongoing computations and hosted data.

In the next sections we analyze the impact of serverless computing on security, discussing the pros and cons of the paradigm in relationship with its contribution to the security posture of the supported ecosystem.

4 Serverless as a Security Enabler

In this section, we discuss some principles and use cases related to the inherent advantages of serverless from a security point of view.

4.1 Increased difficulty in performing attacks

The fact that serverless functions have small code footprints, are stateless, and short-lived, significantly raises the bar for adversaries to successfully execute their attacks. Indeed, serverless imposes strict limits on the time available to adversaries for retrieving sensitive data from functions or for performing *lateral movements* in order to carry out more sophisticated attacks. The highlighted features are important because experience has shown that adversaries who compromise servers often remain undetected for very long periods, carrying out malicious activities at a very slow pace, not to generate signals that could lead to detection—this is commonly known as advanced persistent threats (APTs).

The consequences of such long-lasting attacks can be severe, ranging from intellectual property theft (e.g., trade secrets or patents), compromised sensitive information (e.g., employees and users private data) to total site takeovers. With serverless, long-standing servers do *not* exist, thus adversaries must carry out their attacks—including the reconnaissance phase—again and again, increasing both the attack costs and the risks of being detected. Additionally, by using small, single-purpose functions to realize applications, serverless allows not only the definition of more fine-grained security policies, but also a significant reduction of the impact of attacks. Adversaries who compromise a function can now only exploit the "capabilities" of such a function. Finally, due to the way applications are designed when using the serverless paradigm, not all serverless functions need to send results back to the Internet, hence hampering adversaries from conducting some types of attacks (e.g., those ones that aim to exfiltrate sensitive data).

4.2 Less security responsibilities for software developers

Unlike prior cloud programming models, where software developers play an important role in the security of their applications, serverless security is a shared responsibility between software developers and cloud providers. While the cited model alleviates some security concerns (mainly those caused by infrastructure management), it still requires software developers to be heavily involved in security matters.

When it comes to serverless security, it is common to distinguish between *"security of the cloud"* and *"security in the cloud"*, as below detailed.

"Security of the cloud" is the responsibility of cloud providers and encompasses all measures in place to keep the underlying infrastructure and cloud services (e.g., the execution environments on which functions run or the virtualization layer) secure from adversaries. Although software developers have less control and require trust in the chosen cloud provider, delegating all infrastructure-related security tasks to cloud providers is considered to be an effective mechanism to eliminate a wide number of attacks. Providing, maintaining and operating an infrastructure that is secure by design is the core business model of cloud providers offering serverless and hence one of their main focuses.

"Security in the cloud", instead, is the responsibility of software developers. It refers to the security mechanisms employed to: i. prevent vulnerabilities in the functions; ii. protect the application's data (stored in cloud services); and, iii. secure the entire workflows (e.g., ensuring that all functions are executed with the minimum privileges required). The introduced objectives can be achieved by leveraging cloud services and tools that cloud providers make available to software developers. This gives software developers the ability to control and manage access to resources, monitor components, log information, verify network configurations, protect against DDoS attacks, implement firewalls, inspect traffic or secure access control and key management (among others). The cited concepts are

critical ones for the security of the serverless functions and their workflows, and need to be fully seized by software developers, the alternative being the developers ignoring the consideration of security for their applications, or to make unrealistic assumptions about the security measures put in place by cloud providers—in both cases, a dreadful scenario.

4.3 Resistance to Denial-of-Service attacks

Serverless, by construction, enjoys elasticity—it can adapt to workload changes by provisioning and deprovisioning resources—thanks to its efficient and automatic auto-scaling. As such, serverless platforms provide increased resistance against many different types of DoS/DDoS attacks that aim to overwhelm network bandwidth, trigger many compute-heavy actions in parallel, or exploit flaws in the application, for instance to cause infinite loops. While auto-scaling has already been used in previous computing paradigms (e.g., microservices), before serverless the cited technique required the usage of an external service which was complex to use and had to be configured manually by software developers. In serverless, auto-scaling is considerably simpler, more effective and less costly, making it a fundamental feature for any serverless-based application.

