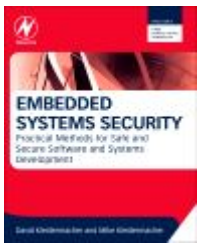
Embedded Systems Security - Part 3: Hypervisors and system virtualization

David Kleidermacher and Mike Kleidermacher - February 19, 2013



Editor's Note: Embedded Systems Security *aims for a comprehensive, systems view of security: hardware, platform software (such as operating systems and hypervisors), software development process, data protection protocols (both networking and storage), and cryptography. In this excerpt, the authors offer an in-depth look at the role of the operating system in secure embedded systems. In part 1, the authors offer an in-depth look at the role of the operating system in secure embedded systems. In part 2, the authors discuss how an OS provides access control for ensuring process security. In this installment, the authors examine the use of hypervisors in implementing system virtualization. In part 4, the authors review the security pitfalls and trends in embedded I/O virtualization.*

Adapted from "Embedded Systems Security" by David Kleidermacher and Mike Kleidermacher (Newnes)

2.6 Hypervisors and System Virtualization

The operating system has long played a critical role in embedded systems. A prime historical purpose of the operating system is to simplify the life of electronic product developers, freeing them to focus on differentiation. The operating system fulfills this mission by abstracting the hardware resources - RAM and storage, connectivity peripherals such as USB and Ethernet, and human interface devices such as touchscreens. The abstraction is presented to the developer in the form of convenient APIs and mechanisms for interacting with the hardware and managing application workloads. The operating system is coupled with a development environment - compilers, debuggers, editors, performance analyzers, and so on - that help engineers build their powerful applications quickly and to take maximum advantage of the operating system.

Of course, this environment has grown dramatically up the stack. Instead of just providing a TCP/IP stack, an embedded operating system must sometimes provide a full suite of application-level protocols and services such as FTP and web servers. Instead of a simple graphics library, the operating system may need to provide sophisticated multimedia audio and 3D-graphics frameworks. Furthermore, as embedded Systems-on-Chip (SoCs) become more capable, application and real-time workloads are being consolidated. The example discussed briefly in Chapter 1 of an automotive infotainment system integrating rear-view camera capability is representative of this trend: the operating system must provide a powerful applications environment while responding instantly to real-time events and protecting sensitive communications interfaces from corruption.

Because of its central role, the operating system has sometimes been the battleground between electronics manufacturers who aim not only to differentiate but also to protect the uniqueness of and

investment in their innovations. The smartphone market is an obvious example: silicon vendors, consumer electronics manufacturers, and service providers at all levels want to control that human-machine interface, which acts as the portal for revenue, loyalty, and brand recognition.

The trend toward consolidation has posed a significant challenge to embedded operating system suppliers who must navigate the myriad of stacks, interfaces, standards, and software packages. This complexity has also fueled a trend toward open source models to reap the benefits of a massive, distributed developer base. Linux is the primary success story. However, while Linux has succeeded in gaining dramatic market share, it has also suffered from tremendous fragmentation. What embedded systems developers have realized is that they must customize and extend Linux to be able to obtain differentiation and platform control. In essence, these Linux-based efforts have become the new do-it-yourself proprietary operating system for a collection of stakeholders.

The key problem is that the typical operating system abstractions - files, devices, network communication, graphics, and threads - have begun to reach their limit of utility. Application developers and electronics suppliers who become too dependent on one operating system abstraction environment can find themselves in dire straights if the operating system fails to meet emerging requirements, runs into licensing or IP rights headwinds, or is simply surpassed by another operating system in the market.

Title-1

Clearly, embedded systems developers need a platform for innovation and differentiation that is flexible enough to accommodate the inevitable rapid evolution in hardware and software technology. Imagine a platform that can run two completely different operating systems at the same time on the same microprocessor.

Imagine a platform that can run the most sophisticated consumer multimedia operating system such as Android while still running a hard real-time communications stack and security-critical functions completely protected beyond the reach of Android - all on the same SoC. Imagine a platform that can meet the aforementioned automotive consolidation requirement: a multimedia automotive infotainment system that also attains a cold boot time in milliseconds to support instant-on rear-view camera (on the same screen as the main infotainment operating system) and communications over real-time automotive networks (e.g., Controller Area Network - CAN) - all on the same SoC to save on size, weight, power, and cost.

