Electronic Design

By RICHARD JAENICKE, Director of Marketing, Safety, and Security-Critical Products, Green Hills Software, www.ghs.com/integrity-178

What's the Difference in Security Between Virtual Machines and Containers?

Virtual machines and containers are widely used in embedded systems to consolidate workloads and enable DevSecOps, but which type of virtualization is more secure and is there a way to make DevSecOps secure?

Hardware virtualization using virtual machines (VMs) has several use cases in embedded systems, ranging from workload consolidation to running applications on legacy operating systems. Operating-system virtualization utilizing containers opens up an additional use case for enabling a DevSecOps (development, security, and operations) environment by packaging an application's dependencies with the application. Many differences exist between VMs and containers in performance, scalability, and portability, but this article will focus on security.

All types of virtualization provide a basis for security with a goal of isolating software running in a VM or container from other VMs and containers. Different types of virtualization solutions achieve isolation to varying degrees, and how tightly the isolation function is coupled to the virtualization function also can impact security.

After reviewing the basic technology of VMs and containers in the first section, the second section compares the security posture of those solutions based on three key indicators: the size of the trusted computing base (TCB), the number and the rate of Common Vulnerabilities and Exposures (CVEs) uncovered, and the types of security certifications that have been achieved. The third section then examines how to apply those findings to increase security in a DevSecOps environment.

Overview of Virtualization Types

In hardware virtualization, the VMs are controlled by a virtual-machine monitor, more commonly referred to as a hypervisor. Each VM simulates the same underlying hardware but can have a different guest OS. The hypervisor allocates a separate address space to each VM, which the hypervisor enforces using the CPU's memory management unit (MMU).

All of the software inside the VM executes as if it were running directly on the physical hardware isolated from other VMs and their software stacks. When a VM needs access to some shared physical resource, such as the network, display, or disk drives, the hypervisor intervenes and controls that access directly.

Originally there were two types of hypervisors: Type 1 hypervisors run directly on the physical host hardware, whereas Type 2 hypervisors run on top of an operating system. From a security standpoint, Type 1 hypervisors execute in privileged kernel mode, while Type 2 hypervisors execute in user space.

VMware ESXi and Xen Project are examples of Type 1 hypervisors, and VMware Desktop is an example of a Type 2 hypervisor. More recently, there are also hybrid hypervisors that combine elements of Type 1 and Type 2. For example, KVM is the virtualization layer in the Linux kernel, and it runs in kernel mode on bare metal.

For real-time and embedded systems, Type 1 hypervisors are commonly reduced in size and functionality to fit within resource constraints of embedded systems as well as provide more determinism than a server or workstation hypervisor. Improving determinism is particularly important for hypervisors used in real-time systems since a Type 1 hypervisor is

the source of extra latency for every application running in such a system *(Fig. 1)*. That's a critical issue for latency-sensitive real-time applications.

Type 2 hypervisors impose additional latency for applications running in VMs because there are now three layers of software: the guest OS, hypervisor, and host OS. However, with a Type 2 hypervisor, not all applications must run in VMs on the hypervisor. This makes a lot of sense in a workstation setting, for example, when a Linux workstation needs to run a small percentage of its applications on a Windows OS.

