Virtualization: Issues, Security Threats, and Solutions

MICHAEL PEARCE, The University of Canterbury
SHERALI ZEADALLY, University of The District of Columbia
RAY HUNT, The University of Canterbury

Although system virtualization is not a new paradigm, the way in which it is used in modern system architectures provides a powerful platform for system building, the advantages of which have only been realized in recent years, as a result of the rapid deployment of commodity hardware and software systems. In principle, virtualization involves the use of an encapsulating software layer (Hypervisor or Virtual Machine Monitor) which surrounds or underlies an operating system and provides the same inputs, outputs, and behavior that would be expected from an actual physical device. This abstraction means that an ideal Virtual Machine Monitor provides an environment to the software equivalent to the host system, but which is decoupled from the hardware state. Because a virtual machine is not dependent on the state of the physical hardware, multiple virtual machines may be installed on a single set of hardware. The decoupling of physical and logical states gives virtualization inherent security benefits. However, the design, implementation, and deployment of virtualization technology have also opened up novel threats and security issues which, while not particular to system virtualization, take on new forms in relation to it. Reverse engineering becomes easier due to introspection capabilities, as encryption keys, security algorithms, low-level protection, intrusion detection, or antidebugging measures can become more easily compromised. Furthermore, associated technologies such as virtual routing and networking can create challenging issues for security, intrusion control, and associated forensic processes. We explain the security considerations and some associated methodologies by which security breaches can occur, and offer recommendations for how virtualized environments can best be protected. Finally, we offer a set of generalized recommendations that can be applied to achieve secure virtualized implementations.

Categories and Subject Descriptors: Software [**Operating Systems, Security and Protection, Performance**]; Computer Communication Networks [**Network Protocols, Local and Wide-Area Networks, Network Operations, Network Architecture and Design**]

General Terms: Design, Security, Reliability, Performance

Additional Key Words and Phrases: Encryption, virtualization, threat, virtual machine, virtual machine monitor

ACM Reference Format:
Pearce, M., Zeadally, S., and Hunt, R. 2013. Virtualization: Issues, security threats, and solutions. ACM Comput. Surv. 45, 2, Article 17 (February 2013), 39 pages.
DOI = 10.1145/2431211.2431216 http://doi.acm.org/10.1145/2431211.2431216

Authors' addresses: M. Pearce, Department of Computer Science and Software Engineering, The University of Canterbury, Christchurch, New Zealand; S. Zeadally (corresponding author), Department of Computer Science and Information Technology, University of the District of Columbia, Washington, DC; email: szeadally@udc.edu; R. Hunt, Department of Computer Science and Software Engineering, The University of Canterbury, Christchurch, New Zealand.
Permission to make digital or hard copies of part or all of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies show this notice on the first page or initial screen of a display along with the full citation. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, to republish, to post on servers, to redistribute to lists, or to use any component of this work in other works requires prior specific permission and/or a fee. Permissions may be requested from Publications Dept., ACM, Inc., 2 Penn Plaza, Suite 701, New York, NY 10121-0701 USA, fax +1 (212) 869-0481, or permissions@acm.org.
© 2013 ACM 0360-0300/2013/02-ART17 $15.00
DOI 10.1145/2431211.2431216 http://doi.acm.org/10.1145/2431211.2431216

1. INTRODUCTION

Although it may seem that system virtualization[1] is a development of the last decade or so, it has a history of research going back over 40 years [Fuchi et al. 1969; Rosin 1969], with some of the foundational research undertaken in the early 1970s [Goldberg 1973, 1974]. The last decade has seen an explosion of development in the area of system virtualization, as the applications of virtualization of commodity hardware have spurred an ever-increasing set of uses.

In essence, system virtualization is the use of an encapsulating software layer that surrounds or underlies an operating system and provides the same inputs, outputs, and behavior that would be expected from physical hardware. The software that performs this is called a Hypervisor, or Virtual Machine Monitor (VMM). This abstraction means that an ideal VMM provides an environment to the software that appears equivalent to the host system, but is decoupled from the hardware state[2]. For system virtualization, these virtual environments are called Virtual Machines (VMs), within (or upon) which operating systems may be installed. Since a VM is not dependent on the state of the physical hardware, multiple VMs may be installed on a single set of hardware.

Figure 1 shows an example implementation of system virtualization, where replicas of seven physical systems (running six different operating systems) at the top of the figure execute on a single hardware system (at the bottom) through the use of a VMM.

A virtual machine is the logical equivalent of a physical one, and multiple VMs on the same hardware are (ideally) as logically separate as air-gapped machines are physically, or (if networked) as logically separate as different systems on the same network. This isolation of VMs, when combined with the equivalence mentioned earlier, has many implications for security. However, the realities of a system's security implications in the real world are set not just by theory, but also by factors such as assumptions, implementation specifics, and user priorities. System virtualization is no exception.

This work is intended as an introduction to the security concerns, considerations, and implications arising from use of virtualized systems. In Section 2, we present some basic concepts of system virtualization. Section 3 presents the system virtualization architecture. We describe some of the motivating factors for virtualization's use to improve security in Section 4. We discuss what security means in virtualization in Section 5. The security threats that arise as a result of strong virtualization properties are presented in Section 6. We present some security implications that can result from weak virtualization properties in Section 7. We discuss security implications resulting from the virtualization control and network channels in Section 8. We make some recommendations that can be deployed to achieve secure virtualization implementations in Section 9. Finally, we make some concluding remarks in Section 10.

2. BACKGROUND

2.1. Basic Concepts

Most computer systems require the running of more threads than the processor can support directly. On a single-core consumer machine there is only a single thread of CPU execution.[3] Operating Systems (OSs) allow a higher level of interface for software by not only offering process multiplexing, but also handling many of the underlying hardware

[1]Also known as platform virtualization, but we will continue to use the term system virtualization.

[2]However, in reality most VMMs do not provide fully equivalent environments, merely sufficiently similar ones.

[3]For simplicity we ignore features such as hyperthreading.

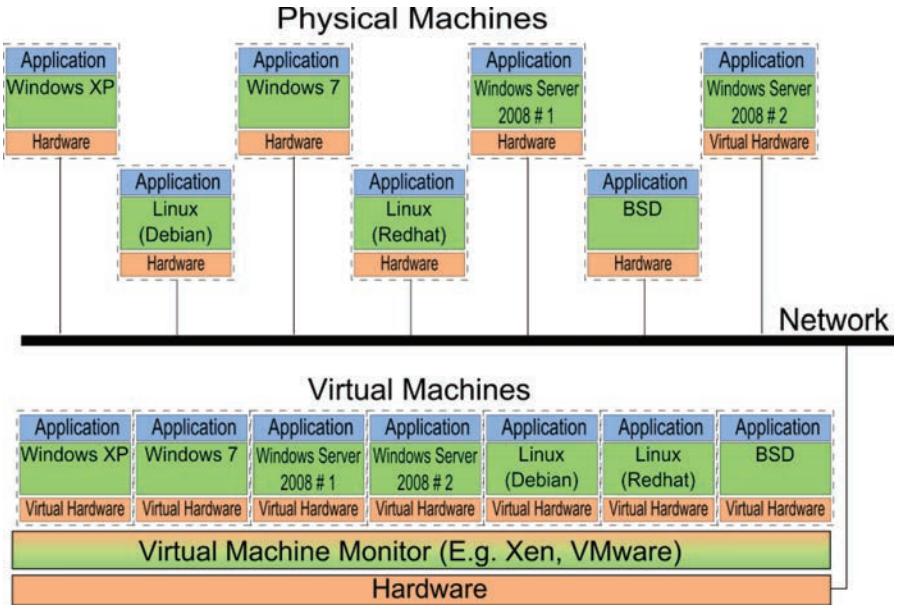

Fig. 1. Seven physical systems (top) and a virtualized equivalent implementation (bottom).

management issues. Thus, operating systems offer a level of abstraction above the hardware, on which multiple processes can run concurrently. This arrangement only enables a single operating system to operate on a single hardware system at any given time in most circumstances.

Operating systems operate on the hardware as privileged software, and are generally able to perform any operation the hardware supports, whereas programs running inside an operating system are less privileged, and generally cannot perform operations except those that the operating system permits. These privilege levels are often called *rings*, with the lower numbered ring (i.e., ring 0) having higher privileges than those with higher designations [Bratus et al. 2009]. Operating system kernels generally run in the lowest ring, and thus have control over everything running in the lower privilege (higher numbered) rings.

A Virtual Machine Monitor is a highly privileged piece of software that runs either alongside or under an operating system, it is designed to be "*an* efficient, isolated duplicate *of the real [physical] machine*" [Popek and Goldberg 1974]. Furthermore, it is possible for a single VMM to run on multiple networked physical systems. In this article we will discuss VMMs as they relate to a single hardware system.

A *virtual machine monitor* is distinct from an *emulator*. An emulator intercepts all instructions, whereas a virtual machine monitor need only intercept sensitive instructions (those which could interfere with the operation of the VMM itself). All nonsensitive instructions should be executed directly on the hardware where possible.

There are two main types of virtual machine monitor: *Type I* (Bare Metal) and *Type II* (Hosted). These are described briefly next, and illustrated in Figure 2.

—A classical *Type-I* VMM is installed as the primary boot system on the hardware. The VMM executes at the highest level of privilege and has full control over any virtual machines that use it.

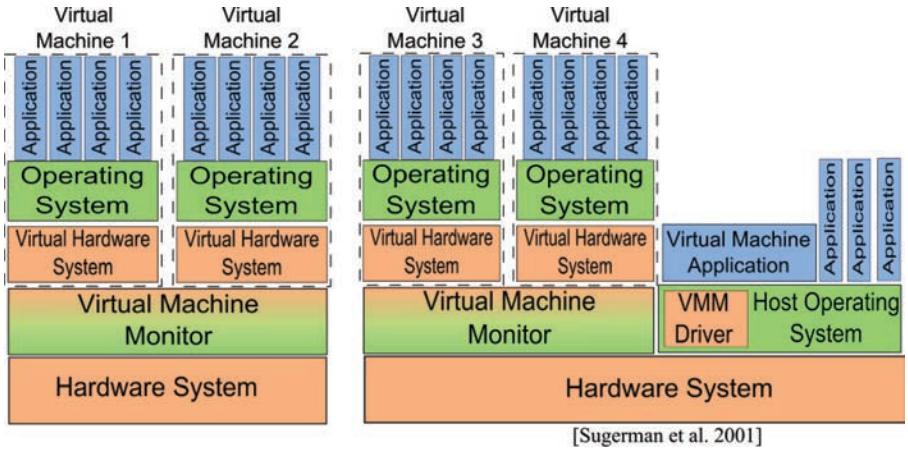

Fig. 2. Generalized architectures for Type-I (left) and Type-II (right) Virtual Machine Monitors [Sugerman et al. 2001].

—A hosted (including *Type II)* VMM, as popularized by products such as VMware Workstation, has a more complicated architecture.[4] A hosted VMM sits alongside or above a host operating system above the hardware, and may share drivers from the host operating system to handle Input/Output (I/O). This cooperative model results in a VMM system that does not require hardware-specific drivers for VMM I/O operations, and allows the use of virtual machines within an existing environment. As a result entry barriers to VM use are reduced, as the existing OS does not need to be overwritten or migrated to a multiple boot arrangement [Sugerman et al. 2001].

2.2. Motivations for System Virtualization

System virtualization is widely used for a variety of applications, such as, among other things, the consolidation of physical servers [Scott et al. 2010], isolation of guest OSs, and software debugging [Bratus et al. 2008].

There are many other uses to which system virtualization lends itself, and many different motivators for adoption of system virtualization technologies. System virtualization has been attracting a lot of attention, particularly in the last case, because of various technological trends. Some of these trends include increasing commodity operating system complexity, increasing cost of supporting hardware and software systems, and the availability of inexpensive, powerful, and flexible commodity hardware [Ivanov and Gueorguiev 2008; Wlodarz 2007].

A modern commodity OS such as Windows or Linux is very complex (tens of millions of Lines Of Code (LOC) in the latest desktop OSs), and this results in a much larger vulnerability surface than can be easily or provably secured [Franklin et al. 2008a; Seshadri et al. 2007]. Furthermore, OSs add a single point of failure for everything (processes and data) in them. This difficulty in securing a single complicated point of failure represents a security risk for the data and processes on the system. Consequently, with ever-decreasing commodity hardware costs, most organizations achieve

[4]Although strictly speaking a Type-II VMM is hosted, a hosted VMM is not necessarily a Type-II VMM. A hosted VMM is so-called due to its reliance upon a host OS. Simply speaking, a Type-II VMM sits roughly alongside the host OS and shares drivers, whereas a hosted VMM may sit above the host OS, share no drivers, and handle all hardware calls via the host OS. In this work we shall use the term hosted in the broadest sense.

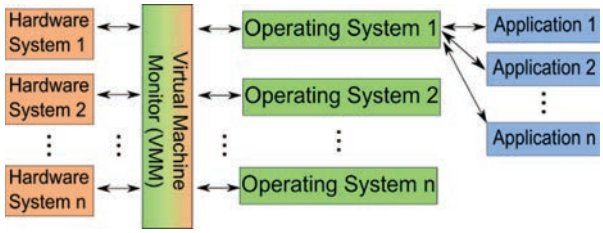

Fig. 3. The extra layer of abstractions that a VMM offers.

different operational and security-based requirements through the use of multiple physical systems.

The deployment of multiple physical systems to mitigate potential security risks stemming from a single point of failure has been enabled by increasing performance and flexibility, and reductions in the price of commodity hardware and networks. However, physical systems are associated with other significant costs (that include operational, physical, and technical) in addition to the initial purchasing outlay. Each physical machine requires physical space, cabling, energy, cooling, and software administration. Additionally, physical separation adds communication overheads such as data transfer and storage latencies. For some classes of problems workarounds and optimizations exist (such as improved caching for frequently accessed data, or power management to reduce energy costs) but for others these costs are hard to overcome or justify. Since every physical system has a cost, one of the most important issues that arises to be avoided is systems that are underutilized.

Underutilization of computer systems comes in two main relevant forms: desktop machines that are rarely used to their full potential (for instance, systems left operational overnight for maintenance purposes, but generally remaining idle [Domingues et al. 2005; Newsham and Tiller 1994]), and organizational systems or servers that are not running efficiently [Vasan et al. 2010; Kist 2009; Barroso and Hölzle 2007]. The spare cycles are available for use on desktop systems, and organizations have an interest in consolidating physical hardware where possible to keep overheads low and efficiency up.

2.3. Implications of Virtualization

By removing the dependency of operating systems on a system's physical state, system virtualization allows multiple operating systems to be installed on a VMM, and thus multiple operating system VMs (called *guest operating systems*) can be installed on each physical system. Allowing multiple VMs on the same hardware offers many advantages. Near-complete isolation between guest operating systems on the same hardware protects against OSs being a single point of failure. It also allows OS consolidation from different machines as is necessary to reduce system underutilization and maintain efficiency of operation.

