TECHNICAL WHITE PAPER: **July 2025**

Container Security in VMware Tanzu Platform

A Guide for InfoSec Teams

vmware® by Broadcom

Table of contents

Introduction

As container adoption has surged, so too have the security threats targeting containerized environments. A recent report by Kaspersky reveals that 85% of geo-distributed businesses utilizing container development methods experienced cybersecurity incidents in the past year, leading to data leaks, financial losses, and diminished customer trust.

Containers have become integral to Infrastructure and Operations (I&O) organizations, offering agility, scalability, and consistency across development and production environments. Their lightweight nature and ability to encapsulate applications with their dependencies make them ideal for rapid deployment and efficient resource utilization.

However, the very features that make containers advantageous also introduce unique security challenges. Common attack vectors include:

- **Image Vulnerabilities:** Malicious or outdated code within container images can be exploited if not properly scanned and maintained.
- **Insecure Configurations:** Misconfigurations in container settings can expose systems to unauthorized access and potential breaches.
- **Runtime Exploits:** Attacks targeting the container runtime environment can lead to unauthorized actions or access within the host system.
- **Exposed Secrets:** Embedding sensitive information, such as API keys or credentials, within container images can lead to unauthorized access if these secrets are compromised.

Given these challenges, it's imperative for I&O leaders to integrate robust security measures throughout the container lifecycle. This includes implementing best practices for image management, configuration, runtime security, and secret handling. By doing so, organizations can harness the benefits of containerization while mitigating associated risks.

In this paper, we will describe the security features in VMware Tanzu Platform to help address these challenges. As VMware Tanzu's flagship platform offering, Tanzu Platform brings consistency and control to the platform engineering experience, while delivering developers and application teams an experience that allows them to quickly move from idea to code, and code to production – accelerating application delivery and enabling faster time to market.

Objectives, Definitions, and Scope

In this paper, we'll review the most important container security concepts. Our goal is to help Information Security (InfoSec) teams evaluate the built-in container security features of Tanzu Platform. Logging, networking, and routing security capabilities are omitted.

We do not discuss Kubernetes, or how VMware Kubernetes solutions secure containers. As such, container registries and other orchestrator-specific components are not reviewed.

What is a Container? And How Does It Lead to Better Business Outcomes?

When people refer to a container they usually mean a container image or a container instance. A **container image** is an ordered collection of root filesystem changes, and the corresponding execution parameters for use within a container runtime. A **container instance** is a running instance of an application based on a container image. So what makes an application process a container? The magic of containers stems from a set of Linux kernel functions that isolate a given process from the rest of the processes on a machine. Further, the container limits the resources (CPU, memory, disk, etc.) available to a given process.

In this paper, we'll refer to **containers, application instances**, and **application containers** interchangeably. Consider these terms synonymous. Likewise, we'll refer to **VMs** and **hosts** as equivalent. (We also refer to **cells**, which are the VMs that Tanzu Platform uses to host application containers.)

As a deployment unit, container images use Linux kernel constructs in a repeatable and reusable way. Container images also bundle everything an application needs to run, like code, binaries, libraries, runtime, configuration, and filesystem. As a result, IT operations teams can avoid the setup and ongoing maintenance of specialized environments for different applications. This is a significant shift compared to traditional methods. Further, it yields a more efficient hand-off between developers and operations.

Another benefit: containers are very lightweight compared to VMs. What do we mean by lightweight? Application containers don't have an OS kernel. Containers running on a VM share the OS kernel of that VM. This is another significant shift in software deployment.

Containers can start and stop far faster than a VM. Containers offer greater elasticity compared to VMs, resulting in greater speed and agility.

There is also a direct financial benefit: containers can lower your infrastructure cost. Containers give higher density on the same hardware investment. Figure 1 explains how containers increase your application density.

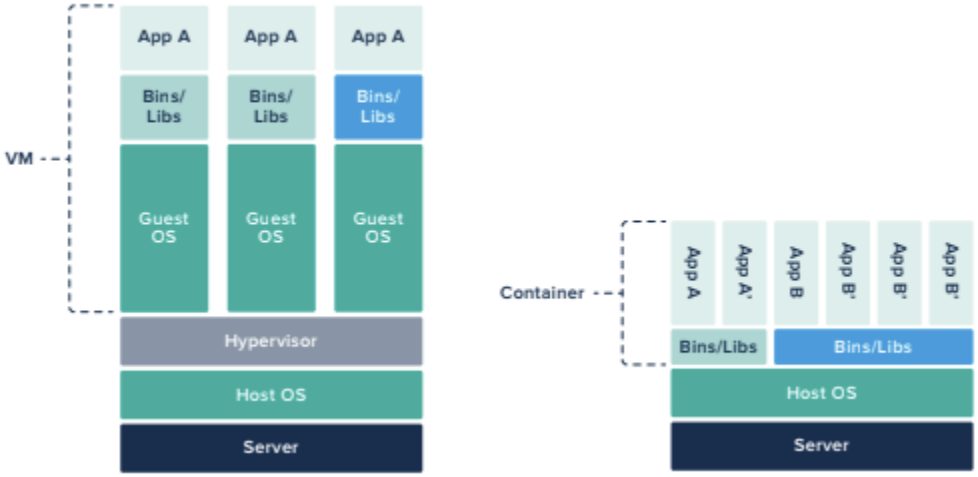

Figure 1: A comparison of virtual machines and containers.

Containers Need Extra Layers of Security. Here's What You Need to Know.

Containers can deliver wonderful business benefits because they share the OS kernel and resources. When surrounded by the right tech stack, containers can help provide speed, agility and higher density than VMs. But there's no such thing as a free lunch. These benefits come with complexity. You need to incorporate additional security provisions compared to a VM-centric model. The shared OS kernel and resources makes containers more risk-prone compared to VMs. (For an in-depth look at the threat vectors for containers, review the **Appendix**.) That said, the upside of containers is well-worth the extra cost for security.

In fact, distributed systems now embed many powerful container security concepts "out of the box." In this section, we'll review these concepts, then examine how Tanzu Platform implements each one.

Namespaces: A Fundamental Isolation Concept

A namespace is a Linux kernel feature that allows a given process to have an isolated view of system resources.

With namespaces, many different processes can run on the same host and share the same kernel. This way, "global" resources, say a virtual machine, are isolated so that a process has a restricted view of the system. Namespaces are the first line of defense. They prohibit one container from disrupting another container, or host for that matter. What happens without namespaces? Very bad things! For example, an application written by a bad actor can send a kill signal to another application process. Or malicious code can escalate privileges and take control of your system.

Here's a look at different namespaces and what each does.	
Namespace	What it isolates
IPC (Interprocess Communications)	Interprocess message queues. These are interprocess communication resources that allow processes to communicate with each other using shared memory.
network	This namespace governs the network devices, stacks, and ports used by applications to communicate over a network.
mount	The first namespace introduced in the Linux kernel. It controls what mount points a process can view, thereby governing access to the filesystem.
PID	Provides a namespaced tree of process IDs. It controls what you will see with the ps command in a container.
user	Perhaps the most important element of container security. This provides namespaced version User and group IDs for containers.
UTS	Manages the hostname and NIS domain name for the container.

Let's take a closer look at each namespace, and how they work with containers.

First up: the big picture. Each VM will have global namespaces that govern resources at VM level (Figure 2).

Figure 2: Global Namespaces apply at the VM level.

Now let's add containers to the picture.

A container can share a namespace with the host, or even with other containers. If a container shares a network namespace, it will get the same network interfaces, IP address, and port space (Figure 3).

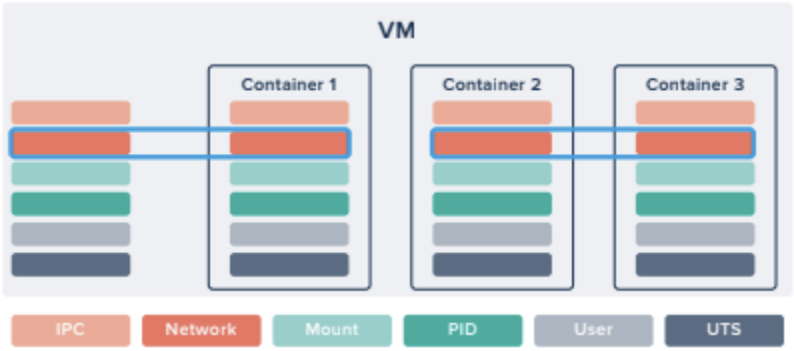

Figure 3: Shared namespaces with containers.
Notice that Container 1 shares a network namespace with the host VM.

Let's see how Tanzu Platform implements namespaces.

Namespaces in Tanzu Platform

Tanzu Platform guarantees complete isolation for each application instance because it uses all the namespaces for each container. A given container can't "see" anything on any other container, and it can't communicate with any other container. This is a foundational element of secure multi-tenancy. It's something often overlooked in homegrown container management solutions.

