FOI
Swedish Defence Research Agency
Information Systems
Box 1165
SE-581 11 Linköping

Phone: +46 13 37 80 00
Fax: +46 13 37 81 00

www.foi.se

Virtual Machines

Security Qualities

ALF BENGTSSON, LARS WESTERDAHL

FOI-R--2904--SE
ISSN 1650-1942

User Report
December 2009

Information Systems

Alf Bengtsson, Lars Westerdahl

Virtual Machines

Security Qualities

Titel	Virtuella maskiner, säkerhetsegenskaper
Title	Virtual Machines, Security Qualities
Rapportnr/Report no	FOI-R--2904--SE
Rapporttyp Report Type	Användarrapport User Report
Månad/Month	December
Utgivningsår/Year	2009
Antal sidor/Pages	36 p
ISSN	ISSN 1650-1942
Kund/Customer	Försvarsmakten
Kompetenskloss	26 IT-säkerhet
Extra kompetenskloss	
Projektnr/Project no	E53055
Godkänd av/Approved by	Magnus Jändel

FOI, Totalförsvarets Forskningsinstitut	FOI, Swedish Defence Research Agency
Avdelningen för Informationssystem	Information Systems
Box 1165	Box 1165
581 11 Linköping	SE-581 11 Linköping

Sammanfattning

Huvudsyftet med projektet *Objekt- och tjänstebaserad säkerhet* är att klargöra vilken säkerhetsfunktionalitet som skulle kunna finnas i form av tjänster, samt att utreda vilken säkerhetsfunktionalitet som skulle kunna vara möjlig att knyta till distribuerade informationsobjekt. Virtuella maskiner är relevanta för båda delarna av projektet. Därför rapporteras en litteraturstudie som har fokuserat på isolations- och separeringsförmågan hos en hypervisor, vilken är en minimal monitor för virtuella maskiner. Isolations- och separeringsförmågan är central för att uppnå assurerad säkerhet. Rapporten inleds mer allmänt om hypervisor, och inriktas sedan på öppen källkods-hypervisorn Xen.

Studiens slutsats är att separationsförmågan i en hypervisor är av samma slag som i en separation kernel. Till skillnad från separation kernel finns det dock ingen säkerhetsassurerad hypervisor. Litteratur om vilken assuransnivå som skulle kunna uppnås refereras. Likaså refereras forskningsprojekt om tvingande åtkomstkontroll (mandatory access control) i Xen hypervisor. Det senare är ett steg mot visionen om objektbaserad säkerhet.

Referenser till risker med virtuella maskiner ingår också.

Nyckelord: virtuella maskiner, hypervisor, Xen, separation kernel, mandatory access control, tjänstebaserad säkerhet, objektbaserad säkerhet

Summary

The main goal of the project *Object- and Service-Based Security* is to clarify which security functionality could be transformed into services, and to consider which security functionality that potentially could be bound to distributed information objects. Virtual machines are relevant for both parts of the project. Therefore, a literature study is reported, concentrating on the isolation and separation capabilities of hypervisors, which are minimal virtual machine monitors. The isolation and separation capabilities are central to achieve assured security. The report starts with general hypervisors, and then focuses on the open source hypervisor Xen.

The conclusions of the study are that the separation capability of a hypervisor is of the same kind as that of a separation kernel. However, unlike separation kernel there is no authoritatively security evaluated hypervisor. Some literature, on what assurance level that could be achieved, is referred to. Likewise, research projects on mandatory access control in the Xen hypervisor are referred. This is a step towards the vision of object-based security.

References on risks with virtual machines are also included.

Keywords: virtual machines, hypervisor, Xen, separation kernel, mandatory access control, service-based security, object-based security

Table of Contents

1 Introduction

1.1 Motivation

The main goal of the project *Object- and Service-Based Security* is to clarify what security functionality could be transformed into services, complementing the functionality that is traditionally enforced locally in each unit of the total system. The quest for service orientation is primarily due to better flexibility and management. Additionally, it has potential to promote object-based security. The latter term refers to a vision to bind security attributes to each individual information object and have mandatory control of the object, even across system boundaries.

We realized early that virtual machines are relevant for both parts of the project. Virtual machines can be configured and distributed from a central service. They can also be stripped down to minimal complexity, which should promote mandatory control. Accordingly, we decided to carry out a theoretic study of publicly available literature on security qualities of virtual machines.

1.2 Topics of interest

This theoretic study shall focus on topics relevant to the two parts of the project. The topics of interest in the study have been, in order of interest:

1. Separation capability. It is absolutely essential that virtual machines, running on the same physical machine, are independent of each other. There must be no data leakage between virtual machines.
2. Configuration. It shall be possible to configure each virtual machine to only contain components that are necessary for the application running in the virtual machine.
3. Policies. It shall be possible to have security policies which authorize how virtual machines interact.
4. Assurance. To allow sensitive information in virtual machines, they must conform to the proper assurance level.
5. Usability and deployment are important, although hard to assess in a theoretical study.

1.3 Report layout

The report is organized as follows. Section 2 gives a background to why virtual machine monitors have relevant security qualities. The main quality discussed is that of a separation kernel. Section 3 is a general description of virtual machines, ending in the concept of a minimal hypervisor. Risks with virtual machines are also included. In Section 4 the open source hypervisor Xen is described. Section 5 describes two security extensions to Xen, packaged as Xen Security Modules. Finally, conclusions are presented in Section 6.

2 Background

It is a well known "old truth" that modularity and minimalism are good from a security point of view. Rather than having a big monolith, a system should consist of smaller modules, easier to grasp. Likewise, controlling directives like access control policies should be partitioned into smaller building blocks. The reason is of course that it is easier, and less expensive, to assure and verify small modules than big ones. However, the modules must be totally separated and independent of each other, otherwise the total system effectively will be a monolith, hard to assure. One way to achieve separation would be to use separate hardware for each module. This, however, is in most cases impossible, for practical and economical reasons. Instead, the alternative is to have a secure way to separate software modules inside a host machine. Some terms, used in this context, and later discussed:

- SK, Separation Kernel. Mechanisms whose primary function is to isolate and separate partitions and control information flow between these partitions.
- VM, Virtual Machine. A relatively large partition, which emulates a whole physical machine.
- VMM, Virtual Machine Monitor, running in the hosting physical machine. The software needed to establish and control the guest virtual machines.
- Hypervisor. A class of VMMs, running directly on the hardware in the host machine. A hypervisor is akin to "a minimal operating system".

The concept of "separation kernel" was first coined in 1981 in a seminal paper of Rushby [1]. Quoted from this paper:

> "However, the type of kernel which I am proposing differs from a VMM in that there is no requirement for it to provide VMs which are exact copies of the base hardware (or even for all the VMs to be alike)—but there is a requirement for it to provide communications channels between some of its VMs. In order to avoid confusion with established terminology, I shall call this new type of security kernel a 'separation kernel'."

In his paper Rushby observes that an SK in many ways is identical to a VMM, Virtual Machine Monitor, which was a known concept. The differences are that the VMM controls whole machines, larger than an SK-module, and that a VMM does not enforce a communications policy.

Rushby argues in his paper that it is possible to build a small SK, which completely isolates modules (processes) from each other. Completely isolated modules, not communicating or sharing anything, are of course of limited use.