5 Serverless as a Security Risk

In this section, we detail several aspects of serverless that can negatively affect security. Table 1 compares the level of security offered by the serverless paradigm against the competing ones for a few interesting dimensions.

5.1 Larger attack surface

Serverless computing exposes a significantly larger attack surface compared to its predecessors for three main reasons:

First, as functions are stateless and are only intended to perform a single task, they are required to constantly interact with other functions and (shared) cloud services. However, the definition and enforcement of security policies—specifying which functions and cloud services can be accessed by each function—in such dynamic and complex environments is very complex and thus prone to errors [35, 36].

Second, functions can be triggered by many external and internal events (e.g., 47 event types in Amazon Lambda, 11 event types in Azure and more than 90 event types in Google) with multiple formats and encoding. To further complicate matters, the trend is that the number of events supported will increase even more to allow other applications to also benefit from serverless offered advantages. What above creates many possible entry-points for adversaries to gain control of functions; even more than when using microservices due to the fact that serverless applications are stateless and event-driven.

Third, serverless platforms include a number of new components and cloud services, many of which are shared across users. Again, the fact that serverless functions are stateless, simple, and event-driven, together with the fact that cloud providers want to provide greater application performance with reduced costs and achieve a much more optimal use of their resources, means that serverless platforms include many more components that are shared between users with respect to previous computing paradigms. Such shared components can enable new forms of side or covert channels that can allow adversaries to leak sensitive data or to violate the specified security policies.

The combination of these factors together opens new doors for adversaries to mount attacks and makes it harder for cloud providers to defend against them.

5.2 Proprietary cloud provider infrastructures

Cloud providers are now the ones responsible for conducting all operational and infrastructure tasks, including those aimed to protect their infrastructures and the hosted applications from internal and external threats. Unfortunately, cloud providers typically keep most information about their infrastructures confidential, making it difficult for security experts to scrutinize the security of serverless platforms. Within the security research community, it is widely known that relying on *security-through-obscurity* alone is a dangerous approach that may conceal insecure designs. Motivated by the above rationales, researchers have devoted significant efforts into reverse-engineering and documenting how the serverless platforms of the main cloud providers were developed in an attempt to understand their core design decisions (e.g., [22]). Yet, there are still many components within serverless platforms that remain unexplored to date and hence whose security level is unknown.

5.3 Security vs. performance vs. cost.

Ideally, cloud providers would like to develop serverless platforms that jointly maximize the security and performance of their infrastructures while maximizing their revenue and keeping the incurred cost as low as possible. However, the cited dimensions are conflicting with each other. Therefore, it is important to find a balance between them. Experience has shown that cloud providers, when it comes to which dimension to curb in order to keep cost under control, do not have security at the top of their priority list of features to

Table 1 Security comparison between monolithic applications, microservices, and serverless—in serverless, we consider security to be a shared responsibility because many of the responsibilities that software developers used to have are now shifted to cloud providers.

	Monoliths	**Microservices**	**Serverless**
Feasibility of long-lasting attacks	High	High	Low
Main responsible for security	Mostly app owners	Mostly app owners	Shared responsibility
Entry points for adversaries	Few	Medium	Many
Resistance to Denial-of-Service attacks	Low	Medium	High
Resistance to Denial-of-Wallet attacks [26]	High	Medium	Low
Communication with other components	None	Medium	High
Visibility of underlying infrastructure	High	High	Low

preserve. Next, we show how the selection and usage of execution environments as well as the chosen function placement strategy can influence the security, performance and cost of serverless platforms and the applications they host.

Execution environments. The selection of the execution environment in which functions are executed is crucial for cloud providers since it strongly impacts the security and performance of their serverless platforms (see Table 2 for more details). For example, containers entail less overhead and provide greater resource utilization than VMs but this also results in weaker isolation guarantees. A possible solution would have been to combine traditional VMs and containers together (e.g., by placing all containers of a user inside a VM). However, this would have prevented reaping the isolation benefits VMs offer and the performance advantages containers provide. The synthesis was provided by cloud providers: they have opted for developing their own execution environments and open-sourcing their code. Without loss of generality we focus on the execution environments proposed by Amazon and Google. However, the conclusions we reach are also applicable to other well-known execution environments like Microsoft's Hyper-V Technology [38], IBMs Nabla Containers [39] and Kata [40].