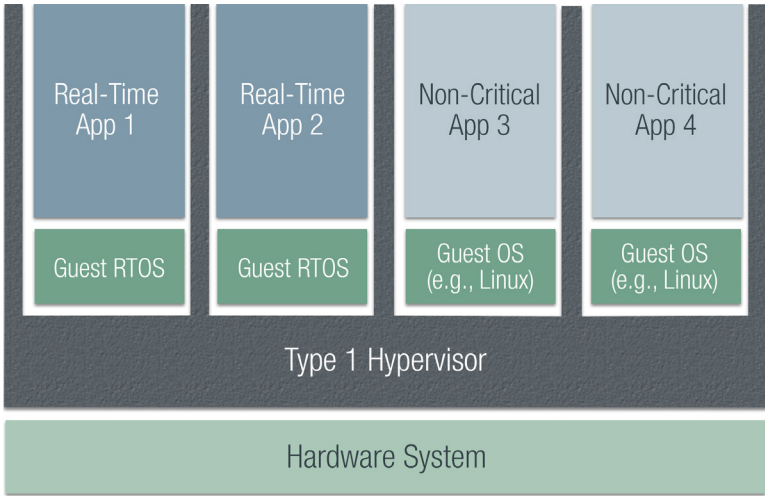
In that example, Linux applications run natively outside the hypervisor and at full performance. Similarly, in a real-time system, real-time applications can run directly on the real-time OS (RTOS), while only the non-real-time applications incur the extra latency of running on a Type 2 hypervisor *(Fig. 2)*.

Hybrid hypervisors aim to achieve the best of both approaches. For real-time embedded systems, INTEGRITY-178 tuMP from Green Hills Software uses a combination of Type 1 and Type 2 approaches that separates the isolation and virtualization mechanisms.

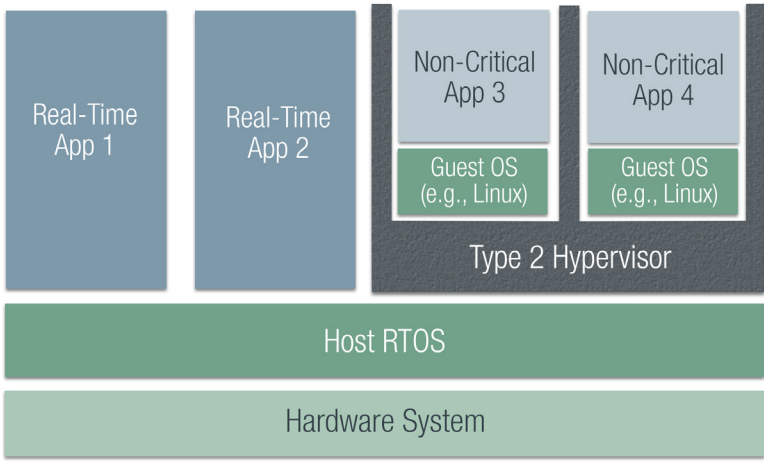
As with a Type 1 hypervisor, the fundamental separation mechanisms run in kernel mode on bare metal. For INTEGRITY-178 tuMP, that's a hardened separation microkernel, which isolates groups of non-kernel software into isolated partitions. However, the rest of the virtualization, mostly the hardware simulation, runs in user mode on top of the INTEGRITY-178 tuMP RTOS.

Each virtualization layer instance runs in user mode inside of an isolated partition with a VM running on top of it *(Fig. 3)*. This combination yields a security-critical real-time hypervisor with advantages in both security and performance.

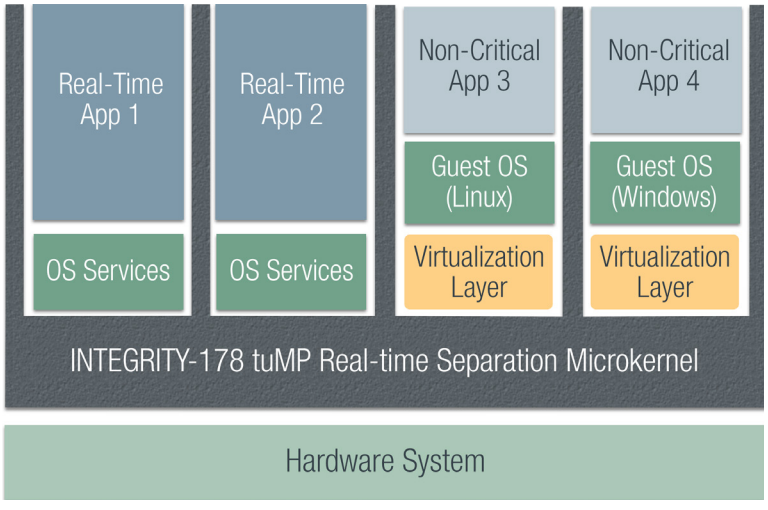
Containers virtualize the operating system running on the host system into isolated user-space instances. All of the containers running on the same system share the services of a single operating system kernel, but they can run different distributions (e.g., CentOS, RHL, Ubuntu). Though that makes the container lighter weight, it's less flexible than VMs that can have completely different guest operating systems.



