Comprehensive Security Assurance Measures for Virtualized Server Environments

Ramaswamy Chandramouli (mouli@nist.gov)

National Institute of Standards & Technology, Gaithersburg MD, USA

1. INTRODUCTION

Virtualization is the dominant technology employed in enterprise data centers and those used for offering cloud computing services. This technology has resulted in what is called a virtualized infrastructure. From a computing and communication point of view, the two forms of virtualization that have made significant impacts are Server (or Hardware) virtualization and Operating System (OS) virtualization. Server virtualization is enabled by software called a Hypervisor —functionally, an operating system kernel with some additional kernel modules that provides an abstraction of the hardware, enabling multiple independent computing stacks called virtual machines (VMs), each with its own OS and applications, to be run on a single physical host. While access to CPU and memory (to ensure process isolation) are handled directly by the hypervisor (through instruction set (CPU) virtualization and memory virtualization respectively with or without assistance from hardware), it handles the mediation of access to devices by calling on software modules running either in the kernel or in dedicated VMs called Device-driver VMs. This physical host is called a virtualized server or hypervisor host.

Operating system virtualization, on the other hand, is enabled purely by using OS kernel-level features (e.g., namespaces, Cgroups, etc. in Linux OS distributions) that allow for the definition of encapsulated entities called containers, each running as an isolated process (i.e., hosting one or more applications) on the same OS kernel. The creation, configuration, and running of containers is enabled by software called *container runtime*, which makes direct Application Programming Interface (API) calls to the OS kernel for performing these functions. Thus, we see that hypervisor software provides abstraction of the hardware while container runtime software enables the creation of an artifact (called a container) that provides abstraction of the OS.

The initial motivation for server virtualization—even before their deployment in data centers used for cloud services—is better utilization of hardware resources with the added benefit of reduced floor space and power consumption. After the advent of cloud services, virtualized servers have become the de facto component of data centers' infrastructure, especially for those offering Infrastructure as a Service (IaaS). This is because a VM image, being a complete computing stack with its virtual hardware resource definitions and OS (called Guest OS) can be offered as a basic computing unit to the cloud service consumer (CSC) for this type of cloud service.

Out of the two forms of virtualization referred to above (i.e., hardware virtualization and OS virtualization), the focus of this manuscript is on hardware virtualization and its resulting artifact virtualized server. The data center ecosystem consists of multiple virtualized servers with its hardware, the core virtualization software (the hypervisor), and VMs. The ecosystem, together with the network inside of each virtualized server (called virtual network) and the linking of virtualized servers, constitutes the *virtualized server environment*. The goal of this manuscript is to develop security assurance for all

components of a virtualized server environment. The approach adopted in this manuscript for realizing this goal is as follows:

- Study the functions of various components in a virtualized server environment
- Identify threats to the secure execution of those functions
- Develop the security assurance measures to counter those threats

For the hypervisor, which is the core component of the environment, there are multiple commercial product offerings. Since the objective of this manuscript is to outline product-agnostic security assurance measures, the approach adopted is to identify a set of baseline or canonical functions of the hypervisor that will form the basis for threat identification.

The overall organization of this manuscript is as follows. In Section 2, a brief technology overview of components in a virtualized server environment is provided. The hardware functions in a virtualized server are briefly described in Section 3. Section 4 identifies and elaborates on the baseline functions of the hypervisor and the threats to those functions. The threat to the secure execution of VM-resident programs, such as Guest OS and applications, form the subject matter for Section 5. Section 6 describes typical virtual network configurations in a virtualized server and the protections required for those configurations. The security assurance measures for hypervisor, VM, and virtual networks are developed in Sections 7, 8, and 9, respectively. The security assurance for booting a virtualized server platform is described in section 10. Section 11 provides the summary and conclusions.

2. VIRTUALIZED SERVER ENVIRONMENT – A TECHNOLOGY OVERVIEW

From the perspective of this manuscript, a virtualized server environment consists of the following components:

- A physical host, called a virtualized server or hypervisor host, with server virtualization software (hypervisor and its associated modules), along with multiple computing stacks (i.e., Virtual Machines or VMs) running on it. The hypervisor host has hardware extensions to assist virtualization.
- A virtual network, or software-defined network, inside the virtualized server, consisting of software-defined network devices. This network is configured with network segmentation techniques such as Virtual Local Area Network (VLAN) and overlay-based network (e.g., VXLAN) that span multiple virtualized servers and enable logical segmentation of the VMs distributed throughout the data center.

A Virtualized server can have two different types of hypervisors: one that can be mounted directly on the hardware (called bare metal) and the other that requires an OS (called host OS) for its installation. These two types of hypervisors are also called Type 1 and Type 2 hypervisor, respectively. The VMs, also called Guests, host and run the application programs with the help of an OS (called the Guest OS). The virtualized server platforms, consisting of Type 1 and Type 2 hypervisors, are shown in Figure 1.

In addition to classification based on the platform on which it is mounted (bare metal or host OS), hypervisors can be classified based on the type of virtualization they provide for devices. In one approach, called Full Virtualization, the hypervisor will expose the interface of a well-known hardware device that is available in the real world to the VM, and it will completely emulate the behavior of that device. Emulation allows the programs running in VMs to use the guest OS drivers that were designed to interact with the emulated device without installing any special driver or tool specified by the hypervisor vendor. In another approach called para-virtualization, the hypervisor provides an interface of an artificial device

to the guest that has no corresponding hardware device. This artificial device is a software-only device that presents a lightweight interface designed and optimized to work in virtual environments. However, the performance improvement made possible with para-virtualization requires that the guest OS and device drivers be modified to communicate directly with the hypervisor through a special interface called hypercall interface.

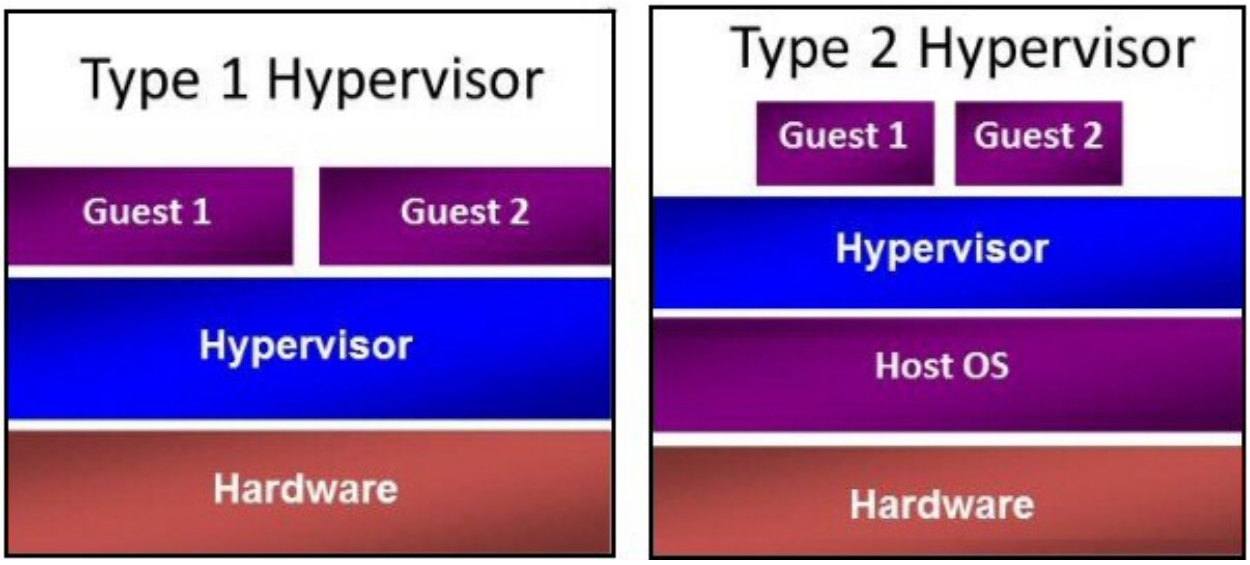

Figure 1: Virtualized Server Platforms with Type 1 and Type 2 Hypervisor

The hardware extensions in a hypervisor host assist virtualization through functions such as instruction handling and memory management. Hardware features, such as CPU/Instruction Set virtualization and memory virtualization, respectively, enable these functions and are described in detail in Section 3.