Amazon designed Firecracker[2], a new execution environment that builds upon the KVM hypervisor to create and manage so called microVMs through a new virtual machine monitor and a new API. Following this trend, Google has developed g-Visor[3], a user-space application kernel that sits between the containerized application and the host OS and hence provides an additional layer of isolation per container. Although Firecracker and g-Visor approaches are promising, neither their attack surface nor their security mechanisms have yet been properly evaluated by security experts. Thus, research should focus on understanding their weaknesses and limitations.

[2] `https://firecracker-microvm.github.io/`

[3] `https://gvisor.dev/`

Cold containers vs. warm containers. Repeatedly booting a function from scratch inside a newly-generated container (i.e., a *cold* container) can be an expensive operation latency-wise. It is worth reminding that most serverless functions are executed only for a very short period of time and hence the container's booting latency would be similar to the function's execution time. Another reason why the use of cold containers is an issue (from the point of view of the cloud provider) is that customers are not billed for the time it takes for their containers to boot.

Warm containers (i.e., containers that are reused to run multiple instances of the same function) reduce the functions' startup times and improve efficiency, e.g., by keeping and reusing local caches or maintaining long-lived connections between invocations. However, the advantages offered by warm containers come at the expense of providing fewer security guarantees. To prevent such attacks, application owners can disable the possibility of reusing the same execution environment to run the same function multiple times. Yet, disabling warm containers is not always be a viable option since this can degrade the application's performance.

Deterministic vs. random scheduling. Let us consider the process adopted by cloud providers to assign functions to worker nodes. From a security point of view, randomized scheduling algorithms are preferred over deterministic ones because they offer stronger protection against attacks that could exploit co-location. However, randomized scheduling algorithms do not consider functional aspects such as worker nodes' resource utilization or the existence of warm containers when choosing the worker nodes that will execute the functions. This leads to a non-optimal allocation of functions that can negatively affect the overall performance of both the applications and the underlying serverless infrastructure. In practice, to prevent the latter issue, cloud providers typically opt for deterministic scheduling algorithms that lead to a more optimal use of the available resources and less latency overhead. Nevertheless, this approach can be vulnerable to attacks by adversaries that can obtain information about (or tamper with) the scheduling algorithms internals. Thus, research is required to first understand

Table 2 Comparison between multiple execution environments.

Features	Traditional VM	Docker Containers	g-Visor (Google)	microVMs (Amazon)
Number of functionalities in host OS kernel [37]	Almost none	Almost all	Less than in containers	More than in VMs
App startup times	Very high	Medium	Medium	High
Isolation guarantees	Medium-high	Low	Medium	High
Complexity	High	Medium-low	Medium-low	Medium
Written in safe prog. languages	No	Yes (Go)	Yes (Golang)	Yes (Rust)

all possible attack vectors within this context, and then to develop scheduling algorithms that are resistant to attacks.

6 Security Attacks and Countermeasures

In this section, we present the main types of attacks against serverless. We group them into two main categories: (i) *application-level attacks* that exploit vulnerabilities in the functions' code; and, (ii) *infrastructure-level attacks* that take advantage of the way the serverless architectures are designed and operated. As application-level attacks have already been covered in a report by OWASP [13], in this section we mainly focus on infrastructure-level attacks and briefly mention the most important security issues at the application level.

6.1 Application-level attacks

In serverless computing, software developers are still responsible for guaranteeing the security of their applications, i.e., the security in the cloud. Hence, if software developers do not adhere to standard secure coding practices and write their functions' code in an insecure manner, their functions could contain vulnerabilities that can make them vulnerable to traditional application-level attacks such as Cross-Site Scripting (XSS), Command/SQL Injection, Denial of Service (DoS), and many more. With serverless computing, the cited attacks (or variants of them) remain possible; the only difference is that sometimes they are carried out in a slightly different way (or with a slightly different goal in mind). OWASP has released a report detailing the serverless attack surface as well as the feasibility and impact of a variety of well-known application-level attacks when launched against serverless applications [13]. Inspired by this report, next we briefly describe the most important application-level security risks and attacks for serverless functions.