1. Type 1 hypervisors run directly on the hardware, and every application gets additional latency from the virtualization layer.

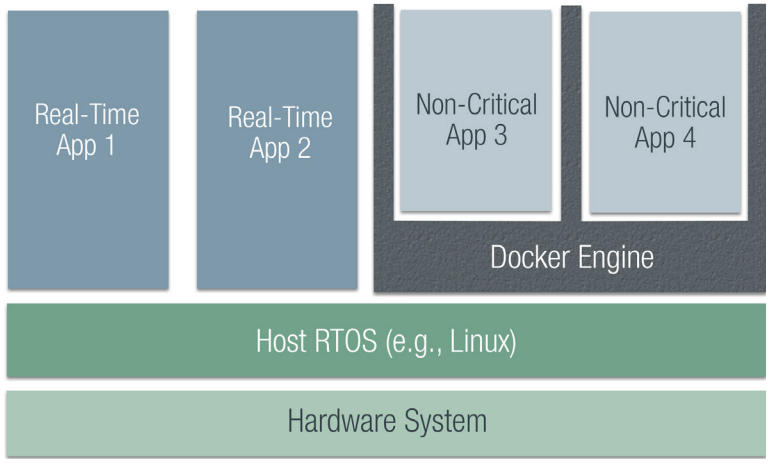


2. With Type 2 hypervisors, applications in VMs get extra latency from both the virtualization layer and the host OS, but applications requiring the highest performance can give up the isolation of a VM to run directly on the host OS.



3. A security-critical hybrid hypervisor provides isolation to all applications, and only applications that need a guest OS pay the virtualization performance penalty.

Although some operating systems, particularly Linux, have native support for containers, the most commonly used container runtime is Docker Engine *(Fig. 4)*. Because the container image contains all of the application dependencies (e.g., programming language runtime and standard libraries), applications in containers developed on one system can execute on a different production system. Container development and runtime tools, like Docker, and container orchestration platforms, such as Kubernetes, make it easy to update software applications and form the basis for a DevSecOps architecture. But containers depend on support from the OS and are susceptible to vulnerabilities in the OS kernel. If the OS is Linux, that's a very large number of vulnerabilities.



4. A container runtime such as Docker Engine runs on top a host OS, but applications can run outside of containers as well.

Size of the Trusted Computing Base

The security of a system depends on many components, including all of the hardware, the firmware, and each software component. The virtualization solution is just one element, but any software module running in privileged kernel mode has the highest security consequences.

One principle of software security is that the likelihood of a breach is proportional to the size of the attack surface. The attack surface is the sum of all entry points and vulnerabilities in a system that can be exploited to carry out an attack.

Although virtualization solutions aren't directly involved in limiting and securing entry points, they do play a role in limiting the spread of a breach by running application code in VMs or containers with some amount of isolation. Conversely, virtualization code adds to the total code that can contain vulnerabilities.

The effect of a successful attack depends on whether the breach occurred in the trusted computing base, or TCB, which is the set of hardware, firmware, and software that enforces the security policy. If part of the TCB is breached, the security properties of the whole system could be affected.

The general goal is to minimize the size of the TCB and the number of interfaces to it so that the TCB can be verified more easily. If the TCB can be made small enough, it's possible to evaluate its security formally. Formal verification uses mathematical proofs to show correct operation and is the strictest form of evaluation.

