I/O Virtualization Architecture for Security

J. Lakshmi, S. K. Nandy, Indian Institute of Science, Bangalore, India

{jlakshmi, nandy}@serc.iisc.ernet.in

Abstract - Prevalent and popular virtualization technologies have concentrated on consolidating servers based on the CPU component of the workload. Other system resources, particularly I/O devices like network interfaces and disks have been always designed to be in control of the OS that is managing system resources. Sharing of these devices has been through the OS abstraction layers, with device always being accessed and managed by the privileged OS kernel. Extending such systems for virtualization has resulted in sharing the device access path along-with the shared device, which makes this software layer the vulnerable component for the whole system. In this paper we argue that virtualization gives a simple and efficient mechanism for isolation, and hence improved security, if architected correctly. We analyze the existing I/O virtualization architectures with a view towards identifying the security issues and propose an enhancement that addresses these issues.

Keywords: I/O virtualization, security threats, unconstrained DMA, denial of service attack.

1. Introduction

System virtualization enables the hosting of multiple, independent virtual machines (VMs) on a single physical machine [1]. Each VM can be used to host an application complete with its environment. This not only allows for more efficient and improved utilization of system resources but also offers complete software isolation of independent applications. The software isolation provides necessary security by encapsulating the application from threats arising from software bugs or loopholes external to it.

Co-hosting multiple VMs on a single physical machine is normally achieved by sharing the same hardware across many VMs. Unconstrained and unchecked resource usage in such systems can lead to denial of service type of attacks [2]. It is envisaged that resource usage monitoring and regulation will be the norm rather than an option in virtualized servers. However, the effectiveness of the resource controls will decide if denial of service threat can be deterred. Resource sharing mechanisms differ depending on the nature of the resource. Traditional OSes have used the process abstraction for resource allocation and management. Various process schedulers have proven to be robust and very effective in time-sharing the CPU resource. Based on the goals of resource sharing, round-robin and credit based schedulers have been quite successful in managing CPU sharing without causing deadlocks or overpowering the CPU for exclusive use by a single process. Extending the same model to virtualized servers by treating each independent VM as a process that is scheduled by the VMM has proven to be quite effective in terms of both performance and security.

I/O resources like memory, disks and NICs are treated rather differently as compared to the CPU resource. Current systems are designed to be managed by single OS. As a result of which these system resources are abstracted within the OS layers and projected as high-level resources like memory pages, disk files, sockets, packets, etc. The process is allocated these high-level resources and usage is mostly managed using these abstractions. Actual data transfer to and from the physical device is managed and handled by the OS. Almost all of the these resources are accessed using functions of the privileged OS kernel, which in the case of general purpose OS like the Linux, is a non-pre-emptive kernel. In such systems, security breaches can happen because of faulty or buggy device drivers that capture the CPU in an infinite loop. This issue is easily solved by restricting the VM's CPU timeshare. Also, for devices that have drivers supporting unconstrained DMA, the drivers themselves become the security threat [3]-[6]. Moving such device drivers from within the kernel space to user space has been proven to be the simplest method for solving the problem of unconstrained DMA. It is easy to support this idea on virtualized servers since all the VMs reside on a higher protection ring when compared to the virtual machine monitor (VMM) or the hypervisor. Hence, a device driver attempting unconstrained DMA will

naturally trap with a memory protection fault if it tries to read or write outside the VMs address space. Apart from these, the support for resource usage controls for these I/O devices lies high in the abstraction layers of the OS. Because of this the usage of the device tends to be unrestricted even though the application experiences constraints [3]. This can lead to denial of service attack on the device [7]. To prevent this, the device should provide resource usage controls. In order to support such a model, it is essential to have micro-architecture support from the I/O device, which the existing devices lack.

Rest of the paper is organised as follows: in section 2 we identify the security loopholes within the existing NIC virtualization architectures; in section 3 we propose I/O virtualization architecture to overcome the identified limitations; section 4 discusses the results obtained for the proposed architecture, in a simulation environement; section 5 details the related work and in section 6 we conclude the paper highlighting its contributions.

2. I/O Virtualization and Security Issues

Virtualization leads to sharing of limited I/O devices by multiple, independent VMs. To guarantee performance and security it is necessary to provide concurrent device access to the shared I/O devices. Due to lack of micro-architecture support for concurrent device access, many of the prevalent virtualization architectures extend the "single-OS single-hardware" model to support virtual devices. Popular, prevalent I/O virtualization technologies use either a hosted VM or the VMM itself to contain the necessary software abstraction to support the virtual devices and concurrent access. As a consequence all shared devices also cause device access path to be shared. This model of sharing has two major pitfalls:

1. Device access path is a software layer, which when shared can potentially cause all the sharing VMs to fail due to its failure.
2. Since virtual device is an abstraction supported in software, device usage controls tend to be coarse grained and hence can be ineffective. This could lead to an easy denial of service attack.

