Micro-size it

8 best practices for microservices app sec

Marco Troisi, Senior Software Engineer, ServisBOT

There is virtually no situation in software architecture that entirely frees you from security conside microservices, some issues become more distinct and a lot harder. However, there are also a few fea that can bolster security.

With microservices, the network is still a bottleneck. Things like access control, which the industry thoroughly within the realm of monolithic applications, assumes a new, almost unexpected, level c complexity. This paves the way for debates and scrutiny over whether a microservices architecture problems than it creates. Your decision to use microservices should always be conditional.

When you've done your due diligence and decided that microservices are right for you, it's time to make sure that all of your applications' security demands are met. Here are eight best practices for securing your microservices.

[Get Report: Buyer's Guide to Software Test Automation Tools]

1. Use OAuth for user identity and access control

While OAuth2 isn't perfect, it's a widely adopted standard. The advantage of using it is that you car platforms that will greatly accelerate your development phase. By the same token, several solution security level of your OAuth-based authorization service have already been built by some of the big smartest engineers around.

[Webinar: How to Fit Security Into Your Software Lifecycle With Automation and Integration]

2. Use 'defence in depth' to prioritize key services

Assuming that a firewall on your network perimeter is enough to protect your software is a big mis depth" is defined as "an information assurance concept in which multiple layers of security control throughout an information technology system."

In plain English, what you need to do is identify what your most sensitive services are, and apply a layers of security to them, so that a potential attacker who is able to exploit one of your security la figure out a way to beat all your other defenses on your critical services. This is by all accounts easier said than done, but several resources are available.

Security is typically a job better left to experts and not to amateurs. A proper defense in depth strategy is more likely to succeed if it's established by people who actually know what they're doing.

What's great about microservices is that they make it easier to adopt this strategy in a very granular and strategic way—by focusing your security efforts and resources on specific microservices. The architecture also makes it easier for you to diversify the layers of security you wish to adopt on each microservice. By so doing, an attacker who is able to exploit one of your services may not necessarily be able to figure out how to exploit the second one.

3. Don’t write your own crypto code

Over the years, many people have invested incredible amounts of money, time, and resources into building libraries that handle encryption and decryption. If you hired 10 smart and competent security people, put them all in a room and asked them to come up with the best possible library for encryption and decryption, I doubt they would come up with something as good as the best open source crypto libraries that are already out there.

Most of the time, when it comes to security you shouldn't try to roll your own new solutions and algorithms unless you've got strong and specific reasons to, and you've got people skilled enough to create something nearly as good as the open source tools already available (tools that have been heavily battle tested by the community).

In most cases, you should use NaCl/libsodium for encryption. It's been around for several and it's fast, easy to use, and secure. While the original implementation of NaCl is written in C, it also supports C++ and Python. And thanks to the libsodium fork, several adapters for other languages like PHP, Javascript, and Go are available.

This section wouldn't be complete without mentioning the wildly popular Bouncy Castle library. If you're working with Java or C#, your best bet is to go with this one. If you want to learn more about encryption, read this developer's guide.

High testing coverage here is more essential than ever. Every time a part of your system is updated, you want to make sure you catch any issue early enough and in as much detail as possible.

Make sure that your platform is mostly "atomic". What that means is that everything should be wrapped within containers so that testing your application with an updated library or language version is just a matter of wrapping a different container around it. Should the operation fail, reversing everything is fairly easy and, most importantly, can be automated.

CoreOS, RedHat's Atomic Linux, and Ubuntu's Snappy Core are also projects you want to keep an eye on, as they try to bring about the same concept on an OS level.

5. Use a distributed firewall with centralized control

For the most part, this is still uncharted territory, but I believe that a firewall that allows users more granular control over each and every microservice (as attempted by Project Calico) has got to be the way we build firewalls for microservices. If not now, at least at some point in the future.

6. Get your containers out of the public network

Amazon, with their AWS API gateway, probably made this whole notion more mainstream and easy to adopt than anyone else before.

An API gateway establishes a single entry point for all requests coming from all clients. It subsequently knows how to provide an interface for all of your microservices.

By using this technique you can secure all of your microservices behind a firewall, allowing the API gateway to handle external requests and then talk to the microservices behind the firewall.

Moreover, as the Netflix experience teaches us, using an API gateway is a great way to optimize requests based on the client, especially in the case of mobile devices.

7. Use security scanners for your containers

Within your automated testing suite, it would make sense to include periodic vulnerability and security scanning for your containers. The chief open source actor in this space appears to be Clair, from CoreOS. Docker Security Scanning and Twistlock are a couple of commercial options.

Something else to keep in mind here is that the container image itself may not necessarily be trusted unless its signature has been verified. rkt does that by default, while Docker introduced a similar feature a while ago after several security vulnerabilities were found.

8. Monitor everything with a tool

Built originally by engineers at SoundCloud, Prometheus is an open source monitoring platform and a part of the Cloud Native Computing Foundation. It's being supported and adopted by some of the biggest players in the industry, like SoundCloud themselves, CoreOS, and Digital Ocean.

Other monitoring solutions include InfluxDB, statsd and several well-known commercial platforms.

Don't reinvent the wheel

While the above is not intended to be an exhaustive list, it touches on the issues you are most likely to face when building applications based on a microservices architecture.

When it comes to security, reinventing the wheel is rarely a good idea. Always be researching the best practices adopted by the industry and suggested by experts.

Here are some additional resources I've bookmarked:

- Project Cockpit - administer your Linux servers via a web browser
- Securing Micro-services with a Distributed Firewall (video)
- Docker and High Security Microservices
- Defense in depth conference
- Network Perimeter Security: Building Defense In-Depth (book)
- Docker Containers: Build and Deploy with Kubernetes, Flannel, Cockpit, and Atomic (Negus Live Linux) (book)
- Container Defense in Depth
- API gateway pattern

Share your best practices or resources on which you rely for securing microservices in the comments below.

Topics: App Dev & Testing, App Dev

More from TechBeacon

Forget monoliths vs. microservices. Cognitive load is what matters.

Tool up your API integration testing with OpenAPI

3 steps to improve your test-dev relationship