All Physical hosts or servers are connected to the data center network (or become nodes of the data center network) using a physical device called a Network Interface Card (NIC). An independent computing stack such as a VM requires a similar connection to the networking infrastructure of the data center. This is enabled by an artifact called a Virtual NIC (vNIC), which is the software defined analog of the physical NIC (pNIC). In addition, since there are multiple VMs or containers inside a single physical host, there is the need to provide interconnection among the multiple VMs within it. This requirement necessitates the creation of a software-defined network within a physical host (called virtual network) with switching/bridging functions performed by software-defined entities (called virtual switches/virtual bridges), which are software analogs of the corresponding physical network devices.

3. VIRTUALIZED SERVER HARDWARE FUNCTIONS

As already stated, the hardware of a virtualized server provides two features to assist the virtualization function of the hypervisor: Instruction Set Virtualization and Memory Virtualization. These hardware-based functions provided by chip vendors are mature technologies that have been utilized for more than a decade and whose known vulnerabilities have already been addressed. Therefore, no threats need to be considered for these functions.

Instruction Set Virtualization: The processor architecture of the hardware is generally designed to operate OS instructions at a higher privilege level than the application instructions. However, in a virtualized server, the guest OS instructions cannot be executed at the highest privilege level (e.g., Ring 0 in x86 architectures) since the hypervisor that mediates the access of various VMs to hardware resources of the virtualized server must operate at a higher privilege level than any guest OS. To facilitate this, hardware architectures (e.g., Intel, AMD[1]) provide two modes of operation (host and guest) for the processor, each with four hierarchical privilege levels (Ring 0 thru Ring 3). Additionally, among the two modes, the host or root mode has a higher privilege for executing CPU instructions than the guest or non-root mode, and it is in the former mode that hypervisor instructions are executed. The guest mode is used for executing instructions from guest OSs and VM-based applications.

Contribution to Hypervisor Security Assurance Verification: By running the hypervisor in root mode and guest OSs in non-root mode at privilege or ring level 0, the hypervisor is guaranteed safety from at least any instruction set-type attacks by any Guest OS. This safety is ensured by allowing the hardware to trap privileged instructions from a guest OS to run in non-root mode. Additionally, when the hypervisor does not have to perform additional functions (e.g., translating sensitive instructions using techniques such as binary translation) for handling the instructions, the code executing with privileges is reduced in the hypervisor, making the trusted computing base (TCB) smaller and enabling better assurance verification.

Memory Virtualization: Hardware-assisted memory virtualization is provided using two levels of page tables (Guest page table and Host page table). The guest page table, maintained by a guest OS, translates from guest virtual to guest physical addresses, whereas the host page table translates from guest physical to host physical addresses.

Contribution to Hypervisor Security Assurance Verification: The availability of a hardware-based host page table eliminates the need for the hypervisor to generate and maintain shadow page tables, providing the same security advantage (i.e., smaller TCB) as for Instruction Set Virtualization.

4. HYPERVISOR BASELINE FUNCTIONS AND THREATS

The hypervisor is the core component in the virtualized server platform, and its baseline functions are as follows [1]:

- HY-BF1: VM Process Isolation – The hypervisor, in addition to its software-based tasks, leverages the hardware extension features in two ways to enforce process isolation. First, it runs in higher privilege mode (i.e., host mode) and uses the special instruction *vmrun* to switch the CPU to lower privilege mode (i.e., guest mode) for VMs to begin execution. Second, before VMs start running, it creates a data structure called Virtual Machine Control Block (VMCB) for recording the

[1] Any mention of commercial products or organizations is for informational purposes only; it is not intended to imply recommendation or endorsement by the National Institute of Standards and Technology, nor is it intended to imply that the products identified are necessarily the best available for the purpose.

execution state of VMs, and it leverages the memory management features (e.g., two layered page tables) of the hardware to enforce separation of memory address spaces for VMs.

- HY-BF2: Devices Mediation & Access Control – Mediates access to all devices (e.g., Storage, Network, etc.)
- HY-BF3: Execution of Guest Instructions through Hypercall Interface – This functionality is only applicable to para-virtualized hypervisors, which handle certain device access instructions from guests directly through its hypercall interface rather than through the combination of *vmexit* and host mode transition events.
- HY-BF4: VM Lifecycle Management – Performs all functions including creation and management of VM images, control of VM states (Start, Pause, Stop, etc.), VM migration, creation of snapshots, VM monitoring, and policy enforcement
- HY-BF5: Management of Hypervisor – Setting various configuration parameters, such as CPU and memory allocation logic, including those for the Virtual Network inside the hypervisor; also includes tasks such as updates and application of patches to hypervisor modules

To execute the above baseline functions, different software modules are needed, which makes the hypervisor a non-monolithic software. The software module that carries out each baseline function along with the location in the overall virtualized server platform architecture where each resides is given in Table 1 below.

Table 1: Hypervisor Baseline Functions & Deployment Locations

Baseline Function	Component (Software Module)	Location
VM Process Isolation (HY-BF1)	Hypervisor Kernel	Either an OS kernel (along with a kernel module) itself or a component installed on a full-fledged OS (Host OS)
Devices Mediation & Access Control (HY-BF2)	Device emulator or Device driver	Either in a dedicated VM (called Device-driver VM) or in the hypervisor kernel itself
Execution of Guest Instructions through hypercall interface (HY-BF3)	Hypervisor Kernel	Pertain to only para-virtualized hypervisors and handled by hypercall interfaces in that type of hypervisor
VM Lifecycle Management (HY-BF4)	A management daemon	Installed on top of the hypervisor kernel but runs in unprivileged mode
Management of Hypervisor (HY-BF5)	A set of tools with CLI (command line interface) or a GUI	A console or shell running on top of the hypervisor kernel

The tasks involved in implementing each of the above baseline functions are described in more detail in the following subsections and accompanied by statements of potential threats to secure execution of these tasks. However, the virtual network configuration tasks (in HY-BF5), including the set-up for VM network traffic monitoring (in HY-BF4), are discussed under a separate section (Section 6) due to their critical roles in the security of the entire virtualized server environment.