This abstraction from the hardware state allows not only multiple operating systems to coexist on the same hardware, but for one VMM to run on multiple different networked physical systems concurrently. By utilizing a VMM to mediate between the OS and the hardware, virtualization changes the one-to-one mapping of OSs to hardware (as shown in the top part of Figure 1) to many-to-many (as shown in Figure 3).

Although many real-world systems implement this model only loosely, as a VM does not usually run on multiple systems concurrently, allowing one VMM to be migrated across multiple physical systems seamlessly while running has improved the offerings for high-performance and high-availability systems and cloud computing, as well as

for the commoditization of processing power in general. While we focus in this article on system virtualization, there are many other virtualization technologies that overlap with what we discuss, such as storage virtualization and network virtualization. We discuss these briefly in Section 2.6.

2.4. System Virtualization

The requirements for system virtualization are defined and discussed in detail in Popek and Goldberg's *Formal Requirements for Virtualizable Third-Generation Architectures* [Popek and Goldberg 1974]. The requirements and definitions given are still used to define virtualization, and their criteria are used to assess VMMs, although the criteria have become broader (as we discuss when we describe hybrid virtualization strategies later). Virtualization as we describe it in this section is classical virtualization as defined by Popek and Goldberg [1974] and used in Adams and Agesen [2006]. In the later section of this work, we describe methods of virtualization that do not strictly fit these requirements. To explain the requirements for a classically virtualizable CPU architecture, we need to define two properties an instruction must have, and three properties of a virtualized architecture. A more in-depth discussion on virtualizability of CPU architectures can be found in Adams and Agesen [2006].

Two *Instruction Properties*[Popek and Goldberg 1974]:

> *Privilege level: Privileged, or nonprivileged*. Does this instruction require a process to be highly privileged to call it directly? If the CPU *traps* and switches control to supervisory software (running in low rings) when the instruction is called from a process running in user mode (high rings) then the instruction is privileged, as it requires privilege to be executed.
>
> *Sensitivity: Sensitive or innocuous*. Does this instruction have the capacity to interfere with something that the VMM should have complete control of? (A *sensitive* instruction has the capacity to interfere with VMM operation, whereas an *innocuous* one does not.) For example, reading VMM program memory will not interfere with VMM behavior, but writing to it could.

Three *Virtualized Architecture Properties* [Popek and Goldberg 1974]:

> *The Efficiency Property* states that a VMM must run innocuous instructions directly on the CPU, and not intervene where it is not necessary.
>
> *The Resource Control Property* requires that a VMM retain full control over resources, and it must not be possible for a virtualized guest to access or manipulate resources without the VMM's explicit authorization.
>
> *The Equivalence Requirement Property* requires that a VMM perform indistinguishably from a physical machine given the same context, with two exceptions added by VMM overheads: namely, timing and resource availability. Timing will differ because of processing overheads introduced by the VMM, and resources available will differ because the VMM utilizes some resources.

In order to be virtualizable, an architecture must be capable of trapping all sensitive instructions and calling the VMM. An architecture is fully virtualizable "...if the set of sensitive instructions for that computer is a subset of the set of privileged instructions" [Popek and Goldberg 1974] If this is not the case, and some sensitive instructions are not capable of being trapped, then the architecture is not virtualizable (as shown in Figure 4).

For nonvirtualizable architectures, other measures can be taken to construct a *Hybrid Virtual Machine*, where the problem instructions (as shown in Figure 4) are dealt with by nonvirtualized means. This will involve not strictly meeting the efficiency property in order to maintain the resource control property. The efficiency property is what differentiates virtualization from emulation: a full VMM will intervene only on

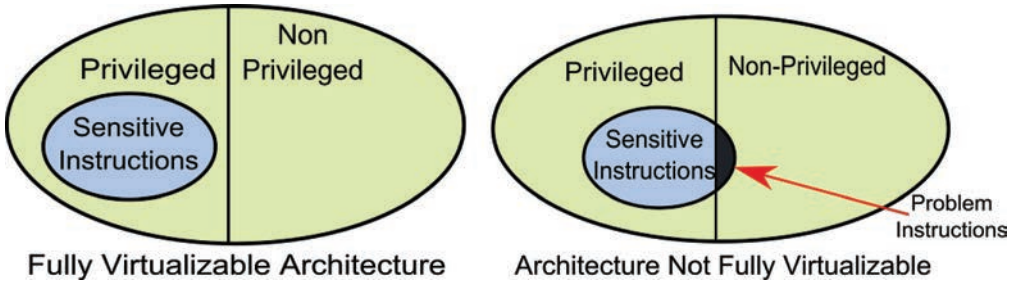

Fig. 4. The relationships between instruction sensitivity and privilege as relates to architecture virtualizability.

sensitive instructions, and the innocuous instructions will be executed on the hardware without VMM intervention. Conversely, a full emulator will intercept instructions whether sensitive or innocuous. This makes a VMM usually, but not always, far more computationally efficient than an emulator (as we explain in the case of binary translation in the next section).

It is also worth noting that with regard to resources, a modern VMM can actually offer a very different level of resources than strictly equivalent to the host minus overheads. This is because of additional features such as disk compression, I/O virtualization, deduplication, and shared paging.

2.5. Virtualizing Nonvirtualizable Architectures

The x86 architecture is not fully virtualizable [Adams and Agesen 2006; Garfinkel et al. 2007; Rose 2004; Rosenblum and Garfinkel 2005], so various methods have been taken to achieve the widespread virtualization of these systems now in use. The most important of these are paravirtualization, binary translation, and hardware-assisted virtualization [Advanced Micro Devices 2008]. For more in-depth discussion on each of these methods and their relative performances refer to Adams and Agesen [2006].

Binary translation is very similar to emulation, and involves running guest code (both OS and application code) on an interpreter that handles any sensitive instructions correctly. However, this method can have a heavy performance overhead, which optimizations are used to overcome. Examples of optimizations include switching between virtualized and translated instructions depending on the privilege level of the code according to the guest VM, and adaptive binary translation, which changes the code being translated in an effort to improve performance (in some cases outperforming a classical VMM because intercepting instructions causes a context switch which uses a lot of clock cycles on current hardware, a situation which binary translation can minimize [Adams and Agesen 2006]).

Paravirtualization involves porting guest operating systems so that they do not use Nonprivileged sensitive instructions, but instead use ones that better cooperate with the VMM [Barham et al. 2003; Rose 2004]. The necessary guest OS modifications have been publicly released for the Linux kernel [Yamahata 2008], and the process has been discussed (but no working port released due to intellectual property reasons) for Windows XP [Barham et al. 2003]. There are, however, various paravirtualized device drivers available for Windows XP that come with some commercial and open-source products. Paravirtualized device drivers are designed to operate using Nonsensitive instructions from the guest OS (in which they are installed), and to interface in a way that minimizes context switching while operating in a way that is transparent to the guest OS.

Hardware assisted virtualization. Intel and AMD have implemented additional functionality to their CPUs to support hardware virtualization. The x86 hardware virtualization technologies used at present are Intel's Virtualization Technologies VT-X and VT-I [Uhlig et al. 2005]), and AMD's AMD-V [Advanced Micro Devices 2010] (formerly Secure Virtual Machine or SVM). The use of these technologies for virtualization offers a better level of CPU equivalence, and when combined with other technologies that can assist virtualization such as I/O Memory Management Units (IOMMU) (which handle VM memory to physical mappings in hardware)[5], very high levels of execution transparency and performance can be achieved. Note that the concept of full transparency remains controversial, as discussed briefly in Section 6.2. Virtualization is used in more than just physical systems, as we will discuss in the next section.

2.6. Other Virtualization Types

Virtualization is used to abstract resources and devices in general. The most important types discussed in this work are as follows.

—System (platform or server) virtualization is the main focus of this work.
—Network virtualization can be of two types:
  —External: imposing a logical view of a physical network, used in Virtual Local Area Networks (VLANs), and Virtual Private Networks (VPNs) [Chowdhury and Boutaba 2009].
  —Internal: a "network in a box" implementing networks using pseudo network devices that appear to behave like physical devices. This is used to control communications between software, virtual machines, or to provide the appearance of a network to external devices.

In addition, the following technologies and uses of the term virtualization are often encountered.

—Storage virtualization: providing a different logical view of physical storage. Usually this is a single consolidated volume that spans multiple physical devices. Storage Area Networks (SANs) are a common example of this type of virtualization [Soundararajan and Anderson 2010], and are also often used to store the disk images from system virtualization.
—Application or container virtualization: this is a technique where the underlying operating environment of an application is virtualized. This will commonly be the operating system surroundings, and the result is an isolated container in which the application can run. Examples of this include application virtualization [Reuben 2007], FreeBSD-style jails, and some methods of sandboxing [Laadan and Nieh 2010].
—Desktop virtualization has two quite different usages.
  —Thin client - remote desktop usage which is similar to application streaming, except that it is targeted at the user experience. The entire operating OS appears to the user to be running locally, but is in fact running elsewhere. This is also referred to as Virtual Desktop Infrastructure (VDI) [Miller and Pegah 2007].
  —System virtualization in a Type-II environment - used by some to refer to the running of a virtualized machine on a standard workstation. Generally this is an incorrect usage of the term *desktop virtualization*, as the whole OS is virtualized and not just the desktop.

[5]An IOMMU allows hardware management of memory mapping.

2.7. Scope of This Work

This work will focus on particular areas and issues of importance in areas related to the security of system (or platform) virtualization, of which network virtualization is an important part. The virtualization market has many vendors/manufacturers at present. Some of the most important ones include VMware, Citrix, and Microsoft, but there are many other offerings both open-source and commercial. However, it is worth noting that most of the published literature has been focusing on VMware products, Xen and Xen-based products, Microsoft's products, and the open-source projects KVM, QEMU, and Bochs.

Our security discussions on virtualization will remain generally platform independent, and we will not focus on any specific provider's products. However, we will use predominant products in examples, since most research efforts discuss or use them. The commodity hardware we will refer to is the most common at present, the x86 and x64 architectures. The virtualization platforms we focus on will be proprietary VMware products and the open-source Xen hypervisor due to the volume of research literature available on them.

In this work we do not discuss desktop, storage, or application virtualization because of space limitations. Furthermore, we also do not discuss many of the delivery methodologies or models for virtualized services. Therefore, we will not discuss cloud computing delivery models or similar issues. A more in-depth discussion of security issues in cloud computing can be found in Subashini and Kavitha [2011], Christodorescu et al. [2009], Ristenpart et al. [2009] and Siebenlist [2009].

3. SYSTEM VIRTUALIZATION ARCHITECTURE

Virtualization has a large number of functional components, and most are used to improve the way it is used and controlled. It is worth noting that the architecture and data flows we discuss in this work are fairly broad, and do not apply in every case of virtualization. Some virtualization implementations do some things differently, although most of the core components are the same.

3.1. Key System Components

Figure 5 shows the components in a simplified Type-II VMM system with two guest operating systems installed and VMM components labeled. Note that a Type-I VMM system is comprised of a subset of the components given in Figure 5, and does not contain a host OS (5), VMM drivers[6] (6), a VM application (7), or any applications executing over the host OS.

A virtualized system will then generally always have some hardware (4), a VMM (3), and at least one virtual machine that consists of virtual hardware (2) running a guest operating system (1). The guest operating system also has applications running on it. This is a general model, and different implementations may use slightly different ways to structure and organize components and separation.[7]

3.2. High-Level Data Flows and Entry Points

The control and data flows inside a platform that implements system virtualization are of two primary types: software or control channels, and network channels. Figure 6 gives a detailed view of the data flow between the important components of

[6]Drivers which may be shared with the host OS, may access hardware directly, or alternatively may not be used, with all device calls going via the host OS.

[7]For instance, some Type-I implementations implement a control VM that runs in a trusted domain and is used to configure the VMM operation (e.g., Xen, Terra). We have not shown the control VM in this diagram, but it is included in the data flow diagram Figure 6.

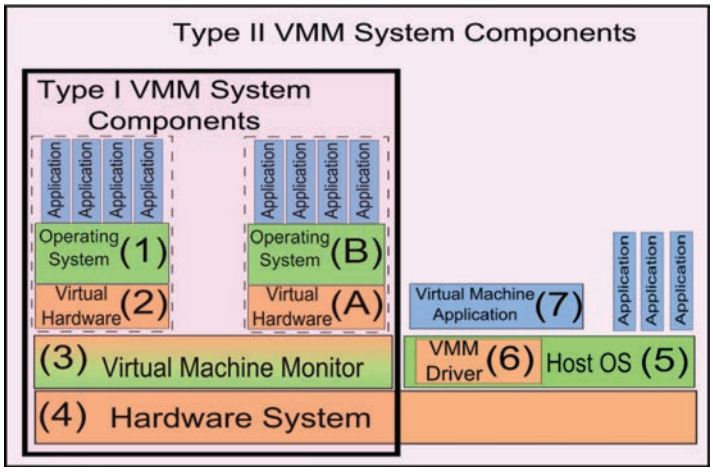

Fig. 5. The core components in virtualization architectures.

virtualization systems (both Type-I and Type-II components are shown on the same diagram). Similar to Figure 5 we have included Type-I and Type-II components on the same diagram.

It is worth pointing out that in Figure 6 much of the overall architecture follows a layered stack model, with each component communicating with the components above and below it in the stack. There are a few communication flows that do not strictly fit this model, with some horizontal communications between applications and the network, as shown in Figure 6.

3.3. Control Channel Data Flows

A VMM which performs anything other than the minimal core requirements (supplying a virtualized system, with no additional features such as cloning or networking) requires control channels to allow configuration and management of the VMM and its child VMs. Such management operations include changing settings, controlling the operational status of VMs and the VMM, and facilitating screen and keyboard access. These control channels can be either software or network based, but in either case they offer a high level of access should they be compromised. The main channels are to control the VMM and the VM.

The VM control channels (as described shortly) are used to control and automate actions in the guest operating system that are useful to the VMM by interacting with drivers or other software installed. Examples of these actions include initiating operating system shutdown, transferring files, and running software on the guest OS. The VMM control channels (described in Section 8.1) are used to control more general operations (such as shutting down or restarting VMs, and changing VM settings) to the virtual machine and VMM.

3.4. Network Communication Data Flows

Software is no more secure simply by virtue of being in a virtualized environment. It is, however, true that software resides in a more tightly managed environment as a result of virtualization. Furthermore, since virtualization is often used for networked software, the software is exposed to both internal and external threats on the network. Threats that come through networking can affect the VMM, software, and operating systems running in the VMM, and other software applications.