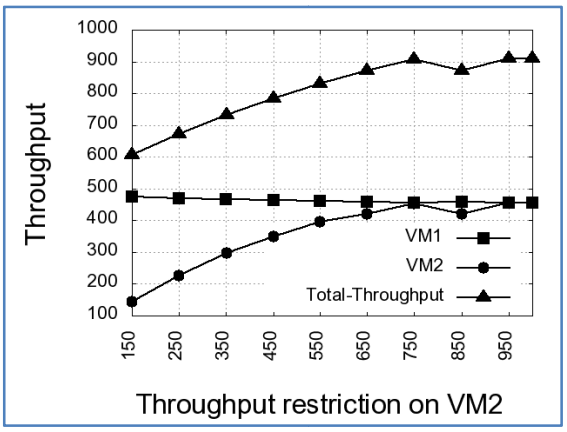

Figure 1: Device bandwidth throttling due to ineffective resource controls.

Figure 1 illustrates the effect of coarse grained resource controls, as implemented in *Linux netfilter* module, on application performance. The graphs depict the throughput performance of the standard *httperf* benchmark on a consolidated Xen server hosting two VMs. Each VM is identical in configuration and supports an independent *httperf* client. Both VMs are configured to share a single physical NIC supporting independent virtual NICs. For the experiment, VM1 is allowed to use all the available device bandwidth. And, VM2 is constrained on the maximum throughput it can support, which is plotted on the X-axis of the graph. Each VM is subjected to the maximum *httperf* load it can take on an isolated virtualized server. As can be observed, from the plots, we notice that the device is capable of supporting total-throughput of 900 replies/s but this does not seem to be translating into throughput improvement for VM1. The reason for this is because *netfilter* module provides resource usage controls high up in the network stack. As a result, the device receives packets that are later dropped based on the controls imposed. At the application level we see the realization of the control, as depicted by the graph for VM2, but the device bandwidth is wasted and not available for use, as depicted by the graph for VM1. Most consolidation scenarios use the device bandwidth as the guideline for server consolidation, but insufficient controls can potentially lead to service denial.

In this paper we propose that virtualization is a simple and effective isolation mechanism and can provide strong security to individual VMs if architected correctly. Here we take the example of NIC virtualization and detail the micro-architecture

modification to the NIC to support virtual-NICs (vNICs) in hardware. These virtual-NICs can be exported to individual VMs which can access the vNICs using their own resident, native device drivers. This has the following significant benefits:

1. Each vNIC can be carved out as a device partition, based on the device requirement specification of the VM. Appropriate micro-architecture and hardware constructs can ensure that the VM does not monopolize device usage and cause denial of service attack to other VMs sharing the device.
2. By allowing native device driver within the VM for the vNIC, not only is the performance enhanced but the device driver errors are easily trapped by the VMM and robust recovery mechanisms can be built into the VM.
3. This model eliminates sharing of the device access path by allowing direct access to the vNIC by the VM and thereby eliminates the associated failures.

Proposed architecture avoids denial of service by only allocating part of the physical resource to a VM. The architecture allows for unmodified GuestOS on a VM. Hence the security is verified and built outside the VM, and within the VMM. The VMM is architected to manage device access control. Actual data transfer is done by the VM. Also, native device driver access to the virtual device allows the device driver to function as a user-space driver. This model traps any un-authorized memory accesses of the device driver by the VMM. Moving the device driver outside the hosted VM or the VMM, eliminates the shared software layer. Hence, it protects all the VMs from faulty device drivers at any of the VMs sharing the device.