To achieve the smallest TCB, the software running in kernel mode should implement only the four fundamental security policies required to support higher security functionality running in user mode. Those security policies are data isolation, control of information flow, resource sanitization, and fault isolation.

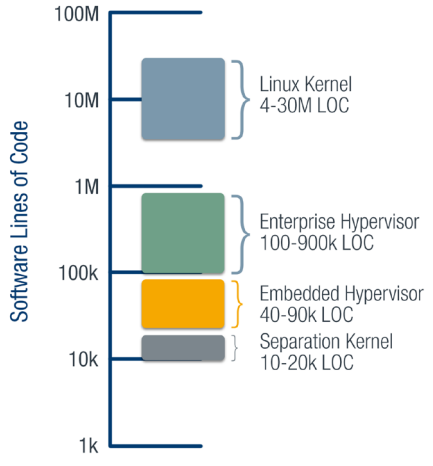
The goal of reducing the software TCB to just the functionality required for those four foundational security policies led to the concept of a separation kernel. A separation kernel divides memory into partitions using a hardware-based memory-management unit (MMU) and allows only carefully controlled communications between non-kernel partitions. Although almost all kernels attempt to enforce isolation by leveraging hardware functionality, such as the MMU and input/output MMU (IOMMU), leveraging hardware resources alone doesn't guarantee isolation, let alone cover the other security requirements.

Looking at the TCB of different virtualization solutions, the TCB of a Type 1 hypervisor is generally smaller than an entire OS but larger than a pure separation kernel. The problem is that software running in kernel mode for a typical hypervisor provides not only the fundamental isolation required, but also full virtualization, including emulation of hardware. The amount of code required for that emulation can be huge, dramatically increasing the TCB.

There's no security reason why the hardware emulation must be included in the kernel, but Type 1 hypervisors include it anyway, mainly to improve performance. Some hypervisors designed for embedded systems attempt to mitigate this by reducing functionality to reduce code size.

The TCB of a Type 2 hypervisor is really just the TCB of the host OS, since a Type 2 hypervisor runs in user space. If that's a general-purpose OS, then the TCB is likely larger than a Type 1 hypervisor.

However, if the OS is designed for security like a separation kernel, then the TCB in a Type 2 system can be smaller than a Type 1 hypervisor. Similarly, a hybrid hypervisor built on a separation kernel has a TCB the size of the separation kernel. Because a separation kernel is designed explicitly to minimize the TCB, it has the smallest TCB of any virtualization solution.



5. The size of the TCB can range from under 20K lines of code for a hybrid hypervisor based on a separation kernel to more than 20M lines of code for containers running on Linux and Docker.

For containers running on top of Linux, the TCB is an order of magnitude larger than a Type 1 hypervisor TCB. Looking at just the amount of unique code used for virtualization, Linux containers utilize the entire syscall interface, which is about 10X larger than the typical hypercall interface of a hypervisor.

The full TCB includes all parts of Linux running in kernel mode. The 5.11 release (2021) of the Linux kernel had more than 30 million lines of code, with about 4 million lines of code for the core functions and the rest as drivers *(Fig. 5)*. Docker can add to the TCB as well because the Docker daemon usually runs as root.

Critical Vulnerabilities Uncovered

The Common Vulnerabilities and Exposures (CVE) database lists publicly known information-security vulnerabilities and exposures. A vulnerability is a weakness in the code that, when exploited, results in a negative impact to confidentiality, integrity, or availability. The number of vulnerabilities reported has been increasing each year, with the National Vulnerability Database (NVD) maintained by NIST showing over 20,000 reported in 2021.

Given the numerous vulnerabilities continually being found, relying on patches can't be the primary mitigation. Many vulnerabilities are revealed to have been in the code for multiple years and multiple generations of software. Even after a vulnerability is discovered, it takes time to create a patch and more time to get the patch distributed and installed.