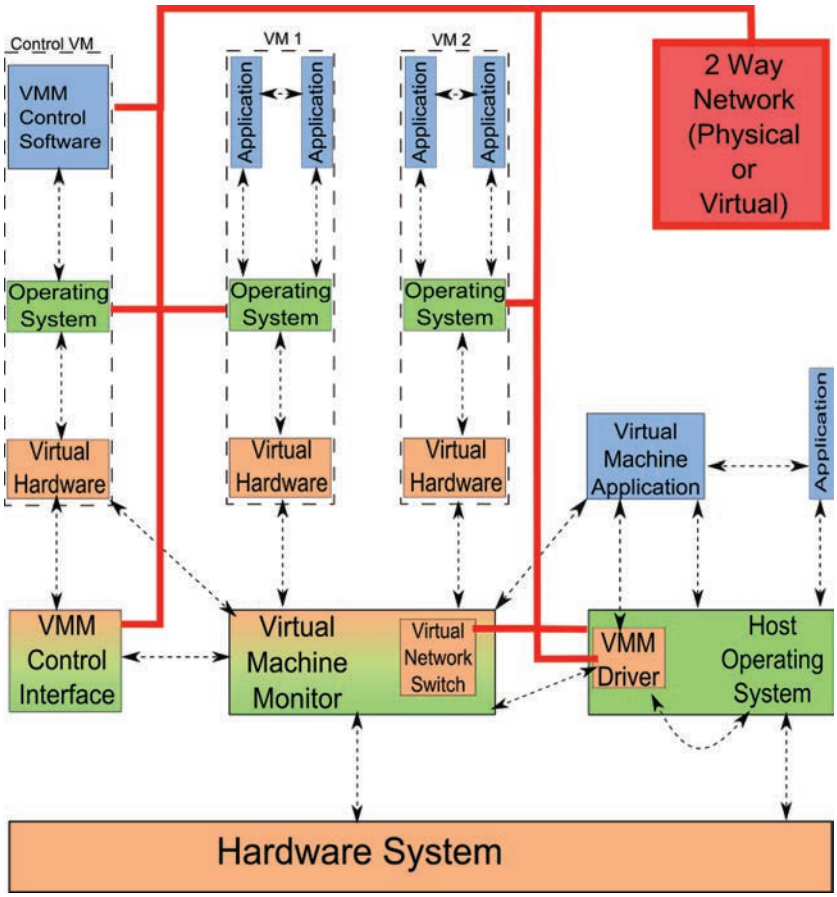

Fig. 6. Data flow composite for Type-I and Type-II VMM implementations. (Type-I systems use components above and to the left of the VMM, while Type-II VMMs use components above and to the right of the VMM.)

3.5. Network Operations

Networking of virtual machines is achieved with the VMM either bridging the virtual adapter to a physical adapter, or acting as a (virtual) network hub/switch on a logical virtual network and facilitating further network routing or bridging from the virtual hub to other networks (virtual or not). The basic types are internal/isolated, bridged (to a host adapter), routed/switched (through the host adapter), and Network Address Translation (NAT) routed (through the host's IP address). A bridged adapter does not need to use virtual networks, through the use of virtual hubs/switches, but the other types use them. Figure 7 shall be used to discuss the related concepts.

A *bridged* virtual network adapter will send and receive traffic on a VM's virtual network adapter directly to a network adapter on the host at layer 2. A bridged adapter appears to be directly connected to the network, and can send, receive, and listen to traffic on the physical network. This usually occurs with little to no traffic intervention from the host (firewall rules, MAC address, or NAT modifications, etc.)[8]. Thus (in Figure 7) operating systems 1 and 2 appear to be on physical network 1, and can

[8]However, whether or not it can act in a promiscuous fashion and see all traffic including that not addressed to it is dependent upon the VMM, drivers, and security policy in use.

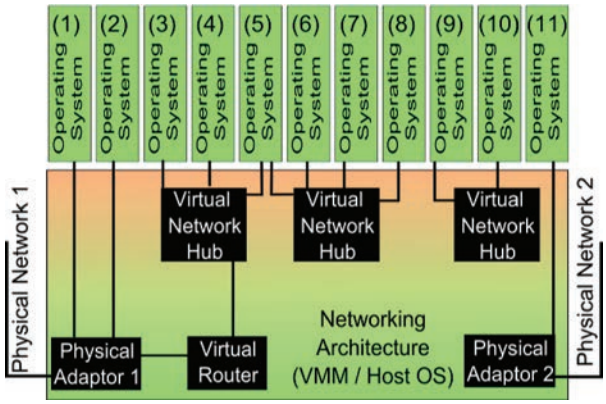

Fig. 7. Ten VMs connected to virtual networks in various arrangements.

communicate with all devices on it, and each other, while operating system 11 can see and send traffic on physical network 2 and only communicate with other systems that the network is connected to.

A virtual network has all of its devices connected to the same virtual hub, and connects these hubs to each other, or the real world, via virtual routers. Operating systems 3, 4, and 5 (shown in Figure 7) are on the same virtual network hub, while 5, 6, 7, and 8 are on another and 9 and 10 are on another still (5 is on two different virtual networks). Virtual networks may be *isolated*, *routed*, or *NAT routed* to an external network.

For networking of devices that should not be bridged, the VMM supplies isolated virtual networks, to which virtual adapters in the guests may be connected. An isolated network can only communicate among its own devices, and possibly to the host, but may not communicate with other networks, virtual or not. Devices on it can communicate among themselves. In Figure 7, the virtual network that 9 and 10 are on is isolated. Thus, OS9 can communicate with 10 via the network, but not any other system.

A virtual network may also be routed such that traffic to or from it goes through the VMM and is routed at layer 3. The VMM translates the layer-2 information and does not reveal Medium Access Control (MAC) addresses. A VM with a routed adapter does not need to be on the same subnet as devices with which it communicates. A NAT routed virtual network translates traffic at layer 4, and uses the host's IP and MAC addresses. Thus, it cannot be directly addressed by hosts external to the virtual network, but can make outbound communications (and accept inbound connections via port forwarding if the VMM supports it).

Which type of virtual networking administrators may wish to use depends upon the isolation and segmentation that they require, in addition to any security concerns. A bridged connection usually allows full LAN access at the MAC level as if the VM were physically on the network switch (and thus full two-directional visibility below the IP level), a routed network offers layer-3 (IP) level two way visibility as if the VM were on another network connected to the router, whereas a NAT routed network offers layer-4 (TCP/UDP) outbound visibility only (unless static port forwarding is used).

4. MOTIVATING FACTORS FOR VIRTUALIZATION'S USE IN SECURITY

The properties of system virtualization offer the promise of improved data and process security in terms of confidentiality, integrity, and availability which are achieved through properties we shall term isolation, oversight, and duplication, respectively.

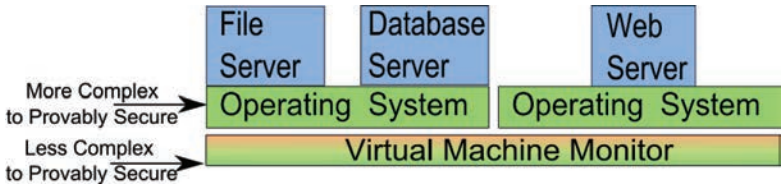

Fig. 8. A VMM offers a bottleneck that should be less complex to provably secure.

However, not all of the assumptions behind these properties are fully correct, and in other cases the principles underlying the additional security may be undermined by other VMM features, as we shall discuss later.

4.1. Isolation (Improved Confidentiality)

By placing operating systems inside virtual machines, a higher degree of isolation can be achieved not only between software running on the same hardware, but also between the guest operating systems and the hardware itself [Ormandy 2007]. This allows risky services to be run in a VM alongside critical ones in a different VM with a lower risk of the critical service becoming compromised via the OS [Madnick and Donovan 1973; Wimmer 2008; Wlodarz 2007]. Intrusion and malware analysis are simplified too, because specimen or sample malicious code may be run in an isolated virtual environment over which the analyst has full control, without requiring the setup of a full physical machine for each sample [Dinaburg et al. 2008; Ferrie 2007a; 2007b; Raffetseder et al. 2007; Wimmer 2008].

That isolation of guest code offers a higher degree of isolation than a complex operating system alone is due to not only the properties of VMMs, but also the fact that VMMs usually have a significantly smaller codebase. The core of a VMM can be built with tens of thousands of lines of code, whereas a modern operating system has millions to tens of millions [Bugnion et al. 1997; Garfinkel and Rosenblum 2003; Seshadri et al. 2007]. The use of a VMM can offer a higher level of containment because it is a single, hopefully less vulnerable, chokepoint which must be breached to break isolation. We illustrate this chokepoint in Figure 8 the database server and the file server are running on a single operating system made of millions of LOC, thus the attack surface between them which must be secured in order to isolate them is quite large. Conversely the Web server is separated from the other servers by a common choke point of only thousands of LOC, which is a smaller surface to secure.

A (hypothetical) fully equivalent virtualized system (where the VMM offers perfect efficiency and resource control) should offer an isolated, yet logically identical environment to a physical one. However, unlike the physical environment, full data and processing oversight can be used. However, the increasing complexity and feature set of modern VMMs (especially their additional control and management channels, discussed later) has led some to question whether this is indeed a valid argument.

4.2. Oversight (Integrity/Repudiation) and Duplication (Availability)

The resource control property of a VMM ensures that nothing happens in the VM that the VMM cannot observe or intervene in[9]. Thus the VMM has not only full low-level visibility of operation, but can intervene in operation of guests. As a result of this, VM operations can be observed irrespective of VM or guest OS state, and this state can be captured, analyzed, replayed, and restored easily. This low-level visibility of VM operation, sometimes called *introspection* [Garfinkel and Rosenblum 2003], is very

[9]However, there are methods to obscure data contents from the VMM, as we discuss later.

useful for Intrusion Detection Systems [Garfinkel and Rosenblum 2003], malware, and rootkit detection/analysis [Jones et al. 2008; Wimmer 2008; Xuan et al. 2009], and software development/patch testing.

The ability to capture and restore system state is a very useful property of VMs, and it allows for very rapid capture and restoration of the state of entire guest OSs to file (with each capture referred to as a *snapshot*). A snapshot can capture the memory, hard disk, and device states of the virtual machine (either while shut down, or while running) and then record the differences from the time the snapshot was taken. Previously captured snapshots can be restored very quickly [Collier et al. 2007], and some VMMs can restore snapshots even while the virtual machine is running, with little downtime. This ability to store and restore multiple past states of a VM has made them very popular for uses such as malware analysis, honeypots, and intrusion detection systems [Kourai and Chiba 2005; Payne et al. 2007].

The ease with which the VM state can be captured and duplicated, combined with the abstraction away from the hardware, significantly improves the availability of virtual machines. Virtual machines may be load balanced, moved between different hardware platforms with negligible downtime (live migration), and backups may be very quickly restored [van Cleeff et al. 2009; Rosenblum and Garfinkel 2005; Whitaker et al. 2005]. As a result of this, VMs offer very high availability levels [Jansen et al. 2008].

5. SECURITY IN VIRTUALIZATION

The properties of virtualization are not only advantageous for security, they can also be detrimental. As virtualization is a large and very vibrant field of research, with new research and threats coming out daily, any coverage can never be fully comprehensive. As stated in the Introduction, this work is intended as an introduction to the security concerns, considerations, and implications arising from use of virtualized systems.

Consequently, this work proposes a general coverage of the security issues surrounding virtualization. We are concerned with threats affecting the following agents: the VMM, VMs, OSs in VMs, software running on those OSs, and the operational environment such as the network. Since we are undertaking a very general approach, there are situations where a specific example will be covered very briefly. For more information we encourage the reader to refer to the appropriate references.

Security, in the context of this work, refers to the disclosure and alteration of data and operations which may be considered sensitive. Related threats are both breaches of expected privilege (those that should not be able to be altered or read without explicit permission) and breaches of other controls that, while often implicitly permitted, may be applied on a case-by-case basis to suit the situation.

Sensitive data includes, but is not limited to, the confidentiality, integrity, and availability of:

—software data (program data in memory, on disk, or in other forms of storage);
—software and hardware operational state (resource allocation levels, program execution paths, etc.);
—control and network channels.

Here "software" refers to the VMM, VM, OSs in VMs and applications running in VMs as discussed before.

5.1. Threat Model in a Virtual Environment

The secure design of any system or application requires an accompanying threat model. This model should address two key aspects: (i) the key security design aspects of a virtual model (including what is considered potentially sensitive information) and (ii) definitions of a set of possible attacks to consider. A threat model can be developed

Fig. 9. Primary steps in the process of threat modeling [Microsoft 2010].

with a focus on the attack or penetration method (a software-centric focus as typified by Microsoft's Security Development Lifecycle), or by an asset-centric model (in which the value of assets is the focus of such a model). This may entail a set of small threat models relating to a specific set of attacks which could occur in specific circumstances or are related to specific assets. Such a set of threat models can assist in assessing the likely harm, priority issues, and ability to limit or recover from such attacks. A generalized threat model that attempts to address all types of attacks, penetrations, and leakages concerning every possible configuration of virtual platforms utilizing any type of supporting hardware (and/or software) and protecting every type of asset is very likely to be too general to be of significant value. Thus, the focus in both academic research and within organizations that have an important stake in threat and intrusion control (e.g., OWASP (Open Web Application Security Project), Microsoft's SDL) [OWASP 2010; Microsoft 2010] has been to develop models related to specific operational scenarios to be used for evaluation.

When commencing any system design in general–and virtualization in particular–it is necessary to apply threat risk modeling to avoid risks associated with (real or virtual system): stability, malfunction, misuse, or information leakage. Various methods of risk assessment exist, but the choice of method is far less important than actually performing a threat risk modeling process. OWASP recommends Microsoft's threat modeling process because it works well for addressing the issues facing Web application security and is equally applicable to virtualization architecture.

Figure 9 shows the classic threat modeling process which (when applied to virtualization in particular) enables stakeholders to:

—define the security requirements and risk appetite for the system under development (Vision);
—communicate security design aspects of the VMM/VM system (Diagram);
—analyze components for potential security issues using a proven methodology (Validate);
—discover and mitigate issues that could put VM infrastructures at risk, as well as prioritize and plan efforts to address significant threats such as backup, failover, resource misallocation and dysfunction caused by malicious software (Identify Threats);
—manage mitigations for security issues, particularly as a result of the complex interaction between VM Monitor, host operating system, drivers, VM operating systems and applications (Mitigate). This enables better security efforts to be conducted for

both new and existing IT infrastructure components in a practical and cost-effective manner.

Two significant papers have become cornerstones in the development of computer security taxonomies [Landwehr et al. 1994; Lindqvist and Jonsson 1997]. Although written before recent resurgence of virtualization, both have applicable contributions to make in the development of taxonomies for virtualization.

Lindqvist and Jonsson [1997] proposed a taxonomy for use in incident reporting, statistics, alerts, and intrusion detection systems. Unlike previous proposals, they take the viewpoint of the system owner rather than that of system developers and vendors. Their work focuses on external observations of attacks, as the system owner may not normally be able to categorize security flaws in detail. This arises because most software and hardware is purchased from system vendors where source-code and structural design are proprietary and not available from the vendor. Their approach to a taxonomy is based upon intrusion techniques and intrusion results. Their intrusion techniques and classification of intrusion results only have limited applicability in the case of virtual machines, since such an architecture can be viewed externally (owner) or internally (system designer).