Key Point
The new level of abstraction needed to cope with increasingly sophisticated, consolidated elec- tronic systems is the operating system itself, not just the computer’s hardware resources.

Developers need the flexibility to run any operating system, not just any application, with ease, on the same hardware. The answer to the operating system dilemma is the embedded hypervisor, implementing system virtualization. System virtualization enables the hosting of multiple virtual computer systems on a single physical computer system. These virtual computer systems are often called virtual machines. The ability to virtualize an entire computer system enables multiple independent operating systems to execute concurrently on the shared hardware platform. These operating systems are often referred to as “guest” operating systems because they are permitted to share the physical hardware as guests of the software layer that owns it. This layer is called a hypervisor. Unlike enterprise hypervisors, the embedded hypervisor is designed specifically for embedded and mobile electronics.

The motivations for system virtualization technology in the data center are well known, including resource optimization and improved service availability. But virtualization technology has broader applications throughout the embedded world, including security- enabled mobile devices, embedded virtual security appliances, trusted financial transactions, workload consolidation, and more. This vision is made possible, in part, due to hardware- assisted virtualization technology that now scales down to embedded and mobile devices. The remainder of this chapter provides an overview of the evolution of embedded hypervisor architectures, including both software and hardware trends, and how they affect the security of system virtualization. We also discuss a range of compelling applications for secure virtualization across communities of interest.

2.6.1 Introduction to System Virtualization

Computer system virtualization was first introduced in mainframes during the 1960s and '70s. Although virtualization remained a largely untapped facility during the '80s and '90s, computer scientists have long understood many of the applications of system virtualization, including the ability to run distinct and legacy operating systems on a single hardware platform.

At the start of the millennium, VMware proved the practicality of full system virtualization, hosting unmodified, general-purpose, "guest" operating systems such as Windows, on common Intel Architecture (IA)-based hardware platforms.

In 2005, Intel launched its Virtualization Technology (Intel VT), which both simplified and accelerated virtualization. Consequently, a number of virtualization software products have emerged, alternatively called virtual machine monitors or hypervisors, with varying characteristics and goals. Similar hardware-assists for system virtualization have emerged in other popular embedded CPU architectures, including ARM and Power.

Title-1

While virtualization may be best known for its application in data center server consolidation and provisioning, the technology has proliferated across desktop and laptop-class systems, and has most recently found its way into mobile and embedded environments.

Key Point
The availability of system virtualization technology across a wide range of computing platforms provides developers and technologists with the ultimate open platform: the ability to run any flavor of operating system in any combination, creating an unprecedented flexibility for deployment and usage.

Yet embedded systems often have drastically different resource and security constraints as compared to server computing. We also focus on the impact of hypervisor architecture upon these constraints.

2.6.2 Applications of System Virtualization

Mainframe virtualization was driven by some of the same applications found in today's enterprise systems. Initially, virtualization was used for time sharing, similar to the improved hardware utilization driving modern data center server consolidation. Another important usage involved testing and exploring new operating system architectures. Virtualization was also used to maintain backward compatibility of legacy versions of operating systems.

2.6.3 Environment Sandboxing

Implicit in the concept of consolidation is the premise that independent virtual machines are kept securely separated from each other (also referred to as virtual machine isolation). The ability to

guarantee separation is highly dependent on the robustness of the underlying hypervisor software. As we discussed earlier and will soon expand upon, researchers have found flaws in commercial hypervisors that violate this separation assumption. Nevertheless, an important theoretical application of virtual machine compartmentalization is to isolate untrusted software. For example, a web browser connected to the Internet can be sandboxed in a virtual machine so that Internet-borne malware or browser vulnerabilities are unable to infiltrate or otherwise adversely impact the user's primary operating system environment.