4.1 Potential Threats to VM Process Isolation (HY-BF1)

The threats to VM process isolation are the results of two primary causes [1]:

Breach of Process Isolation – VM Escape: Major threats to any hypervisor come from malicious VM-resident programs. These programs can subvert the isolation function provided by the Virtual Machine Monitor (VMM)/hypervisor to hardware resources such as memory pages. In other words, these programs can, under some conditions, access areas of memory belonging to the hypervisor or other VMs or devices (e.g., memory mapped devices) that they are not authorized to access. Examples of such attacks include some crafted applications in VM executing arbitrary code on the host OS [2] or VM programs accessing areas of memory that are not allocated to them, thereby causing corruption or information leakage [3]. Extreme attack scenarios may include VMs with malicious programs taking control of the hypervisor to install rootkits or attack other VMs on the same virtualized server. These threats are mainly due to code flaws in the hypervisor.

Denial-of-Service to some VMs: Hypervisor offerings come with sophisticated CPU and memory allocation options. Improper use of these configuration options may result in some VMs hogging resources, resulting in denial-of-service or the inability to meet the critical availability requirement for some VMs.

4.2 Potential Threats to Devices Mediation (HY-BF2)

The applications executing in VMs need to access devices such as video output, network (for communication), or block (storage) devices. There are three common approaches to handling devices by virtualized servers: (a) Passthrough, (b) Emulation, and (c) Para-virtualization [4]. Out of these, the passthrough approach provides exclusive access to a device for a VM. Since this is not a scalable approach, it is adopted for VMs running specialized applications. The para-virtualization approach was generally designed for enhancing performance for accessing devices. In this approach, the hypervisor provides to the guest an interface of an artificial device that has no corresponding hardware counterpart. Therefore, it requires that the hypervisor and guest agree on an interface that takes into consideration the features of the specific hypervisor-guest combination. This naturally means that a generic guest OS device driver cannot be used, and a specially modified device driver is needed to be run in the guest. Calls from these special device drivers are directly handled by the hypervisor through its hypercall interface instead of the usual route of a driver call causing a *vmexit*. Because of the need to use customized device drivers for each environment, the difficulty of providing security guarantees to them (e.g., certification), and the fact that hardware extensions have substantially mitigated performance penalties in full virtualization, para-virtualization has limited deployments. This leaves the emulation approach to handling devices using full virtualization as the most commonly deployed technique in many production environments.

The code for device emulation resides either in the hypervisor kernel or in a dedicated VM. Any I/O call from a guest VM application is intercepted by the hypervisor kernel and forwarded to this code since guest VMs cannot typically access the physical devices directly unless they are assigned to it. This code emulates devices, mediates access to them, and multiplexes the actual devices since each permitted VM has full access to the underlying physical device.

The main threats with respect to devices mediation are: (a) Unauthorized access to memory regions by Direct Memory Access (DMA) capable devices due to faulty device driver code, (b) Unauthorized access to devices by VMs, and (c) denial-of-service due to monopolization of I/O bandwidth.

4.3 Potential Threats to the Execution of Instructions by Hypercall Interface (HY-BF3)

In hypervisors implementing para-virtualization, certain guest instructions (e.g., accessing devices by accessing memory areas assigned to memory-mapped devices) cause a trap directly into the hypervisor instead of through channels enabled by *vmexit* instruction. This mechanism is called a hypercall, and the portion of the hypervisor dealing with such instructions is called a hypercall interface. Lack of proper validation of those instructions (e.g., not checking the scope for an instruction that requests a full dump of a VM's Virtual Machine Control Block, or not checking input values) would cause the entire virtualized server to crash. This is a hypervisor design vulnerability that must be addressed through proper validation and testing of the relevant hypervisor code rather than through any assurance measures in deployment.

4.4 Potential Threats originating from VM Lifecycle Management (HY-BF4)

In most instances, the lifecycle management operations on VMs are performed using commands submitted through a GUI or a scripting environment, both of which are supported by a management daemon at the back-end. This is a standard architectural paradigm for any management software. Vulnerabilities and potential threats are not virtualized server environment-specific and are therefore outside of the scope of this manuscript. Instead, the threat analysis in this context is to identify some VM lifecycle management operations that might be sources of potential threats for other baseline functions. This analysis reveals the following:

- Retrieving and deploying VM images that do not conform to the enterprise security profile in the image library, including those with outdated guest OS versions and patches, could result in a potential breach of process isolation described in Section 4.1. Similar potential threats exist if VMs are instantiated from snapshots taken at a considerable time in the past.
- Migrating VMs from one virtualized server to another (a process called VM Migration) involves transferring a running VM's memory content and processor state. The execution of this operation without necessary safeguards such as encryption of migration traffic etc., could result in the operation of a compromised VM in the destination platform, thereby affecting all three aspects of security—confidentiality, integrity and availability.

4.5 Potential Threats to Hypervisor Host Administration (HY-BF5)

The tasks under this function relate to the overall administration of a hypervisor host and software, and they are usually performed through user-friendly web interfaces or network-facing virtual consoles. Threats to the secure execution of these tasks are common in any remote administration and are therefore not addressed in this manuscript. However, the core requirement in a data center with virtualized servers is to have a uniform configuration for entire groups of hypervisors based on different criteria (e.g., the sensitivity of applications, line of business, clients in cloud service environments, etc.). Another requirement is to provide a safe network path for management traffic (packets containing administrative commands), considering that a portion of this network is a software-defined virtual network.

5. THREATS TO THE SECURE EXECUTION OF VM-RESIDENT PROGRAMS

The Guest OS and applications are the VM-resident programs that must execute securely in the presence of a higher privileged hypervisor software executing on the same hardware platform. The hypervisor is responsible for process isolation between VMs and the safe execution of each individual VM. However, a malicious or compromised hypervisor can be a source of threat to VMs for several reasons. First, the data structure that carries the execution state of VMs, called the Virtual Machine Control Block (VMCB), is created and handled by the hypervisor. Second, the hypervisor controls the nested page tables, which are really a pair of tables—one mapping from guest virtual addresses to guest physical addresses and the other mapping from guest physical addresses to host physical addresses. Thus, we see that a hypervisor can read and write the entire guest memory. By monitoring the execution state of a VM, it can also subject it to memory replay attacks [4].

The predominant use case for virtualized server platform is in the Infrastructure as a Service (IAAS) cloud service. In this service, the cloud service provider (CSP) provides the hypervisor while the guest VMs host and run the cloud service customers' (CSC) programs. A malicious hypervisor thus has the potential to affect the integrity and confidentiality of CSC's resources such as data and applications. Since a single cloud data center often hosts multiple guest VMs from different CSCs, data belonging to several VM owners may be breached by a single hypervisor. Therefore, the hypervisor should be treated as untrusted software, and VMs in a cloud data center need to be protected from the hypervisor.

The threats to the secure functioning of guest OS and VM-resident applications are by and large not unique to virtualized server platforms except for the fact that the VM executes as a lower privileged software, and its execution flow is controlled by the higher privileged hypervisor software.