Many embedded systems aren't easily updated, and the end user may not know all of the software components in the system, let alone that a patch should be applied. Even if installed in a timely fashion, some patches do not fix the entire vulnerability. "Spectre" is an example of a vulnerability that hasn't been fully mitigated by a patch, even though it was disclosed in 2018.

Looking at virtualization solutions, container-based systems have the largest number of CVEs because the containers depend on the underlying OS. Containers generally run on the Linux kernel, which has a very sizable code base with over 3,400 vulnerabilities found so far.

More vulnerabilities are continually being discovered, with an average of one every other day in 2021. Of those, half had a severity rating of High or Critical (e.g., privilege escalation, arbitrary code execution). In addition, systems using containers typically add a container runtime engine like Docker, which runs as root and introduces additional vulnerabilities and configuration security challenges.

Type 1 hypervisors also have a significant number of CVEs. For example, Xen has over 600 vulnerabilities in the NVD, with nearly 20 CVEs reported in 2021. Hypervisor vulnerabilities are many and varied, such as VM escape, hyperjacking, denial-of-service (DoS), and side-channel attacks. Some of these vulnerabilities can be very serious because once the hypervisor security is breached, attackers can access all VMs, all applications, and all of the application data inside of them.

Hybrid hypervisors based on a separation kernel fare much better. For example, neither the INTEGRITY-178 tuMP multicore separation kernel nor the INTEGRITY-178 single-core separation kernel has had any security vulnerabilities reported against it since it was first deployed in 2002.

Security Certifications

Certification to government-defined security standards provides strong evidence of the level of security assurance and functionality of a product. Two widely used security certifications are the Common Criteria and the Risk Management Framework.

Common Criteria

The security robustness of computer hardware and software platforms can be specified by evaluation to the "Common Criteria for Information Technology Security Evaluation" (ISO/IEC 15408). Common Criteria targets of evaluation (TOE) are typically evaluated against a government-defined protection profile that includes both functional and assurance requirements.

Evaluations can be done to different levels of depth and rigor, called Evaluation Assurance Levels (EAL), with EAL1 being the least rigorous and EAL7 being the most rigorous. Security assurance requirements and security functional requirements are grouped together into a protection profile tailored to a particular type of product.

Starting with EAL2, each of those EALs requires a vulnerability analysis to a specific level. EAL5 is the first level that protects against attackers of "moderate" potential, and it's only at EAL6 that the resistance to penetration attacks rises to "high" attack potential.

The Common Criteria goes on to say that it's not economically feasible to retrofit security into an existing product line and achieve higher than EAL4. As a result, products that didn't design security in from the beginning can't provide resistance to attackers with even a moderate attack potential.

For example, consider the Protection Profile for General Purpose Operating Systems (GPOS-PP). The GPOS-PP wasn't designed for a specific EAL, but it's based mainly on EAL2 requirements. Additional protection profiles were defined to enhance certain aspects of OS system security. Examples include the Controlled Access Protection Profile (CAPP) and the Labeled Security Protection Profile (LSPP). The assurance requirements for both of those protection profiles are specified at EAL3.

The only government-defined protection profile ever designed for High Robustness or EAL6 and above that applies to operating systems or hypervisors is the "U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness" (SKPP). The NSA-defined SKPP was published in 2007. Although U.S. certification of commercial products to the SKPP ceased in 2011, government programs of record that need high robustness continue to require meeting the security objectives in the SKPP.

Considering Common Criteria applied to vitalization environments, some versions of Linux that include Linux container functionality add in the SELinux security options and have been certified to EAL4+ with CAPP and LSPP. As mentioned above, SELinux has many vulnerabilities discovered each year. That's not surprising given that the CAPP and LSPP explicitly state those protection profiles are only "appropriate for an assumed non-hostile and well-managed user community requiring protection against threats of inadvertent or casual attempts to breach the system security." It's only starting with EAL5 that hostile threats are addressed to any degree.