2.6.4 Virtual Security Appliances

Another example, the virtual security appliance, does the opposite: sandbox, or separate, trusted software away from the embedded system's primary operating system environment. Let's consider anti-virus software that runs on a mobile device. A few years ago, the "Metal Gear" Trojan was able to propagate itself across Symbian operating system-based mobile phones by disabling their anti-malware software. Virtualization can solve this problem by placing the anti-malware software into a separate virtual machine (see Figure 2.18).

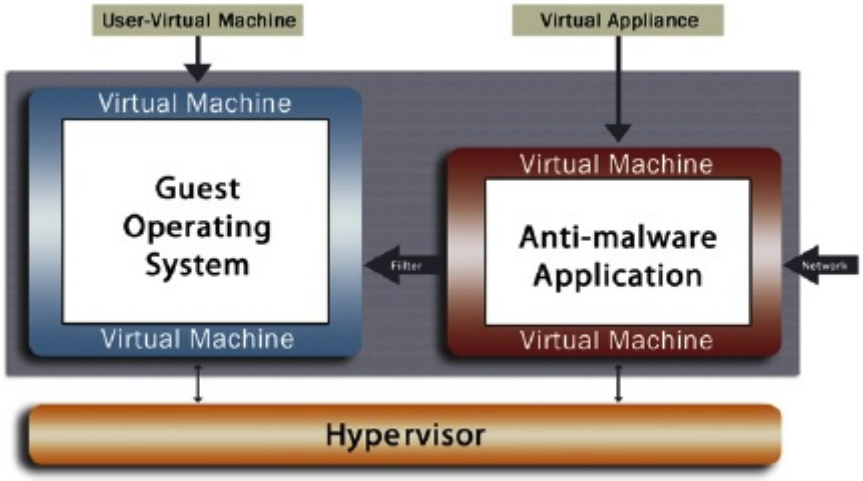


Figure 2.18. Improved security using isolated virtual appliances.

The virtual appliance can analyze data going into and out of the primary application environment or hook into the platform operating system for demand-driven processing.

2.6.5 Hypervisor Architectures

Hypervisors are found in a variety of flavors. Some are open source; others are proprietary. Some use thin hypervisors augmented with specialized guest operating systems. Others use a monolithic hypervisor that is fully self-contained. In this section, we compare and contrast the available technologies, with an emphasis on security impact. Note that this section compares and contrasts so-called Type-1 hypervisors that run on bare metal. Type-2 hypervisors run atop a general-purpose operating system, such as Windows or Linux, that provides I/O and other services on behalf of the hypervisor (see Figure 2.19).

Key Point
Because they can be no more secure than their underlying general-purpose host operating systems (which are well known to be vulnerable), Type-2 hypervisors are not suitable for mission- critical deployments and have historically been avoided in such environments.

Thus, Type-2 technology is omitted from the following discussion.

Title-2

2.6.5.1 Monolithic Hypervisor

Hypervisor architectures most often employ a monolithic architecture, as shown in Figure 2.20. Similar to monolithic operating systems, the monolithic hypervisor requires a large body of operating software, including device drivers and middleware, to support the execution of one or more guest environments. In addition, the monolithic architecture often uses a single instance of the virtualization component to support multiple guest environments. Thus, a single flaw in the hypervisor may result in a compromise of the fundamental guest environment separation intended by virtualization in the first place.