6. PROTECTION FOR VIRTUAL NETWORK CONFIGURATIONS

To link the VMs inside a hypervisor host to each other and to the outside (physical) enterprise network, the hypervisor can define an entirely software-defined network called a virtual network. The components of this virtual network are: (a) one or more software-defined network interface cards, called virtual network interface cards (vNICs), inside each VM and (b) multiple software-defined switches, called virtual switches, operating inside the kernel of the hypervisor. The virtual switches have multiple ports, just like physical switches. One set of ports is used for connecting to the vNICs in VMs. The other set of ports, called uplink ports, are used for connecting the virtual switches to the physical network interface cards (pNICs) of the hypervisor host. Thus, a communication pathway is established for connecting VMs resident inside the same hypervisor host as well as to those resident in other hypervisor hosts. This then enables applications and guest OS instances running inside VMs to interact with computing, network, and storage elements on the data center's physical network. The network traffic flowing inside a virtual network can broadly be classified as [5]

- Management traffic: commands for hypervisor administration and VM lifecycle operations
- Infrastructure traffic: network packets generated during VM migration
- Inter-VM traffic: communication between applications or application tiers running in VMs

Thus, the entire network infrastructure in a virtualized server environment consists of a virtual network inside each hypervisor host and the physical datacenter network linking the various hosts. The threats to this network infrastructure are no different than those encountered in environments that consist of only physical (non-virtualized) hosts. However, defining the virtual network inside each VM entirely by software requires a different set of configurations (virtualized server-specific) and solutions (virtual firewalls) for ensuring secure communication.

There are four common virtual network configuration areas that have a bearing on the security of the network infrastructure in a virtualized server environment [5].

- Network segmentation
- Network path redundancy
- Firewall deployment and configuration
- VM traffic monitoring

A brief overview of the components and techniques involved in the above four configuration areas is necessary to arrive at security assurances associated with their deployment.

Network segmentation: This is a fundamental network configuration in any medium to large data center used for supporting enterprise IT resources or used for offering cloud computing services. This is due to the need for logical separation of applications/VMs with different sensitivity levels or belonging to different organizational entities (departments) or clients (as in cloud service environments). The two techniques commonly found in virtualized server environments are Virtual Local Area Network (VLAN) and Overlay-based virtual networking [5].

VLAN is a network segmentation technique that creates broadcast domains within a large data center network. In a data center with all physical (non-virtualized) hosts, a VLAN is defined by assigning a unique ID called a VLAN tag to one or more ports of a physical switch. All hosts connected to those ports then become members of that VLAN ID, creating a logical grouping of servers (hosts), regardless of their physical locations, in the large flat network of a data center. The concept of VLANs can be extended and implemented in a data center with virtualized hosts using virtual switches with ports or port groups that support VLAN tagging and processing. In other words, VLAN IDs are assigned to ports of a virtual switch inside a hypervisor kernel, and VMs are assigned to appropriate ports based on their VLAN membership. These VLAN-capable virtual switches can perform VLAN tagging of all packets going out of a VM (with the tag depending upon which port it has received the packet from) and can route an incoming packet with a specific VLAN tag to the appropriate VM by sending it through a port with a VLAN ID assignment equal to the VLAN tag of the packet and with a matching media access control (MAC) address. An example of a VLAN configuration inside a virtualized server is shown in Figure 2.

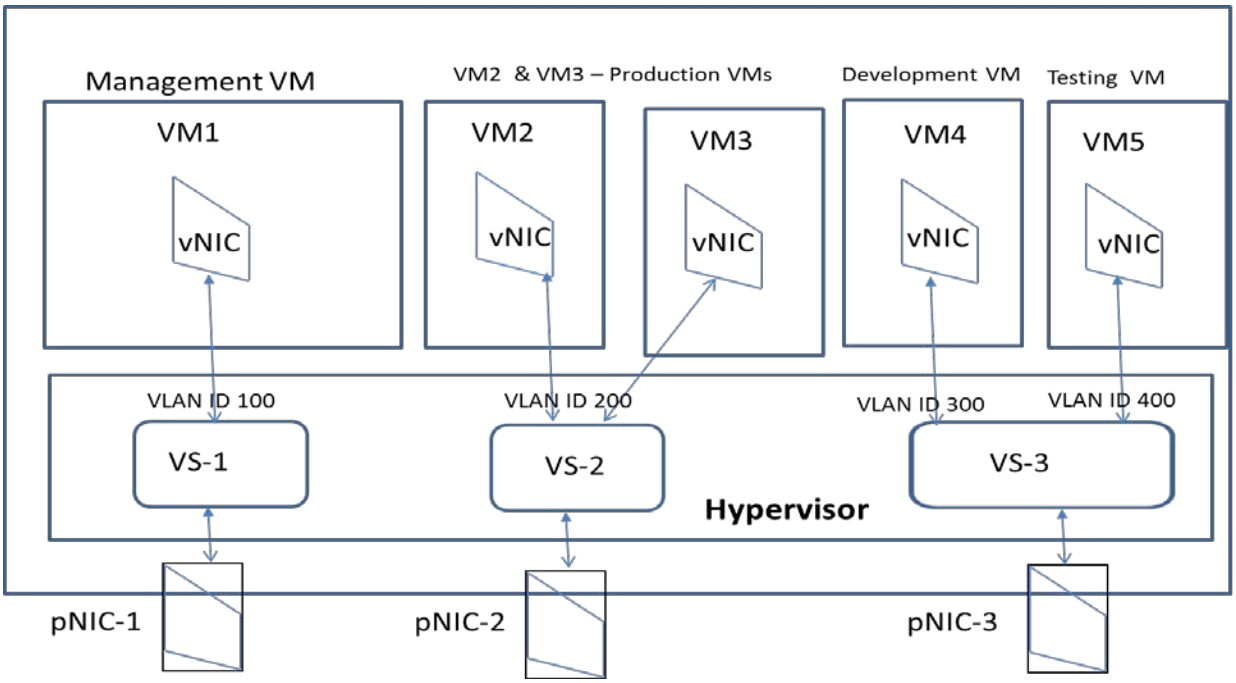

Fig 2. Virtual Local Area Network (VLAN) Configuration in a Virtualized Server

This logical segmentation of traffic inside the virtualized host is then extended to the physical network of the data center by configuring link aggregation (to carry traffic of multiple VLANs) on links between the pNICs of these virtualized hosts and the physical switches in the data center and configuring the receiving ports on the physical switch as trunking ports (capable of receiving and sending traffic belonging to multiple VLANs). A given VLAN ID can be assigned to ports of virtual switches located in multiple virtualized hosts. Thus, the combined VLAN configuration, consisting of the configuration inside the virtualized host (assigning VLAN IDs to ports of virtual switches or vNICs of VMs) and the configuration outside the virtualized host (link aggregation and port trunking in physical switches), provides a pathway for VLANs defined in the physical network to be carried into a virtualized host (and vice versa). This provides the ability to isolate traffic among VMs distributed throughout the data center using logical segments, and thus a means of providing confidentiality and integrity protection to the applications running inside those VMs.

In Overlay-based networking, isolation is realized by encapsulating an Ethernet frame received from a VM by a hypervisor kernel module called the Overlay module. In an example of the encapsulation scheme (or overlay scheme) called VXLAN, the Ethernet frame received from a VM, which contains the MAC address of the destination VM, is encapsulated in two stages: first, with the 24-bit VXLAN ID (virtual Layer 2 (L2) segment) to which the sending/receiving VM belongs, and second, with the source and destination IP addresses of the VXLAN tunnel endpoints (VTEP), which are kernel modules residing in the hypervisors of the sending and receiving VMs, respectively. VXLAN encapsulation thus enables the creation of a virtual Layer 2 segment that can span not only different virtualized hosts but also IP subnets within the data center. A Schematic diagram of VXLAN components is shown in Figure 3.