However, their work does address classical security vulnerabilities such as bypassing controls, active and passive misuse of resources, data leakage, denial of service, and erroneous output. Landwehr et al. [1994] provides a taxonomy which has become, and remains, foundational for many years. The authors make it clear that a general taxonomy addressing intrusions, devices and systems, attack categories and techniques, network categories, and protection systems is too broad to be of a great deal of use and they support this by including combinations of some 50 examples of different systems, architectures, and flaws. Interestingly however, they do follow the concepts developed of a taxonomy based on time of introduction (*when)*, location (*where)*, and origin (*how)* as discussed in Section 9 (recommendations). Subdivisions are provided within each of these categories. However, as the authors state "often, especially at the finer levels, such a partitioning is infeasible, and completeness of the set of categories cannot be assured" [Landwehr et al. 1994].

Many papers relating to attacks, intrusions, and threat models have been written over the last two decades. However, it is the intention in this section to only focus on key contributions which have the potential to enable development of specific threat models relevant to particular implementations of virtualized architectures.

Given the highly integrated nature of virtual architectures and the crucial importance of the security of operating system software, the work of Rashid et al. [2003] is valuable in respect of threat models. The focus here is on *when*, *where*, *what*, and *how* changes can occur to VM architecture.

The issue of *when* focuses on chronological issues such as when a change such as an upgrade or patch should be made, as well as the systems necessary to support this change. For example, upgrades and patches to the VM Monitor or Management VM may not necessarily occur at the same time as changes to various VM operating systems.

The issue of *where* relates to what parts of the software such changes can be made in and which supporting mechanisms are affected. For example, changes to Type-I VMMs may not (necessarily) be applicable to Type-II VMMs. This leads to the very important issues of granularity, change propagation, and systemwide impact.

The issue of *what* changes should occur relates to the type of issue involved, such as issues of availability, extensibility, and others. For example, in a VM architecture, the choice of what drivers might need to be upgraded may well affect the availability of a critical system. A trigger from an IDS engine indicating that firewall rule(s) might

Table I. Simplified Change Taxonomy Based upon "When, Where, What and How" (adapted from Rashid (2003))

Taxonomy Group	Property	Examples
When (chronological issues)	Time of change Frequency of change Change history	System upgrades, patches Coordination with VMM components Version control in conjunction with support of applications
Where (in the system)	Object Granularity Effect of change Dissemination of change	Type II VMMs upgrade Change to virtual object, class, package Affecting a single VM or VM Monitor Change to base VM operating system
What (assets of system)	Availability Reactivity Extendibility	Up time of respective VMs Respond to malware at VM Monitor/OS level How plugins are handled in VM environment
How (manual or automated)	Automation Process control System function modification	Auto-distribution of patches across VMs Logging of malware protection events across VM platform Implementation of upgraded drivers across multiple VMs

need to be changed within a critical period of time can have significantly advantageous (or correspondingly disastrous) consequences.

Finally, the issue of *how* changes should be made can have a critical effect on the operation of VMM architecture. Changes can be made manually, which may imply temporarily taking a VMM system offline (thus affecting *when*) and affecting the automation of the system. Automated tools can be used to manage, control, and measure software changes. However, such procedures require checking and verification that the changes were successful and, in the case of VMM systems, to ensure no catastrophic flow on effects.

It is important to note that these four properties are not necessarily mutually exclusive. For example, a decision *what* to do may well have consequences as to whether or not it is carried out in a timely manner (*when*). Another example might be that a change to a Type-II VMM (*where*) could have implications on *how* this is carried out in order to maintain full availability of the system.

Table I represents a simple sample model, based upon the "*when*, *where*, *what*, and *how*" threat architecture described before.

The answers to these questions can be used to develop a taxonomy relating to the characteristics of software change and the factors which lead to such change(s). The purpose of such a taxonomy is to: (i) define specific software development tools and techniques within the domain of virtualization, (ii) provide an architecture for comparing specific tools and techniques used in virtualization, and (iii) evaluate the use of software development tools or techniques for maintenance and change control.

For this work we have not provided a generalized threat taxonomy, as the general nature of the considerations we give precludes the production of a meaningful and valid taxonomy. We are not presenting a summary of threats to any one area, or to any one type of system. This work discusses considerations for not only virtualized systems, but the software which resides upon these systems. This software often has directly competing security requirements.

Thus, while some of the threats presented are indeed simple to categorize, others are not due to conflicting viewpoints and motivations among stakeholders. For example, fully transparent virtualization can be considered a threat, a security feature,

or a complete irrelevancy depending upon the particular situation and implementation considerations at stake. We have structured the following sections to follow on from the overall system virtualization properties described earlier. In them we discuss considerations around and stemming from strong virtualization properties, those stemming from weak virtualization properties, those stemming from control and communication channels, and those related to use of additional software.

6. SECURITY THREATS RESULTING FROM STRONG VIRTUALIZATION PROPERTIES

The same properties and requirements that define a virtualized system (Efficiency, Resource Control, Equivalence) have security implications that must be considered in a fully virtualized system. Many of these implications are two-sided, where a property of virtualized systems can have effects that can both enhance and compromise security. For instance, some of the motivators given earlier can be quite undesirable if the VMM is not trusted. These threats occur as a result of the VMM trust model, the transparent nature of an ideal VMM, and the introspection capabilities that the resource control requirement allows.

6.1. Untrusted Components and the VMM Trust Model

The implicit trust in a virtualized platform is a major vulnerability area in system virtualization. In a physical system the OS trusts the hardware to a large degree. Likewise, in a VM the operating system is required to trust the virtual hardware, and thus the VMM. The VMM is a single point of failure, and a malicious, compromised, or otherwise problematic VMM may observe or interfere with the VM. Thus secure virtualization relies on the authenticity and integrity of the VMM, and in some cases upon the security or identity of the underlying hardware [Perez et al. 2008]. If any underlying component is compromised it becomes far more difficult to have an assurance of security. For example, for an application running in a guest OS to be secure, it needs to trust the Guest OS, the VMM, and the hardware.

Some authors have proposed the use of the Trusted Computing Group's [TCG 2010] (TCG) Trusted Platform Module (TPM) to offer assurance of hardware and VMM integrity and authenticity. The TPM is a tamper-resistant hardware module that offers a small amount of secure storage, and secure cryptographic functionality outside of the CPU. It is also used by AMD and Intel to offer a *dynamic root of trust* in their secure computing platforms.Through the use of a chain of trust these technologies can offer *attestation* of the integrity or state of both hardware and software system components at system boot time, and while a system is running. The root of this trust chain lies either in the TPM hardware, or in a key held in the TPM hardware [Perez et al. 2008]. (More information on Intel's secure computing platform and AMD's platform can be found in Intel [2003, 2009] and Strongin [2005] respectively. Discussion of trusted computing in general can be found in Irvine and Levitt [2007], Oppliger and Rytz [2005], Sailer et al. [2004], and Trusted Computing Group [2007].)

Attestation can be used to secure applications and operating systems to ensure they will only run in an environment with an attestation chain to a trusted root. In Windows, the trusted computing technology was initially called Palladium, then later renamed to Next-Generation Secure Computing Base (NGSCB) and is discussed in Oppliger and Rytz [2005], Perez et al. [2008], and Wang et al. [2010], however, it was reportedly cancelled in mid 2004 due to market reasons [Perez et al. 2008]. In Linux, IBM's Integrity Measurement Architecture (IMA) may be used [Sailer et al. 2004; Sourceforge 2010], or alternatively another Linux implementation is discussed in Seshadri et al. [2007].

This attestation is used in virtualization to offer a degree of verification of the authenticity and integrity of the hardware and the VMM. The TPM functionality has

been extended by Berger et al. [2006] to offer a virtualized TPM, or vTPM. Trusted computing and attestation have been proposed as ways to assess the security of all aspects of virtualization, most notably the VMM, but also the hardware, the guest OS, and applications running on it. This attestation can be done either locally via internal I/O channels, or remotely through a network [Baldwin et al. 2009; Haldar et al. 2004; Berger et al. 2006; and Dalton et al. 2009]. A more detailed discussion of attestation of VMM and secure virtualized systems is presented in Catuogno et al. [2010], Dalton et al. [2009], Pfaff and Rosenblum [2003], and Tomlinson [2009].

Attestation of the integrity, authenticity, and state of guest software is important for application and data security not only for guest OSs, but also for applications. This is due to the fact that security requirements can depend not only on the integrity of core system components or the application itself, but also on the presence and state of additional software inside the guest OS. This could include the status of security measures such as logging, antivirus, or intrusion detection systems.

A VM is more than the VMM and guest software: it is a container which contains other software and data components such as VM settings, a virtual disk image, and captured VM states. To secure these components the host's TPM may be used to sign and check their signatures. Furthermore, the TPM may contain keys to encrypt/decrypt them.

There has been some discussion on circumvention of trusted execution technologies. For instance, Wojtczuk and Rutkowska [2009] discuss the use of System Management Mode (usually known as SMM), a highly privileged CPU operating state (conceptually similar to ring-2, where ring-1 is hypervisor level and ring-0 is kernel level) which did not securely run the verification and attestation checks. This was due to an implementation flaw which has now been fixed [Wojtczuk and Rutkowska 2009]. This demonstrates that even the most secure system may yet contain unknown or undisclosed flaws. Attestation alone does not remove many security threats surrounding virtualization. Furthermore, even an attested component may be attacked, or used in an attack. Ultimately, the use of trusted computing techniques for virtualization raises the same issues which arise with their use in hardware security. Further detail and discussion of trusted computing's overall security impact can be found in Oppliger and Rytz [2005].

6.2. Transparent Virtualization

If all three of Popek's properties (efficiency, resource control, equivalence) are met in a strong (or perfect) way, then the result is a VMM that is ideal and undetectable to software inside the VM (we will term this a *fully transparent VMM*). This inability to detect a VMM can have quite serious implications for some types of software. Reverse engineering becomes far easier due to introspection capabilities, as any encryption keys, security algorithms, low-level protection, intrusion detection, or antidebugging measures can become simple to compromise. The combination of the basic trust model with transparent virtualization means that a VMM can be undetectable, and is automatically trusted.

Since the equivalence property of virtualization requires VMMs to be equivalent to a real machine, it can be very difficult to detect that software is running in a virtualized environment. Some software and systems are exposed to additional threats if they run in an untrusted virtualized environment. Since a VMM has full control over system resources, it can observe or alter any data inside the VM which can cause potential problems. Examples of software affected by issues resulting from this introspection include time-limited trial software, since the VM can be reverted [Franklin et al. 2008a], and software that encrypts, as encryption keys in memory can be read [King et al. 2006]. Software authors sometimes consider running security threats on virtualized systems,

as it allows better analysis and reverse engineering. For example, consider malware authors and the antivirtualization measures they take [Ferrie et al. 2006; Ford and Allen 2007; Liston and Skoudis 2006; Omella 2006; Wu and Ma 2010].

An undetectable VMM is referred to as *transparent*. However, some authors claim that it is not feasible (or useful) to make a fully transparent VMM [Garfinkel et al. 2007; Gueron and Seifert 2009]. Various VMM detection measures may be used, with the most commonly discussed being checks for discrepancies in logic, resources, or timing between a known physical environment and a known virtualized environment [Garfinkel et al. 2007; Gueron and Seifert 2009]. Examples of VMM detection methods include hardware attestation (as just discussed); observing timing of instructions or I/O (since a VMM will necessarily add some overhead when it intercepts instructions); observing resource availability and location (since VMMs may allocate resources at different addresses than physical machines); and checking for inconsistent or incorrect CPU emulation (for instance by not reproducing known CPU bugs). For discussion of technical discussions of these measures see Duflot [2008], Franklin et al. [2008b], Garfinkel et al. [2007], and Raffetseder et al. [2007]. Near-transparency also makes VMM-based rootkits possible [King et al. 2006]. The use of hardware virtualization mitigates the effectiveness of some of the detection measures, but it does not eradicate them. In the end, though, different parties have differing requirements, so transparency can be judged to improve and/or threaten security depending upon the context and party making the assessment.

6.3. VMM Insertion and "Hyperjacking"

Various methods exist for covert insertion of a VMM under an OS, moving the OS from physical to virtual nearly undetectably, either on boot or while the system is running. These VMM rootkits are a serious security risk, as they may be used to subvert an operating system completely. The methods used to accomplish the VMM insertion are varied. Two methods used include the use of raw disk reads to alter device drivers that are paged out to disk, and the modification of system startup files. For technical discussion of the concepts and proof of concept systems refer to Rutkowska and Tereshkin [2008], Rutkowska [2006], Nomoto et al. [2010], Athreya [2010], Dai Zovi [2006], Gebhardt et al. [2008], Skapinetz [2007], Ford and Allen [2007], Wlodarz [2007], Security [2010], and Carbone et al. [2008]. However, this form of attack is normally extremely hardware and VMM version specific, and to the authors' knowledge has never been observed in the wild.

A similar approach of migrating a running OS onto a hypervisor has also been proposed as a host intrusion detection measure [Nomoto et al. 2010]. Much of the strength of this approach comes from the introspection and intervention mechanisms open to a hypervisor, as we discuss next.

6.4. Introspection and Intervention by VMM

Although the VMM requires full control of resources to function, this can range from the minimum requirements of protecting the VMM's own system resources, to complete control, allowing it to observe and manipulate VM data and operation. The minimal case presents few security issues, but the steps taken to enable this can easily result in more control than is strictly necessary, which can enhance or weaken security as concerns various parties.

Introspection and intervention of the VM from the VMM. is a result of the ability of a VMM to take control of resources, and is when the VMM observes or intervenes in the operation of a guest. Software running at a high privilege level (such as a VMM or OS) may observe and modify system aspects affecting other software, while remaining hidden from the lower privileged software. Process introspection allows the

VMM to observe behavior of processes within the VM. The VMM may also observe I/O channels to, from, and within the VM. This is due to the resource control property of virtualization, and has associated information security issues that can both improve and threaten security.

When combined with transparency, the result is that the VMM cannot only observe and alter any aspect of the VM, it is automatically trusted (unless trusted computing elements are used) and undetectable (as described in Section 6.2). If the VMM is untrusted or VMM compromise is a risk, then the security threats due to introspection and intervention are very serious.

Consider this example: Company A has a virtual server in an outsourced datacenter that undertakes financial transactions. Depending upon the contract with the datacenter, it is likely that the datacenter does not have permission to view or alter any transactions undertaken (based on least need-to-know principles). However, because Company A does not control the underlying VMM, it has no way to ensure that the VMM has not altered transaction details or recorded credentials, a potential problem in many ways, as any local audit trails can be similarly compromised.

The most obvious approach to mitigating introspection and intervention threats from an untrusted VMM is to attest to the authenticity of the VMM and hardware. Attestation has been discussed in more general terms in Section 6.1 before, and is the approach taken by sHype [Sailer et al. 2005]. The high privilege of the VMM and hardware makes protection of software from a hostile VMM difficult. Some approaches can be taken to mitigate the threat of undesired introspection: they include cryptographic and architectural approaches.

Cryptographic approaches attempt to obscure the contents or operation of memory from higher privileged software such as the operating system or VMM.

