The Kubes: Practical Container Orchestration Security

P2P3-W08: The Kubes: Practical Container Orchestration Security

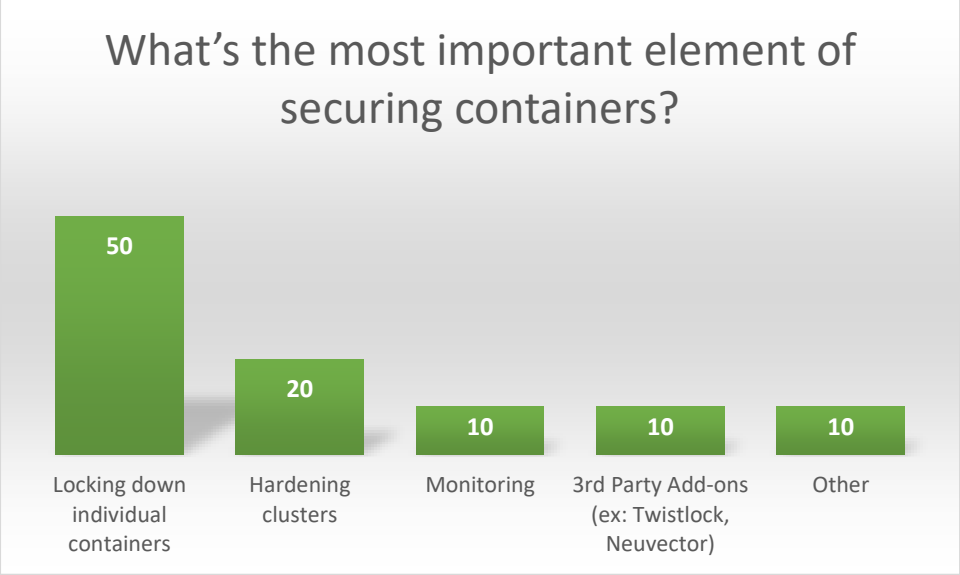
Diana Kelley, Cybersecurity Field CTO, Microsoft

The Kubes: Practical Container Orchestration Security

Companies are adopting and deploying containers at a rapid rate, layering on orchestration tools like Kubernetes to help make usage of containers at scale manageable. But even when using mature orchestration tools (that get even more mature by the day), the work of hardening and securing containers (not to mention the orchestration components) is not always easy and there's no one-size-fits-all approach for every use case and org. The goal of this P2P was to facilitate a lively discussion among peers to discuss wins, misses, and lessons learned on the path to secure container use and container orchestration.

To get the conversation started I queried the attendees in advance with the RSA Conference polling tool.

Pre-Session Responses



Responses were pretty spread out, except for locking down individual containers, which was the clear leader. Surprisingly, actual hardening of the container orchestration or cluster environment was seen as less important and monitoring of the environment even less so.

We had a mix of voices in the room, some from very large companies with active hands-on container orchestration administration and oversight roles, others from smaller firms, and some people who were there to listen and learn because they were at the start of their container orchestration journey. After taking a quick look at the poll responses, we got into the discussion.

Top of mind topics included:

Where does security configuration start? Keeping in mind that there is some active work required on the part of administrators and engineers to secure an "out of the box" Kubernetes implementation (for example, all too often we see etcd – port 2379 - exposed on the open Internet, belying a potentially problematic Kubernetes configuration), what are the key success factors required to make this happen? Is it primarily in securing the underlying OS running the container engine, the Kubernetes console or services on which it depends, or something else?

General consensus of the group was that if the underlying OS hasn't been locked down, the foundation for your containers is vulnerable. Making sure that the underlying substrate – i.e. those virtual or physical hosts running the container engine, the orchestration infrastructure and other components required – is both critical and requires attention and discipline (not to mention understanding of the security model in use) to do well. This is in keeping with recommended industry best practice: layer security all the way up the platform stack for defense in depth.

For example, harden the Linux host by turning off SSH when not required and disabling other unnecessary services; consider kernel modules, like AppArmor or SELinux, to enhance security on those hosts. Also, be sure to turn on MFA for remote access and (very important but may times overlooked) don't let pods/containers run as root.

Following on from this, attendees discussed which team in the organization is responsible for the security of the configurations in use. Answers varied, but some responded that as with other ops activities, the Ops (or DevOps) teams managed the day to day functions, using policies and direction from the security teams.

While on the topic of policies, the conversation shifted into a discussion about how to migrate existing policies especially when moving on-prem non-containerized workloads into cloud-based containerized deployments. Some of the historical constraints of container engines (e.g. namespace issues) can "muddy the waters" when it comes to multi-tenant container use (i.e. when one container engine is used to run containers from multiple tenants). Because of this, it's important that policy address under what circumstances – and for what sort of use case – a multi-tenant container deployment is acceptable in the first place. That aside, moving a container engine onto virtualized infrastructure can make management confusing, since this means both the containers as well as the virtualized OS they're running on are both easily mobile. Therefore, organizations may wish to strengthen policies around inventorying and asset management for both OS virtualization where containers will live and also the containers themselves.