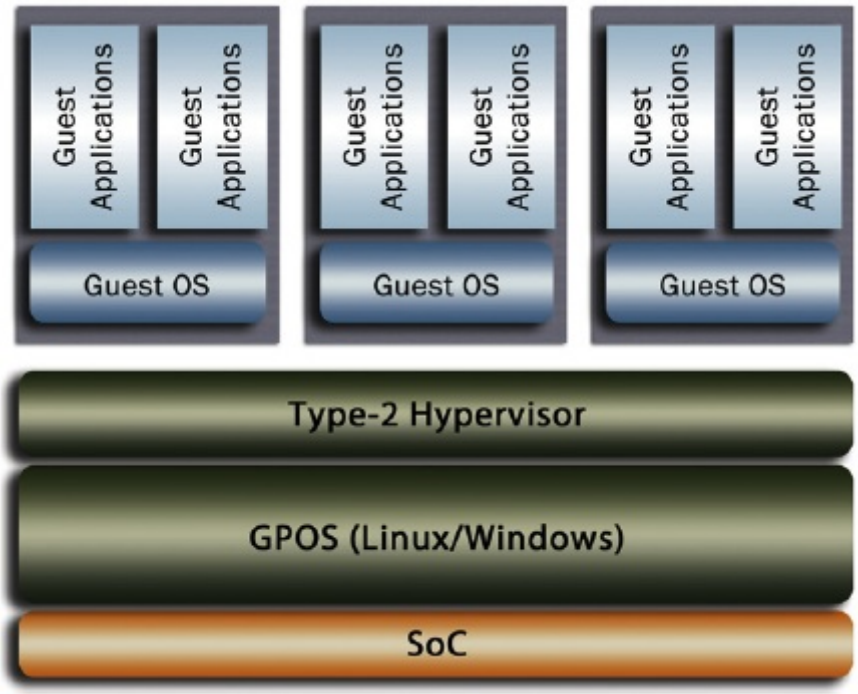


Figure 2.19. Type-2 hypervisor architecture.

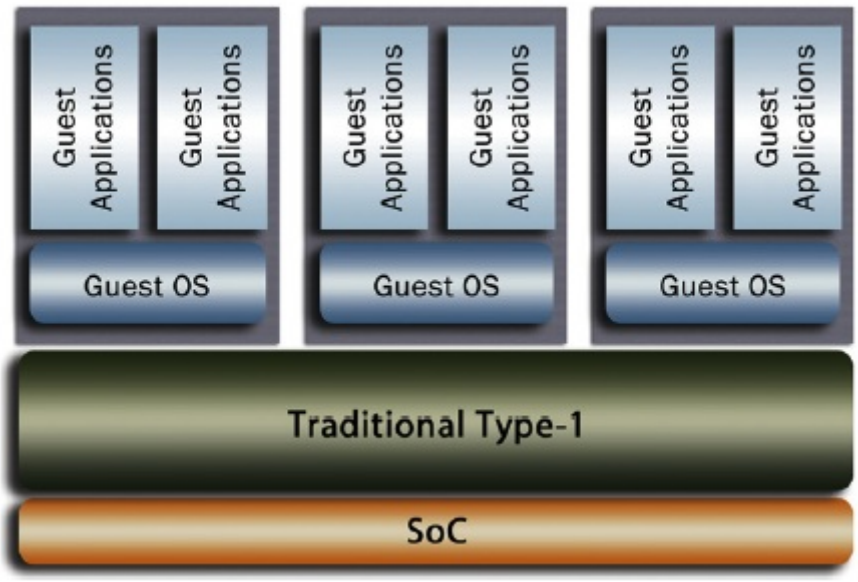


Figure 2.20. Traditional monolithic type-1 hypervisor architecture.

2.6.5.2 Console Guest Hypervisor

An alternative approach uses a trimmed-down hypervisor that runs in the microprocessor's most privileged mode but employs a special guest operating system partition to handle the I/O control and services for the other guest operating systems (see Figure 2.21). Examples of this architecture include Xen and Microsoft Hyper-V. Xen pioneered the console guest approach in the enterprise; within Xen, the console guest is called Domain 0, or Dom0 for short. Thus, the console guest architecture is sometimes referred to as the Dom0 architecture. With the console guest approach, a general-purpose operating system must still be relied upon for system security. A typical console guest such as Linux may add far more code to the virtualization layer than found in a monolithic hypervisor.