But Rushby also argues that the SK can include a function for communications control. A module shall have a communication port and it should be the only, and verifiable, way to send/receive any data. The SK shall be capable to control each connection between the ports of two modules. And each connection can have a separate policy for allowed communication.

Rushby argues that partitioning the overall policy makes it substantially easier to endorse and assure. The then prevailing way was one system where the operating system had a small security kernel[1], a reference monitor. The function of a reference monitor is to examine all security sensitive accesses. When a policy is implemented, e. g. MLS, Multi Level Security, it must be effective in the whole system and enforced by the security kernel. It was learned that this was hard to achieve, unless there were some trusted processes that overruled the policy, which meant that the security was hard to assure.

At a “Classic Papers” track at ACSAC 2007 [2] Rushby summarizes his -81 paper and what has happened since. He pointed out that the former RSRE (Royal Signals and Radar Establishment, Rushby is English) showed interest and started to make three communication systems, based on SK. However, in 1994 this was abandoned. Quote Rushby: “we tend to attribute the problems to technical limitations of the time”. From around mid -90s SK has had renewed interest, notably from NSA[2] in USA. NSA has compiled a Protection Profile, compliant with Common Criteria: “U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness” [3]. One product [4] matching this Protection Profile has been evaluated.

The Common Criteria for Information Technology Security Evaluation [5], CC, is the internationally recognized standard for security evaluation of IT products. A Protection Profile, PP, is an evaluated collection of implementation independent security requirements for a class of target products. Each implemented and realized product can then be evaluated and assured to comply with the PP. The CC defines six building blocks for a PP, including security functional requirements and security assurance requirements. With respect to the latter, seven Evaluation Assurance Levels, EALs, are defined from EAL1 (lowest) to EAL7 (highest). Although the CC is internationally accepted, not all nations follow all of it.

[1] The terminology is confusing, separation kernel vs security kernel. The foremost property of a separation kernel is its isolation capability. The isolation is not a property in a conventional security kernel

[2] nowadays NSA/CSS, National Security Agency/Central Security Service

NSA/CSS is in charge of information security for National Security Systems[3] in the USA. They do not use EAL5-EAL7 for the highest assurance levels. Instead NSA has defined three robustness levels – basic, medium and high. How a robustness level relates to the requirements (both security functional and security assurance) and levels in CC is described in the Consistency Instruction Manuals, for instance [6]. To protect information of high security classification high robustness is required, combined with physical protection and other restrictions (one example is [7]).

The "Protection Profile for Separation Kernels in Environments Requiring High Robustness" [3], is the requirements for the minimal inner kernel, not for a complete system. Quoted from [3]:

> A TOE[4] includes the following security features:
>
> − Information flow control that enforces strict partition isolation, with the exception of explicit interactions specified by the configuration data
>
> − Cryptographic mechanisms that provide functions to verify the integrity of TSF[5] code and data during trusted delivery
>
> − Trusted initialization and recovery functions
>
> − Detection and response to security function failures
>
> − Generation of audit data
>
> Among the features not required are:
>
> − User interfaces during an execution session or initialization
>
> − Identification and Authentication which mandates authorized users to be uniquely identified and authenticated by the TSF[5]
>
> − Discretionary Access Control (DAC) which restricts access to objects based on the identity of subjects and/or groups to which they belong, and allows authorized users to specify protection for objects that they control

[3] "National Security Systems are systems that contain classified information or involve intelligence activities, involve cryptologic activities related to national security, involve command and control of military forces, involve equipment that is an integral part of a weapon or weapon system, or involve equipment that is critical to the direct fulfillment of military or intelligence missions", originally defined in Clinger-Cohen Act 1996

[4] TOE – Target of Evaluation, viz. the Security Kernel

[5] TSF – TOE Security Functions

- Cryptographic services for applications to encrypt, decrypt, hash, and digitally sign data as it resides within the system and as it is transmitted to other systems
- Complete physical protection mechanisms

These features are assumed to be outside the SK.

As already mentioned, an SK is not identical to a hypervisor, but there are many similarities. For instance, according to GreenHills' web page [8], their evaluated separation kernel [4] can act as a hypervisor.

Another indication of the similarities between SK and hypervisor is the Open Trusted Computing consortium (OpenTC) [9]. It is a research project, financed by the European Commission, aiming to develop openly available modules for trusted computing. Much concern is taken to use Trusted Platform Module (see Figure 5), TPM, to sign and verify data and software modules. But the need of isolation and separation is also emphasized. Quoted from OpenTC [10]: ”A core idea of OpenTC is to combine security properties of TC-hardware and isolation properties of virtualisation in order to build trusted platforms. At the lowest level, TC mechanisms are provided by hardware (by the Trusted Computing Module and state of the art CPUs).” For separation they have used both the microkernel L4 [11] and the hypervisor Xen [12]. A microkernel is a minimal operating system, where things like device drivers and file systems run in an unprivileged mode, outside the kernel.

The conclusion of this background discussion is that the desired security properties in a separation kernel also exist in a hypervisor. Consequently, it is appropriate to discuss security qualities of hypervisors. A hypervisor has some advantages over security kernels. The full virtual machine, controlled by a hypervisor, is more well-known and well-tested than an untried partition controlled by a separation kernel.

3 Virtual Machines

Virtual Machines, VM, is the term used when one host machine emulates many guest machines. The Virtual Machine Monitor, VMM, on the host could be a full fledged operating system, like Linux or MS Windows. Alternatively, it could be a much smaller, by a factor 100-1000, special operating system, often called hypervisor. The guest machines, running as applications above the VMM, can also be for example ordinary MS Windows, or it could be, for example, a much stripped Linux, only meant to support a single application. The smaller size, both hypervisor and (stripped) VM, is a good thing from security point of view.

VM is an old concept. It was used in the 70s as a cost-effective way to use expensive hardware, by allowing many machines to run at the same time on the same hardware. When the price of hardware declined, the need for VM also declined. Today there is a reborned interest in VM, due to other types of economy, like footprint area, energy consumption, administration, configuration etc. In addition, as mentioned, VMs are very relevant from security point of view.

3.1 General architecture of Virtual Machines

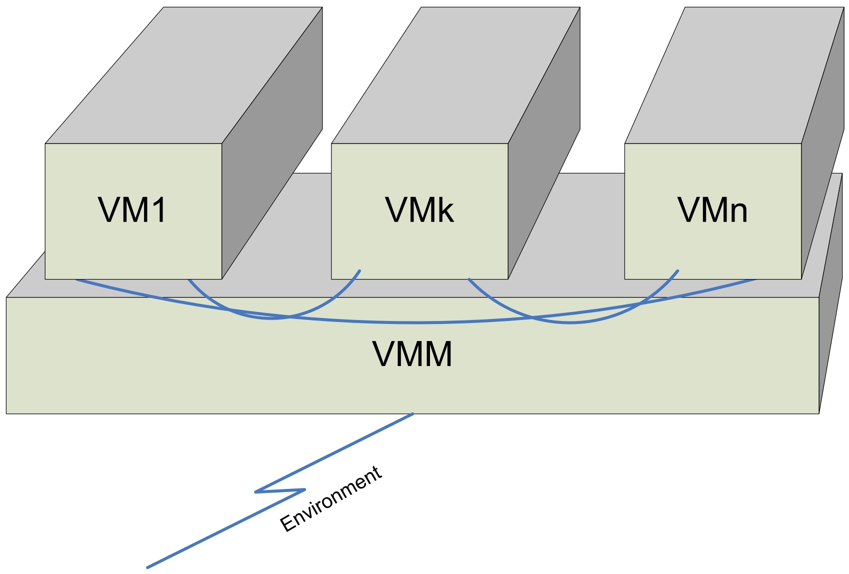

Figure 1 Schematic figure of Virtual Machines.