3. Architecture Proposal

The proposed architecture enhances I/O device virtualization to enable separation of device management from device access. This is done by building device protection mechanisms into the

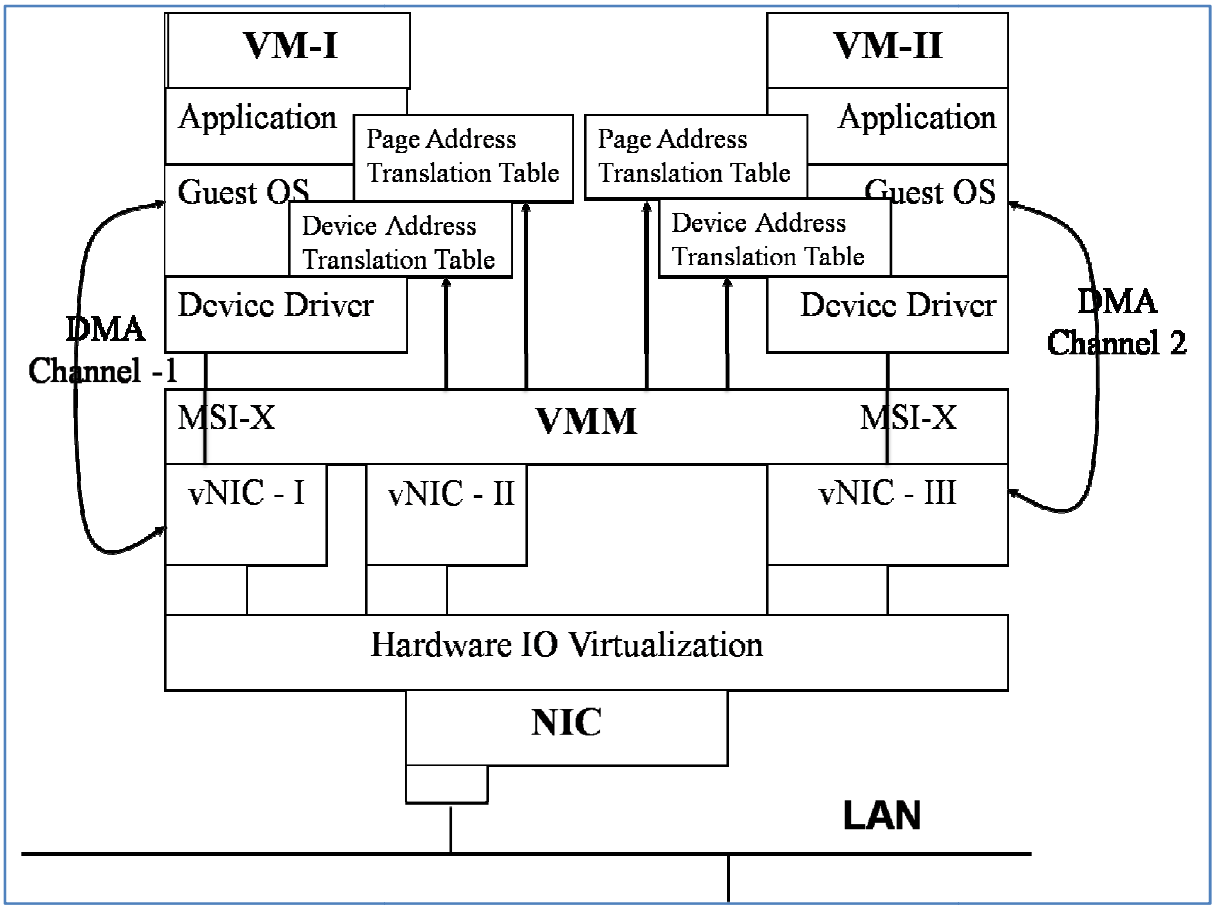

Figure 2: NIC architecture supporting secure, independent reconfigurable virtual-NICs.

physical device to be managed by the VMM. As an example, for the case of NIC, the VMM recognizes the destination VM of an incoming packet by the interrupt raised by the device and forwards it to the appropriate VM. The VM then processes the packet as it would do so in the case of non-virtualized environment. Thus, device access and scheduling of device communication are managed by the VM that is using it. The identity for access is managed by the VMM. This eliminates the intermediary VMM/hosted VM on the device access path and reduces I/O service time, which improves the application performance and consolidation on virtualized servers and reduces security threat due to failure of common shared software. In the following subsections we describe NIC I/O virtualization architecture that addresses the security issues caused due to shared device access path and unconstrained memory access to device driver. With these goals in mind we suggest how the system software layers of the VMM and the GuestOS inside the VM should use the proposed NIC hardware that is enabled for QoS based, secure concurrent device access. Figure 2 gives a block schematic of the proposed I/O virtualization architecture. The picture depicts a NIC card that can be housed within a multicore server. The card has a controller that manages the DMA transfer to and from the device memory. The standard device memory is now replaced by a partitionable memory supported with n sets of device registers. A set of m memory partitions, where $m \leq n$, along-with device registers form the virtual-NICs (vNICs). The device memory is reconfigurable, i.e. dynamically partitionable, and the VMs' QoS requirements drive the sizing of the memory partition of a vNIC. The advantage of having a dynamically partitionable device memory is that any unused memory can be easily extended into or reduced from a vNIC in order to support adaptive QoS specifications. The NIC identifies the destination VM of an arriving packet, based on the logical device address associated with it. A simple implementation is to allow a single physical NIC to support multiple MAC address associations. Each MAC address then represents a vNIC and a vNIC request is identified by generating a message signalled interrupt (MSI). The number of MAC addresses and interrupts supported by the controller restricts the number of vNICs that can be exported. Although, the finite number of physical resources on the NIC restricts the number of vNICs that can be exported, judicious use of native and para-virtualized access to the vNICs, based on the QoS guarantees a VM needs to honor, can overcome the limitation. A VM that has to support stringent QoS guarantees can choose to use native access to the vNIC whereas those VMs that are looking for best-effort NIC access can be allowed para-virtualized access to a vNIC. The VMM can aid in setting up the appropriate hosting connections based on the requested QoS requirements. The architecture can be realized with the following enhancements:

