Containers vs Hypervisors: The Battle Has Just Begun

Let me make one thing clear up front: I like Docker. I think it is good tech. I can see all sorts of scenarios when a lightweight, containerized deployment mechanism could be really nifty, even if it doesn't have the advantage of a hypervisor under it to protect its host operating system. I get it. It's cool.

But there are a few loud, but lonely, voices in the crowd who proclaim, “The battle is over! Containers have won! Hypervisors are obsolete!”

I'm sorry, that's just wrong.

That's like a championship fighter who upon landing a couple good blows on his opponent in the first round yells, “Stop the fight! I've won! Crown me champion!”

We all know it doesn't work that way. Ultimately, the championship goes to the one who wins the battle. And, in this case, the battle isn't over; it's just beginning.

The concepts behind Docker and other Linux containers are solid:

- Very small VMs that allow for much higher server density by removing redundant or unnecessary operating system elements from the VMs themselves.
- Nicely packaged VM stacks, which can easily be transferred, replicated, and controlled, ensuring high levels of portability.
- VM software stacks that are small, removing the problem and tedium of building a large stack of version-specific operating systems and tools that need lots of care and feeding to replicate and maintain.
- Extremely fast startup times that can facilitate a more flexible infrastructure, allowing greater latitude to respond to the needs of the moment – literally.

The challenge, however, is that the security attack surface of a “shared kernel” strategy has its weakest link in that “shared kernel” itself. If one malicious hacker manages to violate that shared kernel, all instances that employ that shared kernel are potentially compromised.

Certainly, a similar argument can be made of traditional hypervisors – if you can violate the hypervisor, you might be able to violate the VMs it powers – but the industry has had many years experience hardening hypervisor installations. While it is not a task for slackers, it is not rocket science either. Hundreds of successful virtualization and cloud providers in the world attest to the manageability of the task; from giants like Amazon, Rackspace, Verizon, and Huawei, down to smaller local and boutique-style service providers, the names of which are far less well known.

Some voices asserting the supremacy of containers cite Google as the proof case. By architecting the Google Docs application stack to work safely on containers, Google has shown the way, they say.

Except Google is Google. They can afford to hire thousands of the best and brightest to do intelligent things that few others can do. After 30 years in this industry, and two decades dealing with customers on site, I doubt that most organizations could readily do what Google has done. If they could, they'd be

Google, too.

The ideal solution is to provide the benefits of containers while actually reducing the attack surface. In the age of the cloud, systems need a higher degree of security than ever before. The shared kernel scenario simply doesn't provide that.

According to reports from a couple different attendees of LinuxCon North America, there were at least two sessions where major Docker users testified they ran Docker in traditional VMs to bolster security. It's really difficult to claim that hypervisors are dead when they still represent the security framework of power users. Maybe that's why these particular users made no such erroneous claim.

However, when it comes to security, unikernels are another story.

Sometimes called “cloud operating systems” or “library operating systems,” unikernels combine many of the advantages of Docker-like container systems with the security footprint of hypervisors and a much smaller attack surface within each VM.

Unikernel systems create tiny VMs. Mirage OS from Xen Project, for example, has created several network devices that run kilobytes in size (yes, that's “kilobytes” – when was the last time you heard of any VM under a megabyte?). They can get that small because the VM itself does not contain a general-purpose operating system per se, but rather a specially built piece of code that exposes only those operating system functions required by the application.

There is no multi-user operating environment, no shell scripts, and no massive library of utilities to take up room – or to subvert in some nefarious exploit. There is just enough code to make the application run, and precious little for a malefactor to leverage. And in unikernels like Mirage OS, all the code that is present is statically type-safe, from the applications stack all the way down to the device drivers themselves. It's not the “end-all be-all” of security, but it is certainly heading in the right direction.

The number of unikernel-type systems today is still small, and their names are not yet well known. In addition to Mirage OS from Xen Project, there are others like OSv from Cloudius Systems, HaLVM from Galois Systems, NEC's ClickOS, and LING (formerly known as “Erlang on Xen”). They provide environments which bind to languages such as Java, C, OCaml, Haskell, and Erlang. And they greatly simplify both the development and deployment processes surrounding the unikernel solution stack.

They are young – and they are the beginning.

I fully expect that five years from now we will look back at the unikernels of 2014 and see these as the seedlings of what will be a growing forest of unikernel-type systems. Unikernels raise the bar on the notion of a containerized VM by matching the touted packaging and deployment benefits while improving software development workflow and resource efficiency. And, perhaps most importantly of all, it does all this while **reducing the external attack surface** through operating environment specialization. Frankly, I can't wait to see what will develop in this space.

“The battle is over?” No chance. We're not even done with Round 1. It’s just beginning to get to really interesting.