Figure 1 is very schematic, just meant to illustrate the concept, Virtual Machines controlled by a Virtual Machine Monitor. The VMM can be of very varying size and complexity. Some varieties:

- The VMM could be an operating system, like MS Windows or Linux, plus a run time system, like Java VM. In this case the VMs in figure 1 rather should be called VAs, Virtual Applications. The main advantage is the emulation. The VMM emulates everything underneath the applications. The applications can therefore be written independently of the environment where they will be executed. But there also exist security qualities in this high level approach of VMM. By adding flow control modules to the Java VM (for example Trishul [13]), it is possible to achieve usage control. This means that the access control, normally provided by the OS, is enhanced by control inside the VM of how information flows between applications. The usage control is very attractive for applications like Digital Rights Management, DRM. It is also a very relevant component in the quest for object based security.

- The VMM could be a middleware running on top of a regular operating system, like MS Windows or Linux. In this case the VMs are complete machines, including an ordinary operating system. Today this is frequently used at regular service centers, to rapidly reconfigure services in response to changing conditions, even without interrupting executions. This also means that it should be possible to configure distributed services from a central point, and to reconfigure in response to changing security conditions. (The reconfiguration possibility is also true for the virtual applications outlined above). Another security quality is that each VM can be stripped to a minimal configuration, just enough to support one delivered service for each VM.
- The VMM could be a minimal kernel, named hypervisor, running directly on the hardware. In effect, it is a minimal basic OS. In this case, the surpassing security quality is the isolation property. The VMs are machines, including an operating system (preferably stripped to a minimum), running on top of the hypervisor.

From now on the focus in this paper will be the minimal VMM, i.e. hypervisor.

VM means that it is possible to build, and configure, several independent and mutually isolated machines inside a single host. And independence and isolation are good security, cf. separation kernel, SK. Note, however, that a system consisting of many independent physical machines, will not be more secure if they are realized as VMs. The challenge to have a correct access control policy will be the same.

It is tempting to add different kinds of functionality to hypervisors, but a hypervisor overloaded with functionality might be nearly as complex as an ordinary OS. One attractive functionality is for example efficient communication between VMs, hosted in the host machine. This can be much more efficient by using shared memory. However, that may sacrifice the isolation property.

IEEE Security & Privacy has a special issue Sep/Oct 2008, “Virtualization and Security: Back to the Future” [14]. The title alludes to the fact that virtualization is an old concept. The Figures 2, 3, 4, 5 and 8 of VMs are copied from [14], through the courtesy of IEEE.

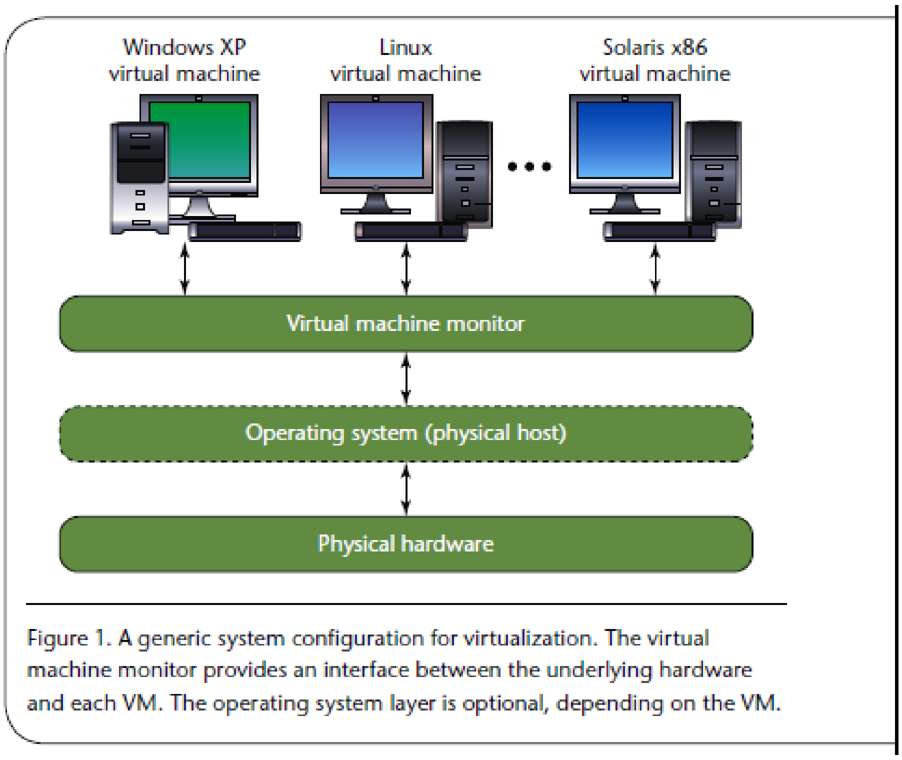

Figure 1. A generic system configuration for virtualization. The virtual machine monitor provides an interface between the underlying hardware and each VM. The operating system layer is optional, depending on the VM.

Figure 2 A generic system configuration for virtualization, from [37], © 2008 IEEE.

The optional host (dotted line) in Figure 2 means that the VMM could be an “ordinary” application running on an “ordinary” host OS. Virtualization as a means to avoid piles of physical computers in computer centers often looks like this. But also NetTop [15] from NSA, on top of SELinux [31] running in VMware, looks like this. Among the drawbacks are overhead (additional layers, VMM + OS) and security assurance. It is exceedingly expensive to assure a large OS, and a large host OS + VMM is even more expensive.

For isolation purposes, like MILS [16], Multiple Independent Levels of Security, it is preferable to have a small (which hopefully could be assured) hypervisor running directly at the hardware, i. e. no big host OS. Essentially, this means that the hypervisor is a “basic OS”, below the “application OS”. A problem is to decide what should be included in the hypervisor, shown in Figure 3.