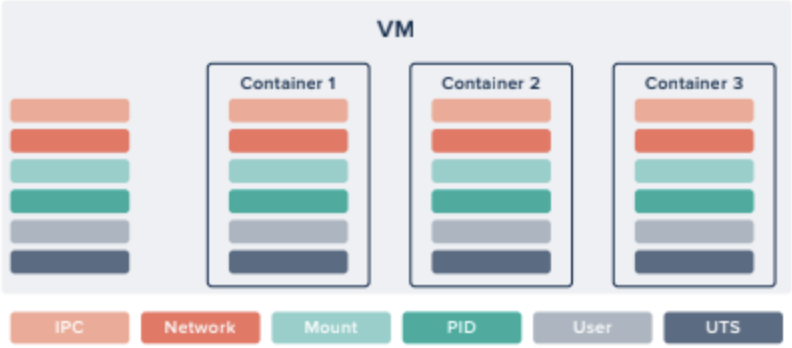

Figure 4: Tanzu Platform creates separate namespaces for each container (from host and each other).

IPC Namespace for Interprocess Communication Resource Isolation

The IPC (POSIX/SysV IPC) namespace provides separation of named shared memory segments, semaphores, and message queues. These objects are only visible to the processes running in the same IPC namespace. Without this particular namespace, processes within the container can see the IPC on the host system, or with another container. Again, this is undesirable as it allows bad actors to gain access to your systems.

Tanzu Platform creates each container within its own IPC namespace, and prevents IPC related attacks and Denial of Service attacks.

PID Namespace for Process Isolation

Process isolation is achieved with a combination of namespaces, specifically the IPC namespace described above and the PID (Process ID) namespace.

The PID namespace provides isolation of process IDs. PID namespace controls the ability of processes to see and interact with each other. This limits cross-container process visibility. It also limits the ability of processes within different PID namespaces to interact with each other (e.g. using signals). The first process in the PID namespace gets PID 1. (PID namespace also provides PID virtualization. Here, each container has a PID 1 process. Additional process PIDs are sequentially numbered.)

Tanzu Platform gives each container its own PID namespace. This protects the container from cross-application attacks, information leaks, malicious use of ptrace, and other such vulnerabilities.

User Namespace Helps You Avoid Privileged Containers

The user namespace, a new addition to the kernel, is perhaps the most critical. Without a user namespace, containers can't be isolated. With user namespaces in place, you sidestep the dangerous condition known as privileged containers.

In a privileged container, root user access in a container maps to the root user on the VM. This condition is unsafe! When the protective boundaries melt away (like in the case of a container breakout), malicious code can take control of the host or VM. When you implement a user namespace, you isolate user IDs (UIDs), and mitigate the impact of a potential breakout.

By default, each application instance created by Tanzu Platform runs as an unprivileged container. This is crucial because a "privilege escalation exploit" can grant malicious code the ability to run code as root, giving the bad actor complete control of the VM.

Even if such an exploit happens in Tanzu Platform, the bad actors aren't able to gain root access to the VM.

Consider Figure 5 below. The diagram depicts how Tanzu Platform creates the application container with a separate user namespace, and how users work across container and host.

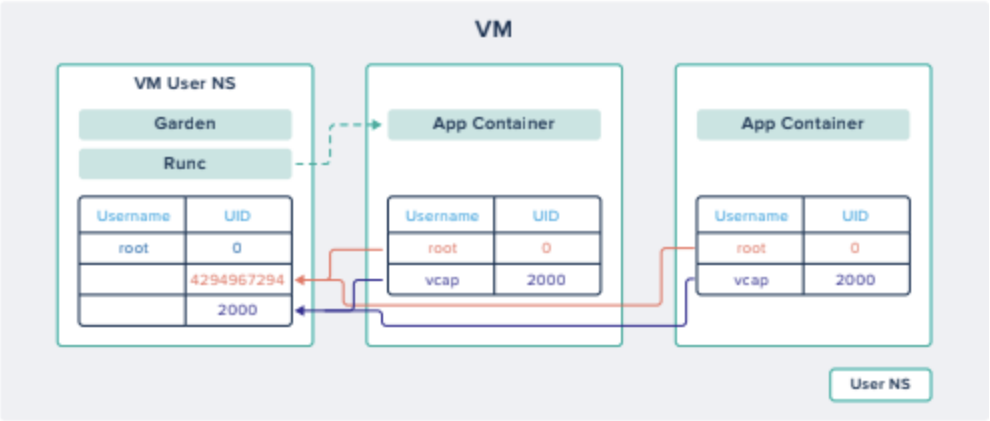

Figure 5: Tanzu Platform creates the application container with a separate user namespace. Also shown: how users map across container and host.

In Tanzu Platform, root in the container maps to the non-root maximus user with (max UID -1) in the host namespace. The maximus on the host is not a named user. So even if there is a container breakout, the user is not a root on the host. The end result: your risk is reduced.

Mount Namespace and `pivot_root` Deliver Filesystem Isolation

This safeguard prevents a given container from gaining illegitimate access to filesystem objects that belong to other containers or the host. The mount namespace isolates the mount points visible to each application instance. This way, a process can issue mount and unmount commands, but only on those specific mount points.

But the mount namespace doesn't guarantee data isolation. That's because containers inherit a view of filesystem mounts from their parent. They can access all parts of the filesystem by default. To achieve data isolation, some container-based systems use the chroot system call. This action changes the root directory of the container. But privileged processes (with CAP_SYS_CHROOT privilege) can escape chroot, sometimes referred to as "chroot jail." That's why Tanzu Platform has something different.

Instead, Tanzu Platform uses the **pivot_root** system call. This mechanism changes the **rootfs** of each application container. (More specifically, pivot_root updates the '/' mount point.) The issue with **pivot_root** is that it changes the '/' mount point. If used without Mount NS this would mean causing change to '/' mount point of the host (or cell to be specific in case of Tanzu Platform). That is why Mount NS separates the mount points of the container from that of the cell and **pivot_root** is used along with it to provide solid file system isolation. This is further augmented by restricting user scope by using User Namespace. Each container can have its own **rootfs** and there is no privilege that would allow escaping the **rootfs**. Tanzu Platform optimizes resources by sharing **rootfs** as a read-only layer of the container file system.

Tanzu Platform Uses Proven Filesystems for Performance, Scale and Security

Tanzu Platform uses a combination of OverlayFS and XFS filesystem drivers. OverlayFS provides layered file system. The base layer is always root file system (RootFS). RootFS provides the root ("/") directory, and all other file systems are mounted atop this file system. RootFS is the read-only layer. Application binaries (bundled as a droplet) are placed on top of the base RootFS, in the read-write layer, a very small part of the filesystem.

The filesystem has to perform at scale and deliver a strong security posture. That's why Tanzu continuously tests various filesystems (like BTRFS and AUFS) for scale, speed and security. We perform these tests because our customers operate at

scale. In fact, some Tanzu Platform deployments run more than 200,000 application instances.

These filesystems have their strengths and weaknesses. So how did Tanzu arrive at the use of OverlayFS and XFS, instead of BTRFS and AUFS? Let's dig into the evaluation process. It's an instructive example that embodies how Tanzu thinks about technical choice within the platform.

In this case:

- AUFS wasn't part of the Linux Kernel. That led to undesired overhead to keep the filesystem updated.
- AUFS doesn't have support for disk quota; loop devices were required for this purpose. But you can't have more than 250 loop devices, which hinders scalability
- AUFS requires root privileges, which means the container manager requires root privileges. This hindered our desire to adopt a rootless container runtime.
- BTRFS supports both layering and quota. But, it ran into performance issues when testing at scale.

Tanzu engineers perform these rigorous tests to ensure the right filesystem gets picked for the right job. In DIY projects, you may end up selecting suboptimal systems. (After all, it is difficult to perform competitive analysis of various filesystems.) Leave this gruntwork to Tanzu, so you can focus on adding business value!

Network Isolation Restricts the "Blast Radius" of Potential Attacks

In Tanzu Platform, each application instance starts in a separate network namespace. The network namespace enables virtualization for ports. All application containers on the same cell listen on a secured port, since each one resides in separate network namespaces. This is shown below in Figure 6.

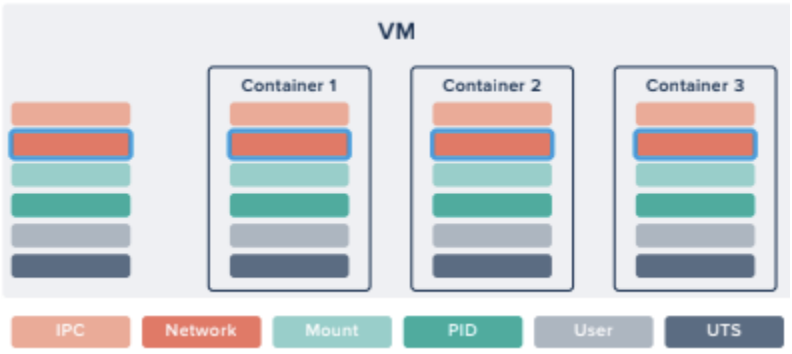

Figure 6: Each container gets a separate network namespace.