Injection. Adversaries can send maliciously-crafted packets to functions in order to exploit weaknesses in the way they parse the input data. Serverless functions can be vulnerable not only to traditional injection attacks (e.g., based on SQL/NoSQL or OS commands), but also to new types of such attacks caused by the fact that there exist many function entry points that can be fully controlled by adversaries. Injection attacks could be launched, for example, to retrieve the functions' source code or secrets stored within the execution environment. To mitigate this concern, each function should always carefully validate and sanitize all received input data before using it (even if the data originates from another function and the said function is considered to be trusted). In principle, validating and sanitizing event data should be no different than validating and sanitizing user data. In practice, however, the former is much more complex due to the large number of events supported and the fact that there are still no widely-available and generic security tools capable of performing this task automatically in order to protect a given application from the described attacks.

Bypass authentication. Serverless functions by themselves lack the necessary information and context to know about other functions and cloud services that are part of the application they belong to. In addition to this, applications typically comprise a plethora of ephemeral functions that can be triggered by many event sources and can make use of a variety of (shared) cloud services. The previous points make it very hard for application owners to apply proper security controls in order to restrict access to their functions at all times. Knowing the difficulty of properly managing security in such complex and dynamic environments, adversaries will try to find ways to trigger functions (or pass malicious data to them)—exploiting both the program logic or resorting to external invocations—while skipping authentication. By doing this, adversaries could exfiltrate private data or tamper with the function's execution flow. A robust access control mechanism is essential in the serverless platforms to determine if a function invocation request is legitimate and has the required permissions to access a function or a piece of data. Currently, cloud providers typically offer access control techniques as part of their cloud service portfolio, with Identity and Access Management (IAM) being the most well-known method which in turn often incorporates traditional role-based access control (RBAC) [41, 42] and attribute-based access control (ABAC) [43]. Moreover, several tools and

services (e.g., [44, 45]) have recently been proposed to ease and automate the creation of credentials and identities that help authenticate the API calls made by users or other workloads.

Privilege misconfiguration. It is widely known that the process of granting permissions to serverless functions is a complex task that often results in functions getting more permissions than the ones they need. There are several reasons why attacks that exploit these weaknesses exist (and will continue to exist at least in the near future). First, software developers often do not have sufficient knowledge to define fine-grained security controls to limit their functions' capabilities. Second, following the tight deadlines to bring their applications to production environments, software developers often do not perform enough testing to verify the set of permissions assigned to their functions. Finally, and most importantly, there is a lack of mechanisms to dynamically and automatically identify and configure the minimum set of permissions needed by applications.

De-serialization and usage of third-party libraries. Serverless functions are written in a number of programming languages, some of which are scripting-based (e.g., Python and NodeJS) that often use serialized data types such as JSON. All these programming languages have their own quirks which can lead to unexpected evaluations of untrusted data. This originates not only from the programming language itself, but from frameworks incorporated into the application—typically, to enable faster code development. Due to the difficulty of protecting against deserialization vulnerabilities, it is strongly recommended to avoid user input deserialization unless absolutely necessary. If the latter is not possible, then software developers must consider and incorporate robust measures that (at least) guarantee that the data has not been tampered with (e.g., through the usage of digital signatures).

In addition, functions often rely on many (potentially insecure) third-party libraries to handle many critical tasks. The problem is that, because of the complexity of the applications, software developers are typically not fully aware of the third-party components used and consequently they do not keep them up-to-date. As a result, functions can contain weaknesses that could allow adversaries to run arbitrary code, leak data, or even worse, gain full control of the functions. To alleviate this concern, software developers should keep good track of the third-party libraries they use, and should apply the necessary measures to ensure that every function builds its own security perimeter. In this regards, it is commendable the initiative related to the SW bill of materials initiative [46].