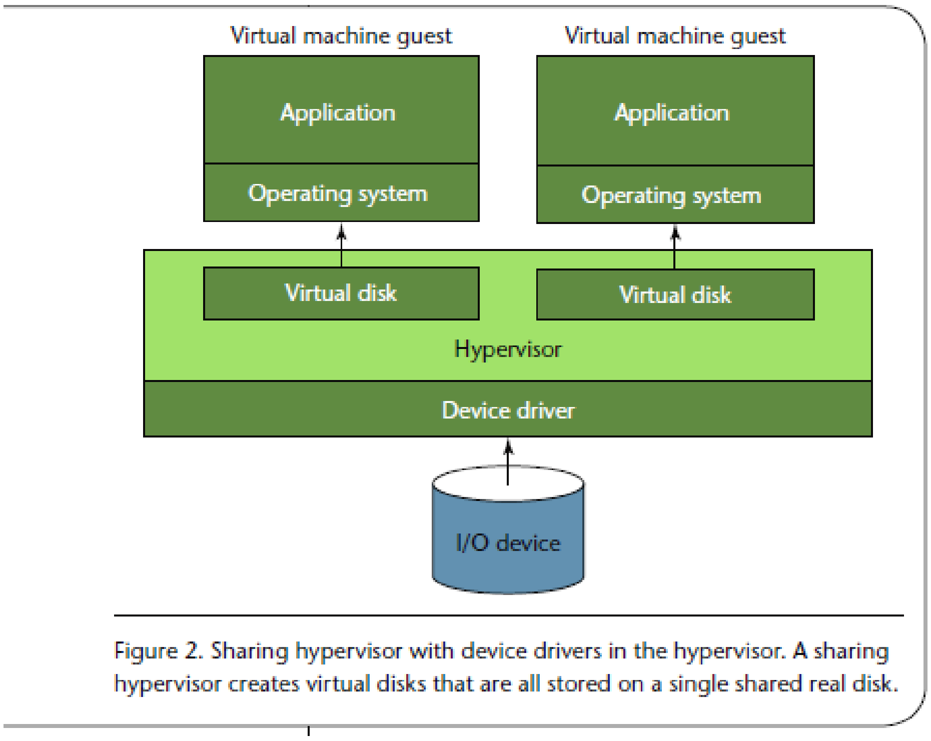
Virtual machine guest
Virtual machine guest
Application
Application
Operating system
Operating system
Virtual disk
Virtual disk
Hypervisor
Device driver
I/O device
Figure 2. Sharing hypervisor with device drivers in the hypervisor. A sharing hypervisor creates virtual disks that are all stored on a single shared real disk.

Figure 3 Device drivers in hypervisor, from [37], © 2008 IEEE.

To be able to share hardware (e. g. radio and/or network devices) the device driver must be moved from application OS to VMM. If too much is moved to the hypervisor, the result would be “a new large OS”, and the security would be hard to assure. Another problem is that some hardware (notably most legacy x86) is not “virtualization friendly”. Some system calls in application OS are therefore changed to special “hypervisor calls” (this is called paravirtualization), which are executed in the hypervisor. This leads to compatibility problems.

One way to handle device sharing is to have a special “device machine”. Figure 4 depicts this.

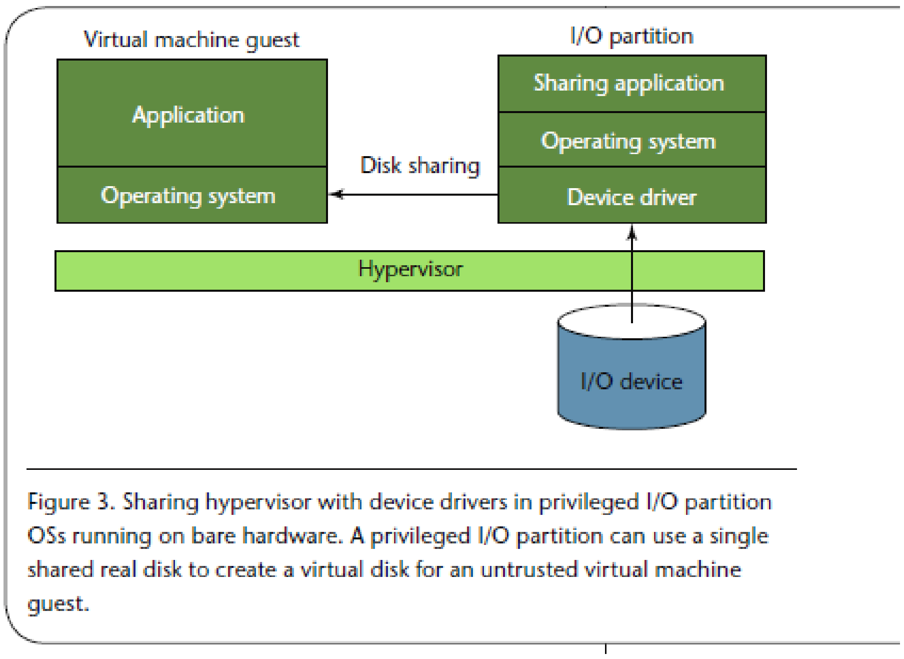

Figure 3. Sharing hypervisor with device drivers in privileged I/O partition OSs running on bare hardware. A privileged I/O partition can use a single shared real disk to create a virtual disk for an untrusted virtual machine guest.

Figure 4 Device drivers in privileged machine, from [37], © 2008 IEEE.

This means communication between VMs, which means some overhead. It also means that the operating system in the guest virtual machines must be modified. In a standard unmodified OS, like Linux or MS Windows, there are device drivers which access the device hardware directly. These drivers are replaced by front-end device drivers in the guest, talking to back-end drivers in the I/O partition. Only the back-end drivers can access the hardware.

A special hardware module, desirable to share between VMs, is the Trusted Platform Module, TPM [17]. It is a hardware module for cryptographic operations and for handling and generation of crypto keys. An important use of TPM is to verify digital signatures of critical software and to verify hardware modules at system startup and reconfiguration. As for other drivers, the driver for TPM should not be inside the hypervisor, but should be in a VM of its own, in a TPM VM. The following Figure 5 is copied from [18], where a TPM VM multiplexes the hardware TPM.

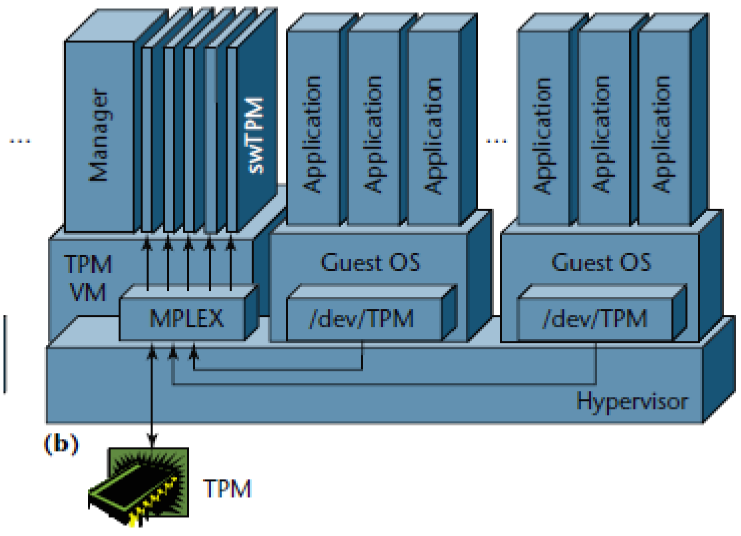

Figure 5 Trusted platform module virtualization, from [18], © 2008 IEEE.

The communication between VMs, e.g. disk sharing in Figure 4 and TPM multiplexing in Figure 5, is by use of so called event channels, controlled by the hypervisor. The main security responsibility of the hypervisor is to endorse the complete isolation of the VMs and that absolutely no information can be passed except via the controlled event channel. An attractive quality would be to be able to formulate a policy for how information is allowed to be passed. Such extensions exist, as described later on. However, policy handling is too complex to implement in a hypervisor. Therefore, it is implemented as minimal extensions in the hypervisor, complemented with a separate VM for policy managing, described later on.

