Secure Virtualization for Real-Time Military Embedded Systems

Contents

Executive Summary 1

Introduction 1

Overview of Virtualization Types 2

- Hardware Virtualization 2
- Operating System Virtualization 3

Security Evaluation Criteria 4

Minimizing the Trusted Computing Base 4

Number of CVEs 6

- Example Attacks on a Hypervisor 6
- Example Vulnerabilities of Containers 6

Control of Information Flows 7

Security Certifications 7

- Common Criteria 7
- Risk Management Framework 8
- Raise the Bar 9

Additional Security Criteria 9

- No Dynamic Resource Allocation 9
- Handling Mixed Assurance Levels 9

INTEGRITY-178 tuMP Separation Kernel 10

- INTEGRITY-178 tuMP Virtualization 10
- INTEGRITY-178 tuMP for Cross Domain Solutions 10

Relationship between Safety and Security 10

Using Containers Securely 11

Conclusion 11

References 12

Executive Summary

Whether used for consolidating workloads, running legacy applications, or supporting DevOps, the choice of a virtualization platform has major implications on security. Although most virtualization platforms are believed to increase the security posture compared to a general-purpose operating system, almost every virtualization platform comes with its own set of vulnerabilities.

Hypervisors are assumed to be secure because they rely on hardware constructs such as the memory management unit (MMU) and IOMMU to provide separation. However, full virtualization requires many more capabilities than those provided by the hardware, and the code base is large enough for popular hypervisors to have dozens of new vulnerabilities discovered each year. Containers, which provide OS-level virtualization, have even more vulnerabilities due to the huge code base of the underlying Linux kernel.

A more secure virtualization solution is built upon a separation kernel, which contains the minimum functionality to ensure isolation and control of information flow among virtual machines running in secure partitions. All other operating system functions are moved out of the kernel into user space. Decomposing hypervisor functionality into isolation and virtualization enables a further reduction in the amount of code running in the kernel by moving the virtualization function into user space as well. A prime example of this is Green Hills' INTEGRITY®-178 tuMP™ security-critical real-time hypervisor, which can achieve EAL 6+ and NSA-defined "high robustness" as well as meet the "Raise the Bar" set of security standards for cross domain solutions. When applied to real-time systems, it can also provide higher performance than a Type 1 hypervisor.

Introduction

Virtualization is used in many real-time and embedded systems to address a variety of goals, such as supporting heterogeneous operating systems, consolidating workloads, enabling DevOps, and increasing security. Heterogeneous operating systems can be used to support legacy applications or combine real-time and non-real-time environments. Workload consolidation is easier with virtualization because applications appear to be running

on their own set of hardware, significantly reducing porting costs. For DevOps, different types of virtualization can be used to move applications between servers for workload balancing and fault handling or to package an application with all of its dependencies for easier deployment and upgrades. Virtualization can also be used to increase the security of general-purpose computing environments by isolating applications from each other when incorporating software and IP from different companies or as part of a zero-trust environment. But how secure is the virtualization technology itself, particularly when applied to military systems?

There are different types of virtualization, and each has a different security profile. Hypervisors, for example, are assumed to be secure because they rely on hardware constructs such as the memory management unit (MMU) and IOMMU to provide separation and because hypervisors generally have a relatively small codebase and limited interaction external to the system. In reality, full virtualization requires many more capabilities than those provided by the hardware, and the code base is large enough for popular hypervisors to have dozens of new vulnerabilities discovered each year. Containers, which provide OS-level virtualization, have even more vulnerabilities due to the even larger code base of the underlying Linux kernel.

Because applications have no awareness of running in a virtualized system, any compromises to the virtualization system are very hard to detect. When a hypervisor runs on bare metal, it has the unique position as the software layer that all others depend upon to provide the base security. If the hypervisor security is breached, the attackers have access to all the VMs, all the applications, and all application data inside of them. This makes the hypervisor a prime target for an attack. Virtualization vulnerabilities are many and varied, such as VM escape, hyperjacking, denial-of-service (DoS), and side-channel attacks. Each of these is explained later in this paper.

Why do virtualization solutions have so many vulnerabilities? The source of many vulnerabilities originates in the tradeoff between performance, security, and cost. Higher performance virtualization requires more code of high complexity running in the kernel, which increases the trusted computing base (TCB). Real-time and embedded systems add more dimensions of performance to trade off, including restricted memory size and a requirement for determinism. In most virtualization systems, security usually takes a back seat to performance and cost because basic security is believed to be good enough.

The rest of this paper reviews the different types of virtualization, lists the criteria for evaluating the security posture, and then compares how each type of virtualization performs against the evaluation criteria.