In Tanzu Platform, containers run on the VXLAN L3 overlay network. All cells in your Tanzu Platform deployment share said network. By default, Tanzu Platform allocates each cell a /24 range that supports 254 containers per cell, one container per usable IP address. This overlay network is not routable, and the overlay IP for the container is not directly routable from outside of the cell. Further, Tanzu Platform blocks east-west traffic. A container can't send requests to a nearby container, even when they are on the same cell. So how does communication happen between containers? With a pair of virtually-linked Ethernet (veth) interfaces.

One of the veth interfaces is assigned to the network namespace of the container; the other is assigned to the VM. A virtual link is established between the pair. Let's take a closer look.

For each application instance in Tanzu Platform, there is a port allocated on the cell. The cell's routing table controls the routing. Gorouter maintains the "maps" for application routes and collections of IP:port combinations. (Here "IP" and "port" are the values that belong to a given cell.) In Figure 7 below, traffic on port 60001 will be DNAT'ed to Container 1 and traffic on

60002 is DNAT'ed to Container 2.

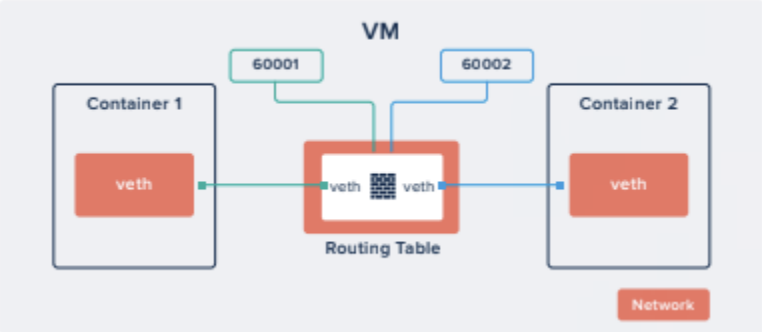

Figure 7: DNAT and Routing Table for containers on a cell.

By default, Tanzu Platform includes a firewall that prevents containers on the same VM from communicating with each other. The system protects against ARP spoofing attacks with network isolation, L3 overlay networking, and firewalls.

Learn more about application routing and network security provided by Tanzu Platform in this video series.

Denial of Service (DoS) Protection

Denial of Service (DoS) attacks are nearly as old as the Internet itself. Wikipedia defined this attack as:

"…a denial-of-service attack (DoS attack) is a cyber-attack in which the perpetrator seeks to make a machine or network resource unavailable to its intended users by temporarily or indefinitely disrupting services of a host connected to the Internet. Denial of service is typically accomplished by flooding the targeted machine or resource with superfluous requests in an attempt to overload systems and prevent some or all legitimate requests from being fulfilled."

The world of containers has its own flavor of this. Here, malware works its way into your containers and attempts to consume your host resources. This, in turn, brings down all the applications on that host.

Tanzu Platform includes several features to protect you from this type of attack.

Resource Limiting with Cgroups

In a multi-tenant environment, no single application instance should hog all system resources. When this happens, your apps will become unstable, and your entire system can grind to a halt. Control groups (cgroups) provide a mechanism for controlling access to resources.

Cgroups also provide resource limiting, prioritization, and accounting. This feature is the first line of defense against a denial of service attack.

When Tanzu Platform schedules containers, the Rep on the cell first allocates the required resources. Then, it makes a claim for an application instance. The auctioneer picks the most suitable cell, and "awards" the containerto that particular cell.

Note that the resource claim takes place before the creation of the container. Once the resource is allocated by Garden-runc, there is no way for the container to get any extra resources. For example, the memory control group function allocates memory quota for the container. If the application instance exceeds its memory quota, Tanzu Platform kills the container, and starts a new instance.

Allocation of CPU by Shares

CPU allocation is a bit tricky, as there are layers of virtualization. But the same idea applies. Instead of allocating CPU on a percentage basis, cgroup uses "shares" to allocate CPU for application instances. The container gets CPU allocated based on the ratio of shares for the container and the sum total of all shares associated with containers. Figure 8 shows an example.

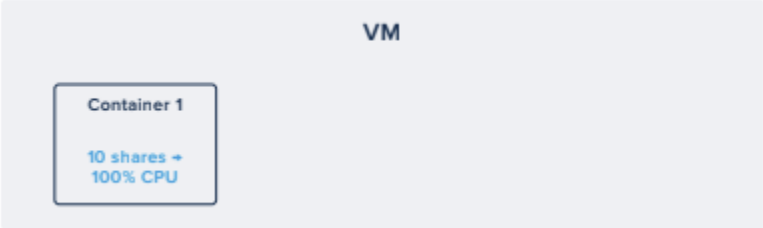

Figure 8: A single container on a VM gets 100% CPU.

In this case, there is only one container on the VM. So, it gets 100% of the available CPU cycles.

In Figure 9, a second container comes up with exactly the same amount of shares. Since they both have the same amount of shares, each gets 50% of the available CPU.

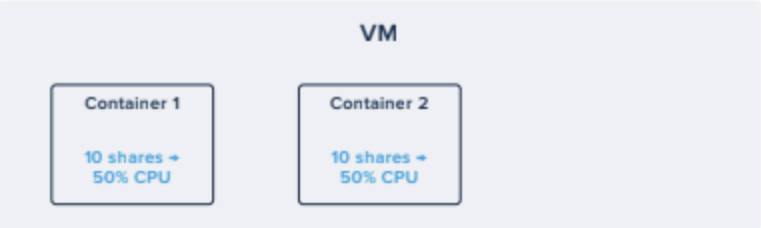

Figure 9: Multiple containers on VM get a percentage of CPU based on shares.

Figure 10 shows a third container with twice the amount of shares. Based on the ratio, the third container gets 50% of the CPU. Tanzu Platform divides the remaining 50% equally between the first two containers.

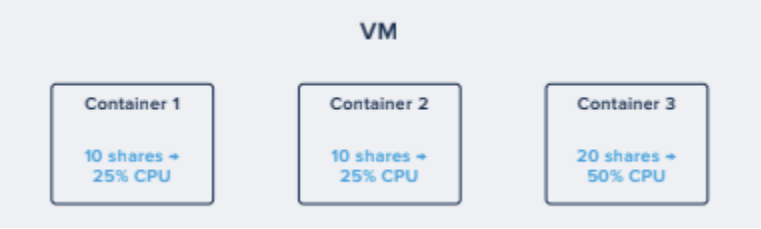

Figure 10: Multiple containers on VM get a percentage of CPU based on shares.

Can a malicious container still cause denial of service, simply by allocating itself the largest number of shares? No, because Tanzu Platform protects against that. It grants CPU shares linearly compared to memory allocation (Figure 11). Tanzu Platform caps shares for each container at 1024 so no container ever receives more than that many.

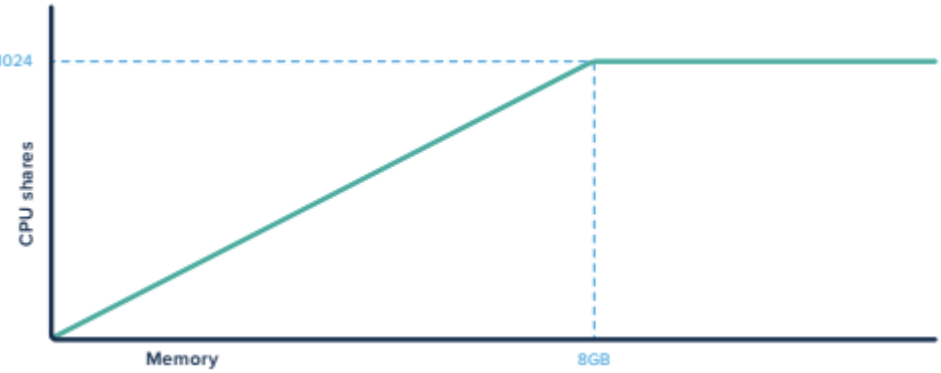

Figure 11: CPU shares capped at 1024.

Another important aspect: share allocation minimizes CPU allocation. When a container becomes idle, Tanzu Platform distributes shares back to active containers. Let's revisit our example from Figure 10. If Container 3 becomes idle, Containers 1 and 2 will each again get 50% CPU. When Container 3 becomes active again, the shares for Container 1 and 2 drop down to 25% each.

Protection Against Fork-bomb Attack

Another well known DoS attack vector is fork-bomb. Attackers can create many processes from within a container, and overwhelm the PID limit of the system. This brings the system to a complete halt. The PID cgroup prevents this kind of attack by restricting the number of tasks that a container can create. Tanzu Platform creates a PID cgroup for every container, which prevents intentional (or accidental) fork-bombs.