What is just described could be expressed as having a minimal hypervisor, aided by helper VMs. The rationale is of course that the security of the hypervisor should be possible to assert. But a helper VM may very well include parts critical for the security, e.g. policy managing. The TCB, Trusted Computing Base, of a system is thus not the hypervisor alone. The TCB also includes parts from helper VMs, which means that they shall also be kept to a minimal complexity, to be possible to assert. Nonetheless, it is less difficult to assert a TCB module contained in a stripped VM, than a module inside a large OS.

3.2 Risks with virtual machines

Virtual machines have been shown to have benefits when it comes to reducing cost and space for hardware, as well as providing the possibility to segment functionality. Apart from economic and physical benefits, VMs can improve security as well. The foremost property for security improvement is the ability to provide isolation.

From an attackers' point of view, attacking a hypervisor should yield the most benefit. The hypervisor controls the guest machines and thus a compromised hypervisor will give access to the guest machines. However, few known attacks have been launched against hypervisors [39]. This may be due to the fact that hypervisors can be scaled down to only containing the minimum required functionality, and thus becoming a purpose-built application. In combination with being small and having limited external access, hypervisors are naturally less exposed to attackers [38]. This is not to say that VMs, VMMs or hypervisors are without flaws of their own. According to the National Vulnerability Database (NVD) there are 329 known vulnerabilities for VMs, 4 for VMMs and 7 for hypervisors [40].

Risks are a combination of threats, vulnerabilities and the ability to exploit these vulnerabilities. In a study by Ormandy [41], six commonly available VMs were compared. The scope of the study was to test the ability of a VM to maintain its isolation property in a hostile environment. A hostile environment is here considered to be a situation where untrusted code is executed or untrusted data is being processed inside the VM.

The study was set up so that, in case of failure of maintaining isolation, the failure of the VM could be measured gradually. Failing was categorized as:

- Total compromise: The VMM is fully compromised and can execute arbitrary code with full privileges on the physical system.
- Partial compromise: The separation between the VMM and the physical host fails. The VMM leaks information about the host or cannot control hostile processes within itself.
- Abnormal termination: Unexpected terminations of the VMM (in effect a denial of service attack) which results in the inability for the host administrator to reach guest applications.

In the study [41] three open source VMs were tested, (Bochs[6], QEMU[7] and Xen[8]) as well as three proprietary machines, (VMware[9]) and two undisclosed

[6] http://bochs.sourceforge.net/ (2009-11-10)
[7] http://www.qemu.org/ (2009-11-10)

popular on the Macintosh platform and on the Microsoft Windows platform, respectively.

The result of the study was that no virtual machine was immune to compromise. By the use of two simple tools (Crashme[10] and iofuzz) known vulnerabilities where identified and exploited. It is important to note that all VMs could not be scrutinized to their full potential. Xen, for instance, a VM with a good security design utilizes hardware support that was unavailable to the author at time of the study. It is also worth to note that the study was performed in 2007, which means that vulnerabilities in the tested VMs may have been dealt with later.

As shown above, there are still many known flaws in available VMs, VMMs and hypervisors. However, known vulnerabilities are most often due to implementation errors and as such can be dealt with. The Xen platform is most interesting to explore further with regards to hardware support that was unavailable for the above study.

[8] http://xen.org (2009-11-10)
[9] http://www.vmware.com (2009-11-10)
[10] http://www.codeplex.com/crashme/ (2009-11-10)

4 Xen

There are many proprietary VMMs, of the hypervisor category, aimed at virtualizing a particular type of hardware. The prevailing open source hypervisor, for general use, is Xen [12]. The University of Cambridge has for many years had a lead in the Xen development [19]. The following overview figures are inspired by Xen architecture overview in [20].

Xen essentially consists of two parts; the minimal hypervisor and the controlling machine, called Domain 0. They are both started when the system is booted. The Domain 0 is a dedicated VM, running a special version of Linux as a guest machine. Vital modules, like device drivers and control modules are usually included in Domain 0. It is therefore essential to protect Domain 0 from unauthorized access. External communication with Domain 0 shall be restricted to configuration files and to special messages. No ordinary communication, like logging in, is allowed. Domain 0 is part of TCB, which shall be as small as possible. Proposals to reduce the size of Domain 0 are elaborated later on.

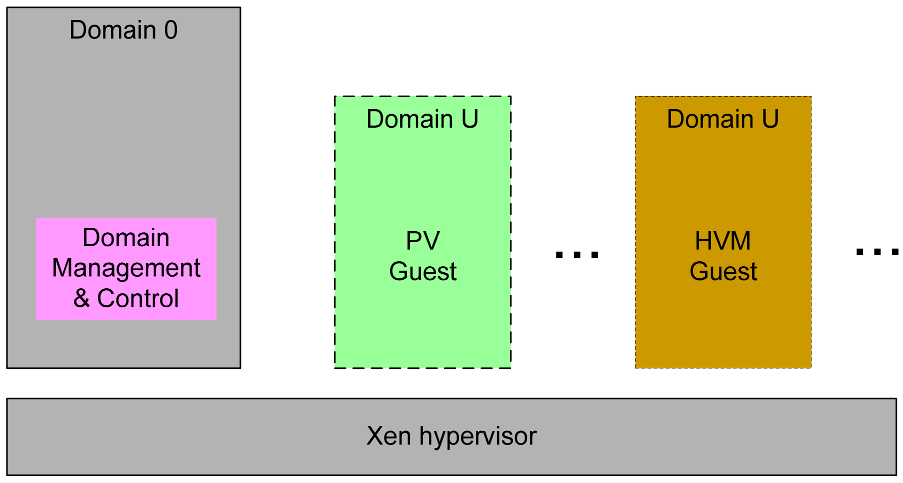

Figure 6 Basic organization of Xen.

Each application guest machine runs in a Domain U (U for User). A Domain U is built, started, stopped and controlled from Domain 0. The system can be sketched as Figure 6. The guests can be of two kinds, paravirtualized, PV, and fully virtualized, HVM[11], respectively. Before version Xen 3.0 only PVs could be built.

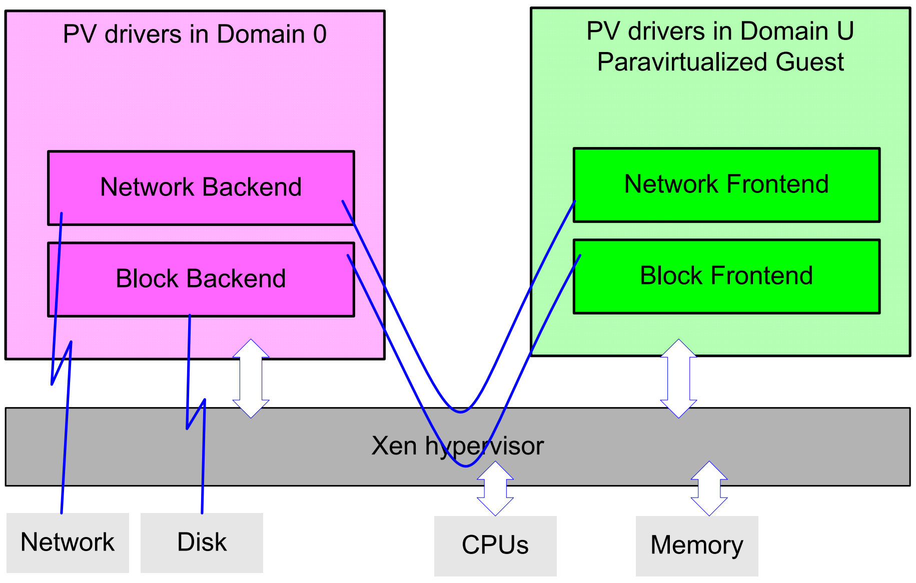

Figure 7 A paravirtualized guest in domain U.