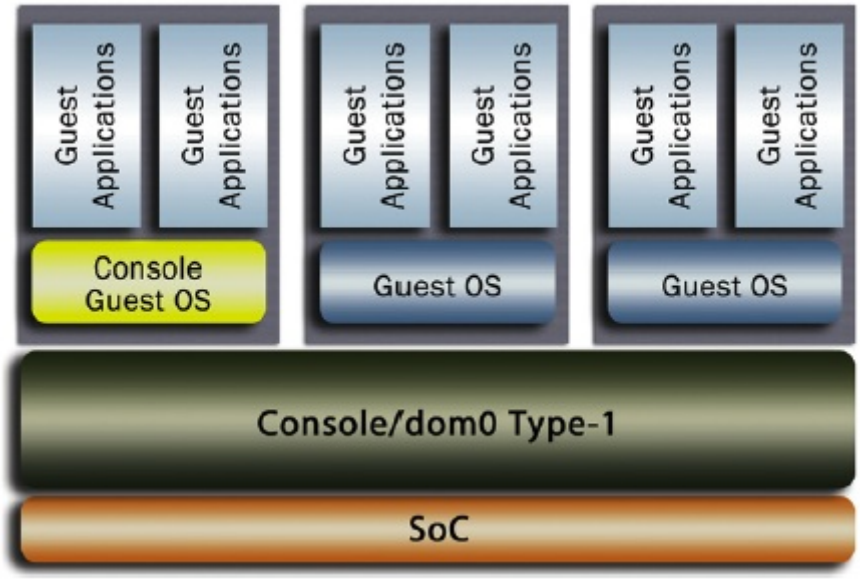


Figure 2.21. Console guest or Dom0 hypervisor architecture.

2.6.5.3 Microkernel-based Hypervisor

Key Point
The microkernel-based hypervisor, a Type-1 architecture, is designed specifically to provide robust separation between guest environments.

Figure 2.22 shows the microkernel-based hypervisor architecture. Because the microkernel is a thin, bare-metal layer, the microkernel-based hypervisor is considered a Type-1 architecture.

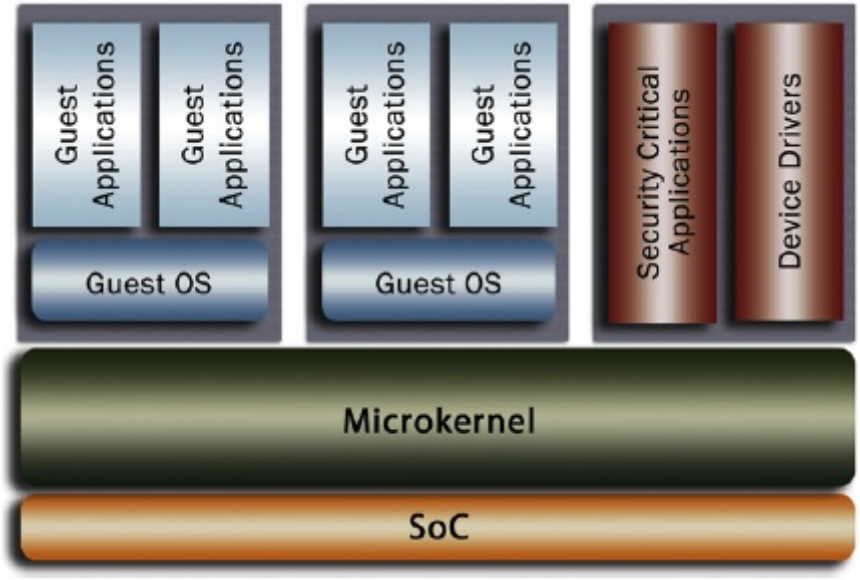


Figure 2.22. Microkernel-based Type-1 hypervisor architecture.

This architecture adds computer virtualization as a service on top of the trusted microkernel. In some cases, a separate instance of the virtualization component is used for each guest environment. Thus, the virtualization layer need only meet the equivalent (and, typically, relatively low) robustness level of the guest itself. In the microkernel architecture, only the trusted microkernel runs in the highest privilege mode. Examples of embedded hypervisors using the microkernel approach include the INTEGRITY Multivisor from Green Hills Software and some variants of the open standard L4 microkernel.

2.6.6 Paravirtualization

System virtualization can be implemented with full virtualization or paravirtualization, a term first coined in the 2001 Denali project.9 With full virtualization, unmodified guest operating systems are supported. With paravirtualization, the guest operating system is modified to improve the ability of the underlying hypervisor to achieve its intended function.