Overview of Virtualization Types

Understanding the different types of virtualization can help with understanding the security vulnerabilities. At the top level, there is hardware virtualization and software virtualization. Hardware virtualization creates virtual machines (VMs) that simulate the physical hardware and run their own guest operating system (OS) isolated from other VMs. Software virtualization most commonly happens at the OS level, where the operating system kernel enables multiple user-space instances that are isolated from each other. Each of those user-space instances, also called containers, runs on top of the host OS kernel, although the virtualized OS may be a different distribution. Originally, containers were created as a lightweight alternative to VMs, but the most common use of containers is part of a DevOps architecture.

Hardware Virtualization

In hardware virtualization, the VMs are controlled by a virtual machine monitor (VMM), more commonly referred to as a hypervisor. Each VM simulates the same underlying hardware but can have a different guest OS. The hypervisor allocates each VM a separate address space, which the hypervisor enforces using the CPU's memory management unit (MMU). All 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, while Type 2 hypervisors run on top of an operating system. 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 and provide more determinism than a server or workstation hypervisor. Improving determinism is particularly important for hypervisors used for real-time systems because a Type 1 hypervisor is the source of extra latency for every application running in such a system. That is a critical issue for latency-sensitive real-time applications (Figure 1).

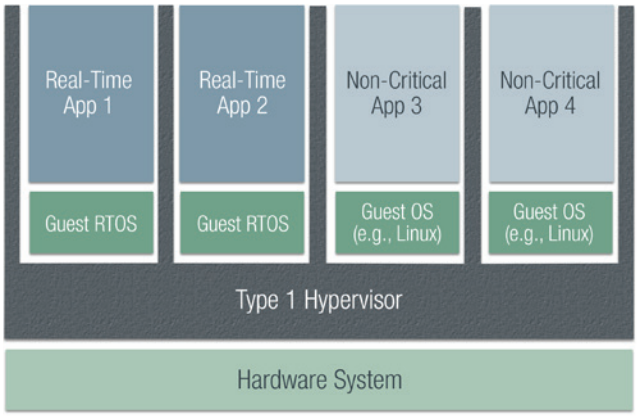


Figure 1: Type 1 hypervisor with some real-time applications.

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 have to 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 of 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 (Figure 2).

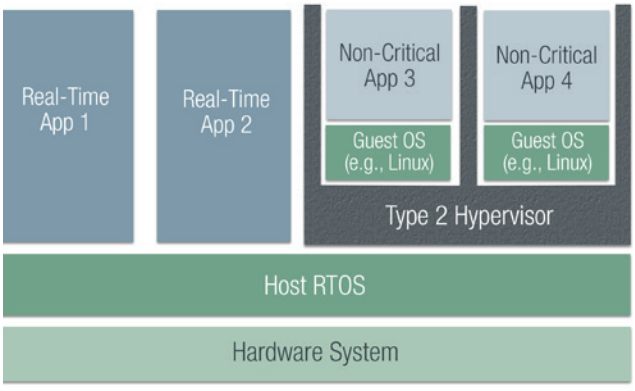


Figure 2: With a Type 2 hypervisor, real-time applications can avoid the virtualization performance penalty but get no isolation.

INTEGRITY-178 tuMP from Green Hills Software uniquely uses a combination of a Type 1 and a Type 2 approach 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 is a hardened separation microkernel. 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 (Figure 3). This combination yields a security-critical real-time hypervisor with advantages in both security and performance.

INTEGRITY-178 tuMP takes isolation to a higher level than a Type 1 hypervisor in that it also includes time partitioning and multicore interference mitigation. Time partitioning enables scheduling processes at different security levels in different time slots to reduce opportunities for lower security processes to gain access to higher security information. Multicore interference mitigation aids in preventing delays in access to shared resources from being used as covert timing channels between processes on different cores.

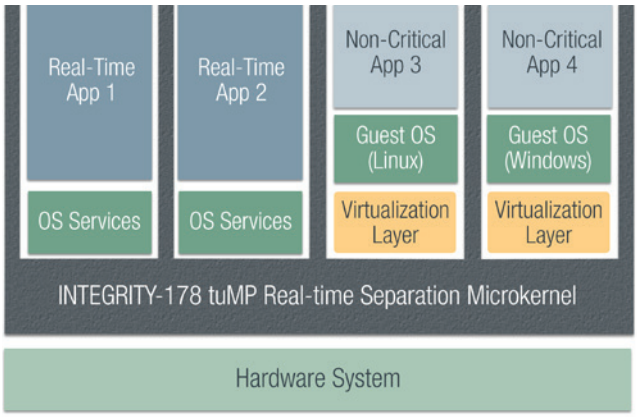


Figure 3: Security-critical real-time hypervisor with the isolation provided by a separation microkernel and virtualization provided in user space where needed.