Cgroups and Whitelisting Device Access

Devices are a kernel resource which are not namespace aware. Tanzu Platform restricts access to devices using cgroups by whitelisting trusted device nodes. The following device nodes are whitelisted by Tanzu Platform:

/dev/full
/dev/fuse
/dev/null
/dev/ptmx
/dev/pts/*
/dev/random
/dev/tty
/dev/tty0
/dev/tty1
/dev/urandom
/dev/zero
/dev/tap /dev/tun

For more details, refer to this video series.

Disk Quota Underscores the Importance of a Trustworthy Filesystem

Another critical resource is disk. Typically, a user quota aims to limit disk. But, in the case of Tanzu Platform, all containers use the same VCAP user. So we can't employ user-based disk quotas. Instead, Tanzu Platform uses directory-based quotas provided by XFS at the read-write layer in the container filesystem. This is why Tanzu Platform uses XFS and OverlayFS. (Tanzu performed extensive testing on AUFS and BTRFS prior to selecting XFS as the long-term filesystem in Cloud Foundry.)

Tanzu Platform Reduces Your Attack Surface

Every bit of code presents some risk. That's why many InfoSec pros will tell you that "less is more" when it comes to security. When you remove unnecessary bits, you reduce the attack surface of your systems.

To this end, Tanzu relentlessly slims down our entire stack, all the way down to the OS of the VM. In this section, we review our methodology with respect to Linux capabilities.

Tanzu Removes Linux Capabilities to Reduce Threat Vectors

So far, we've discussed many security features Tanzu has added to reduce the risk you face from containers. But what do containers look like without these protections?

For starters, the UID 0 (i.e. a root user) has complete control over the system. And the same is true for two processes—**privileged processes** (whose effective user ID is 0 i.e. root), and **unprivileged processes (**whose effective UID is nonzero).

Unprivileged processes are subject to full permission-checking based on the credentials of a given process. (Usually effective UID, effective GID, and supplementary group list). This means a vulnerability in, say a setuid binary, started by a lower-rights user can be exposed to a privilege escalation to root and related attacks.

As it happens, privilege escalations via setuid binaries are a common threat. Many of us have used **/bin/ping utility** which is a setuid root binary. **Ping** only requires one privileged operation: it opens up raw sockets. But a vulnerability in **ping** can help attacker gain root access, and therefore act as the full root user!

Without any controls, the effective UID of the process is either root or a non-root user. That's why any responsible InfoSec team needs to take a hard look at Linux capabilities. Each capability represents a privileged action. Starting in kernel version 2.2, capabilities can be separately added or dropped. This means more control for system administrators.

Now that you have this control, this ability to toggle various parts of the kernel, how will you enforce your chosen configuration?

That's the beauty of Tanzu Platform. It enforces a best-practice configuration for all your apps running across the entire platform. First off, the platform builds unprivileged containers by default. Tanzu Platform also drops several unneeded capabilities for every container it creates. After all, every dropped capability further limits actions the container UID can perform. In particular, Tanzu Platform drops these Linux capabilities for unprivileged containers:

- CAP_DAC_READ_SEARCH
- CAP_LINUX_IMMUTABLE
- CAP_NET_BROADCAST
- CAP_NET_ADMIN
- CAP_IPC_LOCK
- CAP_IPC_OWNER
- CAP_SYS_MODULE
- CAP_SYS_RAWIO
- CAP_SYS_PTRACE
- CAP_SYS_PACCT
- CAP_SYS_BOOT
- CAP_SYS_NICE

- CAP_SYS_RESOURCE
- CAP_SYS_TIME
- CAP_SYS_TTY_CONFIG
- CAP_LEASE
- CAP_AUDIT_CONTROL
- CAP_MAC_OVERRIDE
- CAP_MAC_ADMIN
- CAP_SYSLOG
- CAP_WAKE_ALARM
- CAP_BLOCK_SUSPEND
- CAP_SYS_ADMIN

Tanzu elects to drop these particular capabilities because many have potential attack vectors. Consider these examples:

- CAP_SYS_MODULE – allows the container to load and unload kernel modules. Here, an attacker can cause privilege escalation and modify the kernel of the VM.
- CAP_SYS_ADMIN – this is "as good as root" capability, need to say more?
- CAP_NET_ADMIN – allows an attacker to enable promiscuous mode for the attacked network interfaces and sniff across namespaces.
- CAP_DAC_READ_SEARCH – allows Tanzu Platforms discretionary access control such as file read/write/ exec permission checks. Attacker can gain access to any file using opaque id through Tanzu Platforms namespace restrictions.
- CAP_SYS_RAWIO – attacker can exploit known access issues for /dev/mem, dev/kmem, or procfs entries related to devices.
- CAP_SYS_PTRACE – allows ptrace which can be exploited as detailed in Defense in Depth.
- CAP_MAC_ADMIN – allows container to override the Mandatory Access Control system.

We are always looking for ways to reduce components in this way. You can expect Tanzu to continue to strip out unnecessary bits at every opportunity.

How Tanzu Platform Hardens the OS and RootFS

Containers, though isolated, are a function of the Linux kernel. So any viable enterprise platform must tune and vet the OS as well as the container filesystem. OS hardening is a task that the operations team must perform for all critical systems. This job is especially challenging for distributed systems running at scale. Tanzu Platform answers the call with an opinionated view of OS management using stemcells.

A stemcell is a versioned OS image wrapped with IaaS specific packaging. It contains a minimum of OS features to securely configure the VM. Tanzu Platform uses hardened OS stemcell for cells and hardened RootFS for containers.

BOSH, the release engineering toolchain that packages Tanzu Platform components, doesn't require any packages or libraries for software that runs atop VMs. Put another way, BOSH releases are self-contained. The stemcell doesn't contain any superfluous packages, libraries, or configurations. This model reduces your attack surface, and is more secure than traditional methods.

Here are a few examples of our hardening strategy for stemcells and the RootFS:

- **Filesystem Hardening**: Tanzu locks down the tmp filesystem, because bad actors often target this directory. Other problematic directories are in a separate partition with restricted access. This way, malware can't easily spread.
- **Minimization of Attack Surface**: We've removed unnecessary items like Network Information Service and rsh tools. We also disable talk, telnet, tftp and many other services. Less is more!
- **Network Security**: We modified the IPv4 network configuration to be safer. Also, we disable unneeded protocols like SCTP, DCCP, LDAP, and RDS.

We run more than 270 tests for each stemcell; these tests are widely used in industry and government sectors. Tanzu also continuously scans for CVEs; we release patches as soon as possible. (Refer to tanzu.vmware.com/security for details on this process.) Platform engineers that run Tanzu Platform don't have to deal with highly customized "snowflake" servers. They can adopt immutable infrastructure practices, and deliver massive efficiency and security gains through automation, like zero downtime patching.

Tanzu Platform Offers "Defense in Depth" for Application Containers

The strategies discussed in this paper provide solid isolation and security for containers. However, it's not enough; overlapping layers of security are essential. A vulnerability in any one layer shouldn't give access to other layers of Tanzu Platforms to a malicious container or code within a container. Remember, there are several dangerous places and paths in the Linux kernel. Once malicious code gets in, your risk goes way up!

To this end, using a platform offers "defense in depth": multi-faceted protections across system boundaries.

Here's a look at how Tanzu Platform helps secure systems and data in this comprehensive way.

AppArmor, Part of Tanzu Platform, Improves Your Security Posture

AppArmor is a Mandatory Access Control System (MAC) that's part of the mainline Linux kernel. Here, "access control" refers to the control over access to system resources after a user's account credentials and identity have been authenticated, and system access is granted.

Most applications have Discretionary Access Control where a user controls access to the files and resources it owns. On the other hand, MAC is controlled by the OS and can't be overridden by users. MAC is the strictest level of access control. AppArmor is a path-based system implemented as a Linux Security Module (LSM) or Loadable Kernel Module (LKM). AppArmor helps enforce MAC policies by applying profiles to processes that enable restriction of privileges. The privileges can be restricted at the level of Linux CAPs and file access.

AppArmor works great with runC. (Kudos to Docker for contributing their AppArmor integration to the Open Container Initiative.) And since runC is the default container runtime in Cloud Foundry, Tanzu Platform users get AppArmor compatibility in Garden. It augments security provided by the namespaces, cgroups, and Linux capabilities.

AppArmor is easy to adopt because it focuses on programs, not users. It restricts a given program's access inside a container to system resources like network and disk. Administrators use AppArmor to create enforcement policies that are simple to author and easy to audit. This last point is often underappreciated. Without an easily auditable system, problematic security loopholes can exist via a policy that isn't consistent with itself.

In Tanzu Platform, AppArmor is pre-configured with a default policy and enforced by default for all unprivileged containers. Let's take a closer look at the implementation.