Paravirtualization is sometimes able to improve performance. For example, device drivers in the guest operating system can be modified to make direct use of the I/O hardware instead of requiring I/O accesses to be trapped and emulated by the hypervisor. Paravirtualization may be required on CPU architectures that lack hardware virtualization acceleration features.

Key Point
The key advantage to full virtualization over paravirtualization is the ability to use unmodified versions of guest operating systems that have a proven fielded pedigree and do not require the maintenance associated with custom modifications.

This maintenance saving is especially important in enterprises that use a variety of operating systems and/or regularly upgrade to new operating system versions and patch releases.

Title-3

2.6.7 Leveraging Hardware Assists for Virtualization

Key Point
The addition of CPU hardware assists for system virtualization has been key to

the practical application of hypervisors in embedded systems.

Intel VT, first released in 2005, has been a key factor in the growing adoption of full virtualization throughout the enterprise-computing world. Virtualization Technology for x86 (VT-x) provides a number of hypervisor assistance capabilities, including a true hardware hypervisor mode that enables unmodified guest operating systems to execute with reduced privilege. For example, VT-x will prevent a guest operating system from referencing physical memory beyond what has been allocated to the guest's virtual machine. In addition, VT-x enables selective exception injection so that hypervisor-defined classes of exceptions can be handled directly by the guest operating system without incurring the overhead of hypervisor software interposing. While VT technology became popular in the server class Intel chipsets, the same VT-x technology is now also available in Intel Atom embedded and mobile processors.

In 2009, the Power Architecture governance body, Power.org, added virtualization to the embedded specification within the Power Architecture version 2.06 Instruction Set Architecture (ISA). At the time of this writing, Freescale Semiconductor is the only embedded microprocessor vendor to have released products, including the QorIQ P4080 and P5020 multicore network processors, supporting this embedded virtualization specification.

In 2010, ARM Ltd. announced the addition of hardware virtualization extensions to the ARM architecture as well as the first ARM core, the Cortex A15, to implement them. Publicly announced licensees planning to create embedded SoCs based on Cortex A15 include Texas Instruments, Nvidia, Samsung, and ST-Ericsson.

Prior to the advent of these hardware virtualization extensions, full virtualization was possible only using dynamic binary translation and instruction rewriting techniques that were exceedingly complex and unable to perform close enough to native speed to be practical in embedded systems. For example, a 2005-era x86-based desktop running Green Hills Software's pre-VT virtualization technology was able to support no more than two simultaneous full-motion audio/video clips (each in a separate virtual machine) without dropping frames. With Green Hills Software's VT-x-based implementation on similar class desktops, only the total RAM available to host multiple virtual machines generally limits the number of simultaneous clips. General x86 virtualization benchmarks showed an approximate doubling of performance using VT-x relative to the pre-VT platforms. In addition, the virtualization software layer was simplified due to the VT-x capabilities.

2.6.7.1 ARM TrustZone

Key Point

An often overlooked and undervalued virtualization capability in modern ARM microprocessors is ARM TrustZone.

TrustZone enables a specialized, hardware-based form of system virtualization. TrustZone provides two zones: a "normal" zone and a "trust" or "secure" zone. With TrustZone, the multimedia operating system (for example, what the user typically sees on a smartphone) runs in the normal zone while security-critical software runs in the secure zone. Although secure zone supervisor mode software is able to access the normal zone's memory, the reverse is not possible (see Figure 2.23). Thus, the normal zone acts as a virtual machine under control of a hypervisor running in the trust zone. However, unlike other hardware virtualization technologies such as Intel VT, the normal zone guest operating system incurs no execution overhead relative to running without TrustZone. Thus, TrustZone removes the performance (and arguably the largest) barrier to the adoption of system virtualization in resource- constrained embedded devices.