Figure 7 depicts a PV Guest, controlled by Domain 0. Paravirtualization, PV, means that the guest is not fully virtualized; it has been modified in various ways. This is because in some hardware architectures there are instructions which may have side effects. In a PV guest operating system these instructions are replaced by hypercalls, which are trapped and handled by Xen hypervisor.

External devices, like network and disk adapters, are other parts hard to virtualize. The Xen hypervisor provides a bus abstraction for communication between domains, called XenBus. This is used to construct paravirtualized split-drivers for network and disks. The hypervisor monitors the addresses used for access to physical disk/network. The accesses are allowed only from a privileged driver domain, usually domain 0, where backend drivers are run. The hypervisor sets up two XenBuses between the driver domain and each PV guest. These channels are used by frontend drivers in the guest. Thus, conventional I/O drivers in the guests are replaced by frontend drivers, written for each type of guest OS. A XenBus is essentially a buffer in shared memory, synchronized via an event channel. This results in a very efficient communication. The memory sharing is monitored by

[11] The surprising acronym HVM comes from Hardware Virtualized Machine

the hypervisor. It is also possible to set up channels between two PV guests, provided that the guests have proper drivers for the communication at issue.

The Xen hypervisor can virtualize different architectures; notably x86 types from Intel and AMD, but also other processors like PowerPC and ARM for mobile platforms. The older types of x86 processors are hard to virtualize, and the modifications with hypercalls in guest operating systems are rather extensive. Therefore, there are no paravirtualized versions of proprietary OSs, like MS Windows, at least not openly available. PV guests are available for most Linuxes and Unixes. However, recent versions of x86 have extensions which make them more virtualization friendly. Intel's new hardware is called Intel VT [21], and AMD's is called AMD-V [22]. Xen version 3.0 can virtualize these processors, without the need of special hypercalls. The need for paravirtualized split-drivers can be circumvented by use of device emulation, for a restricted set of drivers, by use of the package QEMU [23]. The backend driver in Domain 0 is replaced by qemu-dm, "qemu device manager". The device emulation comes at the price of lower performance, however. But altogether, this means that Xen version 3.0 can run fully virtualized guests, HVM in Figure 6, also for MS Windows.

The security qualities of Xen can be sectioned into two groups, the separation capability and the Trusted Computing Base, respectively. The separation capability means that the hypervisor acts as a separation kernel in that any guest VM runs entirely independent of any other VMs and that there is no uncontrolled way to pass information between VMs. As was sketched in Figure 7, the hypervisor controls allocation of memory and CPUs. The hypervisor is responsible for sanitizing these resources when they are reallocated between VMs. The only way to pass information between two VMs is via the previously described event channel and shared memory. These used to be statically configured in configuration files. However, Xen 3.0 has included extensions, which facilitate policy controlled communication between VMs. One extension, named sHype, originated from IBM. Xen 3/sHype is described in Section 5. It could be mentioned that it is possible to communicate between VMs in the same way as between physical machines, e. g. via TCP-sockets. The policy for this lies outside Xen, however. It is up to the VMs to decide.

The TCB, Trusted Computing Base, is the hypervisor itself, plus Doman 0. According to [24] Xen 3/sHype meets the criteria for level EAL 4, Common Criteria Protection Profile CAPP [25], which is the same level as for some evaluated Linuxes and some MS Windows. No product has been evaluated, however.

In order to reduce the TCB, and purportedly reach higher evaluation levels, there are projects to restructure Domain 0. Domain 0 (configuration management, communication policy,…) runs in privileged mode, which means that it is in the TCB. One project to restructure Domain 0, to make it easier to assure is [26]. Quoted from their report: "In this paper, we introduce our work to disaggregate the management virtual machine in a Xen-based system."

As was earlier mentioned, open source Xen can purportedly be assured to Common Criteria assurance level EAL4. NRL, Naval Research Laboratory, in the US, has analyzed what should be done to make Xen compliant with EAL5. They call such a modified Xen for Xenon [27] [28]. Quoted from their report [27]:

> “This paper explains the Xenon project’s approach to re-engineering Xen’s internal structure into a higher-assurance form. If conventional open source software cannot be brought into this form with moderate amounts of re-engineering then higher-assurance open source software is probably not practical. Our results indicate that moderate amounts of re-engineering will be sufficient for all but a small part of the code. The remaining code is small enough to be addressed in a reasonable time, even though more effort is required”.

Xenon’s primary concern is to act as a high assurance separation kernel, SK. Xenon was realized by restricting a stable version of Xen. Among the restrictions are support for one architecture (x86_64) only, and support for paravirtualization only.

5 XSM, Xen Security Modules

Xen versions 1 and 2 did not have any flexible way to model the security functionality. The only applicable way was to statically configure a set of VMs, as earlier described in Section 4. However, the Xen open source development team is building XSM, Xen Security Modules, available as an option in Xen version 3.

NSA has taken a major role in the XSM development. George Coker, NSA, has made two presentations on XSM, [29] and [30], at two "Xen Summit meetings". Some conclusions from the presentations:

- The rational for XSM is that new usage models for Xen have different security goals. Therefore, there should not be a "hardwired" security model. Xen should rather be capable of supporting many models through configuration, without changes to Xen mechanisms. Examples of new usage models could be decomposing of privileged domain 0, cf. earlier mentioned [26]. Another new usage model could be to isolate, mediate access to, and guarantee invocation of services (e. g. encryption).
- The XSM implementation is derived from Linux Security Modules (LSM), included in Linux kernel 2.6.13.4, which is the basis of Xen. In this Linux kernel there are hooks inserted in places relevant for access control. This essentially means that it is possible to make hooks in the Xen hypervisor which call external modules for access control. These external modules implement the desired policy, e. g. MLS. The rules in the policy are thus concentrated to a few modules, which are more easily configured than the Xen system itself. According to the NSA presentations [29][30], there was no performance degradation due to the hooks in the hypervisor.

In the Xen 3 distribution three security modules are included – Flask developed by NSA, ACM/sHype developed by IBM, and Dummy. The latter is what it says, a dummy module meaning no added security policy.

5.1 XSM Flask

Flask [34], Flux Advanced Security Kernel, was a research project conducted by the Flux group at University of Utah, supported by NSA. It ended around year 2000, and then NSA implemented it as SELinux [31]. Many modern Linuxes can, by using the recently mentioned LSM, be configured into an SELinux.

Two pillars in the Flask architecture are separation between policy enforcement/decision, and type label matching, respectively. In SELinux the enforcement is done at the LSM access hooks, which call a separate decision service. The calls include type labels. The calling subject has type labels bound to it, and so has the requested object. These labels are matched in the decision service. Different policies have been implemented, including RBAC, Role Based Access Control and MLS, Multi Level Security.