Operating System Virtualization

OS virtualization virtualizes the operating system running on the host system into isolated user-space instances. Examples include Linux containers, FreeBSD jails, and AIX Workload Partitions. All of the containers running on the same system share the services of a single operating system kernel (Figure 4), making them less flexible than VMs that can have different guest operating systems.

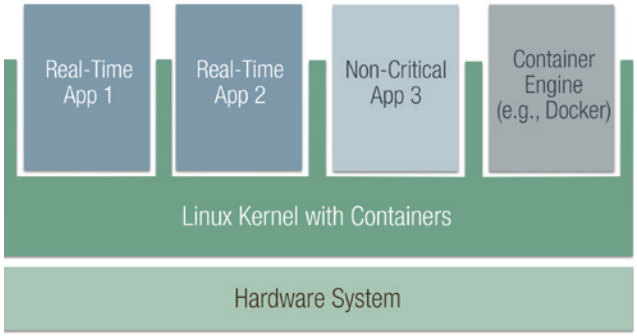


Figure 4: Container architecture

Linux containers are supported directly by the Linux kernel, based mainly on the mechanisms of Linux namespaces and cgroups (short for "control groups"). Linux namespaces divide and assign kernel resources, while cgroups enable fine-grained control and enforcement of limits on the use of those resources.

Containers gained popularity with the introduction of Docker, which uses containers as a software packaging and delivery platform. Because the container image contains all the application dependencies (e.g., version of programming language and standard library), 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 DevOps architecture. But containers depend on support from the OS and are susceptible to vulnerabilities in the OS kernel. If that OS is Linux, that is a very large number of vulnerabilities.

Security Evaluation Criteria

The security of a system depends on many components, including all 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.

Some of the most critical factors in evaluating the security posture of a virtualization solution are:

1. The size of the Trusted Computing Base
2. The number CVEs open
3. How information flows are controlled
4. The types of security certifications achieved

The most secure solutions also address static vs. dynamic resource allocation and how to handle applications at mixed assurance levels running concurrently. An additional consideration is the performance penalty for that type of virtualization.

Minimizing the Trusted Computing Base

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 are not directly involved in limiting and securing entry points, virtualization solutions 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 (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 is possible to evaluate its security formally. Formal verification uses mathematical proofs to show correct operation and is the strictest form of evaluation. As an example, the INTEGRITY-178 kernel from Green Hills Software has been formally verified.

Minimizing the TCB requires moving many non-critical services out of the TCB, which in turn requires both the ability to isolate those services and provide secure communication between trusted and non-trusted components. Examples of non-critical services include networking stacks, file systems, hardware emulation, and non-security-critical device drivers. In general, the software portion of the TCB is what is running in privileged kernel mode. Therefore, minimizing the TCB requires minimizing the amount of kernel code.

To achieve the smallest TCB, the kernel should implement only the four fundamental security policies required to support higher functionality running in user mode. Those security policies are data isolation, control of information flow, resource sanitization, and fault isolation. (See the sidebar for more detail.) The kernel leverages hardware functionality, such as the memory management unit (MMU) and Input/Output MMU (IOMMU), to help enforce isolation, but that does not cover all the security requirements.

The four fundamental security policies in a minimized TCB, which form the basis of the INTEGRITY-178 tuMP separation kernel:

- **Data Isolation**: Information in a partition is accessible only by that partition, and private data remains private.
- **Control of Information Flow**: Information flow from one partition to another originates only from authorized sources, is delivered only to intended recipients, and the source of information is authenticated to the recipient.
- **Resource Sanitization**: The microprocessor itself will not leak information from one partition to another as it switches from partition to partition
- **Fault Isolation**: A failure in one partition will not cascade to any other partition. Failures are detected, contained, and recovered from locally.

The goal of reducing the software TCB to just the functionality required for 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. Dramatically reducing the size of the kernel code enables dramatically increasing the rigor of scrutiny. This makes it possible to certify a separation kernel to the highest levels of security functionality, such as the NSA-defined Separation Kernel Protection Profile (SKPP) for environments requiring "high robustness," as well as the highest levels of security assurance, such as Common Criteria EAL 6 and above.