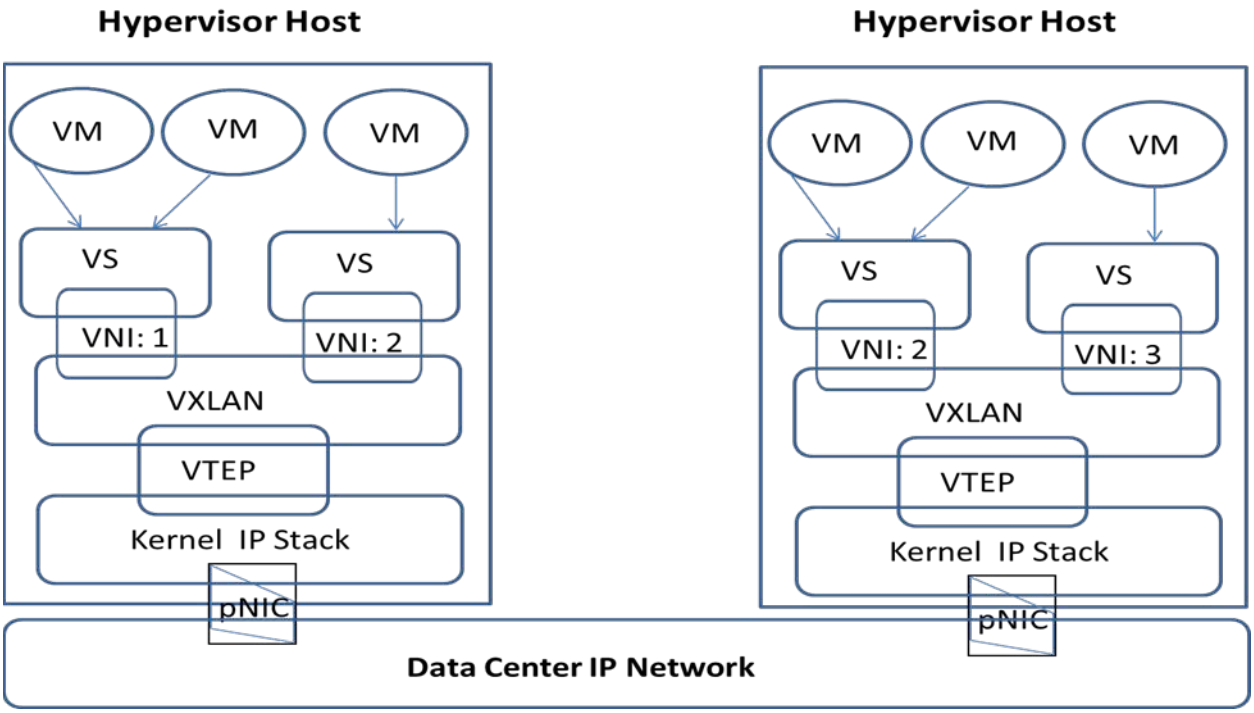

Figure 3: Virtual Network Segmentation using Overlays (VXLAN)

A particular tenant can be assigned two or more VXLAN segments (or IDs). VXLAN-based network segmentation can be configured to provide isolation among resources of multiple tenants of a cloud data center. The tenant can make use of multiple VXLAN segments by assigning VMs hosting each tier (web, application, or database) to the same or different VXLAN segments. If VMs belonging to a client are in different VXLAN segments, selective connectivity can be established among those VXLAN segments belonging to the same tenant through suitable firewall configurations, while communication between VXLAN segments belonging to different tenants can be prohibited.

Network path redundancy: Hypervisors offer a configuration feature called network interface card (NIC) teaming, which allows administrators to combine multiple pNICs into a NIC team for NIC failover capabilities in a virtualized host. The members of the NIC team are connected to the different uplink ports of the same virtual switch. Failover capability requires at least two pNICs in the NIC team. One of them can be configured as “active” and the other as “standby.” If an active pNIC fails or traffic fails to flow through it, the traffic will start flowing (or be routed) through the standby pNIC, thus maintaining continuity of network traffic flow from all VMs connected to that virtual switch. This type of configuration is also called *active-passive NIC bonding*.

Firewall Deployment and Configuration: Software-defined firewalls, called virtual firewalls, are generally the ones that are deployed on virtualized server platforms. There are two kinds of virtual firewalls—subnet-level virtual firewalls and kernel-level virtual firewalls. Subnet-level firewalls run in a dedicated VM, which is usually configured with multiple vNICs. Sometimes they come packaged as a virtual security appliance. Each vNIC in a subnet-level firewall is connected to a different subnet or security zone of the virtual network. Kernel-level firewalls, as the name denotes, are run as loadable (hypervisor)

kernel modules and use the hypervisor's introspection application programming interface (API) to intercept every packet coming into and out of an individual VM.

VM Monitoring: Firewalls only ensure that inter-VM traffic conforms to organizational information flow and security rules. However, to identify any malicious or harmful traffic coming into or flowing out of VMs and to generate alerts or take preventive action, it is necessary to set up traffic monitoring capabilities to monitor all incoming/outgoing traffic of a VM. This requires functionality to send copies of those packets to a network analyzer application. The purpose of a network analyzer application is to perform security analysis, network diagnostics, and network performance metrics generation. One of the techniques by which the above referred operation can be implemented is called port mirroring where the packets (or copies of the packets) flowing into and out of the port of a virtual switch (to which the monitored VM is connected and is called the source port) is forwarded to another port (called the destination port) which may be another virtual port or an uplink port. The entity holding the network analyzer application is connected to the destination port.

7. SECURITY ASSURANCE FOR HYPERVISOR BASELINE FUNCTIONS

7.1 Security Assurance for VM Process Isolation (HY-BF1)

To ensure the isolation of processes running in VMs, the following requirements must be met [1]:

(a) The privileged commands or instructions from a Guest OS to the host processor must be mediated such that the core function of the VMM/hypervisor as the controller of virtualized resources is maintained.
(b) The integrity of the memory management function of the hypervisor host must be protected against attacks such as buffer overflows and illegal code execution, especially in the presence of translation tables (e.g., host page table) that are needed for managing memory access by multiple VMs.
(c) Memory allocation algorithms must ensure that payloads in all VMs are able to perform their functions.
(d) CPU/GPU allocation algorithms must ensure that payloads in all VMs are able to perform their functions.

The requirements (a) and (b) are to be met by the hypervisor code by proper implementation of the data structures, such as Virtual Machine Control Block (VMCB) and second level page tables, that translate guest physical address to host physical address. In addition, hardware extension features, such as Instruction Set Virtualization and Memory Virtualization (described in Section 3), provide isolated execution environments for guests and hypervisor instructions as well as secure memory management through hardware page tables and should be leveraged by the hypervisor. The requirements (c) and (d) are meant to ensure the availability of application services running in VMs. The enablers are some features in memory allocation and CPU allocation algorithms and the assurance requirements they should meet are given below:

(1) The hypervisor should have configuration options to specify a guaranteed physical RAM for every VM that requires it as well as a limit to this value and a priority value for obtaining the required RAM resource in situations of contention among multiple VMs. Further, the over-commit feature that enables the total configured memory for all VMs to exceed the host physical RAM should be disabled by default.

(2) The hypervisor should provide features to specify a lower and upper bound for CPU clock cycles needed for every deployed VM as well as a feature to specify a priority score for each VM to facilitate scheduling in situations of contention for CPU resources from multiple VMs.