Very few Type 1 hypervisors have been certified to the Common Criteria. Only VMware vSphere 5.0 has been certified, and that was to EAL4+ but not to a government-defined protection profile.

The INTEGRITY-178 separation kernel and RTOS have been certified multiple times to EAL6+ and "High Robustness" using the SKPP. No other commercial operating system or hypervisor has been certified to EAL6+ or High Robustness.

As part of certification to the SKPP, INTEGRITY-178 underwent independent vulnerability analysis and penetration testing by the NSA to demonstrate that it doesn't allow hostile and well-funded attackers with high attack potential to violate the security policies. Extending that security design to multicore processors, INTEGRITY-178 tuMP continues to meet the SKPP's rigorous set of functional and assurance requirements. The hybrid hypervisor solution builds upon that secure kernel by adding virtualization in an isolated user-space partition.

Risk Management Framework

The Risk Management Framework (RMF) is a U.S. federal government policy and set of standards developed by the National Institute of Standards and Technology (NIST) for the assessment and authorization of mission systems. The overview document is NIST SP 800-37, "Risk Management Framework for Information Systems and Organizations: A System Life Cycle Approach for Security and Privacy." That document defines six steps: Categorize information systems, Select security controls, Implement, Assess, Authorize, and Monitor.

One of the most challenging steps is selecting the security controls. NIST SP 800-53 "Security and privacy controls for Federal Information Systems and Organizations" lists more than 800 security controls to select from, many of which don't apply to embedded systems. It's up to the program to go through all of the RMF controls and determine which apply.

Note that the RMF is used to certify whole systems, including hardware, firmware, and software. An individual component like virtualization can only offer help in meeting a subset of requirements. For example, an operating system or hypervisor that provides secure audit capability can contribute to the RMF audit capability (AU family of controls).

RMF controls are applied to functions or subsystems and assessed at an impact level (low, moderate, or high) for each of confidentiality, integrity, and availability. For example, a mission computer for a helicopter could be assessed as requiring high confidentiality, high integrity, and moderate availability.

Many risk-management and data-management platforms can help automate some of the burdens of meeting the selected security controls. Looking more specifically at virtualization, some hypervisors claim to aid in meeting RMF controls without providing any specifics. Docker provides a broad set of guidance on which security controls apply to its Docker Enterprise Edition.

The INTEGRITY-178 tuMP hybrid hypervisor doesn't offer direct guidance on meeting the RMF. Instead, it provides detailed information on meeting the much more rigorous security objectives defined in the SKPP. As a result, INTEGRITY-178 tuMP provides security above and beyond the RMF.

How to Use Containers More Securely

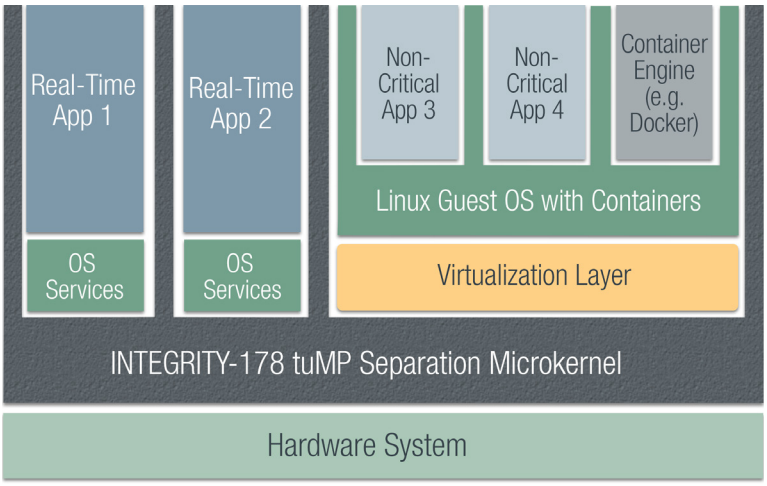
For organizations that want to use containers to implement a DevSecOps environment, the challenge is finding a way to secure the containers and the underlying OS. Some DevSecOps environments attempt to improve container security by using "hardened containers," which rely primarily on scanning for known vulnerabilities and compliance to security policy.