Tanzu Platform Uses AppArmor to Mitigate Threat Vectors

With AppArmor, containers aren't allowed to remount **/dev/pts** and get access to the host's terminal. Kernel file system such as /proc and **/sys** have sensitive data and configuration that can be used for lateral attacks. AppArmor blocks access to the dangerous locations in **/proc** and **/sys** to thwart several threat vectors. The following snippet shows blocked locations for **/proc.**

```
deny @{PROC}/* w,   # deny write for all files directly in /proc (not in a subdir)

# deny write to files not in /proc/<number>/** or /proc/sys/**   deny @{PROC}/{[^1-9],[^1-9][^0-9],[^1-9s][^0-9y][^0-9s],[^1-9][^0-9][^0-9]

[^0-9]*}/** w,   deny @{PROC}/sys/[^k]** w,  # deny /proc/sys except /proc/sys/k*

(effectively /proc/sys/kernel)   deny @{PROC}/sys/kernel/{?,??,[^s][^h][^m]**} w,  # deny everything except

shm* in /proc/sys/kernel/   deny @{PROC}/sysrq-trigger rwklx,   deny @{PROC}/kcore rwklx,
```

This AppArmor profile snippet locks down access to other important components:

- It denies write access to **/proc/sys/kernel** (except **/proc/sys/kernel/shm***). This directory contains a variety of different

configuration files that affect the operation of the kernel.

- It denies write access to **/proc/sys/kernel/modprobe. modprobe** is a utility to add and remove modules from the Linux kernel. The path to this utility is configurable via **/proc/sys/kernel/ modprobe**. A bad actor could change the value in this file to point to a malicious code that will be executed when the kernel attempts to load a module.
- It denies all access to /proc/kcore. The **/proc/kcore** file represents the physical memory of the system and is stored in the core file format. Access to this allows a container to read host memory.
- It denies all access to **/proc/sysrq-trigger**. Writing to this file allows executing System Request Key commands. For example, you can reboot the VM by writing to this file.

Seccomp Blocks Harmful System Calls

Seccomp (Secure Computing) is also part of the mainline Linux kernel. Seccomp uses a Berkeley Packet Filter (BPF) to filter system calls made by restricted programs. It works like this.

Seccomp BPF provides filtering by blacklisting and whitelisting system calls. With blacklisting, the dangerous system calls are explicitly mentioned, and all others are allowed. And with whitelisting, the system calls required by the container are explicitly mentioned and only those system calls are allowed. Whitelisting means all unlisted system calls are blocked. This approach reduces the surface area for break-out exploits inside a container. Why? Even in the event of a container breakout, malicious code can't execute dangerous system calls.

This whitelisting approach also provides a sensible entry-level lockdown mechanism for containerized applications. In Tanzu Platform, this is set up out-of-the-box. We use the industry-standard seccomp profile to maximize compatibility with existing images.

Tanzu Platform's Defense-in-Depth Protections Help in These

The end goal is to "sandbox" the application container so that it is truly isolated from other containers, and from the VM. Namespaces, cgroups, AppArmor and seccomp are the features that serve the purpose. Individually, they are not sufficient to provide necessary sandboxing for the container. That's why Tanzu Platform uses all these features together, and more, to create containers that provide the required isolation.

Let's now consider a few common threat vectors and see how Tanzu Platform protects against them.

Protect Against `ptrace` Escape Container Attacks by Dropping Vulnerable Components

There are situations where even seccomp can't prevent certain attacks. For example, **ptrace** is used for debugging and tracing purposes. With **ptrace**, a t**racer process** attaches to a **tracee process**. The tracer gets extensive control over the tracee, including its memory and registers. In fact, the tracer can change the registers of the tracee process.

Ptrace system calls by Tanzu Platforms seccomp. This is possible because seccomp filtering is applied before the tracer is notified, and before the tracee process actually executes through the code that initiates the syscalls. An attacker can change the registers of the syscalls, and inject malicious code. Since this is done after seccomp has had a chance to filter the syscalls, the attacker can use **open_by_handle_at syscall** to escape the container. In this case, Tanzu Platform blocks **ptrace** syscall by removing the CAP_SYS_PTRACE capability.

Protecting Against the Shocker Exploit by Dropping Vulnerable Components

Open_by_handle_at can be exploited; it uses an opaque file identifier that can be passed across processes. It can be used to gain access to any file, even files outside of the **mount** namespace. The exploit applies to any container, so Tanzu Platform blocks **open_by_handle_at** syscall by removing the CAP_DAC_READ_SEARCH capability.

Protecting Devices and IO with Unprivileged Containers

In a privileged container with CAP_SYS_RAWIO, an attacker can use **/proc** to gain access to the control regions of some devices and their respective IO ports. Even if AppArmor blocks access to **/proc**, a hacker can use the compromised container to infiltrate with system calls. Here, Tanzu Platform once again offers "defense-in-depth" protection since all containers are unprivileged by default. (You'll be glad to know that Tanzu Platform drops CAP_SYS_RAWIO for privileged containers too.) As described in the Resource Limiting / cgroups section, cgroups whitelist the devices a container can access.

Protecting Against Malicious Access to Environment Variables with CredHub

Let's consider the worst case scenario. What happens if a container breakout takes place, and malicious code can read **/proc/[pid]/environ?** In this hypothetical scenario, there could be a 0-day vulnerability that allows the attacker to break out of mount and **PID** namespaces. (Side note: this is why you need to ensure that the OS has minimal surface area, and is continuously patched). In this case, an attacker can read the proc of all containers. This could expose environment variables of other containers. But, Tanzu Platform protects you with yet another security provision – CredHub.

CredHub is a secure credential management component that runs on the BOSH VM to minimize the surface area where credentials can be compromised. It manages credentials like passwords, certificates, ssh keys, RSA keys, and other secrets.

CredHub ensures safe communication throughout the system. It protects secrets for on-platform backing services (like VMware Tanzu for Postgres, VMware Tanzu for MySQL, VMware Tanzu for RabbitMQ,VMware Tanzu for Valkey). What's more, CredHub extends this protection to off-platform services including Oracle, or SAP, and other incumbent applications.

CredHub consists of a REST API and a CLI. It uses the CLI and API to get, set, generate and securely store such credentials. The REST API conforms to the Config Server API spec. CredHub is an OAuth2 resource server that integrates with User Account Authentication (UAA) to provide core authentication and federation capabilities.

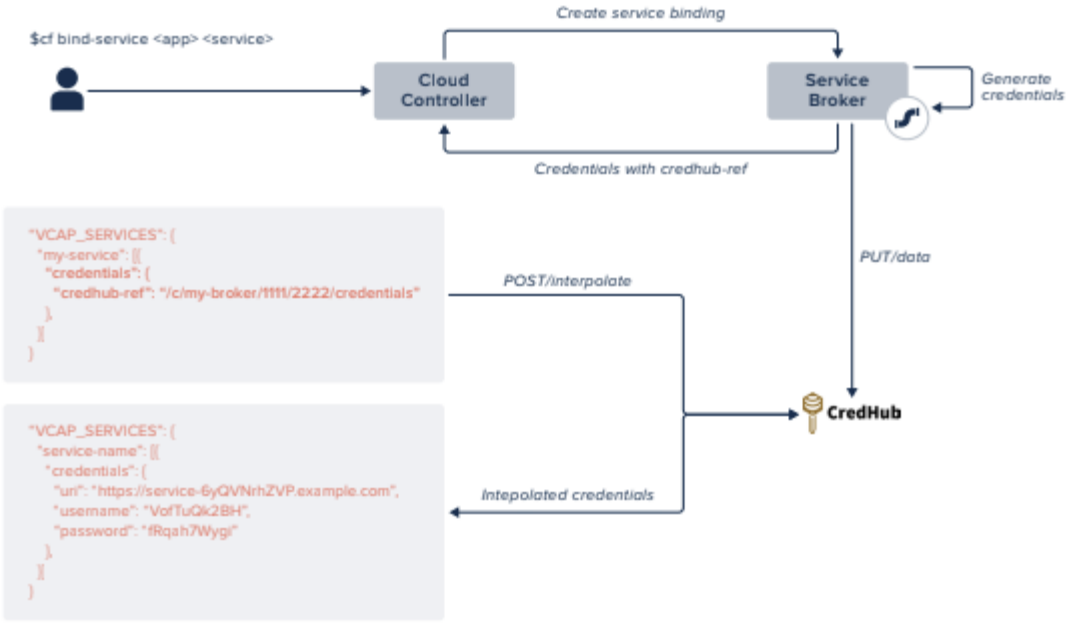

Figure 12: CredHub stores, manages, and rotates system credentials to protect systems and data. Here's how it abstracts these details away for day-to-day use.

How does CredHub help protect credentials? Let's return to our hypothetical scenario.

Even if a malicious code breaks out of a container and gets access to **/proc/[pid]/enviro**n, it will still not be able to access the actual credentials. All an attacker can see is credhub-ref, because the actual values are safely encrypted in CredHub.

Moreover, CredHub ensures that only the specific application instance can interpolate values. The identity of the application is established by the application identity certificate which is rotated every 24 hours.

CredHub itself is managed by Tanzu Platform as part of the application runtime. There is no operator overhead to manage the secrets store, or to integrate it with the platform. These security capabilities differentiate Tanzu Platform from many homegrown DIY options, and other vendor products. (Learn more about CredHub on our techdocs page.)