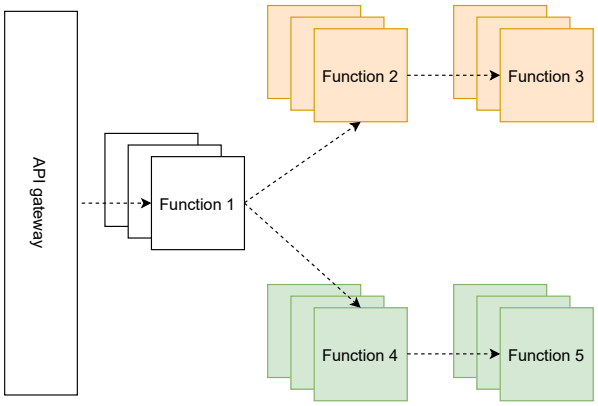

Figure 2 This figure illustrates an application composed by various functions chained together. Let us assume that Function 1 is responsible for authenticating the application's administrators as well as end-users before their requests are passed to other functions. Imagine that the request is passed to Function 2 and Function 3 if the request originates from administrators, while if the request comes from end-users this is handled by Function 4 and Function 5. In such a case, the application's control flow can easily leak the role of the person issuing the request or the times at which the application is accessed by administrators and end-users.

6.2 Infrastructure-level attacks

In the following, we outline possible infrastructure-level attacks within the serverless ecosystem that, to a large extent, remain relatively unexplored. Therefore, we urge the scientific community to investigate them before the full adoption of serverless technology.

Side channel attacks. Adversaries can attempt to exploit the way serverless platforms are designed and implemented in order to conduct new forms of side channel attacks. For example, they could leverage weaknesses in the execution environments where functions are run in order to obtain host-system state information (e.g., power consumption or performance data) or individual process execution information (e.g., process scheduling, cgroups or process running status). This information can help adversaries to uniquely identify a worker node or a function instance, and ultimately to conduct more effective and efficient attacks. Equally, as shown by Figure 2, the sequence of functions traversed in response to external events triggered by users can also reveal information to adversaries (e.g., the role of the person triggering the request). As functions are triggered reactively in response to an action performed by a user, adversaries could gain insights about the users by looking at the functions' metadata (e.g., when or how often functions are called).

More sophisticated side channels can also be devised, based on the fact that there exist many components and cloud services shared across users. In particular,

adversaries are interested in any shared component subject to a change in its state based on the processed data—since these components could leak sensitive data about users and functions through a side channel. Note that side channel attacks in the context of serverless computing have not yet been investigated by the scientific community. Thus, an in-depth evaluation is needed to identify new serverless-specific attacks, then analyze their feasibility, extent and consequences, and finally to propose effective countermeasures in order to defend against them.

Race conditions. Serverless platforms can be vulnerable to attacks caused by inconsistencies in any component whose functionality is distributed across several nodes or that contains multiple replicas. For example, let us assume that software developers decide to modify the code of a given function while several replicas of this function are running. In such a case, there can be a (small) time window where the serverless platform is in an inconsistent state where some incoming requests are handled by an old version of the function and some others by the new version [22]. Such inconsistencies could be caused, for example, by cloud providers reusing execution environments with the old version of the function for a certain period of time. Adversaries could abuse such undesirable behavior to conduct security attacks with the goal of accessing or modifying data that otherwise would no longer be available to them.

Similar attacks could also be carried out when other parameters are modified (e.g., IAM roles, memory sizes, or environment variables) while multiple replicas of the same function are executed. Modifying these parameters at runtime can lead to race conditions that adversaries can exploit to lower the overall security of the serverless platform. While race conditions can also happen in a microservices architecture, the smaller granularity offered by serverless platforms increases the risk of inconsistencies across function versions. Overall, we believe that this research area deserves more attention from the scientific community, both to understand the security threats and to design effective countermeasures against them.