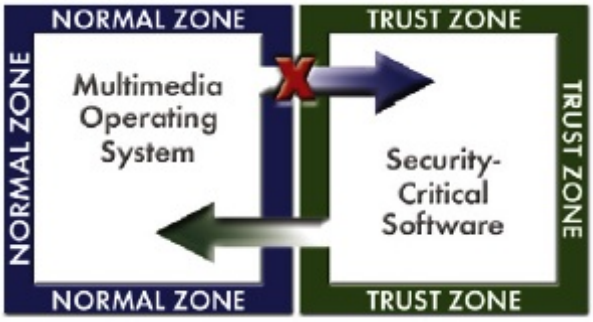


Figure 2.23. ARM TrustZone.

Title-4

TrustZone is a capability inherent in modern ARM applications processor cores, including the ARM1176, Cortex A5, Cortex A8, Cortex A9, and Cortex A15. However, it is important to note that not all SoCs using these cores fully enable TrustZone. The chip manufacturer must permit secure zone partitioning of memory and I/O peripheral interrupts throughout the SoC complex. Furthermore, the chip provider must open the secure zone for third-party trusted operating systems and applications. Examples of TrustZone-enabled mobile SoCs are the Freescale i.MX53 (Cortex A8) and the Texas Instruments OMAP 4430 (Cortex A9).

Trusted software might include cryptographic algorithms, network security protocols (such as SSL/TLS) and keying material; digital rights management (DRM) software; virtual keypad for

trusted path credential input; mobile payment subsystems; electronic identity data; and anything else that a service provider, mobile device manufacturer, and/or mobile SoC supplier deems worthy of protecting from the user environment.

In addition to improving security, TrustZone can reduce the cost and time to market for mobile devices that require certification for use in banking and other critical industries. With TrustZone, the bank (or certification authority) can limit certification scope to the secure zone and avoid the complexity (if not infeasibility) of certifying the multimedia operating system environment.

A secure zone operating system can further reduce the cost and certification time, for two main reasons. First, because the certified operating system is already trusted, with its design and testing artifacts available to the certification authority, the cost and time of certifying the secure zone operating environment is avoided.

Second, because the secure zone is a complete logical ARM core, the secure operating system is able to use its memory management unit (MMU) partitioning capabilities to further divide the secure zone into meta-zones (see Figure 2.24). For example, a bank may require certification of the cryptographic meta-zone used to authenticate and encrypt banking transaction messages, but the bank will not care about certifying a multimedia DRM meta- zone, that, while critical for the overall device, is not used in banking transactions and guaranteed by the secure operating system not to interfere.

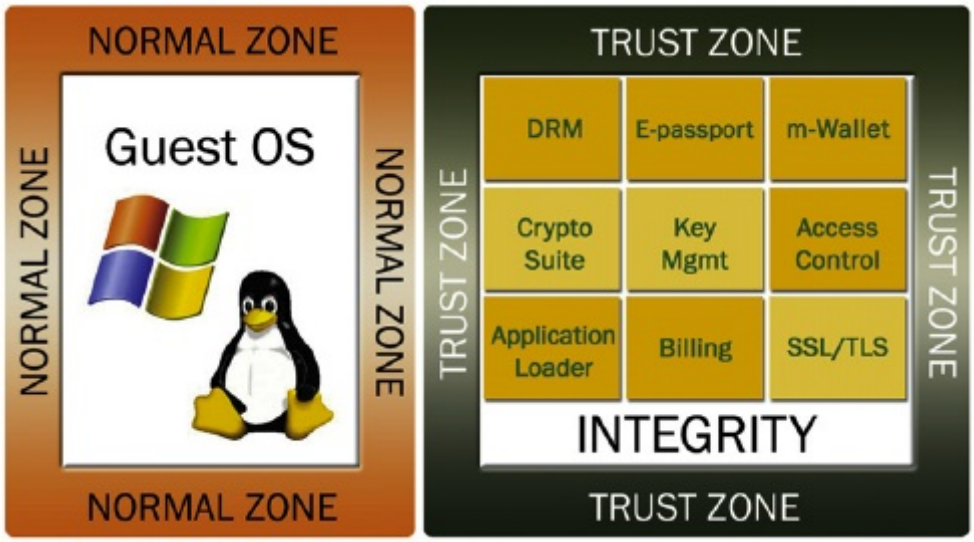


Figure 2.24. TrustZone virtualization implementation with meta-zones within the TrustZone.

