Top 6 API Security Needs for Serverless Apps

Dealing with Shadow APIs

Co-authors: Alban Diquet and Doug Dooley

Serverless computing is gaining momentum among developers. Mid-level software engineers are building applications that deliver at the scale and business value that used to require a few senior architects to help design. These are exciting times for software developers who might not fully appreciate the complexity of capacity planning, infrastructure design, orchestrated automation, and compliance auditing. In a shared security model commonly found in the public cloud, the benefits of making underlying compute infrastructure ephemeral are a welcome change to most DevOps teams. However, to believe there are fewer security problems with serverless applications would be an unwise assumption. Security problems still exist with serverless, but our focus needs to shift particularly to the world of application programming interfaces (APIs) where sensitive data is prominently transferred in these modern application designs. We will provide some basics around serverless applications and some of their top API security needs.

To understand what's so attractive about serverless, let's consider how regular cloud computing works today. It's basically the outsourcing of the operations of a data center to someone else. Whether developers choose Amazon Web Services (AWS), Google Cloud Platform (GCP), Microsoft Azure or some other platform, they still have to plan for the necessary resource usage; i.e., how many servers, how much storage, where to locate instances, and so on. There's still a lot of complexity, particularly with the orchestration of infrastructure as a service. The onus is on the DevOps teams to know how their application consumes resources (compute, memory, storage, network, etc.), as usage may increase or even spike beyond their expectations.

With serverless computing, all of those orchestration and application capacity controls are no longer the responsibility of the DevOps team. Serverless is a technology popularized by AWS with its introduction of Lambda, a function as a service (FaaS) platform. However, Amazon is not alone, with both Google Cloud Functions and Microsoft Azure Functions gaining momentum among their respective customers. Any of these popular FaaS platforms allow a developer to build and deploy an application without knowing in advance all of the necessary infrastructure controls. The developer gains a great deal of flexibility for scaling and controlling the cost of their run-time environment. And perhaps best of all, serverless customers pay for only what is used, typically on a 100-millisecond basis, and never pay for idle-time when the application is not in use.

This is incredibly empowering and democratizing for developers. Now practically anyone can develop code and put it out on the Internet to run without having to go through the gauntlet of checks by a DevOps team, a security team, or other Enterprise IT groups that traditionally had an ability to control the launch of new business applications. With serverless, a developer just deploys the code he or she wants to run, and in mere seconds it is out there on the Internet without the usual barriers and overhead.

Security professionals, however, look at this new world and have some concerns. It means that nearly all levels of developers can deploy code and potentially create exposure for their organization. With serverless computing in its infancy and few real security tools for this technology available today, creating business risk with an application is more than a possibility— in our API-driven world, it's an outright probability. Unintended attack vectors can easily pop up in this application deployment model, and instant deployment of an application can mean instant exposure to potential risks such as unintended data breaches and unauthorized data exfiltration.

Data Leaks Through Shadow APIs

The most likely place where data is going to leak out in a serverless application is through its APIs. In today's programming environment, everything is being shared through APIs. People build microservices that can talk to each other and share information through easy-to-use APIs. The moment those APIs are exposed on the Internet, they create attack vectors for hackers to extract data.

When new Lambda-based microservices get created for others to use, programs will generally interact with these functions through Amazon's API gateway. However, there are programming frameworks that create APIs that completely bypass any gateway. These types of APIs are called ***Shadow APIs*** because they operate for long periods of time undetected before any security teams can discover and inspect their usage. These APIs are generally leveraged by modern software development kits (SDKs) and mobile applications. If an API is not well designed, all the data that passes through it is at risk. Most of these Shadow APIs were created as a utility but some have caused financial and brand damage to companies when exploited by attackers. The concern is that an application or microservice that is pushed quickly without the normal security assurances could conceivably have vulnerabilities. Then anyone who can see that API can take advantage of that serverless function to extract data in ways never intended by the application developer. Further, this API connected to a serverless application can be vulnerable to a Denial of Wallet (DoW) attack, similar to a Denial of Service (DoS) attack except underlying resources automatically scale-up to extremely high prices, which eventually hurts a business's bottom-line.