Long-lasting attacks. As explained in Section 4, traditional long-lasting attacks that target servers are not applicable in the context of serverless computing. However, researchers have reported that it is possible for adversaries to execute a new class of long-lasting attacks by placing malicious code in the (writable) /tmp/ disk space used by warm containers to store temporary information across invocations [47, 48]. The main challenge to perform such attacks is that, as /tmp/ is intended to be used only for maintaining temporary state, their size is relatively small (e.g., 512MB in Amazon Lambda). This poses some restrictions on the type and size of the code adversaries can place inside them. One way for adversaries to overcome this limitation would be to run code that communicates with external endpoints controlled by them. However, most serverless platforms give application owners access to security tools that could preclude such disallowed external communication. Despite this, there is still the need for investigating which attacks could be run from the /tmp/ disk space or any other directory within a given execution environment that is kept intact across multiple invocations of the same function. These attacks are likely to become a more important threat in the near future as serverless platforms evolve to fit the needs of stateful functions (e.g., [49]), since this will require placing more storage closer to the functions. As for countermeasures, in case the usage of warm containers is required to meet the application performance requirements, one possible way to mitigate the exposed issue would be for cloud providers to reduce the size of the /tmp/ folder to the minimum extent possible and to carefully monitor its contents after every function invocation. Here the challenge is how to distinguish between the legitimate data stored in the /tmp/ directory (the ones that come from the application) and the malicious code that adversaries could store therein.

Billing attacks. Though serverless offers increased protection against traditional DoS/DDoS attacks, these attacks can be engineered to lead to new, serverless-specific attacks that take advantage of the fact that application owners are billed based on the amount of resources their functions consume. By sending many requests to functions, adversaries can now perform the so called Denial-of-Wallet (DoW) attacks [26] with the purpose of significantly increasing the cost for application owners. Although some mitigating countermeasures already exist against DoW attacks (e.g., setting an upper limit on invocations concurrency and instances quota on function creation or creating a billing alert to notify application owners if they exceed a predefined spending limit), these attacks are not easy to defend against and require additional control measures: first, to detect abnormal behavior; and, later to discriminate which legitimate invocations to allow, and which ones to drop.

The uniqueness of serverless in this context, is the fact that invocation and billing happen at a very small granularity, i.e., the function. Hence, an adversary can perform these attacks by invoking a function many times, while in other auto-scaling constructions adversaries would require the generation of a high load on

a full container or VM to succeed. As such, the consequences of successfully launching such attacks can be more severe when targeting serverless platforms. Moreover, given the fact that computation can evolve only via function calls, blocking legitimate function invocations would represent a more serious threat than that experienced by the cited auto-scaling twins of serverless.

7 Conclusions

In this paper we have shown that, on the one hand, serverless computing provides additional security features while, on the other hand, it also introduces unique security threats and challenges—clearly differentiating itself from current virtualization technologies. In particular, we have reviewed current serverless architectures, categorized the current security threats, shown actionable hints to improve the current security posture, and highlighted security research directions to make serverless the paradigm of choice when looking for virtualization solutions where security is at a premium. We believe that our contribution, other than being valuable on its own, also paves the way for further research in this domain, a challenging and relevant one for practitioners, Industry, and Academia.

Author details

[1]Telefonica Research, Barcelona, Spain. [2]Hamad Bin Khalifa University (HBKU), College of Science and Engineering (CSE), Information and Computing Technology (ICT), Doha, Qatar.