7.2 Security Assurance for Devices Mediation (HY-BF2)

Among all three approaches for handling devices in virtualized servers (Passthrough, Emulation, and Para-virtualization), emulation presents the greatest advantage in that it enables running VMs using the drivers that are available for that guest OS, without installing any special driver or tool provided by the hypervisor vendor. The advantage of using native OS drivers is that their vulnerabilities have been well-analyzed, published, and remediated.

The first three assurance requirements for secure device access in virtualized servers [1] pertain to emulation while the last requirement pertains to the passthrough scenario:

(1) All device drivers installed as part of a hypervisor platform should be configured to run as lower-privileged level process (guest mode) rather than the privilege level of the hypervisor (host mode). If device drivers are run on the same privilege level as the hypervisor, they should be designed, developed and tested using formal verification to guarantee that the drivers cannot compromise the security of hypervisor execution. This recommendation applies to any code running at the same privilege level as the hypervisor in the kernel (e.g., VMM).

(2) It should be possible to set up an Access Control List (ACL) to restrict the access of each VM process to only the devices assigned to that VM. To enable this, the hypervisor configuration should support a feature to tag VMs and/or have a feature to specify a whitelist, or list of allowable devices, for each VM.

(3) It should be possible to set resource limits for network bandwidth and I/O bandwidth (e.g., disk read/write speeds) for each VM to prevent denial-of-service (DOS) attacks. Additionally, the proper use of resource limits localizes the impact of a DOS to the VM or the cluster for which the resource limit is defined.

(4) Passthrough scenarios generally involve DMA capable devices. A DMA capable device is one that has the capability to read and write directly to and from main memory, allowing the CPU to perform other tasks in parallel. The security assurance required against unauthorized access from DMA capable devices, is that they should only be installed on hardware platforms that have the Input-Output Memory Management Unit (IOMMU) feature that can be configured to confine access by such devices to only the assigned memory regions.

7.3 Security Assurance for VM Lifecycle Management Functions (HY-BF4)

In Section 4.4, two VM lifecycle management operations were identified as potential sources for threats to other baseline functions: VM image management and VM migration. In large virtualized infrastructures, the installed base, consisting of a large number of operational VMs, may span different jurisdictions (departments, lines of business, or clients in infrastructures used for cloud services). For performing lifecycle management operations on these VMs, fine-grained administrative permissions are required to provide security guarantees such as least privilege. The security assurances required for these

operations (VM image management, VM migration, and fine-grained administrative permissions) are described below.

7.3.1 VM Image Management

Since VM-based software (e.g., Guest OS, Middleware, and Applications) shares physical memory of the virtualized host with hypervisor software, it is no surprise that a VM is the biggest source of all attacks directed at the hypervisor. In operational virtualized environments, VMs are rarely created from scratch, but rather from VM Images. VM Images are templates used for creating running versions of VMs. An organization may have its own criteria for classifying the different VM Images it uses in its VM Library. Some commonly used criteria include processor load (VM used for compute-intensive applications); memory load (VM used for memory-intensive applications such as Database processing); and application sensitivity (VM running mission-critical applications utilizing mission-critical data). For each VM image type, the following practices must be followed to provide the necessary security assurance.

(1) Security profiles must be defined for VMs of all types, and VM Images that do not conform to the profile should not be stored in the VM Image server or library. Images in the VM Image library should be periodically scanned for outdated guest OS versions and patches, especially in situations where new OS version releases and/or patches are frequent.

(2) Every VM Image stored in the image library should have a digital signature attached to it as a mark of authenticity and integrity, signed using trustworthy, robust cryptographic keys.

(3) Permissions for checking into and checking out images from the VM Image library should be enforced through a robust access control mechanism and limited to an authorized set of administrators. In the absence of an access control mechanism, VM image files should be stored in encrypted devices that can only be opened or closed by a limited set of authorized administrators with passphrases of sufficient complexity.

(4) Access to the server storing VM images should always be through a secure protocol such as Transport Layer Security (TLS).

7.3.2 VM Live Migration

Live migration is a functionality present in all hypervisors that enables a VM to be migrated or moved from one virtualized host to another while the guest OS and applications on it are still running. This functionality provides key benefits such as fault tolerance, load balancing, host maintenance, upgrades, and patching. In live migration, the state of the guest OS on the source host must be replicated on the destination host. This requires migrating memory content, processor state, storage (unless the two hosts share a common storage), and network state.

The most common memory migration technique adopted in most hypervisors is called pre-copy. In this approach, in the first phase, memory pages belonging to the VM are transferred to the destination host while the VM continues to run on the source host [6]. In the second phase, memory pages modified during migration are sent again to the destination to ensure memory consistency. During the latter phase, the exact state of all the processor registers currently operating on the VM are also transferred, and the

migrating VM is suspended on the source host. Processor registers at the destination host are modified to replicate the state at the source host, and the newly migrated VM resumes its operation. Storage migration is provided by a feature that allows admins to move a VM's file system from one storage location to another without downtime. This storage migration can take place even in situations where there is no VM migration. For example, a VM may continue to run on the host server while the files that make up the VM are moved among storage arrays or Logical Unit Numbers (LUNs).

In the process described above, the memory and processor-state migration functions are inherent aspects of hypervisor design. The storage migration function is an integral part of storage management and is applicable to both virtualized and non-virtualized infrastructures. The network state is maintained after a VM migration because each VM carries its own unique MAC address, and the migration process places some restrictions on the migration target (e.g., the source and target host should be on the same VLAN). Hence, from a security protection point of view, the only aspects to consider are proper authentication and a secure network path for the migration process [1].

During VM live migration, a secure authentication protocol must be employed; the credentials of the administrator performing the migration are passed only to the destination host; the migration of memory content and processor state takes place over a secure network connection; and a dedicated virtual network segment is used in both source and destination hosts for carrying this traffic.

7.3.3 Fine-grained Administrative Privileges for VM Management

The ability to assign fine-grained administrative permissions for the virtualized infrastructure enables the establishment of different administrative models and associated delegations [1].

The access control solution for VM administration should have a granular capability, both at the permission assignment level and the object level (i.e., the specification of the target of the permission can be a single VM or any logical grouping of VMs based on function or location). In addition, the ability to deny permission to some specific objects within a VM group (e.g., VMs running workloads of a designated sensitivity level) despite having access permission to the VM group should exist.

7.4 Security Assurance for Hypervisor Administration Functions (HY-BF5)

Secure operation of administrative functions is critical for any server class software, and hypervisor is no exception to this. The outcome is a secure configuration that can provide the necessary protections against security violations. In the case of a hypervisor, impact of insecure configuration can be more severe than in many server software instances since the compromise of a hypervisor can result in the compromise of many VMs operating on top of it. While the composition of the configuration parameters depends upon the design features of a hypervisor offering, the latitude in choosing the values for each individual parameter results in different configuration options. Many configuration options relate functional features and performance. However, there are some options that have a direct impact on the secure execution of the hypervisor, and it is those configuration options that are discussed in this manuscript.