Architectural approaches involve new hardware architectures or *codesigned* (read as co-designed, not code-signed) VMMs, where the hardware architecture is designed with VMM usage in mind. This technique is used for reasons other than pure security in some VMMs/hardware (such as the IBM AS/400 or Transmeta Crusoe [Smith and Nair 2005]), and is applied for security purposes by more recent approaches such as Bastion [Champagne 2010]. A system utilizing a codesigned VMM has a concealed binary translator that runs in an interface between the physical hardware and software on the system. This VMM is very low level and, because it is essentially a part of the hardware, hardware-level protections may be used to protect it [Champagne 2010].

The introspection capability can also be used to improve the security of a virtualized system. The high privilege level and low-level access of a VMM allows it to observe system aspects that may be inaccessible or hidden from the guest OS. Process introspection allows the VMM to observe behavior of processes within the VM. The VMM may also observe I/O channels to, from, and within the VM.

6.5. VM Cloning and Scaling

In physical systems, scaling up by addition of extra systems is limited by equipment resources, whereas in virtual environments copying a VM consists of copying a file. Since a cloned VM can be difficult or impossible to distinguish from an original, the cloning of VMs is a threat if it happens incorrectly, or to a VM that is not bound to a specific VMM [Garfinkel and Rosenblum 2005; van Cleeff et al. 2009]. Some cloning methods will generate new identifiers in the VM, such as OS Security Identifiers, Hard Drive IDs or virtual network adapter MAC addresses, but not all methods of cloning will do this (for instance, a raw file copy of metadata and disk image files will not do this).

The cloning of virtual machines can cause security concerns due to scaling, management, identity, and data retention issues. Thus, the number of VMs can increase very

rapidly, and become hard for an organization to keep track of and manage [Garfinkel and Rosenblum 2005]. This is complicated further by the concept of identity in virtual machines. A clone of a VM is identical to the original, and the concept of an original can be nearly meaningless [Garfinkel and Rosenblum 2005; Yu et al. 2010]. Locating, tracking, and managing a VM may become very difficult, if not impossible. For example, if an unauthorized clone can be created, then data can become spread between the two versions or one can be used to impersonate the other.

An unauthorized or unwanted clone is a risk because not only will it inherit most permissions from the source, it can also cause address and name collisions on a network. These collisions can cause problems for network resources, as different parts of transactions and communications can be split between different VMs, and possibly different physical machines. The problem is analogous to a misconfigured load-balancing system, and can cause problems for the confidentiality, integrity, and availability of data and systems. Countermeasures include delegating management functionality outside of the guest OS. Detection of VMs on the network can be undertaken using normal network measures such as network Intrusion Detection Systems (IDSs), however, care must be taken to not cause problems for legitimate machines from which a clone was derived [Garfinkel and Rosenblum 2005].

Data retention and control can become difficult to manage because VMs may carry any data previously stored in them when they are cloned. This is important for sensitive data such as personal details or encryption keys, as these must be securely destroyed when no longer needed. Among other places, data can reside in deleted files or memory snapshots, on virtual hard drives, or be retained in previous snapshots. It can also be part of logging data external to the VM that the VMM undertakes [Wimmer 2008]. Methods to reduce these risks include encryption of VM data with keys unknown to the VM, and externalizing encryption from the VM, for example, by storing encryption keys in the VMM or hardware TPM [Dalton et al. 2009; Dewan et al. 2008; Wimmer 2008]. This approach can cause problems for VM migration if measures are not taken to migrate or regenerate keys.

6.6. Nonlinear VM Operation and Monotonicity Issues

Since virtual machines can be cloned, and have their states captured and restored quickly, their execution does not follow a linear path through time, but can be reversed, forked, and subject to similar nonlinear operations. This lack of linearity is referred to as a "lack of monotonicity" by van Cleeff et al. [2009]. The lack of monotonicity can be problematic for data on the VM, configuration, and general logging and monitoring. For application data, the lack of monotonicity can be a problem because snapshots, cloning, and restoration break linear operation of programs and data. Snapshots can fork, merge, and revert, and this behavior must be taken into account for program operation, database operations, and data retention requirements (as for cloning) [Price 2008].

Due to the complete and indistinguishable nature of many types of state capture and restore, applications should store settings, management information, and data external to the VM (if the VMM does not supply partial state operations that leave certain files untouched). Keeping some data separate from the snapshotting process itself presents potential risks if the correct data is not stored or restored correctly, and choosing the correct data is not a trivial problem for systems that were not designed to allow for such operations.

If this is not properly done then threats may arise as a result including the rolling back of updates, configuration, or user settings (such as deactivated accounts) leaving the application vulnerable or nonfunctional. Most notably, if a VM is reverted to a version before a patch was installed, the patch is then removed, thus leaving a system

vulnerable [van Cleeff et al. 2009; Garfinkel and Rosenblum 2005; Yunis and Hughes 2008]. Some vendor's products allow the patching and management of snapshots, but rollback is also a problem for local logging, where logs can be destroyed. Unless special precautions are taken, all records can be destroyed when the system state is restored, which is important not only for security but also for regulatory compliance. Examples of measures to mitigate this threat include the use of a versioning file system such as that in Peterson and Burns [2005], which stores all changes made to all files and enables a view of files at any point in the past.

6.7. Software Decoupling from Physical and Hardware Environment

A virtualized system abstracted away from the hardware is no longer dependent upon its location, and can even be hard to define as a single system. A virtualized system can be duplicated (cloned), and every single instance must be found (at least logically) before it can be managed or secured. Even when threats from cloning and snapshots (as already discussed) are excluded, problems still arise from the abstraction of the VM away from hardware.

Issues may include being unable to locate the physical location of a VM, and those resulting from the implications of hardware consolidation in virtualized systems. A system which cannot be located presents problems in management or administration, as well as presenting potential legal or regulatory jurisdictional issues if there is a requirement for data to stay within a certain jurisdiction. Largely consolidated hardware can result in high impact failure scenarios if care is not taken, as one server may contain a great deal more services than would normally reside on a single system. Conversely, however, the reduced number of systems may also make backups and redundancy easier to manage.

Threats arising from hardware independence are exacerbated by virtualized networking, where a physical network connection may not represent the actual logical location of a system on the network.

Inability to locate a VM. Since a VM is not bound to a physical location it can become complex to locate a single VM if it needs to be managed or isolated. Moreover, when there are difficulties finding the location of a VM, difficulties can arise in managing it. For example, normal measures such as shutting down the physical network port can take out legitimate machines at the same time. Mitigation measures for this type of threat include a combination of normal network intrusion detection measures, and changing management processes to help reduce the incidence or impact of such occurrences.

Hidden or covert VM. A VM is logically the same as a physical machine on the network. If one is running on a system, it represents a risk potentially similar to that of an unauthorized physical machine. Similar technical measures can be taken to those that detect physical machines (for instance, Network Intrusion Detection Systems (NIDS)). In addition, software measures can be taken to check for the presence of VMM software exactly as an administrator would search for other unauthorized software (such as games).

Additional consolidated hardware. Additional hardware is used in virtualized systems, both logical and physical. Logical hardware is used in every virtual machine to interface to the VM. Additional physical hardware arises because more hardware is needed to allow load balancing, as well as server consolidation of many machines into fewer larger-scale servers. This results in a reduced hardware surface, which threatens availability and continuity at a large scale if measures are not taken to protect against failure or compromise. These measures should include standard operational security,

such as load balancing, redundancy, backup, and change management procedures as discussed in Section 9.

Physical location issues. Although virtualization allows software to be run decoupled from hardware, there remain issues which require at least a certain level of physical location control. Of particular importance are any legal, regulatory, or jurisdictional requirements that must be met to fit the appropriate legal system, privacy issues, disclosure or privacy regulations. Such issues are inherently very complex and for more information we refer the reader to Subashini and Kavitha [2011] and Griffin et al. [2005].

However, more practical considerations can also be important, such as the requirement that it is run in a physically secured environment, on a server with ready administrator access, or on a high availability server rather than the standard server nearby.

7. SECURITY IMPLICATIONS FROM WEAK IMPLEMENTATION OF CORE VIRTUALIZATION REQUIREMENTS

Many security implications arise from an improper, incomplete, or compromised implementation of Popek's [Popek and Goldberg 1974] requirements. The two main types are transparency breaches (where any of the three requirements is imperfectly implemented), and resource control breaches.

7.1. VMM Detection and Transparency Breaches

Virtualization transparency is breached when any one of the three requirements is breached, leading to information being disclosed that can be used to deduce the presence of a VMM. The detection of a VMM can cause problems for some uses of VMMs (such as problems for the analysis of malware if it suspects it is being run in a virtualized environment and behaves differently), just as the inability to detect a VMM can cause problems in other cases. A breach in transparency can also cause problems ranging from instability to failure in software that makes assumptions about the environment in which it is being executed. These can include problems resulting from assumptions that are broken by unexpected device states or timing problems. For example, it may well be misleading to extrapolate seek times from that of a stand-alone physical disk to that of a virtualized disk. In particular note that anything that optimizes disk layout for speed (such as a Database Management System (DBMS)) will in a virtualized system perceive a layout that does not necessarily correspond to the physical true layout.

7.2. VMM Compromise

Given that the VMM is the most central part of a virtualized system, if an aspect of it is compromised then the entire system and everything running on it is at risk. The VMM can be compromised in several ways, but in this work we discuss threats specifically related to, respectively, VMM confidentiality (VMM introspection), VMM integrity (VMM alteration), and VMM availability (VMM Denial-Of-Service (DOS)).

VMM introspection. Introspection and intervention are not only issues for virtual machines. A VMM is also at risk, especially in a Type-II implementation, where a host OS is also executing at a privileged trust level beside the VMM. Since any layers at or below the trust level of the VMM may observe (and intervene in) VMM operation, the system hardware is also potentially a risk. Attestation can be used to alleviate much of the associated risk. The entire system may only be as secure as the weakest component, so the VMM may only be as secure as the system on which it resides, and the host system may only be as secure as the VMM. Using a hosted VMM therefore

simply increases the surface to be secured against compromise, and is probably best avoided in situations requiring high security.

VMM alteration. If the VMM is compromised, it can affect all VMs on it, and as a VMM has full hardware control, the underlying hardware below it is at risk too. Few measures have been proposed to overcome this, other than security-aware development of VMMs, and VMM integrity checking and attestation as discussed previously. Security-aware development practices are used to make software that is thoroughly designed and tested, although the level of assurance offered can vary. Hypervisor products have attained at least the Common Criteria Evaluation Assurance Level (EAL) level-5 certification, which indicates that the system has been semiformally designed and tested [CommonCriteria 2008]. A sufficient EAL certification is also needed to permit the use of a product in some high security environments such as the U.S. government or military.

VMM Denial-of-Service. If operation of the VMM software can be interrupted, then operation of all VMs running upon it becomes affected. VMM operation can be negatively affected by resource starvation, or interrupted if a complete shutdown of the VMM is caused or a restart made necessary. Both of these situations cause a Denial-Of-Service (DOS). Should a DOS happen, then data loss can occur, as volatile data can be lost and system data structures can become corrupted. The cause of the DOS or disruption can occur via either bugs in the VMM, local software attack, or network vectors.

Local DOS threats to the VMM. These can include VMM bugs and resource starvation. VMM bugs can be used to crash the emulator, prevent access to administrative channels, or execute privilege escalation attacks (discussed shortly). In our literature research we have encountered bugs in virtualization software that caused the administrative interface to cease functioning, while VMs kept functioning normally. Ormandy [2007] discusses research into host environment risk, and covers many different examples of VMM bugs. *Resource starvation* refers to resources (such as RAM, CPU, or network bandwidth) being starved from another part of a system, and can be a result of either intentional or accidental actions. Resource starvation can cause performance problems and potential failure of some software that requires a certain level of hardware performance. This can occur as a result of any running software, and can significantly affect other parts of the system.

Although it is possible for this to occur from the host, most research is concerned with resource starvation threats from VMs. Resource starvation as a result of VM activity can, if not mitigated, threaten operation of other VMs. Most VMM implementations use resource control to impose resource allocation limits on VMs, and thus on the software running within them.

Overprovisioning is also a DOS risk, and can be quite complex in virtualized environments. Consider page sharing, where multiple VMs have identical memory contents mapped to shared memory locations. This is useful for items which are generally static such as operating system components, but if these shared pages are not taken into account memory can be overprovisioned. Shared pages work as in the following example: If two Windows XP systems are running, each allocated 512MB Random Access Memory (RAM), they each believe they have 512MB accessible. If 200MB of the memory in each VM is identical, then the VM shares those sections, saving the allocation of 200MB. So, the VMs believe they have 1024MB total accessible, but only 824MB is allocated in the host. If this extra 200MB is allocated to a VM, and the shared memory diverges then more memory is allocated than is available, resulting in swapping occurring on the host with a subsequent degradation in performance.

Network VMM DOS threats. These can negatively impact the VMM, by either degrading performance, making the VMM cease functioning, or taking administration capabilities and interfaces offline. This is a particularly significant threat in implementations that use network administration of the VMM and do not facilitate easy local administration. In the same way that a VMM bug can cause a VMM DOS condition locally, a remote bug has the same potential, and the same mitigations. Network flooding attacks should be mitigated using the same measures used in most network flooding attacks, such as good firewalling, and the use of sink holes (network routes that are designed to deal with dangerous network traffic safely).

Host DOS. This occurs when either the hardware or the host (in a Type-II VMM system) causes loss of control or stops operation of the VMM and VMs [Ormandy 2007]. The mitigations and causes are related to those discussed in VMM introspection previously (Section 6.4).

7.3. Resource Control Breaches and Privilege Escalation

For information security, resource control breaches are among the more serious for confidentiality of data. In virtualized systems threat areas include information leakage and privilege escalation.

VM information leakage. Information leakage with regard to virtualization comes in two main types: leakage of information into VMs, and leakage out of VMs. Of particular note with virtualization information leakage vectors is their potentially very large bandwidth.

For leaks out of VMs, a virtual machine may leak information about its operation or resource usage to other VMs through side channels. These side channels include both software and hardware based, with some of the simpler software leakage channels consisting of features installed to facilitate ease of operation between the VM and the host, such as clipboard and file sharing (discussed further in the control and communications section). Leaks of operation data into VMs can include details about the host or the state of the resources on other virtual machines.

Other leak vectors from hardware include cache-based attacks. The VM may be able to detect things that it ought not, such as resource status on the physical machine (CPU or memory usage, etc.), network details, or details about the operations of other VMs. By analyzing page timings it may also be possible to detect shared pages too, thus enabling an attacker to check for certain shared memory pages. Covert information flows are discussed in more detail in Gebhardt & Tomlinson [2008], Jaeger et al. [2007] while Ristenpart et al. [2009] discusses cache issues in the case of the Amazon EC2 shared compute cloud. Much research into attacks and countermeasures has also been undertaken by other researchers [Wang and Lee 2006, 2007].