TrustZone-enabled SoCs are able to partition peripherals and interrupts between the secure and normal states. A normal zone general-purpose operating system such as Android cannot access peripherals allocated to the secure zone and will never see the hardware interrupts associated with those peripherals. In addition, any peripherals allocated to the normal zone are unable to access memory in the normal zone.

Title-5

2.6.8 Hypervisor Security

Some tout virtualization as a technique in a “layered defense” for system security. The theory postulates that since only the guest operating system is exposed to external threats, an attacker who penetrates the guest will be unable to subvert the rest of the system. In essence, the virtualization software is providing an isolation function similar to the process model provided by most modern operating systems.

However, common enterprise virtualization products have not met high-robustness security requirements and were never designed or intended to meet these levels. Thus, it should come as no surprise that the theory of security via virtualization has no existence proof in common enterprise implementations. Rather, a number of studies of virtualization security and successful subversions of hypervisors have been published.

2.6.8.1 SubVirt

In 2006, Samuel King’s SubVirt project demonstrated hypervisor rootkits that subverted both VMware and Microsoft VirtualPC.[10]

2.6.8.2 Blue Pill

The Blue Pill project took hypervisor exploits a step further by demonstrating a malware payload that was itself a hypervisor that could be installed on the fly, beneath a natively running Windows operating system. Secure boot and platform attestation (discussed later in this chapter) are required to prevent hypervisors from being subverted in this manner.

2.6.8.3 Ormandy

Tavis Ormandy performed an empirical study of hypervisor vulnerabilities. Ormandy’s team of

researchers generated random I/O activity into the hypervisor, attempting to trigger crashes or other anomalous behavior. The project discovered vulnerabilities in QEMU, VMware Workstation and Server, Bochs, and a pair of unnamed proprietary hypervisor products.[11]

2.6.8.4 Xen Owning Trilogy

At the 2008 Black Hat conference, security researcher Joanna Rutkowska and her team presented their findings of a brief research project to locate vulnerabilities in Xen. One hypothesis was that Xen would be less likely to have serious vulnerabilities, as compared to VMware and Microsoft Hyper-V, due to the fact that Xen is an open source technology and therefore benefits from the "many-eyes" exposure of the code base. Rutkowska's team discovered three different, fully exploitable, vulnerabilities that the researchers used to commandeer the computer via the hypervisor.[12] Ironically, one of these attacks took advantage of a buffer overflow defect in Xen's Flask layer. Flask is a security framework, the same one used in SELinux, which was added to Xen to improve security. This further underscores an important principle: software that has not been designed for and evaluated to high levels of assurance must be assumed to be subvertible by determined and well-resourced entities.

2.6.8.5 VMware's Security Certification and Subsequent Vulnerability Postings

As VMware virtualization deployments in the data center have grown, security experts have voiced concerns about the implications of "VM sprawl" and the ability of virtualization technologies to ensure security. On June 2, 2008, VMware attempted to allay this concern with its announcement that its hypervisor products had achieved a Common Criteria EAL 4þ security certification. VMware's press release claimed that its virtualization products could now be used "for sensitive, government environments that demand the strictest security."

On June 5, just three days later, severe vulnerabilities in the certified VMware hypervisors were posted to the National Vulnerability Database. Among other pitfalls, the vulnerabilities "allow guest operating system users to execute arbitrary code."[13]

VMware's virtualization products have continued to amass severe vulnerabilities - for example, CVE-2009-3732, which enables remote attackers to execute arbitrary code.

Clearly, the risk of an "escape" from the virtual machine layer, exposing all guests, is very real. This is particularly true of hypervisors characterized by monolithic code bases.

Key Point
It is important for developers to understand that use of a hypervisor does not imply highly assured isolation between virtual machines, no more than the use of an operating system with memory protection implies assured process isolation and overall system security.

Chapter 3 discusses the Common Criteria security evaluation standard and its Evaluated Assurance Levels, and what they imply in terms of security assurance.