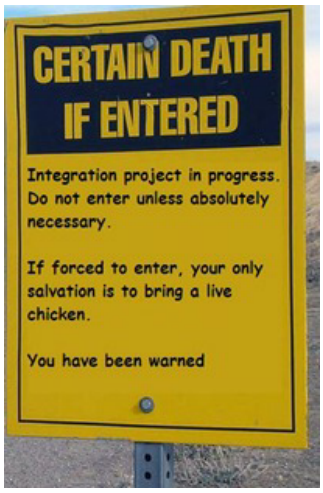
The Impact of Security on Infrastructure Integration

Lori MacVittie, 2010-27-10

Automation implies integration. Integration implies access. Access requires authentication and authorization. That's where things start to get interesting…

Discussions typically associated with application integration – particularly when integrating applications that are deployed off-premise – are going to happen in the infrastructure realm. It's just a matter of time. That's because many of the same challenges the world of enterprise application integration (EAI) has already suffered through (and is still suffering, right now, please send them a sympathy card) will rear up and meet the world of enterprise infrastructure integration head on (we'll send you a sympathy card, as well)



I'm not trying to be fatalistic but rather realistic and, perhaps this one time, to get ahead of the curve. Automation and the complex system of scripts and daemons and event-driven architectures required to achieve the automated data center of tomorrow are necessarily going to raise some alarm bells with someone in the organization; if not now then later. And trust me, trying to insert an authentication and authorization system into an established system is no walk in the park.

If you don't recall why this integration is crucial to a fully dynamic (automated) data center, check out The New Network and then come back. Go ahead, I'll wait. See what I mean? With all the instruction and sharing going on, you definitely want to have some kind of security in place. At least that's what folks seem to bring up. "Sounds good, but what about security? Who is authorizing all these automatic changes to my router/switch/load balancer?"

THE CHALLENGE

The challenge with implementing such a system – whether it's integrated as part of the component itself or provided by an external solution – is maintaining performance. In the past we haven't really been all that concerned with the speed with which configuration changes in the network and application delivery network infrastructure occurred because such modification occurred during maintenance windows with known downtime. But today, in on-demand and real-time environments, we expect such events to occur as fast as possible (and that's when we aren't frustrated the system didn't read our minds and perform the actions on our behalf in the first place).

Consider the performance impact and potential fragility of a process comprising a chain of components, each needing a specific configuration modification. Each component must authenticate and then authorize whatever or whoever is attempting to make the change before actually executing the change. In a multi-tenant infrastructure or a very large enterprise architecture this almost necessarily implies integration of all components with a centrally managed identity management system. That means each component must:

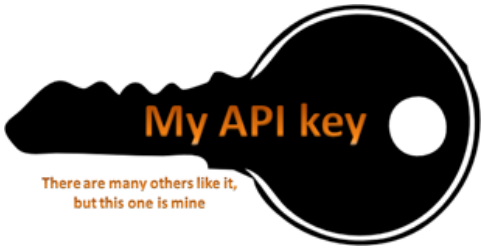
1. Receive a request
2. Extract the credentials
3. Authenticate credentials
4. Authorize access/execution
5. Perform/execute the requested action
6. Write it to a log (auditing, people, AUDITING)
7. Respond to the request with status

The interdependencies between data center components grows exponentially as every component must be integrated with some central identity management system as well as each other and the management console (or script) from which such actions initiated. That's all in addition to doing what it was intended to do in the data center, which is some networking or application delivery networking task. Each integration necessarily introduces (a) a point of failure and (b) process execution latency. That means performance will be impacted, even if only slightly. Chain enough of these integrations in a row and real-time becomes near-time perhaps becomes some-time. And failure on any single component can cascade through the system, causing disruption at best and outages at worst.

But consider the impact of not ensuring that requests are coming from an authorized source. Yeah, potential chaos. No way to really track who is doing what. It's a compliance and infosec nightmare, to say the least. We're at an impasse of sorts, at the moment. We need the automation and integration to move forward and onward but the security risks may be too high for many organizations to accept.

