Micro-size it!

Microservices apps: Do identity, access management without the overhead

Bernard Golden, CEO, Navica

Microservices-based applications stand up better to an external attack than do traditional, monolithic applications. But how do you ensure that internal, intra-application communications are secure? That is, how do you configure identity and access management (IAM) in a distributed application?

This is a trickier question than you might think. With monolithic applications, securing intra-application communication is not a concern because you maintain all intra-process communication within a single executable, and access to all functionality comes from within the same process. However, if someone manages to break into the monolith, that person has access to everything in your executable.

Microservices-based applications consist of distributed components that communicate across a network, so you must ensure that only appropriate traffic comes to each service component, even though those components do not control what sits at the other end of the network connection.

[Get Guide: Best Practices for GDPR and CCPA Compliance]

3 options for identity and access management

The most common IAM technique for external-facing applications, whether on premises or in the cloud, is to combine an initial, heavyweight authorization process with a subsequent, lighter-weight one. On the initial login, users typically must provide a user ID and password. Then, on subsequent accesses, a cookie-based, randomized token gets passed back and forth. The token is accepted as proof of identity, and the application allows further interaction.

This approach means that the initial authentication only occurs once, with the overhead of processing the user ID/password information. Subsequent interactions require only that the submitter pass along the token and that the recipient confirm that the token is already known. This improves overall performance, ensuring that the user experience is not impaired by repeated, lengthy delays as the IAM process executes. The token usually has a timeout set, so that this lighter-weight authentication is only available for a limited time.

So what is the right way to accomplish this same IAM function in a microservices-based application? Your application must ensure appropriate levels of communication security without imposing a performance penalty. For IAM in a monolithic application, there's just one communication: From the user to the application interface. Everything else happens within the application executable. With microservices, you have multiple network communication events as an end-user event cascades through the collection of services associated with a specific branch of execution.

There are three approaches to microservices IAM you can take, and each carries with it benefits and drawbacks.

1. Provide traditional authentication on communications

The user's initial interaction with a microservices-based application should use the same IAM process as does a monolithic application. It should offer a user ID/password, with the externally facing application component performing authentication and authorization. Subsequent communication between the user and this component is then established using the cookie-and-token approach I outlined above.

But what happens as the request cascades through the downstream services? You could have each calling service go through a similar ID/password validation process, but this repeatedly imposes a heavyweight ID/password lookup process. An approach that requires this lookup process for every communication between a caller/called service pair would be even worse. Imagine the aggregate authentication overhead this would inflict on the application, and the corresponding performance hit your application would suffer.

So this approach is not acceptable within the constraints of needing adequate application performance to support user satisfaction.

2. Assume communication security already exists

There's no alternative to using external IAM in the initial heavyweight/subsequent lighter-weight cookie/token approach described above. But once communication enters the internal services that make up the application, you can assume full authentication based on acceptance of the caller/called service pair. In other words, internal communication within the application means that every call to an internal application service will originate from another service.

In your application security design, you might assume that any access by one application service to another is appropriate and that for each internal service, full authentication of the end user has already been performed. Any intra-application processing could then be completed on behalf of an authenticated user without further authentication.

On the positive side, this approach has the benefit of simplicity. Once you have performed the initial authentication, all other application components are then free of IAM overhead. This is bound to improve application performance.

The downside is that, this feels a bit ... squishy? Insecure? Certainly, allowing unfettered network c components flies in the face of long-standing best practices that require some level of authenticati in applications. So that leaves one last option.

[Webinar: Data protection: Your biggest reputational risk?]

3. Use the cookie/token communication mechanism

If you want to avoid relying on heavyweight authentication and authorization processes in your IA don't feel comfortable letting internal microservices components communicate without IAM, consider leveraging the cookie/token approach. The token must be stored inside your application, so you can verify and acc submissions of the cookie by the user's browser. But instead of storing the token inside the user-fa executable, where it is accessible only to that executable, you place the token in a cache residing in cluster.

In this scenario, every communication between service components carries a token, in addition to payload information. The receiving service takes the token, validates it, and, if the token is approp function using the payload provided by the calling service. The return payload also includes the tol service can validate the appropriateness of the returned information.

This approach does require a comparison of the passed token with what resides in the token cache, but that shouldn't take more than a few milliseconds. To be fair, this is a small, but not insignificant, amount of time impose a small performance hit. However, in the interest of overall application security, the overhe should be acceptable.

Performance versus security: Getting to good enough

The challenge of providing IAM for internal microservices reflects the truism in software engineeri improvement brings its own complexity. While the benefits of using microservices are undeniable, fore as well. In the world of monolithic applications, intra-application IAM was unnecessary.

In the microservices world, IAM needs to be tackled. While you can choose a secure, but heavyweight, formal IAM process for every service-to-service communication, the overhead of this approach will probably make your application too slow for your users.

On the other hand, running without any intra-application security maintains the highest possible performance, but you're relying on the assumption that there is no need to check security because heavyweight IAM has already been performed at the user level.

In the end, this topic will no doubt be subject to extensive evaluation in real-world environments, but best practices are converging toward a lightweight, but still present, token-based IAM. This is sufficient for security purposes, and acceptable for application performance purposes.

Are you considering an alternative approach? If you've had this debate in your organization, I'd lik decided to proceed.

Image credit: Flickr

[Webinar: Five steps to implement a universal policy strategy (July 18)]

Topics: Security, Identity & Access Management

More from TechBeacon

How to stop email attacks from compromising your business

Son of Rowhammer: None of us are safe from RAMBleed

How to track security problems in your Kubernetes deployments

Article Tags

Identity and Access Management (IAM)

Microservices App Dev

Brought to you by

Topics

App Dev & Testing
DevOps
Enterprise IT
Security

TechBeacon

About
Our Contributors
Terms of Use
Privacy