This is, not surprisingly, the same approach as in XSM, and NSA accordingly has supported XSM Flask. Quoted from NSA [31]: "The Xen Security Modules (XSM) framework and the Xen Flask security module is an application of the Flask architecture to the Xen hypervisor. This work has been upstreamed to Xen as of Xen 3.2.".

The policy handling in XSM Flask has been reported to be less complex than in SELinux, which has a reputation of being hard to use. However, not many detailed descriptions of XSM Flask have been found. An up to date description of the advancements in SELinux is in [32], where also XSM Flask is briefly mentioned.

5.2 XSM ACM/sHype

IBM has for a long time, ever since their mainframes VM360 and VM370 in the 1970s, been a leading actor regarding research on virtualization. Their research hypervisor, called rHype, was made open source and has been migrated into Xen for support of IBMs Power PC architecture. An extension to rHype, called sHype, for mandatory access control is now included in Xen as XSM ACM/sHype. The basic architecture of XSM Flask and XSM ACM/sHype seems to be the same, but very few detailed papers on XSM Flask have been found.

A paper on sHype is [33]. Three major decisions are the basis of sHype:

1. Build on the existing isolation properties of the hypervisor, which for one thing means minimal code changes in the hypervisor.
2. Use bind-time authorization, to achieve minimal performance overhead. This is implemented as first time access is authorized and the access decision is then cached for use at later accesses.
3. Enforcing formal security policies. Two such policies are implemented in Xen, Chinese Wall and Type Enforcement, respectively.

Both policies are enforced by matching labels bound to the subject (which is requiring the access) and labels bound to the object (which grants/rejects the access). The Chinese Wall policy sets relations between pairs of VMs saying that the two VMs must not run at the same time at the same hardware. This is a means to mitigate the classical covert channel[12], which is a thorny problem. Thus, the policy specifies which combinations that are forbidden. The combinations not specified are allowed. The Type Enforcement policy is the other way around. It specifies "coalitions", which are sets of VMs meant to cooperate. Two VMs may share a virtual resource only if they participate in at least one common coalition. Sharing between other combinations of VMs is forbidden.

The policies are enforced in an extension called ACM, Access Control Module, to the hypervisor. The ACM caches access decisions which have been decided in the Policy Manager, PM, running in its own VM, thus outside the hypervisor. The ACM calls PM when it lacks a cached value. This can be at the first access, or after a request from PM to revoke a cached value. This could for instance be an effect of a changed policy. Since the PM acts as a policy decision point and also as a policy manager, it is crucial for the system and is a part of the TCB. In another VM other security services, like logging and auditing, are run. The architecture can be depicted like Figure 8, which was copied from [18]. The TPM, see Figure 5, could be added to the architecture. The TPM is then used to verify the integrity of policies.

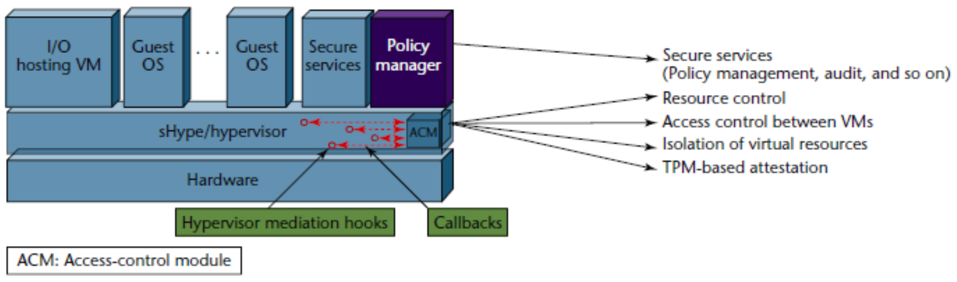
I/O hosting VM
Guest OS
. . .
Guest OS
Secure services
Policy manager
sHype/hypervisor
ACM
Hardware
Hypervisor mediation hooks
Callbacks
ACM: Access-control module
Secure services (Policy management, audit, and so on)
Resource control
Access control between VMs
Isolation of virtual resources
TPM-based attestation

Figure 8 sHype hypervisor security architecture, from [18], © 2008 IEEE.

The labels used in the access control are set by the system owner, for instance as XML-formatted configuration files to the PM. The virtual machine, running PM, shall be isolated from all other machines running user applications. This means that the access control in ACM is a mandatory access control, MAC. As described so far the access control monitors data flow and resource sharing between entire virtual machines.

In [33] a hypervisor call for export of the labels to the VM-layer is described. In case the VM runs a labeled MAC-OS, e. g. SELinux, the labels could be used also in the OS access control. In [35] this is taken one step further. The labels are

[12] A covert channel is an unknown channel where unauthorized information might be transmitted

exported to higher levels, from hypervisor to OS to application, possibly after conversion. This results in a layered policy less complex than a monolithic policy would be, as [33] argues.

Another suggested use of MAC-labels is described in [36]. Their objective is to show a system for distributed MAC, where one common policy authorizes access between VMs at different network nodes. The idea is to have a MAC VM in each node. MAC VMs can establish IPsec tunnels between them. Such a tunnel is a resource that can be labeled and thus controlled by the hypervisor ACM. These labeled IPsec tunnels can be managed in a common policy. A system for distributed MAC is utterly relevant in the quest for object-based security. Note, however, that [36] does not describe labeling of each individual information object. Rather, a label is bound to each IPsec tunnel.

6 Conclusions

The outcome of this theoretic study of publicly available literature on security qualities of virtual machines is that virtual machines play an important role in the undertaking to secure complex and distributed systems. The study has mainly dealt with the open source virtual machine hypervisor Xen. As Xen is an evolving project, it lacks important characteristics like assurance and deployment. Even so, Xen is an essential concept, attracting interest from security organizations, like NSA and NRL in the USA.

In Section 1.2 five topics of interest for the study were listed. Some conclusions for these topics are:

1. Separation capability. A basis for this is called a separation kernel. The Xen hypervisor has most of the properties in such a separation kernel. However, Xen is not evaluated to any assurance level.
2. Configuration. Xen is properly configurable. This conclusion applies to the hypervisor plus chosen control modules. The configuration of a guest virtual machine is essentially the same as the configuration of an equivalent physical machine.
3. Policies. The XSM, Xen Security Modules, facilitates configuration of mandatory security policies. The policies are coarse grained, authorizing interactions between entire virtual machines. There exist research activities to combine virtual machine policies with more fine grained application policies.
4. Assurance. There exists a separation kernel, which allegedly can be used as a virtual machine hypervisor, assured at the US level “high robustness”. Open source Xen is not assured but is considered to reach Common Criteria level EAL4. A restricted Xen is considered to be able to reach level EAL5.
5. Usability and deployment. Xen has been stated to have negligible performance penalty when used in paravirtualized mode. The performance penalty in fully virtualized mode varies.

A final observation is that virtual machines, notably Xen, can be a building block to realize the vision of object-based security. The main contribution in this respect is the potential to build defined isolated and secured environments compliant with mandatory authorization.