For highly secure systems, minimizing the TCB is not the end goal but only a means to ease verification. For systems requiring a high level of security, the end goal is certification as a result of proving the applicable security assurance requirements. The duration, schedule risk, and cost of evaluating, certifying, and deploying software increase non-linearly with the size of the code.[1]

TCB Applied to Virtualization

The virtualization solution with the smallest TCB is the security-critical real-time hypervisor used by the Green Hills' INTEGRITY-178 tuMP RTOS. That is because the hardware emulation part of the virtualization is outside of the TCB. Only the isolation, control flow, and resource sanitization functions are part of the TCB that runs in kernel mode. The resulting code is small enough to be scrutinized, evaluated for security, and even formally proven to be correct.

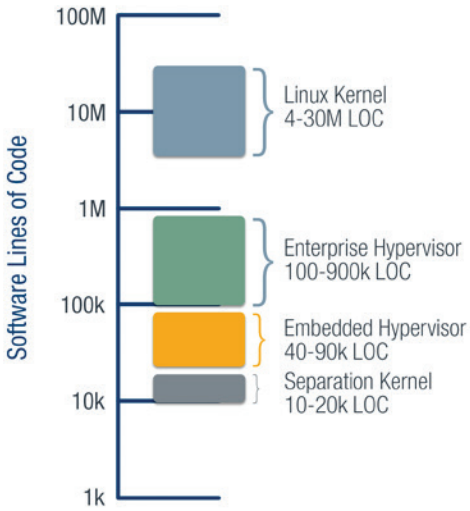
Although the code for the INTEGRITY-178 tuMP virtualization layer is slightly larger than the microkernel, it is not nearly as large as a Type 1 hypervisor. Because it runs in user space and is not part of the TCB, any vulnerabilities in the virtualization code are isolated to a single VM and do not affect other parts of the system, including other VMs, other applications running outside of VMs in their own partition, or the separation microkernel.

The TCB of a Type 1 hypervisor is generally smaller than a full OS but larger than a pure separation kernel. The problem is that 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 is no security reason why the hardware emulation must be included in the kernel, but Type 1 hypervisors include it anyway, mainly to improve performance.

A significant amount of virtualization code is aimed at reducing the performance overhead. Almost every call from the guest OS to the kernel needs to be trapped, examined, and determined if the guest OS is permitted that access. For a virtualization solution to be efficient, it needs to virtualize sequences of instructions instead of single instructions. Such look-ahead capability is just one example of increasing the already large code base of a hypervisor in pursuit of minimizing the virtualization performance penalty.

For a system leveraging Linux containers and running Linux as the host OS, the TCB is an order of magnitude larger than even a hypervisor TCB. Looking at just the amount of unique code used for virtualization, Linux containers utilize the entire syscall interface, which is about ten times larger than the typical hypercall interface of a hypervisor.[2]

The full TCB also includes all the 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.



Number of CVEs

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.

Neither the INTEGRITY-178 tuMP multicore separation kernel nor the INTEGRITY-178 single-core separation kernel has had any security vulnerabilities reported since it was first fielded in 2002.

Given such a large number of vulnerabilities continually being found, relying on patches can't be the primary mitigation.

Example Attacks on a Hypervisor

Hypervisors have a complex code base with a large number of entry points and complicated interconnections, all of which encourages a growing amount of attack vectors. A compromised hypervisor can allow the hacker to gain access to each virtual machine on the system, enabling manipulation and exfiltration of programs and data. Some example types of hypervisor attacks:

VM Escape – A guest OS or one of its applications escapes from the VM encapsulation, allowing it to interact with other VMs, the hypervisor, and possibly the host machine. Example CVE-2018-6981: Multiple versions of VMware ESXi contain uninitialized stack memory usage in the vmxnet3 virtual network adapter that may allow a guest to execute code on the host.

Hyperjacking – Taking control of the hypervisor and either manipulating it or replacing it with an unauthorized one.

Denial-of-Service – A guest OS or its applications cause the hypervisor to crash or otherwise become unresponsive. Example CVE-2019-18423: Multiple versions of Xen allow guest OS users on Arm processors to crash Xen via a XENMEM_add_to_physmap hypercall.

Cross-VM Side-Channels – Accessing information in another VM without attacking the VM or hypervisor. Example: Transferring information through VM memory that was reclaimed by the hypervisor and allocated to another VM via a feature known as memory ballooning.[3]

Many vulnerabilities are discovered 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 are not easily updated, and the end user may not know all 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 has not been fully mitigated by a patch, even though it was disclosed in 2018.

CVEs in Virtualization

Linux containers depend 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). Beyond the Linux kernel, systems using containers typically add a container runtime

Example Vulnerabilities of Containers

Containers share some attack vectors with hypervisors, such as container escape, but also have a broader set of configuration issues due to the size and complexity of the underlying Linux kernel. Two vulnerabilities from early 2022 provide good examples of vulnerabilities in containers. The first directly exploits a vulnerability in the Linux kernel, while the second depends on a configuration issue common in a DevOps installation with Kubernetes.