The following are some security practices that are generic for any server class software. Although applicable to the hypervisor, these are not addressed in this manuscript:

(a) Control of administrative accounts on the hypervisor host itself and least privilege assignment for different administrators
(b) Patch management for hypervisor software and host OS
(c) Communicating with the hypervisor through a secure protocol such as TLS or Secure Shell (SSH)

7.4.1 Centralized Administration

The administration of a hypervisor and hypervisor host can be performed in two ways:

- Having administrative accounts set up in each hypervisor host
- Centralized administration of all hypervisors and hypervisor hosts through enterprise virtualization management software (EVMS).

Centralized management of all hypervisor platforms in the enterprise through enterprise virtualization management software (EVMS) is preferable since security profiles for various hypervisor groups in the enterprise can be defined and easily enforced through EVMS. For any IT data center to operate efficiently, it is necessary to implement load balancing and fault tolerance measures, which can be realized by defining hypervisor clusters. Creation, assignment of application workloads, and management of clusters can be performed only with a centralized management software, making the deployment and usage of an enterprise virtualization management a critical necessity. Hence a security assurance framework for hypervisor administration is as follows:

The administration of all hypervisor installations in the enterprise should be performed centrally using an EVMS. Enterprise gold-standard hypervisor configurations for different types of workloads and clusters must be managed and enforced through EVMS. The gold-standard configurations should, at minimum, cover CPU, Memory, Storage, Network bandwidth, and Host OS hardening, if required.

7.4.2 Securing the Management Network

Management of the hypervisor and its host is performed through administrative commands sent through a management console or command line interface (CLI). This capability can be provided by a dedicated management VM or by a hypervisor kernel module. Part of the network communication path that carries this management traffic is the software-defined virtual network inside the hypervisor host and it is necessary to ensure that a dedicated path is allocated for this. A commonly adopted approach is to allocate a dedicated physical network interface card (pNIC) for handling management traffic, and, if that is not feasible, a virtual network segment (e.g., VLAN) must be assigned exclusively for it.

Protection for hypervisor host and software administration functions should be ensured by allocating a dedicated physical NIC or, if that is not feasible, placing the management interface of the hypervisor in a dedicated virtual network segment (e.g., VLAN) and enforcing traffic controls using a firewall (e.g., designating the subnets in the enterprise network from which incoming traffic into the management interface is allowed).

8. SECURITY ASSURANCE FOR EXECUTION OF VM-RESIDENT PROGRAMS

Providing protected execution for the lower-privileged software is an evolving hardware function and there are not enough threat data available for these functions. However, assurance requirements for this

function can still be identified based on the execution model for VMs and hypervisor instructions in the virtualized server platform.

There are two processor features available to reduce the impact of a malicious, higher privileged software such as the hypervisor on the confidentiality and integrity of lower privileged software. They are:

- A secure region of memory called enclave can be created where the resource-owner marked security sensitive code in VMs can be altered to run. Code running in the enclave cannot be tampered with by the hypervisor or the host OS (in type 2 hypervisor). This feature is implemented in Intel's Software Guard Extension (SGX) [7].
- Encrypt the entire VM's memory so that the hypervisor cannot inspect its data. This is the approach adopted in AMD's Secure Encrypted Virtualization (SEV) [8].

It is not sufficient just to protect a portion or whole of VM's memory while it is executing. The data structures that provide the execution state of VM and the general-purpose registers of the host CPU that contain the values that enable page table walkthroughs to get at the VM's host memory address must also be protected. Hence the assurance requirements for secure VM execution can be stated as follows:

(1) There should be hardware features to protect designated memory areas where VM application code runs. This will protect those applications from malicious or compromised hypervisors.

(2) The Virtual Machine Control Block (VMCB) that contains data about the execution state of VMs and the general-purpose registers used by VMs (that contain entry memory addresses) must also be cryptographically protected to ensure secure VM execution even in the presence of a malicious or compromised hypervisor.

9. SECURITY ASSURANCE FOR VIRTUAL NETWORK CONFIGURATIONS

9.1 Assurance for Network Segmentation

Both techniques discussed for network segmentation – VLAN and Overlay-based networking can span multiple IP subnets and hence can be deployed datacenter wide. However, since a VLAN ID is 12 bits long, the maximum number of segments that can be defined is 4096 (strictly 4094). On the other hand, VXLAN uses a 24-bit segment ID known as the VXLAN network identifier (VNID), which enables up to 16 million VXLAN segments and hence the security assurance recommendation is stated as follows [5]:

Large data center networks with hundreds of virtualized hosts and thousands of VMs and requiring many segments should deploy overlay-based virtual networking because of scalability (Large Namespace) and virtual/physical network independence. However, it is highly advisable that the overall traffic generated by overlay-based network segmentation (i.e., VXLAN network traffic in our context) is isolated on the physical network using a technique such as VLAN to maintain segmentation guarantees. In addition, overlay-based virtual networking deployments should always include either centralized or federated SDN controllers using standard protocols for configuration of overlay modules in various hypervisor platforms.

9.2 Assurance for Network Path Redundancy Configuration

The following operational parameters will provide the necessary assurance that the NIC teaming configuration intended for enhancing the availability of VM-based applications by providing alternate communication pathways will achieve their intended purpose.

Each pNIC member of a NIC team should be driven by different drivers and placed on a separate PCI bus (if available). Further, the network path redundancy inside a virtualized host can be extended to the physical network by connecting each pNIC member of the NIC team to different physical switches.

9.3 Assurance for Firewall Configuration

In the firewall configuration for virtualized servers, the security assurance is dictated by the choice of the appropriate type of virtual firewall (subnet-level or kernel-based), expressiveness of the firewall rules and wherever applicable uniformity in rules for similar traffic flows. The following are the security assurance requirements [5]:

(1) In virtualized environments with VMs running I/O intensive applications, kernel-based virtual firewalls should be deployed instead of subnet-level virtual firewalls, since kernel-based virtual firewalls can potentially perform packet processing in the kernel of the hypervisor at native hardware speeds.

(2) For both subnet-level and kernel-based virtual firewalls, it is preferable that the firewall allows for integration with a virtualization management platform rather than being accessible only through a standalone console. The former will enable easier provisioning of uniform firewall rules to multiple firewall instances, thus reducing the chances of configuration errors.

(3) For both subnet-level and kernel-based virtual firewalls, it is preferable that the firewall supports rules using higher-level components or abstractions (e.g., security group) in addition to the basic 5-tuple (source/destination IP address, source/destination ports, protocol).

9.4 Assurance for VM Traffic Monitoring

The port mirroring technique involves increase in network traffic in the virtualized network inside the hypervisor traffic and must be implemented with care. Minimal assurance for implementing this can be stated as follows:

A port mirroring feature should provide choices in specifying destination ports (either the virtual port or uplink port) so that it creates the flexibility to locate the network analyzer application in another VM on the same or different hypervisor or in any non-virtualized server in the data center.

10. SECURITY ASSURANCE FOR BOOTING A VIRTUALIZED SERVER PLATFORM

Configuration changes, module version changes, and patches affect the content of the hypervisor platform components such as BIOS, hypervisor kernel, and back-end device drivers running in the kernel. To ensure that each of these components that are part of the hypervisor stack can be trusted, it is necessary to check their integrity through a hardware-rooted attestation scheme that provides assurance of boot integrity. Checking integrity is done by cryptographically authenticating the hypervisor components that are launched. This authentication verifies that only authorized code runs on the system. Specifically, in the context of the hypervisor, the assurance of integrity protects against tampering and low-level targeted

attacks such as root kits. If the assertion of integrity is deferred to a trusted third party that fulfills the role of trusted authority, the verification process is known as *trusted attestation*. Trusted attestation provides assurance that the code of the hypervisor components has not been tampered with. In this approach, trust in the hypervisor's components is established based on trusted hardware. In other words, a chain of trust from hardware to hypervisor is established with the initial component (i.e., hardware) called *the root of trust*. This service can be provided by a hardware/firmware infrastructure of the hypervisor host that supports boot integrity measurement and the attestation process. Collectively, this is called a measured launch environment (MLE) in the hypervisor host.