References

1. Armbrust, M., Fox, A., Griffith, R., Joseph, A.D., Katz, R., Konwinski, A., Lee, G., Patterson, D., Rabkin, A., Stoica, I., Zaharia, M.: Above the clouds: A berkeley view of cloud computing. Technical report, University of California at Berkeley (February 2009). http://berkeleyclouds.blogspot.com/2009/02/above-clouds-released.html
2. Lombardi, F., Di Pietro, R.: Security for Cloud Computing. Artech House, Norwood, Massachusetts (2015)
3. Castro, P., Ishakian, V., Muthusamy, V., Slominski, A.: The Rise of Serverless Computing. Communications ACM **62**(12), 44–54 (2019)
4. Jonas, E., Schleier-Smith, J., Sreekanti, V., Tsai, C., Khandelwal, A., Pu, Q., Shankar, V., Carreira, J., Krauth, K., Yadwadkar, N.J., Gonzalez, J.E., Popa, R.A., Stoica, I., Patterson, D.A.: Cloud Programming Simplified: A Berkeley View on Serverless Computing. CoRR (2019). 1902.03383
5. AWS Lambda. https://aws.amazon.com/lambda/ (2021)
6. Azure Serverless | Microsoft Azure. https://azure.microsoft.com/solutions/serverless/ (2021)
7. Serverless Computing Solutions—Google Cloud. https://cloud.google.com/serverless (2021)
8. IBM Cloud Functions. https://www.ibm.com/cloud/functions (2021)
9. Alibaba Cloud Function Compute. https://www.alibabacloud.com/products/function-compute (2021)
10. AWS Lambda Customer Case Studies. https://aws.amazon.com/lambda/resources/customer-case-studies/ (2021)
11. Serverless Computing Market Insights 2022. https://www.digitaljournal.com/pr/serverless-computing-market-insights-2022-business-opportunities-current-trends-and-restraints-forecast-2026#ixzz7W67yDNi4 (2021)
12. Hong, S., Srivastava, A., Shambrook, W., Dumitras, T.: Go Serverless: Securing Cloud via Serverless Design Patterns. In: Proceedings of USENIX Workshop on Hot Topics in Cloud Computing (HotCloud) (2018)
13. OWASP Serverless Top 10. https://owasp.org/www-project-serverless-top-10/ (2021)
14. AWS Serverless Application Repository. https://aws.amazon.com/en/serverless/serverlessrepo/ (2021)
15. Combe, T., Martin, A., Di Pietro, R.: To Docker or Not to Docker: A Security Perspective. IEEE Cloud Computing **3**(5), 54–62 (2016)
16. Gao, X., Gu, Z., Li, Z., Jamjoom, H., Wang, C.: Houdini's Escape: Breaking the Resource Rein of Linux Control Groups. In: Proceedings of ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 1073–1086 (2019)
17. Nam, J., Lee, S., Seo, H., Porras, P., Yegneswaran, V., Shin, S.: BASTION: A Security Enforcement Network Stack for Container Networks. In: USENIX Annual Technical Conference (USENIX ATC), pp. 81–95 (2020)
18. Ory Segal: Serverless Security // Serverless Days TLV. https://www.youtube.com/watch?v=M7wUanfWs1c&t=743s (2021)
19. Event Injection: Protecting your Serverless Applications. https://www.jeremydaly.com/event-injection-protecting-your-serverless-applications/ (2021)
20. PureSec. Hacking a Serverless Application: Demo. https://www.youtube.com/watch?v=TcN7wHuroVw (2019)
21. Yelam, A., Subbareddy, S., Ganesan, K., Savage, S., Mirian, A.: CoResident Evil: Covert Communication In The Cloud With Lambdas. In: Proceedings of the Web Conference (WWW), pp. 1005–1016 (2021)
22. Wang, L., Li, M., Zhang, Y., Ristenpart, T., Swift, M.: Peeking behind the Curtains of Serverless Platforms. In: Proceedings of USENIX Conference on Usenix Annual Technical Conference (USENIX ATC), pp. 133–145 (2018)
23. CVE-2022-0185: Kubernetes Container Escape Using Linux Kernel Exploit. https://www.crowdstrike.com/blog/cve-2022-0185-kubernetes-container-escape-using-linux-kernel-exploit/ (2022)
24. Hacking serverless runtimes: Pro- filing AWS Lambda, Azure Functions, And more. https://www.blackhat.com/us-17/briefings/schedule/#hacking-serverless-runtimes-profiling-aws-lambda-azure-functions-and-more-6434 (2019)
25. Xiong, J., Wei, M., Lu, Z., Liu, Y.: Warmonger: Inflicting Denial-of-Service via Serverless Functions in the Cloud. In: Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS), pp. 955–969 (2021). https://doi.org/10.1145/3460120.3485372
26. Kelly, D., Glavin, F.G., Barrett, E.: Denial of wallet–Defining a looming threat to serverless computing. Journal of Information Security and Applications, 2214–2126 (2021)
27. Many-faced threats to Serverless security. https://hackernoon.com/many-faced-threats-to-serverless-security-519e94d19dba (2021)
28. New Attack Vector - Serverless Crypto Mining. https://www.puresec.io/blog/new-attack-vector-serverless-crypto-mining (2019)
29. Exploring the Unchartered Space of Container Registry Typosquatting. In: USENIX Security Symposium (USENIX Security) (2022)
30. Makrani, H.M., Sayadi, H., Nazari, N., Khasawneh, K.N., Sasan, A., Rafatirad, S., Homayoun, H.: Cloak & Co-locate: Adversarial Railroading of Resource Sharing-based Attacks on the Cloud. In: 2021 International Symposium on Secure and Private Execution Environment Design (SEED), pp. 1–13 (2021)
31. Fang, C., Wang, H., Nazari, N., Omidi, B., Sasan, A., Khasawneh, K.N., Rafatirad, S., Homayoun, H.: Repttack: Exploiting Cloud Schedulers to Guide Co-Location Attacks. In: Proceedings of Network and Distributed System Security Symposium (NDSS) (2022)
32. Razavi, K., Gras, B., Bosman, E., Preneel, B., Giuffrida, C., Bos, H.: Flip Feng Shui: Hammering a Needle in the Software Stack. In: USENIX Security Symposium (USENIX Security), pp. 1–18 (2016)
33. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre Attacks: Exploiting Speculative Execution. In: IEEE Symposium on Security and Privacy (S&P), pp. 1–19 (2019)

34. Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Fogh, A., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., Hamburg, M.: Meltdown: Reading Kernel Memory from User Space. In: USENIX Security Symposium (USENIX Security), pp. 973–990 (2018)
35. Datta, P., Kumar, P., Morris, T., Grace, M., Rahmati, A., Bates, A.: Valve: Securing Function Workflows on Serverless Computing Platforms. In: Proceedings of The Web Conference (WWW), pp. 939–950 (2020)
36. Sankaran, A., Datta, P., Bates, A.: Workflow Integration Alleviates Identity and Access Management in Serverless Computing. In: Annual Computer Security Applications Conference (ACSAC), pp. 496–509 (2020)
37. Anjali, Caraza-Harter, T., Swift, M.M.: Blending Containers and Virtual Machines: A Study of Firecracker and GVisor. In: Proceedings of ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE), pp. 101–113 (2020)
38. Hyper-V Technology Overview. `https://docs.microsoft.com/en-us/windows-server/virtualization/hyper-v/hyper-v-technology-overview` (2021)
39. Nabla containers: a new approach to container isolation. `https://nabla-containers.github.io/` (2021)
40. Kata containers. `https://katacontainers.io/` (2021)
41. Ferraiolo, D.F., Kuhn, D.R.: Role-Based Access Controls (2009). doi:10.48550/ARXIV.0903.2171
42. Colantonio, A., Di Pietro, R., Ocello, A.: Role Mining in Business: Taming Role-Based Access Control Administration. World Scientific, Singapore (2012)
43. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F., Voas, J.: Attribute-Based Access Control. Computer **48**(2), 85–88 (2015). doi:10.1109/MC.2015.33
44. Spiffe: Secure Production Identity Framework for Everyone. `https://spiffe.io/` (2021)
45. Corsha: API Identity & Access Management. `https://corsha.com/` (2021)
46. The Minimum Elements For a Software Bill of Materials (SBOM). `https://www.ntia.doc.gov/report/2021/minimum-elements-software-bill-materials-sbom` (2021)
47. Gone in 60 Milliseconds: Intrusion and Exfiltration in Serverless Architectures. `https://media.ccc.de/v/33c3-7865-gone_in_60_milliseconds` (2021)
48. How AWS Lambda reuses containers (and how it affects you). `https://pfisterer.dev/posts/aws-lambda-container-reuse` (2021)
49. Savi, M., Banfi, A., Tundo, A., Ciavotta, M.: Serverless Computing for NFV: Is it Worth it? A Performance Comparison Analysis. In: IEEE International Conference on Pervasive Computing and Communications Workshops and Other Affiliated Events (PerCom Workshops), pp. 680–685 (2022)