CVE-2022-0492: A vulnerability in the "control groups" feature of the Linux kernel allows attackers to escape containers, escalate privileges, and execute arbitrary commands on a host machine.

CVE-2022-0185: A vulnerability in the Linux kernel provides an opportunity for an attacker with access to a system as an unprivileged user to escalate those rights to root within a Kubernetes cluster. To do this, the attacker must have a specific Linux capability, CAP_SYS_ADMIN, which reduces the risk of breakout in some container cases. But, in most Kubernetes clusters, that capability is enabled.

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.

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.

Control of Information Flows

Control of information flow is one of the four fundamental security policies but is often overlooked compared to vulnerabilities. The control centers on the authorization of an application to provide or receive certain types of information to or from a different application or data repository.

Some different types of authorization include role-based access control (RBAC) and mandatory access control (MAC). Embedded systems often have a predetermined set of applications so that the access permissions can be defined in a static configuration file.

Control of Information Flow Applied to Virtualization

Although many Type 1 hypervisors have some form of mandatory access control available, most default to not including that capability. For example, Xen Security Modules (XSM) implements a type of mandatory access control using a security architecture called FLASK, which is the same architecture used in SELinux. However, XSM is not included by default, and it has a handful of its own vulnerabilities reported each year.

Linux systems can add some level of control over information flow by including SELinux. SELinux enforces mandatory access control policies that restrict user programs and system services to the minimum privilege necessary. However, SELinux is only a partial solution because 1) SELinux does not 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.

The Green Hills INTEGRITY-178 tuMP separation microkernel defaults to no information flow between application partitions. All communication enabled between partitions is limited to authorized sources and destinations in a specified mode, such as read-only or read/write. Those information flows are statically defined and loaded at boot time. Similarly, all resources required by a partition are statically defined and loaded in a configuration file. (See the section below on Additional Security Criteria for the issues with dynamic resource allocation.) Meeting the objectives of the SKPP and "high robustness" guarantees enforcement of that controlled information flow.

Security Certifications

Certification to government-defined security standards provides strong evidence of the level of security assurance and functionality of a product. Three important security certifications are Common Criteria, Risk Management Framework, and the Raise the Bars set of standards for cross domain solutions.

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). Typically, Common Criteria targets of evaluation (TOE) are 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 EAL 1 being the least rigorous and EAL 7 being the most rigorous.

Starting with EAL 2, each of those EALs requires a vulnerability analysis to a specific level. EAL 5 is the first level that protects against attackers of "moderate" potential. It is only at EAL 6 that the resistance to penetration attacks raises to "high" attack potential. The Common Criteria[4] goes on to say that it is not economically feasible to retrofit security into an existing product line and achieve higher than EAL 4.That means that products that did not design security in from the beginning cannot provide resistance to attackers with even a moderate attack potential.

By definition, an EAL addresses only the assurance requirements and not the functional requirements of a security solution. This can lead to a situation where rigorous evaluation methods are applied to a very lax set of security functionality. Security functional requirements are addressed in protection profiles, and it is only when a high EAL is paired with a very demanding protection profile that the highest level of security is achieved.

In U.S. government-defined protection profiles, robustness is a metric that measures the TOE's ability to protect itself, its data, and its resources. Robustness levels are characterized as Basic, Medium, or High. The level of robustness required for a TOE is characterized as a function of the value of the data it protects and the threats identified for the environment in which it is deployed.

An example basic protection profile is the Protection Profile for General Purpose Operating Systems (GPOS-PP). The GPOS-PP was not designed for a specific EAL but is based mainly on EAL 2 requirements. Additional protection profiles were defined to enhance certain aspects of OS system security to a medium level of robustness. 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 EAL 3. Some OS suppliers choose to certify at EAL 4, but that does not increase the security level because it does not change the security functional requirements specified in the protection profile.

The only government-defined protection profile ever designed for High Robustness or EAL 6 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, programs of record that need high robustness continue to require meeting the security objectives in the SKPP.

Common Criteria Applied to Virtualization

Some versions of Linux that include Linux container functionality add in the SELinux security options and have been certified to EAL 4+ with CAPP and LSPP. As mentioned above, SELinux has many vulnerabilities discovered each year. That is not surprising given that the CAPP and LSPP explicitly state that 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 is only starting with EAL 5 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. That was to EAL 4+ but not to a government-defined protection profile.