VM escape. Breaches of the isolation property of virtualized systems can allow privilege escalation of virtualized application code. Incidents of this occurring are referred to as *VM escape* incidents and occur when code from a VM (thus in an unprivileged context) somehow runs itself in a higher privileged context (such as that of the VMM).

VM escape to host. This occurs is when isolation between host and virtualized environment is breached. This occurs when code from a VM runs natively on the host machine, without any VMM control. Caused by bugs or weaknesses in the VMM, these threats are discussed in Criscione [2010], Gebhardt and Tomlinson [2008], Ormandy [2007], and Ramos [2009] and are more common in Type-II VMMs due to the extra attack surface offered by such implementations. Vectors for attack consist of attacking virtualized devices, CPU caches, and Direct Memory Access (DMA) to access host memory directly. Several examples of such attacks are given in Criscione [2010], Kortchinsky [2009], Ormandy [2007], and Vasudevan et al. [2010].

Prevention of escape incidents involves a combination of VMM patching, host security measures (as would be used to secure a physical machine), and measures to detect the malicious code in the VM (as legitimate software does not need such functionality).

VM escape to VM. This occurs is when isolation between VMs is breached and one VM changes or views the operation or data of another VM that it should not be able to access. Compared with the "to host" situation previously discussed the main difference is that the VMM and host are themselves not under threat. Some research has gone into the secure sharing of data and security policies within sets of virtual machines, called *Trusted Virtual Domains* [Griffin et al 2005]. In VMMs that do not support this functionality, one VM modifying another VM's resources is a privilege escalation problem generally caused by VMM bugs. In VMMs which do support virtual domains, situations can occur due to abuse of the resource sharing functionality as well as VMM bugs. We shall refer to these situations as *VMM Trust Boundary Escape.*

These classes of threat can be prevented with VMM patching and configuration management. The threats can also be mitigated if high value storage and communication assets are encrypted, so that if they become compromised the data is of no value unless the compromised asset is decrypted. For example, storing database records encrypted on the virtual disk protects it if that file is compromised or read. An attack would also require compromise of memory containing the encryption key (a less likely outcome) to reveal the encryption key.

VM virtual network escape. This is similar to the preceding, but is at the level of networking logic rather than general I/O. This refers to a VM circumventing intended network bounds. Examples include bugs in the VMM, or bridging through a double homed VM on the network. We will discuss this in more detail in the next section.

8. SECURITY IMPLICATIONS FROM CONTROL, DATA, AND SOFTWARE FLOWS

In addition to the security issues arising as a result of virtualization properties, there are other security issues arising from the implementations of virtualization that can result from the way virtualization is made useful. For example, while VMM and VM data flows, VMM administrative channels, and software installed inside guests all make virtualization useful in real deployments they can also add extra attack surfaces. While the issues discussed previously were virtualization specific, the issues discussed in this section overlap with more general security principles.

8.1. Security Implications from Control Channels

Control channels for VMMs are commonly used for administrative interfaces and VMM Application Programming Interfaces (APIs). VMM APIs[10] facilitate administration of the VMM and VMs running on it. This includes functionality such as changing the operational state of VMs (shutting them down), modifying existing VM settings, cloning new VMs, creating new untrusted VMs, and running commands on the guest OS. Additionally many current VMM solutions offer some form of Web-based interface, with the resulting need to secure those interfaces (as they do not require attackers to have special software and may be susceptible to normal Web application weaknesses).

Threats surrounding control channels consist of both unauthorized access and denial-of-service issues. The threat presented by control channels is exacerbated as some VMMs contain undocumented hidden control channels that function through undocumented device and CPU instructions [Capelis 2007; Jaeger et al. 2007; Ormandy 2007; Vaarala 2006].

[10]Examples of VMM APIs include the VMware Vix API [VMware 2010], or the Microsoft Virtual PC API [MSDN 2010].

For non-network control channels (network channels are discussed later), security measures that should be taken are primarily those that should be undertaken with any software environment. Examples of these include keeping software up to date, configuring (and auditing) sensible permissions using strong access controls (for example, secure passwords and not giving all users full privileges), and disabling any nonessential control channels (for instance, by not installing VMM helper software in the VM unless it is needed for administrative or performance reasons). This also includes securing critical files for VM operation, as if the virtual machine's settings or metadata are compromised then the VM itself can have issues, even if the VMM and guest OS are otherwise secured.

8.2. Security Implications from Dataflows

Figure 6 illustrated many of the important data flows between components in a virtualized system. These flows, both software and hardware, are primarily vertical in function between different layers, with the exception of network channels which are connected to many different components concurrently. Input/output flows and devices in modern systems can be very complex and securing them is not a trivial task, particularly when performance is an issue [Karger and Safford 2008]. I/O and software channels include those used by physical devices and virtualized devices as well as used by VMM APIs (that extend beyond management operations which have already been discussed).

Physical devices potentially present security threats, as they may open up side communication channels that either do not go through the VMM, or use shared resources in the VMM. If every physical device is not completely secured between VMs then the potential exists for I/O side channels to enable isolation or privilege escalation breaches. For instance, Dalton et al. [2009], and Dewan et al. [2008] discuss the potential for VMM resource control to be circumvented through the use of DMA technologies to access memory locations and Dowty and Sugerman [2009] discuss establishing secure Graphics Processing Unit (GPU) isolation in virtualized situations.

Even when the physical devices are secured, however, the location and sharing of device drivers in the VMM architecture can have important security considerations. Karger and Safford [2008] discuss how *pure isolation* hypervisors (where each guest uses a separate device and drivers) can differ from *sharing hypervisors* (where guests use resources on the same device) in the security and performance that they offer. In the example of hard disks, these would use separate hard disks and different areas on the same disk respectively. Full and complete isolation between VMs on the same hardware remains a very difficult problem, particularly in situations which require sharing of display information.

Network channels. These present similar security issues to those presented by control channels and shared devices, but they present a potentially significantly higher risk as they can be potentially accessed remotely, and may be connected to almost every virtualization component. Hidden network channels can also be a risk, as not all virtualized network channels are easily observed and some data sharing and control channels happen over virtual networks too.

To secure these aspects the same normal network hardening and monitoring measures should be deployed in virtual networks as would be deployed to secure a physical network (since a virtual network is logically equivalent to a physical network). These measures include minimizing the attack surface by disabling unnecessary services, deploying access control measures such as firewalls or network control measures such as Quality-Of-Service (QOS), and deploying monitoring and prevention measures such as Network Intrusion Detection Systems (NIDS) and Intrusion Prevention Systems (IPSs).

8.3. Security Implications from Non-VMM Software

Many of virtualization's security threats do not come directly from the properties of virtualization, but from the way in which it is used and implemented. Important examples of the causes and attack points include the decoupling of software from the physical environment and the larger software and hardware attack surface. A virtualized system inherently has more total software than a nonvirtualized system. A Type-II VMM implementation adds further additional attack surface, as the host OS is a threat to VMM components. The additional software attack surface in a VMM implementation includes the VMM software, but in most uses will include additional instances of operating systems and other software inside multiple virtual machines. This means that the overall attack surface of the software running on a system is higher, and this software can have many internal data flows as discussed next.

Guest Denial-Of-Service. In addition to threats normally present, a guest OS is potentially vulnerable to certain DOS situations. Generally these attacks are undertaken via the VMM, but not always. For hosts with multiple VMs resource usage should be monitored, as one VM's operation may be negatively affected by another if resources are overutilized [Ormandy 2007]. Resource starvation is also a potential problem for the exit point of virtual networks, as while the internal rate of virtual networks is very high, most of the time the physical link will have less capacity, so in situations of high load from either internal or external sources network interruption may occur.

Guest software compromise. Because a VM is an equivalent of a physical machine, it shares similar risks if that software is compromised or insecure. This applies to both operating systems and application software, and is exacerbated by the tendency of virtualized environments to run multiple copies of software (one in each VM). If the guest OS or any software running on it is out of date or misconfigured then the whole VM is at risk.

The guest operating system usually has the same vulnerabilities as an identical software configuration (of OS, drivers, and applications) installed on a physical machine. These vulnerabilities can be anywhere in software, and can lead to full VM compromise, and thus potentially network compromise via the compromised VM. Because many VMs are originally cloned from a base image they are often more homogenous in the software and versions they have installed, thus while they may be in more of a known state to administer, this has the downside of potentially having more widely shared vulnerabilities than would be seen in a physical setup.

Virtualization alone offers little extra protection for an OS or software running on it, although introspection can improve detection of hidden processes. The resource control property of VMMs can be used to improve not only detection, but also mitigation and recovery from compromise through introspection and modification of the VMM. The most notable version of this is the use of snapshots to restore to a known good state.

A VMM-assisted IDS or HIDS may help in detection [Garfinkel and Rosenblum 2003; Jin et al. 2009; Kourai and Chiba 2005; Laureano et al. 2004; Litty 2005; Sharif et al. 2009], mitigation [Litty and Lie 2006; Nance et al. 2008], and recovery [Matthews et al. 2005; and Wimmer 2008]. Additionally, approaches to protect the integrity of code running on the guest have been proposed such as Overshadow (which encrypts memory and decrypts only for the authorized application) [Chen et al. 2008], or SecVisor (which ensures only approved code may run in high privilege) [Seshadri et al. 2007].

9. RECOMMENDATIONS FOR SECURE VIRTUALIZATION IMPLEMENTATIONS

The following section builds upon the system virtualization architecture previously described in Section 3 as well as the security implications described in Sections 5–8.

Table II. Development Processes for a Security Policy

Creation of a comprehensive security policy [Mirzoev & Yang 2010, Gebhardt & Tomlinson 2008, Scarfone et al. 2010, Kim 2008, Kirch 2007]		
An existing security policy should be adopted and transformed to satisfy the need of a virtualized environment. This includes a suitable deployment strategy.	Evaluate trust levels and zones for administrators as well as hardware and software components. This requires: • Definition of trusted zones and separate servers either at the hardware or virtual machine level. • Where security requirements are different, we need to create separate zones (physically as well as virtually). Only systems with similar security requirements should be grouped on one host. • Evaluate separation of duties (and create separate policies) for administrators with different responsibilities. For example: separate datacenter administrators, VM administrators, and VMM administrators.	Ensure consistency of the security policy. For example, the same policy should apply regardless of whether the application is running on an OS within a hypervisor or on an OS running on hardware.

The sections to date have provided a detailed explanation of system virtualization and associated security issues. This section provides a template for implementation and verification which can be used by system administrators for the practical implementation and testing of secure virtual platforms.

We focus on four key areas:

(1) rollout planning and managerial issues;
(2) hardening, threat prevention, and vulnerability detection measures;
(3) intrusion detection and prevention measures;
(4) recovery and continuity protection measures.

To a considerable degree the first, third, and fourth items are consistent with any well-designed secure system implementation. However, item two has particular concerns for system virtualization platforms and will therefore be discussed in more detail.

9.1. Rollout Planning and Managerial Issues

The creation of a comprehensive security policy with respect to virtualization follows the same guidelines and principles applicable to any IT system development. Thus, the frameworks defined by Sarbanes-Oxley, Cobit and ISO/IEC 27001 are equally applicable to a virtual machine environment as they are to other IT environments. A brief summary of the security policy definition process is given in Table II.

9.2. Hardening, Threat Prevention and Vulnerability Detection Measures

Hardening is the process by which the security of the virtual system is improved by reducing its exposure to threats and vulnerabilities. In general, a single-function system is likely to be more secure than a multifunction one. The larger the vulnerability surface, the more the virtual machine architecture is subject to threats. Hardening aims to reduce the number of attack vectors, while threat prevention and vulnerability detection measures aim, respectively, to close entry points for attack vectors and to detect them should they exist.

Hardening processes, such as removal of unnecessary usernames and logins, removal of unnecessary software, and disabling unnecessary services, aim to result in a platform which is robust, well-managed, and provides enterprise-grade service for the virtual system users. The virtual architecture should thus be less vulnerable to malware, intrusions, and other threats. Table III provides a summary of the methodology by which such hardening can be achieved.

Table III. Methodology for Platform Hardening in Virtual Systems

Methodology for Platform Hardening and Prevention of Threats [Tolnai & Solms 2010]	
Secure hardware	• Use attestation for verification where supported • Control physical access • Use BIOS passwords to prevent reboot attacks • Remove unnecessary hardware

Secure host operating system	• Use attestation for verification
	• OS Hardening: Follow standard hardening procedures used for OSs
	  ○ Patch and control change management
	  ○ Full disk encryption if possible
	  ○ Disable/remove unnecessary services and software
	  ○ Install Host IDS and anti-virus
	• Implement standard network security measures

Secure hypervisor	• Use attestation and integrity checks • Patch and update attestation records • Use care with resource allocation to VMs • Monitor hypervisor for signs of compromise

Secure management interfaces	• Minimize attack surface
	  ○ Disable unneeded network and/or local admin interfaces
	  ○ Firewall access from untrusted areas
	  ○ Keep management networks separate from core and guest networks
	• Strong authentication, encrypted communication.
	• Least privilege user access
	• Log and audit events
	• Secure local and remote hypervisor management interfaces
	• Encrypt communications for remote administration

Secure virtual machine	• Harden guest machine as would be done with any machine in a physical environment
	  ○ Patch
	  ○ Remove unnecessary drivers and software
	  ○ Only install helper software if needed
	  ○ Disable/remove all unnecessary virtual hardware (CPUs, RAM, media devices)
	  ○ Prevent virtual machines from utilizing physical resources
	• Control allocation of physical resources
	• Integrity validation, signature checking, or virtual encryption (on guest) to prevent unauthorized copying.
	• Monitor change management procedures, and remote auditing/control.
	• Use some form of secure time sync functionality

Once the desired hardening measures have been applied, they should be verified and tested thoroughly. While verification should also be undertaken during predeployment, it should not be treated as fully comprehensive until a final, representative deployment is tested, to ensure that no issues develop undetected right before deployment. This testing of a representative system should involve both “white box” verification and “black box” testing, and should also be repeated regularly to account for errors that

Table IV. Summary of Intrusion Detection/Prevention Measures

Detection measures (Standard)		
Deploy security mechanisms for intrusion detection and prevention	• Virtual IDS/IPS systems • Firewalls • Anti Virus agents • Integrity checking on all associated VM files	These are required in virtualized data centers in order to manage access credentials and monitor virtual machine behavior to track anomalies.

may occur in the configuration over time, as well as to handle new bugs and weaknesses that were not known at the time of hardening.

"White box" verification processes use knowledge of the internal settings, components, and component interaction to assess conformance to the planned settings and behavior, and are able to concentrate on the system's data flows or expected weak points according to a threat model, if one is used.

"Black box" testing is undertaken by using tools or skilled auditors that attempt to cause some form of undesired behavior. Black-box testing will tend to discover fewer flaws than white-box testing, but the threats that it finds will be demonstrable rather than merely theoretical (as many white-box threats are) and should thus receive a greater level of attention.

Testing of system components that are facing untrusted data sources is especially important. Commonly untrusted data sources will consist of Internet or network connections, but can also consist of more subtle trust boundaries such as those surrounding untrusted code (for example, malware under analysis) or nonsecured hardware (as often occurs in situations where hardware or space is leased).