Protecting Against Man-In-the-Middle attacks

Defense-in-depth also helps protect against man-in-the-middle (MIM) attacks, thanks to network isolation. Here, container network connectivity is only routed (at Layer 3), not switched (Layer 2). There is also no possibility of a rogue VM entering the network, because BOSH manages theVMs (and their IP addresses). The IaaS provider also helps assure security of the underlying infrastructure.

Protecting Against Container Runtime Overwrite Vulnerability

A container vulnerability emerged (CVE-2019-5736). It seemingly affected all container platforms that use runC. A similar CVE was revealed for LXC too. An attacker can overwrite the container runtime (runC or LXC) binary and gain root access to the host. The default AppArmor profile has no protection against this. This vulnerability particularly affected Kubernetes environments as it allows privileged containers. As described in the User Namespaces, all containers created by Tanzu Platform are unprivileged. That means the **root** on the host is not mapped in the container, and **root** in the container is mapped to **maximus**. Because of this, no Tanzu Platform customers were impacted by the CVE. Secure by default!

Protecting Against IaaS Breaches with mTLS to the Application Container

Consider a threat vector where an attacker manages to breach IaaS security, or a disgruntled employee deploys malicious code. Will Tanzu Platform allow attackers to send illegitimate requests to the application container? No! Here's why.

Every container created by Tanzu Platform has a Envoy sidecar proxy within it. Envoy works as a local reverse proxy. Tanzu Platform provides it a X.509 certificates for TLS termination. (Tanzu Platform encrypts every request to the application container.)

Tanzu Platform rotates these certificates every 24 hours. Envoy Proxy also has an option to enable mTLS with routing tier (GoRouter or Istio). With this option, Envoy will verify the identity of the routing tier by requesting and verifying its certificates. This is mutual TLS (mTLS).

With mTLS, the identity verification (using certificates) is mutual. mTLS provides defense-in-depth on top of what is already described in Network Isolation. Note that the unsecured container port is not exposed outside of the container. So the only entry to the container is via a secured port! Best of all, you can enjoy this protection "out of the box." Your developers do not have to write any code! (This video demonstrates verification of mTLS provided by Tanzu Platform).

Immutable Infrastructure Helps Avoid Configuration Drifts Without Off-Platform Toolchain

Configuration drifts and snowflake configurations across environments are also sources of unpatched vulnerabilities. How do you ensure there are no accidental changes to configuration? What is the source of truth for configuration? How do you ensure governance at scale? Buildpacks and BOSH to the rescue.

Buildpacks Configure Frameworks and App Dependencies

What are buildpacks?

Buildpacks manage the framework dependencies when you push applications to Tanzu Platform. Deploy an app via the cf push command. The platform will then automatically associate the app with the right buildpack (Java, .NET, Node.js, whatever is needed). You don't have to think about dependencies, middleware or runtime components—the platform does it for you.

Buildpacks also bring uniformity to the application runtime across deployments. Once a new buildpack is available, operators make a simple update to the manifest and cf push the application with the new buildpack. Operators can choose to use blue-green or canary-style deployment to roll out upgrades. This systematic approach reduces your risk from human error. Tanzu Platform will kill existing containers as needed and create new ones. (This video captures the essentials of buildpacks.)

BOSH Helps You Adopt Immutable Infrastructure Practices

Tanzu Platform embraces immutable infrastructure and treats all containers and VMs as cattle, not pets. Tanzu Platform ensures that there are no snowflake VMs in the environment. The platform replaces running VMs with new ones generated from the "golden" image managed by BOSH. This is a paradigm shift in securing applications at scale. Any unhealthy VM is replaced with a new one. Unhealthy containers are replaced with new ones. Repave the whole platform with zero downtime, and you get brand new VMs and containers, all recreated from golden images.

Here's How Tanzu Platform Creates Secure Containers

The security constructs detailed so far are undifferentiated heavy lifting. They don't add any differentiating business value for you, even though they are necessary for securing the application containers. It's much easier to engineer applications when you know your platform is building secure containers for your applications. With that goal in mind, let's describe how it works in Tanzu Platform.

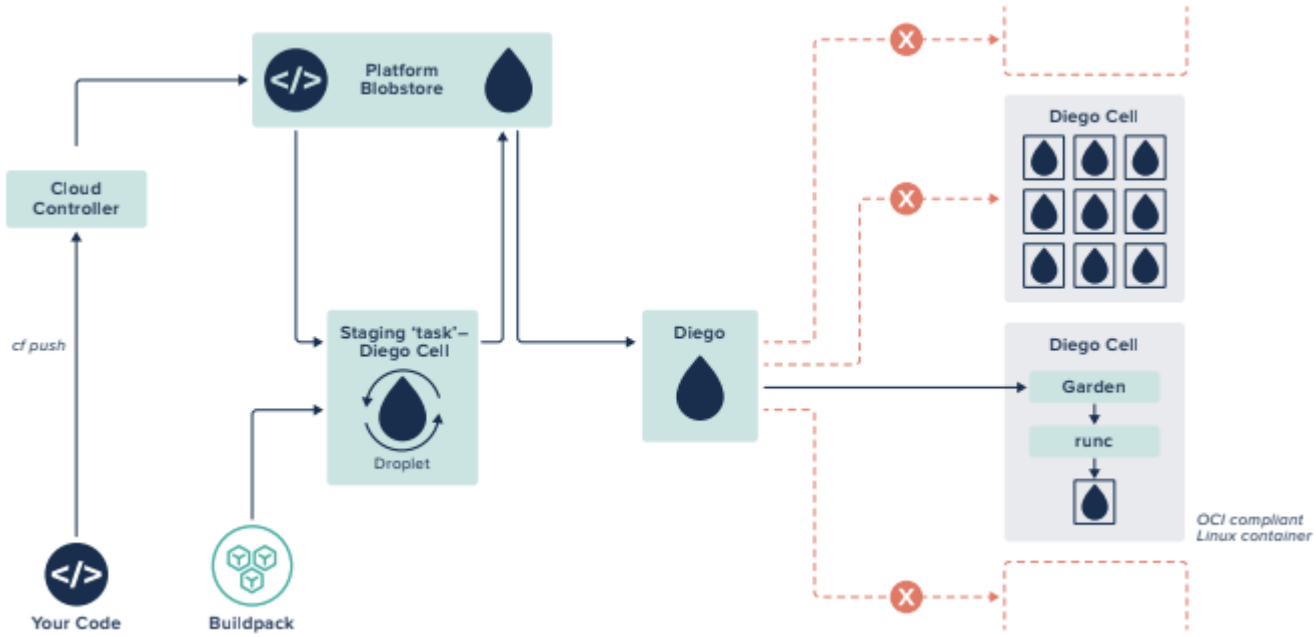

Figure 13: An overview of the Tanzu Platform container creation process.

First, the developer pushes the app to the platform, with the cf push command. (Tanzu Platform documentation describes this process in detail.) For every application instance, Tanzu Platform goes through an "auction" of all available cells. Then, it picks the most suitable cell for deployment. This happens for both long-running processes (LRPs) and short-lived workloads called tasks.

A "Rep" is part of the cell used in Auctions for LRPs and tasks. It claims an LRP instance based on available resources. Tanzu Platform also grants an LRP instance claim to a Rep. Rep then starts the container creation process with the help of Garden, a pluggable, API-enabled container runtime manager.

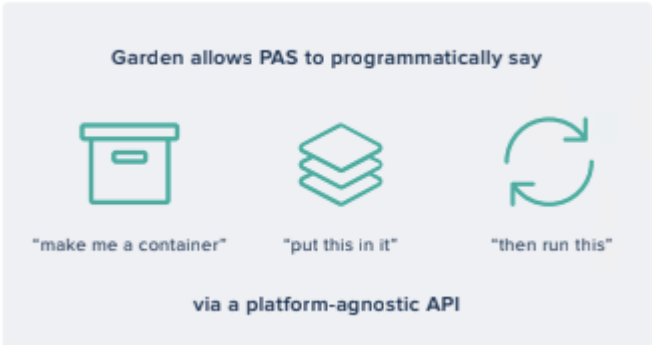

Figure 14: The top functions of the Garden API.

Containers created by Garden are Open Container Initiative (OCI) compliant. OCI is an open industry standard with two specifications. The **image specification** defines how the container image looks on disk. The r**untime specification** outlines how to run container filesystem bundles that unpacked on disk.

Garden has three important plugins: OCI-compatible file system management, OCI container runtime and Container Networking Interface (CNI). Garden orchestrates container creation using these three plugins. The default plugin implementations on Linux are GrootFS, runC, and Silk, respectively.

Rep then invokes the Create Container call from the Garden API to make a secure container from the droplet, which proceeds to kickstart the process of container creation.