The INTEGRITY-178 separation kernel and RTOS have been certified multiple times to EAL 6+ and "High Robustness" using the SKPP. No other commercial operating system or hypervisor has been certified to EAL 6+ 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 does not 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.

Meeting the objectives in the SKPP demonstrates that INTEGRITY-178 and INTEGRITY-178 tuMP support information flow control, resource isolation, trusted initialization, trusted delivery, trusted recovery, and audit capabilities. As stated in the SKPP, "when integrated within a high-assurance security system architecture, the SKPP separation mechanisms are appropriate to support critical security policies for the Department of Defense (DoD), Intelligence Community, the Department of Homeland Security (DHS), [and] Federal Aviation Administration (FAA)."

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 key document for RMF is NIST SP 800-53: Security and privacy controls for Federal Information Systems and Organizations. The RMF contains more than 800 controls to select from, many of which don't apply to embedded systems. It is up to the program office or federal agency to go through all the RMF controls and determine which apply. CNSSI-1253 provides guidance on selecting RMF controls for national security systems.

Note that the RMF is used to certify whole systems, including hardware, firmware, and software. An individual component like virtualization can only provide 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 are 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.

Some hypervisors claim to aid in meeting RMF controls without providing any specifics. The objectives defined in the SKPP are much more rigorous than the RMF controls that apply to operating systems and hypervisors. As a result, INTEGRITY-178 tuMP provides security above and beyond the RMF.

Raise the Bar

Raise the Bar (RTB) is a set of cybersecurity standards published by the National Cross Domain Strategy & Management Office (NCDSMO) in the National Security Agency (NSA). First published in 2018, the RTB standards are a set of security guidelines and requirements for cross domain solutions (CDS) deployed by the U.S. government to protect National Security Systems (NSS).

The RTB standards go well beyond the RMF controls that many government agencies implement. RTB standards ensure CDS systems are at low risk of failing, even under persistent attack. RTB is designed to combat evolving threats by continually improving CDS effectiveness. CDS systems are a critical part of a multi-domain operational environment where warfighters need data at their fingertips to make real-time decisions.

INTEGRITY-178 tuMP is the first and only RTOS to be part of a successful certification to Raise the Bar standards. The ability of the RTOS to meet this evolving set of standards is directly rooted in the INTEGRITY-178 pedigree of achieving the SKPP for “High Robustness.”

Additional Security Criteria

The most secure systems have additional security criteria, such as static vs. dynamic resource allocation and how to handle applications at mixed assurance levels running concurrently.

No Dynamic Resource Allocation

Dynamic allocation of memory and other resources can result in one application being denied resources because the resource pool has been exhausted by other applications. Dynamic allocation generally comes with dynamic reclamation of resources that are no longer needed. That can result in residual information being passed from one VM or container to another. Other applications can also see the change in the amount of available resources after allocation, which could form the basis of a covert channel.

Most hypervisors allow dynamic resource allocation, risking reduced availability if the resources become exhausted as well as the transfer of information either through residual information or covert channels.

Likewise, the dynamic agility of containers is one of its biggest advantages, but it is also a security weakness. That agility requires dynamic allocation and reuse of resources, which is much harder to secure than statically allocated resources.

With the INTEGRITY-178 tuMP separation kernel, all resources required by a partition are statically defined and loaded through a configuration file. That means there is no change of an application in one partition using resources from applications in another partition.

Handling Mixed Assurance Levels

Because modern CPUs have a significant amount of shared resources, it is not practical to try to prevent all possible yet unlikely information leaks between applications running concurrently. This is primarily a concern when there are applications of different security assurance levels. The mitigation is to reduce the risk to an acceptable level by only allowing applications at the same assurance level to run concurrently. Time partitioning (of the CPU) provides this capability by assigning processes at the same assurance level to their own unique time slots.

Neither Type 1 hypervisors nor container environments include time partitioning in their schedulers. That means there are no mechanisms for ensuring that processes at different security levels run at different times unless concurrent execution is disabled in the hypervisor or container OS. Some RTOSes have time partitioning, but running one of those inside a VM does not increase the security posture of the underlying hypervisor.

INTEGRITY-178 tuMP includes time partitioning in its high-assurance scheduler. The software architect manages the secure execution of partitions at different security levels by restricting concurrent execution to partitions at the same security level. Shared resources can be sanitized either after every time slot or after execution of the highest security level if the time slots are arranged in increasing security levels.