Some hardware platforms provide support for MLE with firmware routines for measuring the identity (usually the hash of the binary code) of the components in a boot sequence. An example of a hardware-based cryptographic storage module that implements the measured boot process is the standards-based Trusted Platform Module (TPM), which has been standardized by the Trusted Computing Group (TCG) [9]. The three main components of a TPM are: (a) Root of Trust for Measurement (RTM) – makes integrity measurements (generally a cryptographic hash) and converts them into assertions, (b) Root of Trust for Integrity (RTI) - provides protected storage, integrity protection, and a protected interface to store and manage assertions, and (c) Root of Trust for Reporting (RTR) - provides a protected environment and interface to manage identities and sign assertions. The RTM measures the next piece of code following the boot sequence. The measurements are stored in special registers called Platform Configuration Registers (PCRs).

The measured boot process is briefly explained here using TPM as an example. The measured boot process starts with the execution of a trusted immutable piece of code in the BIOS, which also measures the next piece of code to be executed. The result of this measurement is extended into the PCR of the TPM before the control is transferred to the next program in the sequence. Since each component in the sequence in turn measures the next before handing off control, a chain of trust is established. If the measurement chain continues through the entire boot sequence, the resultant PCR values reflect the measurement of all components.

The attestation process starts with the requester invoking, via an agent on the host, the TPM Quote command. It specifies an Attestation Identity Key (AIK) to perform the digital signature on the contents of the set of PCRs that contain the measurements of all components in the boot sequence to quote and a cryptographic nonce to ensure freshness of the digital signature. After receiving the signed quotes, the requester validates the signature and determines the trust of the launched components by comparing the measurements in the TPM quote with known good measurements.

The MLE can be incorporated in the hypervisor host as follows:

- The hardware hosting the hypervisor is established as a root-of-trust, and a trust chain is established from the hardware through the BIOS and to all hypervisor components.
- For the hardware consisting of the processor and chipset to be established as the root-of-trust and to build a chain of trust, it should have a hardware-based module that supports an MLE. The outcome of launching a hypervisor in MLE-supporting hardware is a measured launch of the firmware, BIOS, and either all or a key subset of hypervisor (kernel) modules, thus forming a trusted chain from the hardware to the hypervisor.

- The hypervisor offering must be able to utilize the MLE feature. In other words, the hypervisor should be able to invoke the secure launch process, which is usually done by integrating a pre-kernel module into the hypervisor's code base since the kernel is the first module installed in a hypervisor boot up. The purpose of this pre-kernel module is to ensure the selection of the right authenticated module in the hardware that performs an orderly evaluation or measurement of the launch components of the hypervisor or any software launched on that hardware. The Tboot is an example of a mechanism that enables the hypervisor to take advantage of the MLE feature of the hardware.
- All hypervisor components that are intended to be part of the Trusted Computing Base (TCB) must be included within the scope of the MLE-enabling mechanism so that they are measured as part of their launch process.

The MLE feature with storage and reporting mechanisms on the hardware of the virtualized host can be leveraged to provide boot integrity assurance for hypervisor components by measuring the identity of all entities in the boot sequence, starting with firmware, BIOS, hypervisor and hypervisor modules; comparing them to "known good values;" and reporting any discrepancies. If the measured boot process is to be extended to cover VMs and its contents (guest OS and applications), a software-based extension to the hardware-based MLE implementation within the hypervisor kernel is required. The security assurance for ensuring a secure boot process for all components of a hypervisor platform can now be stated as follows [1]:

The hypervisor that is launched should be part of a platform and an overall infrastructure that contains: (a) hardware that supports an MLE with standards-based cryptographic measurement capabilities and storage devices and (b) an attestation process with the capability to provide a chain of trust starting from the hardware to all hypervisor components. Moreover, the measured elements should include, at minimum, the core kernel, kernel support modules, device drivers, and the hypervisor's native management applications for VM Lifecycle Management and Management of Hypervisor. The chain of trust should provide assurance that all measured components have not been tampered with and that their versions are correct (i.e., overall boot integrity). If the chain of trust is to be extended to guest VMs, the hypervisor should provide a virtual interface to the hardware-based MLE.

11. SUMMARY AND CONCLUSIONS

Server or Hardware virtualization is an established technology in data centers used for supporting enterprise IT resources as well as cloud services. The core entity in this technology is a set of software modules called the hypervisor. The hypervisor provides abstraction of the hardware resources, such as CPU, memory, and devices (the first two with some assistance with hardware extensions), and enables the running of multiple computing stacks called VMs, each with its own OS and applications, to be run on a single physical host. Such a physical host is called a hypervisor host or virtualized server. The network linking the multiple VMs within a hypervisor and with VMs located in other hypervisor hosts is a combination of a software-defined network (called virtual network) and the physical network infrastructure and constitute the virtualized server environment.

Since hypervisors come in several architectural flavors (Type 1 vs Type 2, Full vs Para-virtualized), this manuscript identified five baseline functions for the hypervisor. Analyzing these baseline functions,

together with functions of other components of the virtualized server environment (i.e., the hardware, the VMs, the Virtual Network), enabled identification of threats to these functions as well as threats originating from these functions. The threats were then used as the basis for developing appropriate security assurance measures for countering each threat.

12. REFERENCES

[1] Chandramouli R., *Security Recommendations for Hypervisor Deployment on Servers*, NIST Special Publication SP 800-125A, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-125A.pdf

[2] *Heap-based buffer overflow in the IDE subsystem in QEMU*, https://www.cvedetails.com/cve/CVE-2015-5154, January 2018

[3] *Allowing guest OS users to execute arbitrary code on the host OS*, https://www.cvedetails.com/cve/CVE-2015-3214, January 2018

[4] Hetzelt F, Buhren R, *Security Analysis of Encrypted Virtual Machines*, Proceedings of the13th ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE'17), April 2017, Xi'an, China

[5] Chandramouli R., *Secure Virtual Network Configuration for Virtual Machine (VM) Protection*, NIST Special Publication SP 800-125B, http://nvlpubs.nist.gov/nistpubs/SpecialPublications/NIST.SP.800-125B.pdf

[6] S.Shirinbab S., l. Lundberg I., and Illie D., *Performance Comparison of KVM, VMware and Xenserver using a Large Telecommunication Application*, Proceedings of the Fifth International Conference on Cloud Computing, GRIDs, and Virtualization (CLOUD COMPUTING), 2014.

[7] *Intel Software Guard Extensions (Intel SGX)*, https://software.intel.com/en-us/sgx, January 2018

[8] Kaplan D, Powell J, and Woller T., *White Paper AMD Memory Encryption*. http://amd-dev.wpengine.netdna-cdn.dom/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, January 2018.

[9] Trusted Platform Module (TPM) Main Specification: http://www.trustedcomputinggroup.org/resources/tpm_main_specification