API KEYS MAY HOLD the KEY

Most Web 2.0 applications and cloud computing management frameworks leverage an API key to authorize a specific action. Given the fact that Infrastructure 2.0 is largely driven by a need to automate via open, standards-based APIs, it seems logical that rather than continue to use the old username-password or SSL client certificate methods of the past that infrastructure vendors would move toward API key usage as well.



Consider the benefits, especially when attempting to normalize usage of infrastructure with more traditional components. While it's certainly true that cloud computing providers, who build out frameworks of their own to manage and meter and ultimately bill customers for usage, they still need a way to interface with the infrastructure providing services in such a way as to make it possible to meter and bill out that usage, as well. Wouldn't it be fantastic if infrastructure supported the same methods of authentication as the cloud computing and dynamic data center environments they enable?

But Lori, you're thinking, the use of API keys to authenticate requests doesn't really address any of the challenges.

Au contraire mon frère, but it does!

Consider that instead of needing to authenticate a user by extracting a username and password and validating them against an identity store that you can simply verify the API key is valid (along with some secret verification code, like the security code on your credit card) and away you go. You don't need to verify the caller, just that the call itself is valid and legitimate based on the veracity of the API key and security code, much in the way that credit cards are validated today.

While this doesn't eliminate the need to verify credentials per se, it does do three things:

> 1. **Decreases the time necessary to extract and validate.** If we assume that the API key and associated security code are passed along in, say, the HTTP headers, extraction should be fast and simple for just about every network component in the data center (I am assuming SSL/TLS encrypted transport layer here to keep prying eyes from discovering the combination). Passing the same information in full payload is possible, of course, but more time consuming to extract as the stream has to be buffered, the data found and extracted, and then formatted in a way that it can be verified.
>
> 2. **Normalizes credentials across the infrastructure.** If there were, say, some infrastructure standard that specified the way in which such API keys were generated, then it would be possible to share a single API key across the infrastructure. Normalization would enable correlation and metering in a consistent way and if it is only the security code that changes per user, we can then leverage that as the differentiator for authorization of specific actions within the environment.
>
> Imagine that we take this normalization further and centrally log using a custom format that includes the API key and service invoked. A management solution could then use those aggregated logs and, indexing on the API key, compile a list of all services invoked by a given customer and from that generate – even in real-time – a current itemized billing scheme.

3. **Eliminates dependency on third-party identity stores.** By leveraging a scheme that is self-verifiable, there is no need to require validation against a known identity store. That means any piece of infrastructure supporting such a scheme can immediately validate the key without making an external call, which reduces the latency associated with such an act and it eliminates another potential point of failure. It also has the effect of removing a service that itself must be scaled, managed, and secured which reduces complexity for cloud computing providers and organizations implementing private cloud computing environments.

THE INTERSECTION of INFOSEC and INFRASTRUCTURE INTEGRATION

Traditional enterprise application integration methods of addressing the challenge of managing credentials internally often leverages credential mapping or a single, "master" set of credentials to authenticate and authorize applications. This method has worked in the past but it also imposes additional burdens on the long-term maintenance and management of credentials and introduces performance problems and does not support a multi-tenant architecture well.

An API key-based scheme may not be "the" solution, but something has to be done regarding security and its impact on infrastructure that necessarily needs to turn on a dime and potentially support multiple tenants. Security is an integral part of an enterprise architecture (or should be) and there are alternative methods to the traditional username/password credential systems we've been leveraging for applications for what feels like eons now. It's not just a matter of improving performance, that's almost little more than a positive side effect in this case; it's about ensuring that there exists a security model that's feasible and flexible enough to fit into emerging data center models in a way that's more aligned with current integration practices.

Infrastructure 2.0 has the potential to change the way in which we architect our networks, but in order to do so we may have to change the way in which we view authentication and authorization to those network and application network components that are so critical to achieving a truly automated data center.

F5 Networks, Inc. | 401 Elliot Avenue West, Seattle, WA 98119 | 888-882-4447 | f5.com

F5 Networks, Inc.	F5 Networks	F5 Networks Ltd.	F5 Networks
Corporate Headquarters	Asia-Pacific	Europe/Middle-East/Africa	Japan K.K.
info@f5.com	apacinfo@f5.com	emeainfo@f5.com	f5j-info@f5.com