INTEGRITY-178 tuMP Separation Kernel

The INTEGRITY-178 tuMP safety and security-critical RTOS has a separation microkernel architecture, with OS services (including virtualization) implemented in user space instead of in the kernel. Designed from the beginning for both safety and security, INTEGRITY-178 tuMP provides the high level of data isolation, control of information flow, resource sanitization, and fault isolation required for a separation kernel of high robustness. Because the separation kernel is the only software running in privileged kernel mode, the four foundational security policies are non-bypassable, evaluateable, always invoked, and tamperproof (NEAT), providing the high assurance level needed to run systems with the highest level of security, including multi-level security (MLS) and cross domain solutions (CDS). Green Hills develops and maintains SKPP-compliant processes and life-cycle data for INTEGRITY-178 tuMP security customers. This security evidence can be used to demonstrate meeting any project-specific security objectives related to the operating system.

INTEGRITY-178 tuMP Virtualization

INTEGRITY-178 tuMP provides virtualization as a security-critical real-time hypervisor that combines the best attributes of Type 1 and Type 2 hypervisors. Like a Type 1 hypervisor, the INTEGRITY-178 tuMP separation microkernel runs directly on the physical hardware and isolates applications running in user space partitions. Like a Type 2 hypervisor, the virtualization layer with the hardware emulation runs in a user space partition on top of the INTEGRITY-178 tuMP RTOS and only runs in partitions that require a different OS. There can be multiple partitions running with a virtualization layer, and the separation microkernel enforces isolation between those different virtualization instances as well as isolation from applications not using virtualization (Figure 3). The virtualization layer in user space makes use of the fundamental separation mechanisms implemented in the separation microkernel, including the use of MMU and IOMMU. The result is a combined virtualization solution that is both more secure than a Type 1 hypervisor and also has higher system performance because the real-time applications run directly on the RTOS instead of on top of the virtualization layer.

INTEGRITY-178 tuMP for Cross Domain Solutions

Continuing to meet the objectives of the SKPP forms the basis for INTEGRITY-178 tuMP to be used for cross domain solutions. However, CDS applications are large, complex code bases that require support for a broad set of functionality from the OS.

Being a true RTOS, INTEGRITY-178 tuMP provides many more OS services than a Type 1 hypervisor. Most security-critical applications, such as cross domain guards, require OS functions like tasking services, semaphores, and message passing. When using a standard hypervisor-based solution, one would need to run a secure guest OS on top of a secure Type 1 hypervisor to provide that level of service while taking a performance penalty from the extra software layer.

INTEGRITY-178 and INTEGRITY-178 tuMP have been successfully deployed in numerous cross domain solutions. One of the most recent is the U.S. Navy Tactical Combat Training System Increment II (TCTS II) from Collins Aerospace. That system is completing successful certification to the "Raise the Bar" standards for cross domain solutions.[5] When used in a tactical setting like the TCTS II system, the RTB standards enable multi-domain operations across a connected battlespace by ensuring that a CDS is at low risk of failing, even when under persistent cyberattack. As a component of the TCTS II program, INTEGRITY-178 tuMP is the first and only RTOS to be part of a Raise the Bar certification.

Relationship between Safety and Security

Similarities exist between the concept of separation for security and the concept of partitioning for safety. Safety partitioning and security separation share the fundamental requirements of data integrity and availability. Security separation has the additional requirement of confidentiality, which includes verification that there are no unintended side effects. Although ensuring confidentiality, such as all potential read accesses to memory or devices are authorized, is not generally required for safety-critical systems, it can be perceived as a benefit due to the strengthening of arguments of independence.[6]

Given the overlapping and complementary relationship between safety and security, it is achievable for a single product to meet the high-assurance requirements for both safety and security. The INTEGRITY-178 tuMP RTOS meets the requirements for both DO-178C at DAL A and the SKPP at high robustness. Several deployed military systems make use of that capability by deploying classified applications and safety-critical applications in the same system, sometimes with virtualization included for legacy applications (Figure 5). INTEGRITY-178 tuMP includes multicore interference mitigation using shared-resource bandwidth allocation and monitoring (BAM), and INTEGRITY-178 tuMP is the only RTOS or hypervisor to be part of a multicore certification to DO-178C and CAST-32A multicore objectives, and that was to Level A.

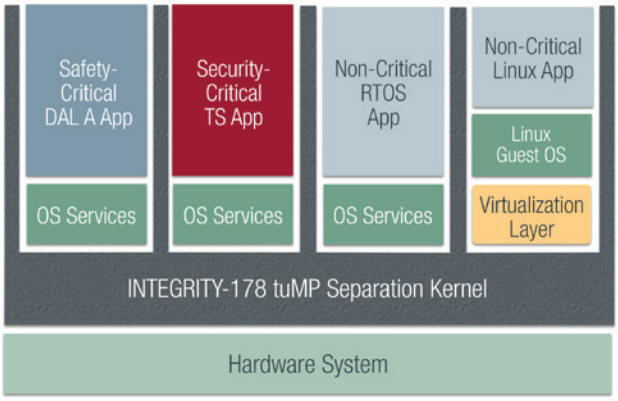


Figure 5: INTEGRITY-178 tuMP simultaneously hosting DAL A, Top Secret, and Linux applications.