References

[1] "Design and Verification of Secure Systems", Reprint of a paper presented at the 8th ACM Symposium on Operating System Principles, Pacific Grove, California, 14–16 December 1981. (ACM Operating Systems Review Vol. 15 No. 5 pp. 12-21), http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.62.1726&rep=rep1&type=pdf

[2] B Randell, J Rushby, "Distributed secure systems: Then and now", Proceedings of the Twenty-Third Annual Computer SecurityApplications Conference, pages 177–198, IEEE Computer Society, Miami Beach, FL, December 2007. Invited "Classic Paper" presentation, http://www.cs.newcastle.ac.uk/publications/trs/papers/1052.pdf

[3] Information Assurance Directorate, National Security Agency, Fort George G. Meade, MD, 20755-6000. U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, June 2007. Version 1.03.

[4] Green Hills Software, INTEGRITY-178B Separation Kernel, http://www.niap-ccevs.org/cc-scheme/st/vid10119/

[5] The Common Criteria Portal, http://www.commoncriteriaportal.org/

[6] National Security Agency, Information Assurance Directorate, Consistency Instruction Manual for development of U.S. Government Protection Profiles for use in Medium Robustness Environments, February 2005. Release 3.0., http://www.niap-ccevs.org/pp/ci_manuals.cfm

[7] NATIONAL SECURITY AGENCY, "INFORMATION ASSURANCE GUIDANCE FOR SYSTEMS BASED ON A SECURITY REAL-TIME OPERATING SYSTEM", SSE-100-1, December 2005 http://www.nsa.gov/ia/_files/SSE-100-1.pdf

[8] Green Hills Software Inc, http://www.ghs.com/

[9] Open Trusted Computing (OpenTC) consortium, http://www.opentc.net/

[10] General activities of OpenTC, http://www.opentc.net/index.php?option=com_content&task=view&id=13&Itemid=28

[11] The L4 µ-Kernel Family,
http://os.inf.tu-dresden.de/L4/

[12] Xen Hypervisor ,
http://www.xen.org/

[13] Srijith K. Nair, Patrick N.D. Simpson, Bruno Crispo, Andrew S. Tanenbaum, "Trishul: A Policy Enforcement Architecture for Java Virtual Machines", http://www.few.vu.nl/~srijith/publications/techreports/Trishul-IR-CS-45.pdf.

[14] Virtualization and Security: Back to the Future, IEEE Security&Privacy, 2008, volume 6, Issue 5,
http://ieeexplore.ieee.org/xpl/tocresult.jsp?isYear=2008&isnumber=4639007&Submit32=View+Contents

[15] NetTop, NSA,
http://www.nsa.gov/research/tech_transfer/fact_sheets/nettop.shtml

[16] J. Alves-Foss, P. W. Oman, C. Taylor, W. S. Harrison, "The MILS architecture for high-assurance embedded systems", International Journal of Embedded Systems, Volume 2, Number 3-4 / 2006, pp 239-247,
http://inderscience.metapress.com/app/home/contribution.asp?referrer=parent&backto=issue,9,10;journal,6,9;linkingpublicationresults,1:110847,1

[17] The Trusted Computing Group,
https://www.trustedcomputinggroup.org/home

[18] R. Perez, R. Sailer, L. van Doorn, Virtualization and Hardware-Based Security, IEEE Security&Privacy, 2008, volume 6, Issue 5, pp 24-30

[19] The Xen™ virtual machine monitor,
http://www.cl.cam.ac.uk/research/srg/netos/xen/

[20] Xen Architecture Overview,
http://wiki.xensource.com/xenwiki/XenArchitecture?action=AttachFile&do=get&target=Xen+Architecture_Q1+2008.pdf

[21] Virtualization technologies from Intel,
http://www.intel.com/technology/virtualization/

[22] AMD Virtualization,
http://www.amd.com/virtualization

[23] Qemu homepage,
http://www.nongnu.org/qemu/

[24] R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, L. var Doorn, "sHype: Mandatory Access Control For XEN",

http://www.docstoc.com/docs/5691508/sHype-Mandatory-Access-Control-For-XEN

[25] CONTROLLED ACCESS PROTECTION PROFILE, http://www.commoncriteriaportal.org/files/ppfiles/capp.pdf

[26] D. G. Murray, G. Milos, S. Hand, "Improving Xen security through disaggregation", Virtual execution environments ´08, http://portal.acm.org/citation.cfm?id=1346278

[27] J. McDermott, J. Kirby, B. Montrose, T. Johnson, M. Kang, "Re-engineering Xen internals for higher-assurance security", Center for High Assurance Computer Systems, Naval Research Laboratory, USA, Information security technical report 13 (2008) 17–24, Elsevier, available at www.sciencedirect.com

[28] J. McDermott, Naval Research Lab, "Xenon Assurance Modifications to Xen Code", http://www.xen.org/files/xensummitboston08/XenSummitSpring08.pdf

[29] G. Coker, National Information Assurance Research Lab, National Security Agency (NSA), "Xen Security Modules (slides)" - Xen Summit, July 2006 http://xen.xensource.com/files/summit_3/coker-xsm-summit-090706.pdf,

[30] G. Coker, National Information Assurance Research Lab, National Security Agency (NSA), "Xen Security Modules (slides)" - Xen Summit, April 2007 http://xen.org/files/xensummit_4/xsm-summit-041707_Coker.pdf, 2007.

[31] SELinux Related Work, http://www.nsa.gov/research/selinux/related.shtml

[32] J. Morris, Red Hat Asia Pacific Pte Ltd, "Have You Driven an SELinux Lately?", An Update on the Security Enhanced Linux Project, http://kernel.org/doc/ols/2008/ols2008v2-pages-101-114.pdf

[33] R. Sailer , T. Jaeger , E. Valdez , R. Caceres , R. Perez , S. Berger , J. L. Griffin , L. van Doorn, "Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor", Proceedings of the 21st Annual Computer Security Applications Conference, p.276-285, December 05-09, 2005 [doi>10.1109/CSAC.2005.13], http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?tp=&arnumber=1565255&isnumber=33214

[34] Flask: Flux Advanced Security Kernel, http://www.cs.utah.edu/flux/fluke/html/flask.html

[35] B. D. Payne, R. Sailer, R. Caceres, R. Perez, W. Lee, “A Layered Approach to Simplified Access Control in Virtualized Systems”, ACM SIGOPS Operating Systems Review, 2007, http://portal.acm.org/citation.cfm?id=1278905

[36] J. M. McCune, T. Jaeger, S. Berger, R. Caceres, and R. Sailer, ‘‘Shamon—A System for Distributed Mandatory Access Control,’’ Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC), Applied Computer Security Associates, 2006, pp. 23–32

[37] P. A. Karger, D. R. Safford, I/O for Virtual Machine Monitors – Security and Performance Issues, IEEE Security&Privacy, 2008, volume 6, Issue 5, pp 16-23

[38] A. M. Antonopoulos, “Virtualization Risk Analysis”. Memertes Research Group Inc., 18 March, 2008. http://www.nemertes.com/issue_papers/virtualization_risk_analysis (2009-11-10)

[39] N. MacDonald, “Hypervisor Attacks in the Real World”. 20 February, 2009. http://blogs.gartner.com/neil_macdonald/2009/02/20/hypervisor-attacks-in-the-real-world/ (2009-11-10)

[40] National Vulnerability Database. National Institute of Standards and Technology (NIST). http://nvd.nist.gov/ (2009-11-10)

[41] T. Ormandy, “An Empirical Study into the Security Exposure to Hosts of Hostile Virtualized Environments”. 2007. http://taviso.decsystem.org/virtsec.pdf (2009-11-10)