9.3. Intrusion Detection and Prevention Measures

Intrusion Detection/Prevention System (IDS/IPS) are of paramount importance in all computer network and host systems. Such systems are equally applicable to virtual system architecture. This requires detecting (logging) intrusions, categorizing them, and in some cases providing feedback to security devices in order to change rule sets, thus achieving an IPS architecture. Further, IDS/IPS systems can be used for purposes such as documenting existing threats, identifying problems with the security policy, and preventing users from violating security policies. IDS/IPS systems have become a necessary component to the security infrastructure of nearly every system. Table IV provides a brief summary of the methods used.

9.4. Recovery and Continuity Protection Measures

Recovery and continuity planning is as equally applicable to IT systems in general as it is to virtual systems. Neither is this process (necessarily) distinct from the processes described in Sections 9.1–9.3 earlier, as clearly one may well be a consequence of the other. Well-defined procedures exist for backup and recovery, which themselves are intimately tied to the level of risk deemed appropriate. Again, such procedures are equally applicable to virtual systems as they are to any computer or network system architecture. Table V provides a brief summary of these processes.

Any implementation of virtual machine architectures will require a careful and thorough application of the principles discussed in Sections 9.1–9.4. Not all aspects will be required in every implementation. However, neglecting a meticulous process of security policy implementation, hardening and threat prevention, intrusion analysis and recovery, and continuity measures in such an environment could well lead to catastrophic results often seen in poorly implemented systems in other areas of

Table V. Summary of Recovery and Continuity Measures

Recovery and continuity protection measures		
Normal procedures as with any computer system recovery	• Backup regularly • Create trusted builds and employ tight change management controls on uses • Integrate with integrity checking and attestation	Recovery measures need to address the often complex interaction of the VM components.

computer-communications architectures. Even a simple oversight such as neglecting to change a default password can have serious consequences.

As with any aspect of security, the secure deployment of system virtualization cannot be undertaken with well-planned deployments and operational security alone. The deployed architecture should have regular security audits and penetration tests performed to detect any undiscovered or new types of vulnerability.

10. CONCLUSION

This survey introduced some of the foundational details of system virtualization as is now prevalent in many computing contexts, with a focus on the security implications thereof. While virtualization is an old paradigm, it is also one which has been given new vitality with today's hardware and software architectures, including both hardware systems such as TPMs as well as software systems such as tailored and selectively tuned operating systems designed to support specific applications. Due to the high privilege of the VMM and hardware, few other measures can be taken at present. We examined the various technologies associated with virtualization and in particular, the security issues related to such a tight integration of modern hardware and software. Furthermore, virtualization is being associated with not just multiple host operating systems, but also virtual routing and associated virtual networking, all of which create challenging security issues.

Virtualization has given a new dimension to software studies (for example, malware analysis), where conventional test beds were limited in capacity and conventional simulation often lacked practicality and accuracy because of the complex processes involved. The ability to capture (multiple) snapshots and to be able to roll back and restore a machine's state are of significant value in today's complex design architectures.

This article has demonstrated that implicit trust in a virtualized platform can be a major vulnerability area in system virtualization. To a considerable degree, the OS trusts the hardware in a physical system. Similarly, in a VM the OS trusts the virtual hardware, and thus the VMM. The VMM is a single point of failure, and a malicious, compromised, or otherwise problematic VMM may interfere with the VM. Secure virtualization relies on the authenticity and integrity of the VMM, and in some cases upon the security or identity of the underlying hardware. Since a cloned VM can be difficult or impossible to distinguish from an original, the cloning of VMs is a threat if it happens incorrectly, or of a VM that is not bound to a specific VMM.

A virtualized system abstracted away from the hardware is no longer dependent upon its location, and can even be difficult to define as a single system. A virtualized system can be duplicated (cloned), and every single instance must be found before it can be managed or secured. Even when threats from cloning and snapshots are excluded, threats can still arise from the abstraction of the VM away from hardware.

Some research has gone into the secure sharing of data and security policies within sets of virtual machines, called trusted virtual domains. In VMMs that do not support this functionality, one VM modifying another VM's resources is a privilege escalation problem. Although virtualization can be advantageous for security, it can also be its

downfall. The necessity of a thorough implementation and associated test procedures as discussed in the previous section cannot be overemphasized.

Finally, system virtualization is a two-edged sword with regard to security. It must be wielded with skill and care, as it not only offers improvements in the security isolation and accountability of software, its use also brings with it risks that should be handled with care and forethought. A well implemented, deployed, monitored, and managed virtualization solution can offer security advantages for confidentiality, integrity, and particularly availability, but a failure in any one of these aspects can lead to potentially disastrous results.

ACKNOWLEDGMENTS

We would like to thank the anonymous reviewers for their valuable feedback and constructive suggestions which have helped to improve the quality and presentation of this article. We also express our gratitude to Deb Frincke for initiating the early discussions on virtualization which led in part towards the completion of this work. Finally, we are also thankful to Michael Huth and Chris Hankin for their support and encouragements throughout the preparation of this article.

REFERENCES

Adams, K., and Agesen, O. 2006. A comparison of software and hardware techniques for x86 virtualization. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* ACM Press, 2.

Advanced Micro Devices. 2008. AMD-V^{TM} nested paging. http://developer.amd.com/assets/NPT-WP-1 1-final-TM.pdf.

Advanced Micro Devices. 2010. AMD virtualization (AMD-V)^{TM} technology. http://sites.amd.com/us/business/itsolutions/virtualization/Pages/amd-v.aspx.

Athreya, M. B. 2010. *Subverting Linux On-the-Fly Using Hardware Virtualization Technology*. http://smartech.gatech.edu/handle/1853/34844.

Baldwin, A., Dalton, C., Shiu, S., Kostienko, K., and Rajpoot, Q. 2009. Providing secure services for a virtual infrastructure. *ACM SIGOPS Oper. Syst. Rev. 43*, 1, 44.

Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., et al. 2003. Xen and the art of virtualization. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP'03).* ACM Press, 164–177.

Barroso, L. A. and Hölzle, U. 2007. The case for energy-proportional computing. *Comput. 40*, 12, 33–37.

Berger, S., Perez, R., Caceres, R., Sailer, R., Goldman, K. A., and van Doorn, L. 2006. vTPM: Virtualizing the trusted platform module. In *Proceedings of the 15th USENIX Security Symposium*. 1–16.

Bratus, S., Johnson, P. C., Ramaswamy, A., Smith, S. W., and Locasto, M. E. 2009. The cake is a lie: Privilege rings as a policy resource. In *Proceedings of the 1st ACM Workshop on Virtual Machine Security.* ACM Press, 33–37.

Bratus, S., Locasto, M., and Ramaswamy, A. 2008. Traps, events, emulation, and enforcement: Managing the yin and yang of virtualization-based security. In *Proceedings of the 1st ACM Workshop on Virtual Machine Security.* ACM Press, 49–58.

Bugnion, E., Devine, S., Govil, K., and Rosenblum, M. 1997. Disco: Running commodity operating systems on scalable multiprocessors. *ACM Trans. Comput. Syst. 15,* 4, 412–447.

Capelis, D. J. 2007. Virtualization: Enough holes to work vegas. In *Proceedings of Defcon 15*.

Carbone, M., Zamboni, D., and Lee, W. 2008. Taming virtualization. *IEEE Secur. Privacy Mag. 6*, 1, 65–67.

Catuogno, L., Dmitrienko, A., Eriksson, K., and Kuhlmann, D. G. 2010. Trusted virtual domains—Design, implementation and lessons learned. In *Trusted Systems*, Springer, 156–179.

Champagne, D. 2010. Scalable security architecture for trusted software, Princeton University, Ph.D dissertation Princeton, NJ.

Chen, X., Garfinkel, T., Lewis, E. C., Subrahmanyam, P., Waldspurger, C. A., Boneh, D., et al. 2008. Overshadow. In *Proceedings of the 13th International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS).* ACM Press, 2.

Chowdhury, N. M. M. K and Boutaba, R. 2009. Network virtualization: State of the art and research challenges. *IEEE Comm. Mag. 47*, 7, 20–26.

CHRISTODORESCU, M., SAILER, R., SCHALES, D. L., SGANDURRA, D., AND ZAMBONI, D. 2009. Cloud security is not (just) virtualization security. In *Proceedings of the ACM Workshop on Cloud Computing Security (CCSW'09).* ACM Press, 97.

COLLIER, G., PLASSMAN, D., AND PEGAH, M. 2007. Virtualization's next frontier: Security. In *Proceedings of the 35th Annual ACM SIGUCCS Fall Conference.* ACM, 34–36.

COMMONCRITERIA. 2008. Certification report for processor resource/system manager (PR/SM) for the IBM system z10 EC GA1. Tech. rep. BSI-DSZ-CC-0460-2008. *Informationstechnik,* 1–38. http://www.commoncriteriaportal.org/files/epfiles/0460a.pdf.

CRISCIONE, C. 2010. Virtually pwned - Pentesting virtualization. In *Proceedings of Blackhat USA*. http://media.blackhat.com/bh-us-10/presentations/Criscione/BlackHat-USA-2010-Criscione-Virtually-Pwned-slides.pdf.

DAI ZOVI, D. A. 2006. Hardware virtualization rootkits. http://www.orkspace.net/secdocs/Conferences/BlackHat/USA/2006/Hardware Virtualization Based Rootkits.pdf.

DALTON, C. I., PLAQUIN, D., WEIDNER, W., KUHLMANN, D., BALACHEFF, B., AND BROWN, R. 2009. Trusted virtual platforms. *ACM SIGOPS Oper. Syst. Rev. 43,* 1, 36. ACM.

DEWAN, P., DURHAM, D., KHOSRAVI, H., LONG, M., AND NAGABHUSHAN, G. 2008. A hypervisor-based system for protecting software runtime memory and persistent storage. In *Proceedings of the Spring Simulation Multiconference*. Society for Computer Simulation International, 828–835.

DINABURG, A., ROYAL, P., SHARIF, M., AND LEE, W. 2008. Ether. In *Proceedings of the 15th ACM Conference on Computer and Communications Security (CCS'08).* ACM Press, 51.

DOMINGUES, P., MARQUES, P., AND SILVA, L. 2005. Resource usage of windows computer laboratories. In *Proceedings of the International Conference on Parallel Processing Workshops (ICPPW'05).* IEEE, 469–476.

DOWTY, M. AND SUGERMAN, J. 2009. GPU virtualization on vmware's hosted i/o architecture. *ACM SIGOPS Oper. Syst. Rev. 43*, 3, 73.

DUFLOT, L. 2008. CPU bugs, cpu backdoors and consequences on security. *J. Comput. Virol. 5*, 2, 91–104.

FERRIE, P. 2007a. Attacks on more virtual machine emulators. http://pferrie.tripod.com/papers/attacks2.pdf.

FERRIE, P. 2007b. Attacks on virtual machine emulators. http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf.

FERRIE, P., HAPI, H., AND JOY, J. O. Y. 2006. Virus analysis tumours and polips. *Virus Bull.*, 4–8.

FORD, R., AND ALLEN, W. H. 2007. How not to be seen II: The defenders fight back. *IEEE Secur. Privacy Mag. 5*, 6, 65–68.

FRANKLIN, J, SESHADRI, A, QU, N, CHAKI, S., AND DATTA, A. 2008a. Attacking, repairing, and verifying SecVisor: A retrospective on the security of a hypervisor. Cylab Tech. rep. CMU-CyLab-08-008.

FRANKLIN, J., LUK, M., MCCUNE, J. M., SESHADRI, A., PERRIG, A., AND VANDOORN, L. 2008b. Remote detection of virtual machine monitors with fuzzy benchmarking. *ACM SIGOPS Oper. Syst. Rev. 42*, 3, 83.

FUCHI, K., TANAKA, H., MANAGO, Y., AND YUBA, T. 1969. A program simulator by partial interpretation. In *Proceedings of the 2nd Symposium on Operating Systems Principles (SOSP'69).* ACM Press, 97.

GARFINKEL, T, AND ROSENBLUM, M. 2005. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *Proceedings of the 10th Conference on Hot Topics in Operating Systems (HOTOS'05).* Vol. 10, ACM Press, 6.

GARFINKEL, T., ADAMS, K., WARFIELD, A., AND FRANKLIN, J. 2007. Compatibility is not transparency: VMM detection myths and realities. In *Proceedings of the 11th USENIX Workshop on Hot Topics in Operating Systems.* USENIX Association, 1–6.

GARFINKEL, T. AND ROSENBLUM, M. 2003. A virtual machine introspection based architecture for intrusion detection. In *Proceedings of the Network and Distributed Systems Security Symposium.* Vol. 1. 253–285.

GEBHARDT, C., DALTON, C., AND BROWN, R. 2008. Preventing hypervisor-based rootkits with trusted execution technology. *Netw. Secur. 11*, 7–12.

GEBHARDT, C, AND TOMLINSON, A. 2008. Security consideration for virtualization. Tech. rep. RHUL–MA–2008–16. In *Proceedings of the 3rd Asia Pacific Trusted Infrastructure Technologies Conference.* 19–29.

GOLDBERG, R. P. 1973. Architecture of virtual machines. In *Proceedings of the Workshop on Virtual Computer Systems*. ACM Press, 74–112.

GOLDBERG, R. P. 1974. Survey of virtual machine research. *IEEE Comput. 7*, 3, 34–45.

GRIFFIN, J. L., JAEGER, T., PEREZ, R., SAILER, R., VAN DOORN, L., ET AL. 2005. Trusted virtual domains: Toward secure distributed services. In *Proceedings of the 1st IEEE Workshop on Hot Topics in System Dependability.*

GUERON, S. AND SEIFERT, J. P. 2009. On the impossibility of detecting virtual machine monitors. *Emerg. Challen. Secur. Privacy Trust 297*. Springer, 143–151.

HALDAR, V., CHANDRA, D., AND FRANZ, M. 2004. Semantic remote attestation—A virtual machine directed approach to trusted computing. In *Proceedings of the 3rd Conference on Virtual Machine Research and Technology Symposium*. USENIX.

INTEL. 2003. *Intel® Trusted Execution Technology Architectural Overview*. http://www.intel.com/technology/security/downloads/arch-overview.pdf

INTEL. 2009. Intel® trusted execution technology (Intel® TXT) software development guide. *architecture*.

IRVINE, C. E. AND LEVITT, K. 2007. Trusted hardware: Can It be trustworthy? In *Proceedings of the 44th ACM/IEEE Design Automation Conference*. 1–4.

IVANOV, I. AND GUEORGUIEV, V. 2008. Operating systems virtualisation and security-modern aspects and an open trusted computing project. In *Proceedings of the International Scientific Conference Computer Science.* 335–339.

JAEGER, T., SAILER, R, AND SREENIVASAN, Y. 2007. Managing the risk of covert information flows in virtual machine systems. In *Proceedings of the 12th ACM Symposium on Access Control Models and Technologies (SACMAT'07)*. ACM Press, 81.