Garden Uses GrootFS to Create OCI Bundle for Container Image

Garden invokes GrootFS to setup the file system for the container. GrootFS uses cflinuxfs4 to create a base layer. This base layer forms the RootFS of the container. Later, this is mounted as a read-only layer within a container. Here optimization is possible because all containers use the same RootFS. All containers share the same RootFS on the VM as no container is ever allowed to write back to the RootFS. So where do the application logs go? Well, they are in the read-write layer of the container.

As depicted in Figure 15, the Rep maintains a local cache for droplets. GrootFS picks up the droplet from the droplets cache, and places it above the RootFS layer. GrootFS returns to Garden with a list of mountcommands to get access to the newly-laid file system. (The disk limits are properties of the XFS which is established per container.)

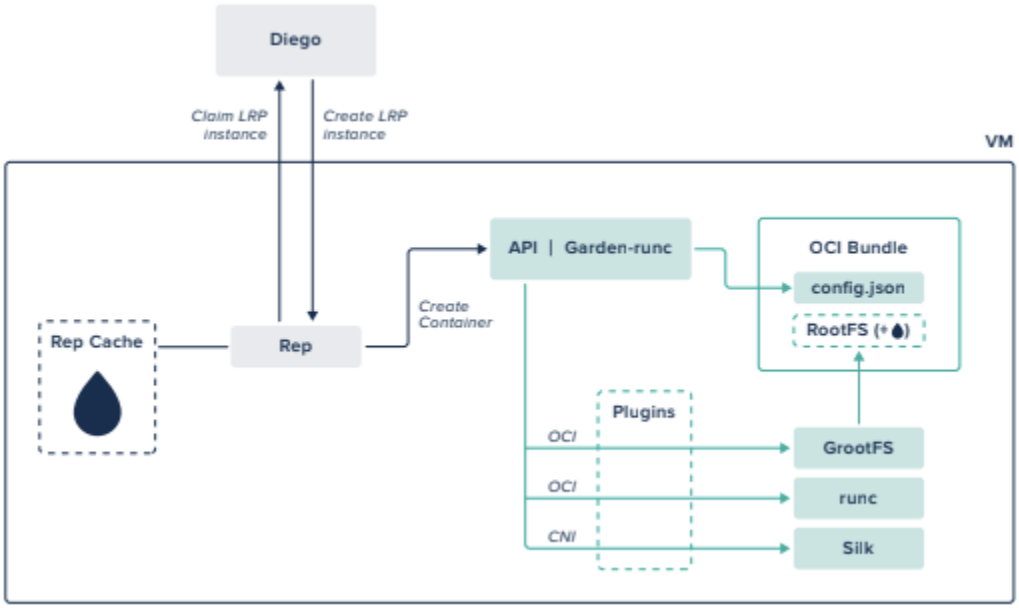

Figure 15: How Tanzu Platform uses container runtimes and plug-ins to create containers from a droplet.

Garden then creates a OCI bundle and **config.json** according to the OCI bundle spec. The **config.json** file has a complete profile of the container. We've discussed many sample values from this container profile already. This profile informs the container runtime how to create the container and how to lock it down. Some of the profile details included in config.json are listed below.

- Mount calls given by GrootFS
- AppArmor profile
- seccomp profile

- List of CAPs to exclude
- Process info—init process (PID = 0)
- **Init** never dies, so it keeps the container alive
- **Init** cleans up "zombie" processes, makes them child of **init**
- UID/GID mappings on the host
- Maximus map to Root on container
- List of Namespaces
- Cgroup config for resource limits and access control

Garden, RunC and CNI Work Together Setup Container Using OCI Bundle

RunC, the OCI container runtime, initiates the process of creating the container for an application using the **config.json**. The container is started with **Init** process and the profile in the **config.json** is applied to the init process, which makes that process into a container. The PID of the init process is in a file in the Bundle folder.

Next, Garden configures networking for the new container. Garden invokes CNI plugin and passes the path to the Network namespace of the container. The path for different namespaces is **/proc/<PID of Init>/ns/**.

(Quick note: Tanzu Platform comes with the Silk CNI plug-in. Tanzu Platform also supports VMware vDefend Distributed Firewall; this option is popular with enterprises that need advanced micro-segmentation features. Both Silk and NSX-T support policy-driven container networking that enhances security for multitenant workloads.)

From here, CNI sets up the host networking firewall using IP/Routing table. The routing table has mapping for a VM given port to the container port. Details of how this mapping works is in the network isolation section. By default, Tanzu Platform configures the firewall to block all other requests to the container. Once this step is done, the container is ready to serve traffic, except it doesn't have the application running within the container yet. That's the final step.

To do this last bit, Garden invokes runC to get the application running inside a container. The runC runs **fork** exec workflow. The **fork** system call starts a child runC process, and it joins the namespaces of the container. The resulting runC then calls exec system call to launch application process. The difference between **fork** and exec is that **fork** creates a child process that duplicates the calling process. **Exec** loads the new program into the current process space and runs it from the entry point. Exec is whatactually starts your application within the container. The application process is a peer of the **init** process in the container. And this is how a new locked-down application container is born!

Conclusion: Rapidly patch any part of the platform with minimal disruption to your business

The universe of containers is an exciting, if daunting one. (Appendix A and Appendix B describe the wild and woolly security landscape in more depth.) The value of containers, particularly when containers are the output of mature build pipelines, makes the complexity trade-off worth it.

We have reviewed many native security capabilities of Tanzu Platform in this paper. It's fitting to conclude the paper with discussion of the most important security advantage you gain with Tanzu Platform. Let's discuss the ability to rapidly patch any part of the platform with minimal disruption to your business.

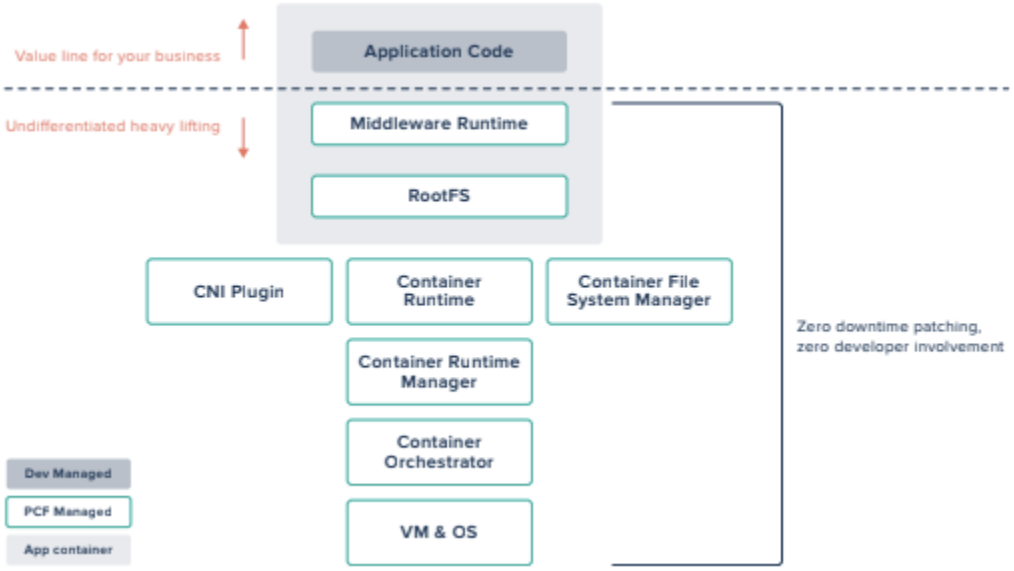

Figure 16: Application container stack

Your peers choose to run their most important applications on Tanzu Platform for this reason. When a CVE hits, Tanzu Platform offers you a fully automated capability to remediate the situation. Most enterprises do not patch software on time due to downtime and risk of failures during patching. That leaves them vulnerable. For instance, the XZ ssh attack were exploited by attackers but the Tanzu Platform was found to not be impacted. With Tanzu Platform, buildpacks handle the middleware runtime. Tanzu continuously scans for potential CVEs in the buildpack along with all other components that make up the Tanzu Platform, and immediately releases fixes.

Zero-downtime patching and upgrade brings another crucial benefit to customers. The container runtime as well as Tanzu Platform itself are fast changing technologies. Tanzu continues to modernize the components used in the platform. And thanks to zero-downtime upgrades provided by BOSH, you can conduct business as usual, pushing your apps to Tanzu Platform. In the future, the underlying components in Tanzu Platform are likely to evolve. Yet your developers will enjoy the same cf push experience.

