Running Kubevirt Workloads with No Additional Privileges

Luboslav Pivarc @ Red Hat
What is coming?

• Kubernetes & Kubevirt crash course
• How is security enforced?
• What is enforced?
• Where did we start, where are we now and Where are we heading?
What is Kubernetes?

“Open-source system for automating deployment, scaling, and management of Containerized applications.”

Source: https://kubernetes.io/
How does the Kubernetes look like?
Node perspective

Pod A
Pod B
Pod C

Containers
What is Kubevirt?

“KubeVirt is a Kubernetes extension that allows running traditional VM workloads natively side by side with Container workloads.”
Kubevirt Integration
Node perspective

Virt-handler

kubelet

Pod A

Virt-launcher

Pod B

Pod C
How is security enforced?
Security policies

• Restricted - hardening best practices
• Privileged - allows for known privilege escalations
What is restricted?

- Capabilities
- Selinux/AppArmor
- Running as Root
- Privileged containers
- Seccomp
- Privilege Escalation (no_new_prsvs bit)
- HostPath volumes
- Host Ports
Kubevirt

- Capabilities
- Selinux/AppArmor
- Running as Root
- Privileged containers
- Seccomp
- Privilege Escalation (no_new_privs bit)
- HostPath volumes
- Host Ports
“That's one small step for security”

- First step was to have unprivileged networking
- We used `NET_ADMIN`, `NET_RAW`, `NET_BIND_SERVICE` because:
  - Pod gets IP address
  - Containers get Interface that requires configuration
  - Configuration of network requires privileges
  - Interface IP needs to be exposed to Guest (DHCP)
Solution

- Offload network setup to privileged component (Virt-handler - privileged container)
- Requires Libvirt “Unmanaged” option
- Existing management tool is losing privileges
- NET_BIND_SERVICE stays around
“As easy as setting user for workload & using Libvirtd in session mode”
Running as any non-root user

• Security policies requires anyuid
• Pre-allocated ranges of uids
• Qemu processes can’t read each other’s disk
• Filesystem permissions are set at build time of container images
• Modifying container FS at runtime can trigger copy
Solution

- Use "EmptyDir" feature that is just tmpfs with relaxed permissions
- Manage the permissions by Kubevirt
Storage for non-root user

- Filesystem/Block Volumes don’t have standardized permissions
- Kubernetes provides feature “fsgroup”
  - Does not always work
  - Restricted by some policies
Solution

• Manage permissions with privileged component (Virt-handler - privileged container)
Devices for non-root user

- Kubernetes expose devices through device plugins
- Devices are exposed with same permissions as on the host
  - This lead to inconsistencies depending on the setup
  - Not usable out-of-box for non-root users most of the time
Solution

- Manage permissions with privileged component (Virt-handler - privileged container)
- Drawback is that we can only manage devices that we know
Capabilities for non-root containers

Transformation of capabilities during execve()

During an `execve(2)`, the kernel calculates the new capabilities of the process using the following algorithm:

\[
\begin{align*}
P'(\text{ambient}) & = (\text{file is privileged}) \ ? \ \emptyset : P(\text{ambient}) \\
P'(\text{permitted}) & = (P(\text{inheritable}) \ & F(\text{inheritable})) \ | \\
& \quad (F(\text{permitted}) \ & P(\text{bounding})) \ | \ P'(\text{ambient}) \\
P'(\text{effective}) & = F(\text{effective}) \ ? \ P'(\text{permitted}) : P'(\text{ambient}) \\
P'(\text{inheritable}) & = P(\text{inheritable}) \quad \text{[i.e., unchanged]} \\
P'(\text{bounding}) & = P(\text{bounding}) \quad \text{[i.e., unchanged]}
\end{align*}
\]

where:

- \( P() \) denotes the value of a thread capability set before the execve(2)
- \( P'() \) denotes the value of a thread capability set after the execve(2)
- \( F() \) denotes a file capability set
Solution

• Non-root containers requires file capabilities on the executed binary
• Ambient capabilities are the future
  – They don’t require changes to image
  – Keeps working with no_new_privs bit
  – Missing support in Kubernetes
Drawback

“File capabilities require always requesting the capabilities for workload, disallowing opt-in approach”
SELinux

Keep me enabled!
What's left to do?

- Arbitrary user running workloads
  - How does user namespaces affect this?
- Remove custom SELinux policy
  - Upstream rules that makes sense for general container use cases
  - Use alternative API that are not requiring privileges
- Upstream support for Ambient capabilities
- Switch to “Restricted first” approach?
Thank you

You can get in touch with Kubevirt

- Twitter @kubevirt
- Slack kubevirt-dev