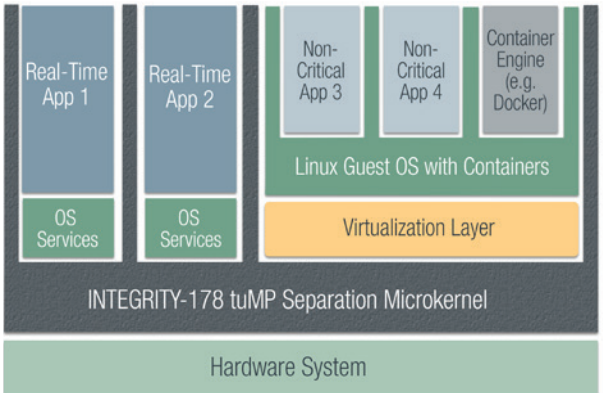


Figure 6: Linux containers running in a secure partition of a separation microkernel

Using Containers Securely

Given that Linux containers are at the core of most modern DevOps solutions, is there a way to use containers in a highly secure manner? Some DevOps 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 cannot "inspect in" quality, no amount of scanning is going to change the inherent security of a set of software. Hardened containers do not attempt to address the biggest security issue, which is the underlying Linux kernel.

Another approach is to run containers inside a VM, which can increase the security to some degree. The 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 is 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 huge TCB for a medium-sized TCB and still are 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 (Figure 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 vis-a-versa. Any applications requiring low latency, however, would be better to run outside of both the container-level virtualization and the VM-level virtualization directly on a bare-metal real-time operating system built upon 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 rely on an operating system and a container engine with a large attack surface with many new vulnerabilities found each year. Type 1 hypervisors for embedded systems have a smaller attack surface than Linux containers but generally are 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 virtualization layer lies outside of 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. INTEGRITY-178 tuMP has been used on a certification to NSA/NCDSMO's "Raise the Bar" set of standards for cross domain solutions.

A secure system can employ containers as a software packaging and delivery platform for DevOps if the isolation is provided by another solution. 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.

References

1. W. Mark Vanfleet, et al, “MILS: Architecture for High Assurance Embedded Computing,” CrossTalk, Aug-2005.
2. James Bottomley, Containers and Cloud Security, 20 June 2018. https://blog.hansenpartnership.com/containers-and-cloud-security/
3. Wang Z, Yang R, Fu X, Du X, Luo B (2016) “A shared memory based cross-VM side channel attacks in IaaS cloud.” 2016 IEEE conference on computer communications workshops (INFOCOM WKSHPS), pp 181–186. http://ieeexplore.ieee.org/document/7562068/
4. Common Criteria for Information Technology Security Evaluation, Version 3.1, revision 5, April 2017. Part 3: Security assurance components.
5. “Raise the Bar: Demanding Cybersecurity Excellence for Cross Domain Solutions in the attlespace,” Modern Battlespace, 3 March 2021. https://modernbattlespace.com/2021/03/03/raise-the-bar-demanding-cybersecurity-excellence-cross-domain-solutions-battlespace/
6. P. Huyck, “Safe and Secure Data Fusion – Use of MILS Multicore Architecture to Reduce Cyber Threats,” 2019 IEEE/AIAA 38th Digital Avionics Systems Conference (DASC), Sep-2019.



Corporate Headquarters

30 West Sola Street • Santa Barbara, CA 93101
ph: 805.965.6044 • fax: 805.965.6343 • email: info@ghs.com • www.ghs.com

European Headquarters

Fleming Business Centre • Leigh Road • Eastleigh • Hampshire S050 9PD • United Kingdom
ph: +44 (0)2380 649660 • fax: +44 (0)2380 649661 • email: info-emea@ghs.com

Safety & Security Critical Products

34125 US Hwy 19 North • Suite 100 • Palm Harbor, FL 34684
ph: 727.781.4909 • fax: 727.781.3915 • email: info-sscp@ghs.com

Green Hills, the Green Hills logo, INTEGRITY, and tuMP are trademarks or registered trademarks of Green Hills Software in the US and/or internationally. All other trademarks (registered or otherwise) are the property of their respective owners. © 2022 Green Hills Software. v0722