However, just as you can't "inspect in" quality, no amount of scanning will change the inherent security of a set of software. Beyond that, hardened containers don't attempt

to address the most significant security issue, namely the vulnerabilities in the underlying OS.

When Linux is the underlying OS, the first option is to beef up the security posture of the Linux kernel using security patches such as SELinux. SELinux enforces mandatory access control policies that restrict user programs and system services to the minimum privilege necessary.

SELinux is effective in preventing the exploitation of some access control vulnerabilities, such as the RunC vulnerability (CVE-2019-5736) that permitted containers in the Docker solution to gain root privileges on the host machine. However, SELinux is only a partial solution because 1) SELinux doesn't protect against other types of vulnerabilities in the Linux kernel or the kernel's security configuration, and 2) SELinux has its own vulnerabilities, with 15 discovered in 2021 alone.



6. For maximum security, containers can run in a secure partition of a separation microkernel.

A second option is to run containers inside a VM, which can increase security to some degree. A VM provides a higher level of isolation, while the container provides convenient packaging and a delivery mechanism for applications to enhance DevOps.

Running inside a VM removes Linux and the container runtime engine from the TCB, but there's a performance penalty for running OS virtualization on top of hardware virtualization. If the VM hosting the containers is running on a Type 1 hypervisor, you have traded a massive TCB for a large TCB and are still susceptible to hypervisor-based attacks.

A more secure solution is to run containers in a partition of a hardened separation microkernel, such as INTEGRITY-178 tuMP *(Fig. 6)*. Other partitions can hold applications at different, mixed security levels. Running the container system in a partition protects the containers from applications running in other partitions and vice versa.

Any applications requiring low latency, though, would be better to run in a separate partition without either the container-level virtualization or the VM-level virtualization. Those low-latency applications can then run directly on a bare-metal real-time operating system built on the hardened separation kernel.

Conclusion

Although all forms of virtualization add some level of security compared to a general-purpose operating system, the level of security varies greatly. Containers provide a basic level of isolation for applications and processes, but they rely on an operating system and a container engine that have a large attack surface and many new vulnerabilities found each year. Type 1 hypervisors for embedded systems have a smaller attack surface than Linux containers. However, they're generally still too large to evaluate fully for security vulnerabilities.

By separating the isolation and virtualization requirements, the INTEGRITY-178 tuMP separation microkernel creates a security-critical real-time hypervisor to maximize security and optimize system performance. The separation kernel presents the smallest attack surface and can be fully evaluated for isolation, security properties, and correctness.

The microkernel's virtualization layer lies outside the TCB and is used only for the applications that need it, leaving the high-assurance applications to run directly on the RTOS. INTEGRITY-178 tuMP meets the requirements of the NSA-defined Separation Kernel Protection Profile for High Robustness.

A secure system can employ containers as a software packaging and delivery platform for DevSecOps if another solution provides the isolation. Linux containers and Docker can run on top of the virtualization layer of the INTEGRITY-178 tuMP RTOS while the separation microkernel provides the certified isolation.

RICHARD JAENICKE is the Director of Marketing for safety- and security-critical products at Green Hills Software. Those products focus on applications that require certification or conformance to DO-178B/C, ED-12B/C, CAST-32A, ARINC 653, EAL6+/SKKP, or the FACE™ Technical Standard. Prior to joining Green Hills, he was at Mercury Systems marketing avionics software and hardware solutions as well as signal-processing systems. Mr. Jaenicke holds an MS in Computer Systems Engineering and has over 25 years of experience working with embedded software and systems.