CVE in Tanzu Platform	
When a CVE Occurs in this Component…	…This workflow updates the stack
Kernel	OS Patching is by Tanzu and orchestrated by BOSH
Container Runtime	Part of Tanzu Platform runtime, patched by Tanzu and orchestrated by BOSH
Container Orchestrator	OS Patching is by Tanzu and orchestrated by BOSH
Container file system (RootFS)	RootFS is a BOSH release, once patched in the runtime, Tanzu Platform rolls out new RootFS to all containers, recreating them as per desired scale.
Application Runtime (Middleware)	Patched using Tanzu Buildpacks
Application Code	Blue-green, canary style deployment patterns for zero-downtime upgrades

We've written about this process. For the OS layer, it goes something like this:

1. A CVE is identified. A fix is supplied, and a new OS image is created.
2. Tanzu conducts end-to-end tests with the new OS image.
3. After the image has passed these tests, it is posted as a new "stemcell" on Broadcom Support Portal.
4. Customers are notified about the availability of these updates.
5. Platform engineers download the new files, then add them to Tanzu Operations Manager. Mature deployments with many Tanzu Platform foundations tend to have automation pipelines for this flow. These pipelines allow engineers to manage versions and configuration for several foundations centrally. (We recommend Concourse.)
6. Once the deployment is started through Ops Manager or a Concourse pipeline, updated stemcells are automatically rolled out to the Tanzu Platform installation.

Tanzu Platform is an excellent option to help you securely implement containers, so you can get on with achieving better business outcomes.

Appendix A: Container Threat Vectors

The most significant threat vector is the Linux kernel itself. After all, containers are built using kernel features. A vulnerability in the kernel can bypass every application security policy you might have. For example, you may have a highly secured application running on SSL, and with top-class perimeter security. However, malicious code on a compromised kernel can read the memory of the application process and extract a private key. Similarly, large kernel footprint increases the attack surface. Consider a legacy network stack. These systems are often neglected, and offer an easy attack vector. And they often have many known vulnerabilities.

You can see the trend of vulnerabilities in the kernel in this chart:

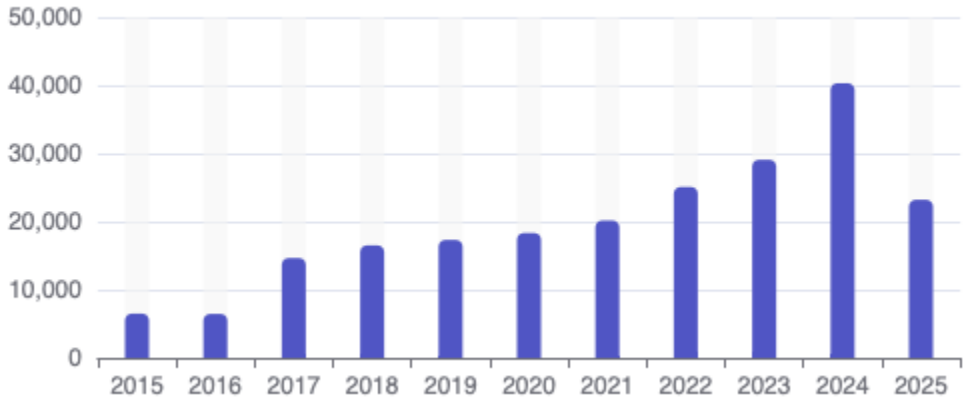

Figure 17: A look at the vulnerabilities by year for the Linux kernel (source: CVE Details).

Here are a few other common threats you need to be aware of:

- Container breakout
- Privilege escalation
- Vulnerabilities in the base image
- Vulnerabilities in dependencies
- Advanced persistent threats
- Denial of service to other containers
- Denial of service to host
- Spoofing, man-in-the-middle attacks
- Kernel modification
- File system modification
- And most importantly—vulnerabilities in the kernel

A more common threat vector: container escapes. Here, malicious code “escapes” from the guest into the host.

Some of the conditions can lead to a container escape include:

- Allowing privileged operations, if there happens to be a vulnerability in the **setuid** binary.
- Granting write access to **/proc** or **/sys.** For example, an attacker can modify the location of **modprobe** utility. From there, they can run their malware when the kernel loads a module.
- Poor networking setup can enable cross-container attacks or expose containers to ARP spoofing attacks.
- Device nodes (placed under **/dev**, separate from the rest of the file system) provide interfaces to device drivers. As such, the drivers present significant attack vectors because they expose interfaces, particularly the storage interface, to code running in the kernel space. Bad actors could gain illegitimate data access, escalate privileges, or mount other attacks.

Appendix B: The Need for Rapid Patching

The biggest risk to an enterprise IT system is an unpatched vulnerability running in production. The unpatched software can be anywhere in the application stack: the OS, middleware, container runtime, container image, application runtime, dependencies, application code, or container scheduler. No matter the location, your risk is significant.

That said, unpatched CVEs in containers are particularly troublesome. Why? Because containers are often built by hand. A single developer will package an app, its dependencies, and an OS into a container image. Then, they ship it off to production. In this scenario, there's no automation around building the container.

There's probably not a record of what's in the container. And there's likely no one else at the company who knows what's in the container except for the original builder. You are right to be anxious about containers in this scenario! (Of course, Tanzu Platform protects you from this workflow, as discussed in this paper.)

Just how real is this scenario? Consider a recent survey on open-source security from Snyk. The chart below shows some alarming figures. Even official images, those published by vendors, have scores of OS vulnerabilities. (We recommend reading the entire paper, The State of Open Source Security Report.)

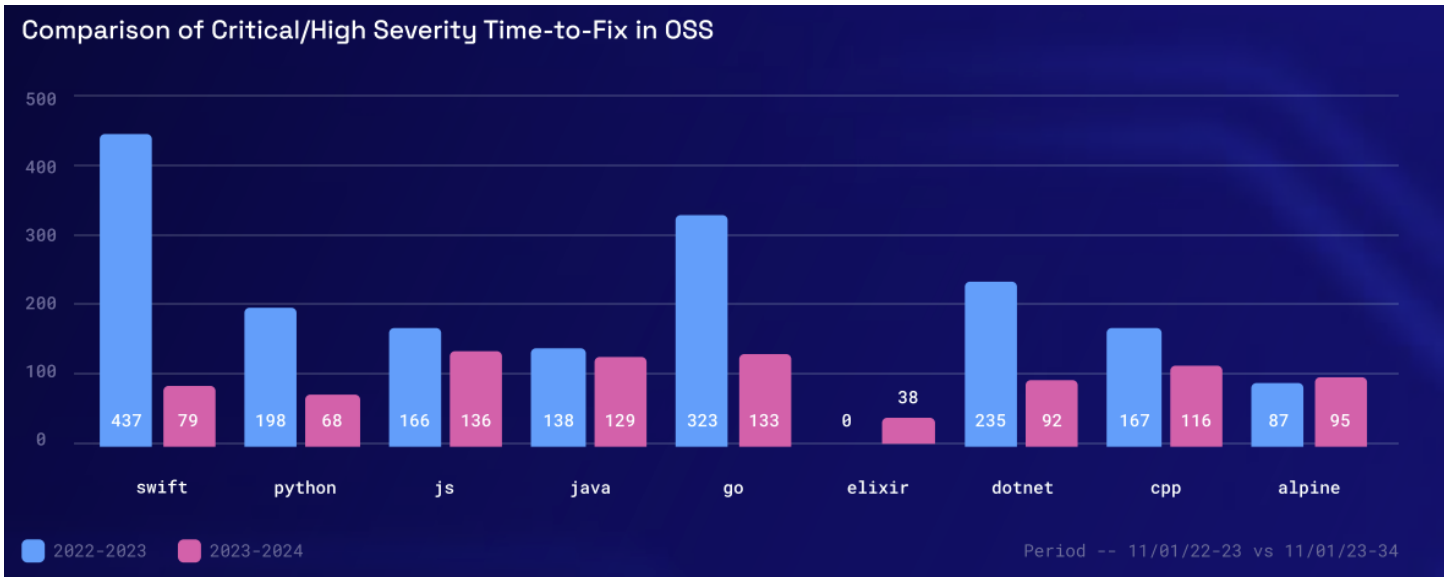

Figure 18: Source: Snyk's The State Of Open Source Security Report 2024

When your application stack isn't updated with fixes for known vulnerabilities, your entire IT infrastructure is at risk. (Refer to Figure 16 in the Conclusion for a look at how Tanzu Platform helps you.) Container images downloaded from open registries often have a large attack surface. If your container uses root to perform common functions, an attacker can take over the whole system. Application code itself can be a source of vulnerability. For example, if the code uses hard-coded credentials, or if it leaks credentials, your security measures are negated.

Consider the different threats in Appendix A. In general, for container-based deployments, the fear of a malicious container or of malicious code within a container is a major issue. The problem gets worse if a container breakout takes place in a multi-tenant environment. That's because malicious code in a container can cause denial of service attacks on other containers, or even on the host. (Sometimes this may not be malicious code, just inadequate containment of the application process. But the net effect is the same.)

Other serious issues are advanced persistent threats (APTs), and vulnerabilities that expose kernel or the filesystem to malicious code. That's why enterprises often cite security as the number one challenge to running containers to production.

The general security posture in most organizations is "secure the perimeter." But that is not enough in the multi-tenant world of containers. You can have the best fence around a tree, but it can't protect the tree from termites. You end up with a hollowed out tree surrounded by a lovely (but ineffective) perimeter.

Item No: Container Security in Tanzu Platform 7/25