There are two factors that really exacerbate these types of problems in modern applications. One is agile development and the other is the automated nature of scaling serverless resources. Agile affects the rate of change and serverless automation affects the scale of exposure—both of which have the potential to magnify the effect of a simple mistake.

In this new age of function as a service, this scenario of unintended consequences is going to happen a lot at the API level and serverless hides these issues from the DevOps and security teams. And if developers aren't even aware there is a problem, and there's no automated service to constantly monitor for these kinds of changes to see if there are violations in security and compliance policies, the vulnerabilities might not be discovered until they are exploited by attackers.

API Gateways Are Not a Security Panacea

So, now we know that serverless computing has a security problem. What can be done about it? Legacy security tools and services, as well as those created for virtual containers, won't work in this new environment. Gartner advocates using an API gateway as a best-practice. While that seems like sound advice, the existing gateways from the serverless vendors have their limitations. The highest rated API gateway on the Gartner Magic Quadrant is Apigee, which was acquired by Google. Unfortunately, the majority of Google customers eschew Apigee because an excessive number of features make the tool too complicated. In terms of market share, Amazon's API Gateway grows in popularity, but it lacks many basic enterprise security functions, making it less useful as a security tool.

But the biggest problems with any API gateway is that it's almost impossible to force developers to use them and they don't work well across cloud platforms. A developer using Lambda to build a serverless application is not likely to choose an API gateway running on GCP, and vice versa. Developers often use multiple clouds for testing and production, making a universal gateway a nice-to-have but not required to get a new application functional and running. The fact is, new third-party security services need to be created with serverless applications in mind.

Checklist: Top 6 API Security Needs for Serverless Apps

In the meantime, when trying to secure your serverless applications, we recommend the following checklist and addressing these questions to start:

1. Enumerate your serverless apps and entry points - Do you have a way to identify all of the new serverless apps being created in your Amazon, Azure, and Google cloud environments? Is this a continuous process or do you have staff doing manual steps for discovery? Are there tools/scripts you can use to help automate? Can you figure out the entry/input points that trigger underlying services, often via API calls?

2. API definition and specification - Do you have an API definition of what your serverless apps can do? Does it meet a standard specification such as Swagger or OpenAPI v3? If not, can you create an API definition through an audit? Does your serverless app connect to an API gateway to help outline what it can do?

3. Authentication and encryption - How does your serverless application verify identity? OAuth has become a de-facto standard in many organizations but making that work seamlessly with all of the API interfaces can be prone to misconfiguration. Also, is SSL/TLS encryption enabled? What versions of TLS are employed? How are encryption keys and app secrets handled and stored? Again, these basic security controls tend to be common areas where attackers take advantage of poor AppSec hygiene.

4. Consumers, receivers, and clients - Can you list all of the client-side API consumers/receivers for your serverless applications, e.g. mobile apps, modern web front-ends, chatbots, IoT backends? Are there consumers you don't expect to be accessing your serverless apps through their APIs? Are there tools you can use to help identify these various consumers of your serverless apps?

5. Data sources - Is source code access part of your CI/CD environment? Can you ensure it falls within your automated SDLC audit and security assessment? Can you identify the types of data sources your serverless apps have access to? The source code is often a starting point to map out what data sources the serverless app is attempting to connect with.

6. Error handling - Do you have an automated way to parse and analyze the logs coming from your serverless applications? Many security and compliance problems can be identified through critical alerts and error messages stored in logs from these serverless apps.

The growth in serverless apps is going to accelerate because the cost and time benefits for developers are overwhelmingly positive. However, these six areas highlight the need for a new approach to security with an architecture that is substantially different than previous techniques and tools used to secure non-modern applications. Many of you will struggle to answer all of these questions above for your serverless app environment.

The good news is that there are new security tools for serverless applications on the horizon. The more advanced API-centric security tools will have the ability to discover new serverless apps and conduct security assessments through their APIs, source code, and logs. Security teams that embrace the need for DevOps-style automation will keep pace with these new modern applications. As an example, organizations that adopted security automation for mobile applications more than five years ago continue to reap the rewards of faster, better application security today. Once the major security challenges of serverless are under control – and they will be, in time – there will be no stopping this new direction in cloud computing.