After the policy discussion, we moved onto Namespaces. The group concurred that namespace-specific permissions provided more control than permissions applied by cluster. Which led to a conversation

about how namespace permissions are not the same as data protection. They are intended to be used to help divide resources among users, utilizing RBAC for AuthZ.

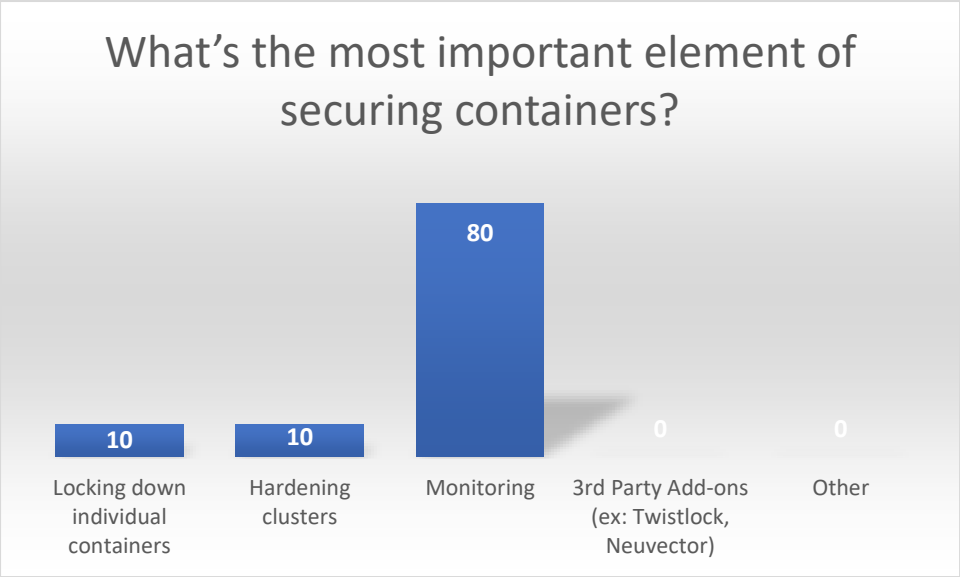
We also talked about trust and how to determine if a container is trusted or not. For example, everyone knows that downloading a random container off the Internet and running it as root probably isn't a great idea. But there's nuance here – for example, what if a developer makes a mistake or if unforeseen/unmanaged dependencies are included in a container that serve to make it less trustworthy? Just like it's important to keep track of the software, libraries, dependencies and middleware that you use in traditional application development or deployment, so too is it important to keep track of these things when using containers as well. Containers, by their nature, serve to help foster a "tear it down and respawn" mentality, thereby helping to reduce the prevalence of "holdover" and transient artifacts that can persist in a long-running production system, but this doesn't give free license to ignore dependencies completely.

Another topic that we spent a lot of time on was encryption, key management, and wrapped secrets. In Kubernetes, "Secret" objects store sensitive information like tokens (e.g. API or OAuth tokens) and passwords (e.g. for middleware) and are attached to pods at runtime. Most attendees reported using some kind of vault (ex: Azure Key Vault, Amazon KMS) to help manage keys, but there were concerns that too many people, including ex-employees, had access to those keys if they weren't rotated frequently. Just like you would have policy and processes around where and how secrets are stored without containers, so too should you extend this to cover how secrets are stored with them.

When it came to firewalls and container-specific firewalls, it's worth noting that this group was not relying on them for segregation. A few people mentioned how quickly firewalls can be circumvented in a container-heavy environment. More traditional layer three type segmentation was reported as being used for instances where stronger isolation was needed.

We wrapped the session with a question about 3rd party add-ins for additional container security, but no one in the room reported using them. At the end of our discussion we had another pass at the poll and the numbers had shifted quite a bit with a strong shift (40 percentage points) towards monitoring.

Post-Session Responses



I'd like to thank everyone who took the time to attend the session and share in the peer to peer conversation. There's no single path to secure container perfection but working together and sharing our experiences with each other we can accomplish stronger, more resilient deployments.

Resources:

- CIS Benchmarks for Kubernetes: https://www.cisecurity.org/benchmark/kubernetes/
- Cloud Native Computing Foundation: https://www.cncf.io/blog/2019/01/14/9-kubernetes-security-best-practices-everyone-must-follow/
- Azure Kubernetes Service (AKS), Security & Monitoring: https://docs.microsoft.com/en-us/azure/aks/intro-kubernetes#access-security-and-monitoring