JANSEN, B., RAMASAMY, H., SCHUNTER, M., AND TANNER, A. 2008. Architecting dependable and secure systems using virtualization. In *Architecting Dependable Systems V*, Springer, 124–149.

JIN, H., XIANG, G., ZHAO, F., ZOU, D., LI, MIN, AND SHI, L. 2009. VMFence. In *Proceedings of the 3rd International Conference on Ubiquitous Information Management and Communication (ICUIMC'09).* ACM Press, 391.

JONES, S. T., ARPACI-DUSSEAU, A. C., AND ARPACI-DUSSEAU, R. H. 2008. VMM-Bbased hidden process detection and identification using lycosid. In *Proceedings of the 4th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'08).* ACM Press, 91.

KARGER, P. A., AND SAFFORD, D. R. 2008. I/O for virtual machine monitors: Security and performance issues. *IEEE Secur. Privacy Mag. 6*, 5, 16–23.

KING, S. T., CHEN, P. M., VERBOWSKI, C., WANG, H. J., AND LORCH, J. R. 2006. SubVirt: Implementing malware with virtual machines. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'06).* IEEE, 314–327.

KIST, A. A. 2009. Staged request routing for reduced carbon footprints of large scale server systems. In *Proceedings of the Australasian Telecommunication Networks and Applications Conference (ATNAC)*. IEEE, 1–5.

KORTCHINSKY, K. 2009. Cloudburst—A VMware guest to host escape story. http://www.blackhat.com/presentations/bh-usa-09/KORTCHINSKY/BHUSA09-Kortchinsky-Cloudburst-SLIDES.pdf.

KOURAI, K. AND CHIBA, S. 2005. HyperSpector: Virtual distributed monitoring environments for secure intrusion detection. In *Proceedings of the 1st ACM/USENIX International Conference on Virtual Execution Environments*. ACM, 197–207.

LAADAN, O. AND NIEH, J. 2010. Operating system virtualization: Practice and experience. In *Proceedings of the 3rd Annual Haifa Experimental Systems Conference*. ACM 1–12.

LANDWEHR, C. E., BULL, A. R., MCDERMOTT, J. P., AND CHOI, W. S. 1994. A taxonomy of computer program security flaws, *ACM Comput. Surv. 26*, 3, 211–254.

LAUREANO, M., MAZIERO, C., AND JAMHOUR, E. 2004. Intrusion detection in virtual machine environments. In *Proceedings of the 30th Euromicro Conference*. IEEE, 520–522.

LINDQVIST, U. AND JONSSON, E. 1997. How to systematically classify computer security intrusions. In *Proceedings of the IEEE Symposium on Security and Privacy.* IEEE Computer Society.

LISTON, T. AND SKOUDIS, E. 2006. On the cutting edge: Thwarting virtual machine detection. In *Proceedings of the SANSFIRE Conference*. 1–27.

LITTY, L. 2005. Hypervisor-Based intrusion detection. Master's thesis, Department of Computer Science, University of Toronto, Canada.

LITTY, L. AND LIE, D. 2006. Manitou: A layer-below approach to fighting malware. In *Proceedings of the 1st Workshop on Architectural and System Support for Improving Software Dependability.* ACM, 6–11.

MADNICK, S. E. AND DONOVAN, J. J. 1973. Application and analysis of the virtual machine approach to information system security and isolation. In *Proceedings of the Workshop on Virtual Computer Systems*. Vol. 4102, ACM Press, 210–224.

MATTHEWS, J. N., HERNE, J. J., DESHANE, T. M., JABLONSKI, P. A., CHERIAN, L. R., AND MCCABE, M. T. 2005. Data protection and rapid recovery from attack with a virtual private file server and virtual machine appliances. In *Proceedings of the IASTED International Conference on Communication, Network and Information Security.* 170–181.

MICROSOFT. 2010. Microsoft security development lifecycle (SDL) threat modeling tool. http://www.microsoft.com/security/sdl/adopt/threatmodeling.aspx.

MILLER, K. AND PEGAH, M. 2007. Virtualization: Virtually at the desktop. In *Proceedings of the 35th Annual ACM SIGUCCS Conference on User Services.* 255–260.

MSDN. 2010. Windows virtual pc interfaces. http://msdn.microsoft.com/enus/library/dd796756(VS.85).aspx.

NANCE, K., HAY, B., AND BISHOP, M. 2008. Virtual machine introspection observation or interference? *IEEE Secur. Privacy Mag. 6*, 5, 32–37.

NEWSHAM, G. R. AND TILLER, D. K. 1994. The energy consumption of desktop computers: Measurement and savings potential. *IEEE Trans. Ind. Appl. 30*, 4, 1065–1072.

NOMOTO, T., OYAMA, Y., EIRAKU, H., SHINAGAWA, T., AND KATO, K. 2010. Using a hypervisor to migrate running operating systems to secure virtual machines. In *Proceedings of the IEEE 34th Annual Computer Software and Applications Conference*. IEEE Computer Society, 37–46.

OMELLA, A. A. 2006. Methods for virtual machine detection. *Grupo S21sec Gestión SA*. http://www.s21sec.com/descargas/vmware-eng.pdf.

OPPLIGER, R. AND RYTZ, R. 2005. Does trusted computing remedy computer security problems? *IEEE Sec. Privacy Mag. 3*, 2, 16–19.

ORMANDY, T. 2007. An empirical study into the security exposure to hosts of hostile virtualized environments. In *Proceedings of the CanSecWest Applied Security Conference.* 1–10.

OWASP. 2010. (Open Web Application Security Project) OWASP threat risk modeling. http://www.owasp.org/index.php/Threat_Risk_Modeling.

PAYNE, B. D., CARBONE, M. D. P. D. A., AND LEE, W. 2007. Secure and flexible monitoring of virtual machines. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC'07).* IEEE, 385–397.

PEREZ, R., VAN DOORN, L., AND SAILER, R. 2008. Virtualization and hardware-based security. *IEEE Sec. Privacy Mag. 6*, 5, 24–31.

PETERSON, Z. AND BURNS, R. 2005. Ext3cow: A time-shifting file system for regulatory compliance. *ACM Trans. Storage 1*, 2, 190–212.

PFAFF, B. AND ROSENBLUM, M. 2003. Terra: A virtual machine-based platform for trusted computing. In *Proceedings of the Symposium on Operating Systems Principles* (*SOSP'03).* ACM, 193–206.

POPEK, G. J. AND GOLDBERG, R. P. 1974. Formal requirements for virtualizable third generation architectures. *Comm. ACM 17*, 7, 412–421.

PRICE, M. 2008. The paradox of security in virtual environments. *Comput. 41*, 11, 22–28.

RAFFETSEDER, T., KRUEGEL, C., AND KIRDA, E. 2007. Detecting system emulators. In *Information Security.* Lecture Notes in Computer Science, vol. 4779, Springer, 1–18.

RAMOS, J. 2009. Security challenges with virtualization. Ph.D. thesis, Libson University Faculty of Computing, 121.

RASHID, A., MENS, T., BUCKLEY, J., AND ZENGER, M. 2003. Towards a taxonomy of software evolution. In *Proceedings of the International Workshop on Unanticipated Software Evolution*. 1–18.

REUBEN, J. 2007. A survey on virtual machine security. http://www.tml.tkk.fi/Publications/C/25/papers/Reuben_final.pdf.

RISTENPART, T., TROMER, E., SHACHAM, H., AND SAVAGE, S. 2009. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. *Artif. Intell.,* 199–212.

ROSE, R. 2004. Survey of system virtualization techniques. http://www.robertwrose.com/vita/rose-virtualization.pdf.

ROSENBLUM, M. AND GARFINKEL, T. 2005. Virtual machine monitors: Current technology and future trends. *Comput. 38*, 5, 39–47.

ROSIN, R. F. 1969. Contemporary concepts of microprogramming and emulation. *ACM Comput. Surv. 1*, 4, 197–212.

RUTKOWSKA, J. AND TERESHKIN, A. 2008. Bluepilling the xen hypervisor. http://invisiblethingslab.com/bh08/part3.pdf.

RUTKOWSKA, J. 2006. Subverting vista kernel for fun and profit. http://www.blackhat.com/presentations/bh-usa-06/BH-US-06-Rutkowska.pdf.

SAILER, R, ZHANG, X., JAEGER, T., AND VAN DOORN, L. 2004. Design and Implementation of a tcg-based integrity measurement architecture. In *13th USENIX Security Symposium.* Vol. 8.

SAILER, R., JAEGER, T., VALDEZ, E., CACERES, R., PEREZ, R., BERGER, S., GRIFFIN, J. L., AND VAN DOORN, L. 2005. Building a mac-based security architecture for the xen open-source hypervisor. In *Proceedings of the 21st Annual Computer Security Applications Conference (ACSAC'05)*. 276–285.

SCOTT, S. L., VALLÉE, G., NAUGHTON, T., TIKOTEKAR, A., ENGELMANN, C., AND ONG, H. 2010. System-Level virtualization research at Oak Ridge National Laboratory. *Future Generation Comput. Syst. 26*, 3, 304–307.

SECURITY, N. 2010. Virtualisation worries. *Netw. Secur. 5*, 20.

SESHADRI, A., LUK, M., QU, N., AND PERRIG, A. 2007. SecVisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In *Proceedings of the 21st ACM SIGOPS Symposium on Operating Systems Principles.* ACM, 335–350.

SHARIF, M. I., LEE, W., CUI, W., AND LANZI, A. 2009. Secure in-vm monitoring using hardware virtualization. In *Proceedings of the 16th ACM Conference on Computer and Communications Security (CCS'09).* ACM Press, 477.

SIEBENLIST, F. 2009. Challenges and opportunities for virtualized security in the clouds. In *Proceedings of the 14th ACM Symposium on Access Control Models and Technologies (SACMAT'09).* ACM Press, 1. http://portal.acm.org/citation.cfm?doid=1542207.1542209.

SKAPINETZ, K. 2007. Virtualisation as a blackhat tool. *Netw. Secur. 10*, 4–7.

SMITH, J. E. AND NAIR, R. 2005. The architecture of virtual machines. *IEEE Comput. 38*, 5 , 32–38.

SOUNDARARAJAN, V. AND ANDERSON, J. M. 2010. The impact of management operations on the virtualized datacenter. In *Proceedings of the 37th Annual Iternational Symposium on Computer architecture (ISCA'10).* ACM Press, 326.

SOURCEFORGE. 2010. Integrity measurement architecture (IMA) - SourceForge.net. http://sourceforge.net/projects/linux-ima/.

STRONGIN, G. 2005. Trusted computing using amd "pacifica" and "presidio" secure virtual machine technology. *Inf. Secur. Tech. Rep. 10*, 2, 120–132.

SUBASHINI, S. AND KAVITHA, V. 2011. A survey on security issues in service delivery models of cloud computing. *J. Netw. Comput. Appl. 34*, 1, 1–11. http://linkinghub.elsevier.com/retrieve/pii/S1084804510001281.

SUGERMAN, J., VENKITACHALAM, G., AND LIM, B.-HONG. 2001. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *Proceedings of the Usenix Annual Technical Conference*. Vol. 7. USENIX Association, 1–15.

TCG. 2010. Trusted computing group. http://www.trustedcomputinggroup.org/.

TOMLINSON, C. 2009. Trusted virtual disk images. In *Proceedings of the 1st International Conference Future of Trust in Computing.* 197.

TRUSTED COMPUTING GROUP. 2007. *TPM Main Part 1 Design Principles Version 1.2 (Level 2 Revision 103). ReVision*, 182.

UHLIG, R., NEIGER, G., RODGERS, D., SANTONI, A. L., MARTINS, F. C. M., ANDERSON, A. V., ET AL. 2005. Intel virtualization technology. *Comput. 38*, 5, 48–56.

VAARALA, S. 2006. Security considerations of commodity x86 virtualization. Helsinki University of Technology-Telecommunications.

VAN CLEEFF, A., PIETERS, W., AND WIERINGA, R. J. 2009. Security implications of virtualization: A literature study. In *Proceedings of the International Conference on Computational Science and Engineering*. IEEE, 353–358.

VASAN, A., SIVASUBRAMANIAM, A., SHIMPI, V., SIVABALAN, T., AND SUBBIAH, R. 2010. Worth their watts? An empirical study of datacenter servers. In *Proceedings of the 16th International Symposium on High-Performance Computer Architecture*. IEEE, 1–10.

VASUDEVAN, A., MCCUNE, J., QU, N., AND VAN DOORN, L. 2010. Requirements for an integrity-protected hypervisor on the x86 hardware virtualized architecture. In *Proceedings of the 3rd International Conference on Trust and Trustworthy Computing (TRUST'10)*. Springer, 141–165.

VMWARE. 2010. VIX API. http://www.vmware.com/support/developer/vix-api/.

WANG, S.-X., WANG, Y.-C., AND TIAN, W. Z. 2010. Research on trusted computing implementations in windows. In *Proceedings of the International Conference of Information Science and Management Engineering*. IEEE, 446–449.

WANG, Z. AND LEE, R. 2006. Covert and side channels due to processor architecture. In *Proceedings of the 22nd Annual Computer Security Applications Conference (ACSAC'06)*. 473–482.

WANG, Z. AND LEE, R. 2007. New cache designs for thwarting software cache-based side channel attacks. *ACM SIGARCH Comput. Archit. News 35*, 2, 494.

WHITAKER, A., COX, R. S., SHAW, M., AND GRIBBLE, S. D. 2005. Rethinking the design of virtual machine monitors. *Comput. 38*, 5, 57–62.

WIMMER, M. 2008. Virtual security. In 1st *Conference on Computer Security Incident Handling.* Vol. 20.

WLODARZ, J. J. 2007. Virtualization: A double-edged sword. http://arxiv.org/abs/0705.2786.

WOJTCZUK, R. AND RUTKOWSKA, J. 2009. Attacking intel trusted execution technology. http://www.invisiblethingslab.com/resources/bh09dc/Attacking Intel TXT-paper.pdf.

WU, X. AND MA, W. 2010. Hypervisor based detection and prevention for packed malware. http://www.ece.tamu.edu/~tristanw/files/Wu_Xiaojian_Ma_Weiqin_Report.pdf.

XUAN, C., COPELAND, J., AND BEYAH, R. 2009. Toward revealing kernel malware behavior in virtual execution environments. In *Recent Advances in Intrusion Detection*. Springer, 304–325.

YAMAHATA, I. 2008. Paravirt_ops on IA64. *Kernel.org*. http://www.kernel.org/doc/Documentation/ia64/paravirt_ops.txt.

YU, L., WENG, C., LI, M., AND LUO, Y. 2010. Security challenges on the clone, snapshot, migration and rollback of xen based computing environments. In *Proceedings of the 5th Annual ChinaGrid Conference*. IEEE, 223–227.

YUNIS, M. AND HUGHES, J. 2008. Real security in virtual systems: A proposed model for a comprehensive approach to securing virtualized environments. *Issues Inf. Syst. IX,* 2, 385–395.

Received January 2011; revised July 2011; accepted September 2011