Virtual-NIC: In order to define vNIC, the physical device should support timesharing in hardware. For a NIC, this can be achieved by using MSI and dynamically partitionable device memory. These form the basic constructs to define a virtual device on a physical device as depicted in Figure 2. Each virtual device has a specific logical device address, like the MAC address in case of NICs, based on which the MSI is routed. Dedicated DMA channels, a specific set of device registers and a partition of the device memory are part of the virtual device interface which is exported to a VM when it is started. This virtual interface is called the vNIC which forms a restricted address space on the device for the VM to use and remains in possession of the VM until it is active or relinquishes the device. The VMM sets up the device page translation table, mapping the physical device address of the vNIC into the virtual memory of the importing VM, during the vNIC creation and initialization. The device page translation table is given read-only access to the VM and hence forms a significant security provisioning on the device. This prohibits any corrupt device driver of the VM GuestOs to affect other VMs sharing the device or the VMM itself. Also, for high-speed NIC devices, the partitionable memory of the vNIC is useful in setting up large receive and segment offload capabilities specific to each vNIC and thus customize the sizing of each vNIC based on the QoS requirements of the VM.

Accessing virtual-NIC: To access the vNIC, the native device driver is hosted inside the VM. This device driver manipulates the restricted device address space which is exported through the vNIC interface by the VMM. The VMM identifies and forwards the device's interrupt to the destination VM. The GuestOS of the VM handles the I/O access and thus directly accounts for the resource usage it incurs. With the vNIC interface, data

transfer is handled by the VM. The VM sets up the Rx/Tx descriptor rings within its address space and makes a request to the VMM for initializing the I/O page translation table during startup. The device driver uses this table along-with the devices address translation table and does DMA directly into the VMs address space. Any violation by the device driver to memory outside its address space is trapped by the VMM as a protection error. The error can be used to initiate preventive measures (like taking the VM offline) on the corrupt driver or the VM. Thus only the misbehaving VM is isolated without affecting any of the other VMs sharing the device.

QoS and virtual-NIC: The device memory partition acts as a dedicated device buffer for each of the VMs. With appropriate logic on the NIC card, QoS specific service level agreements (SLAs) can be easily implemented on the device that translate to bandwidth restrictions and VM based processing priority. The key is being able to identify the incoming packet to the corresponding VM. This is done by the NIC based on the packet's associated logical device address. The NIC controller decides on whether to accept or reject the incoming packet based on the bandwidth specification or the current free memory available with the destination vNIC of the packet. This gives a fine-grained control on the incoming traffic and helps reduce the interference effects. The outbound traffic can be controlled by the VM itself, as is done in the existing architectures. By provisioning the resource usage controls on to the device, and managing device timesharing in hardware, monopoly on the device by a single VM is eliminated. Also, on virtualized servers it is necessary to control the incoming traffic since unrestricted incoming traffic at the device can potentially cause denial of service attack on the entire group of device sharing VMs.

With these constructs, the virtualized NIC is now enabled for carving out secure, customized vNICs for each VM, based on its QoS requirements, and support native device access to the GuestOS of the VM.

4. Results

The proposed architecture involves changes in the complete end-to-end architecture of I/O device virtualization. The hardware changes proposed for the NIC are difficult to implement on a virtualized server unless a physical prototype is available and appropriate changes are effected in the virtualization stack. This is the aim of future work. But the effect of moving the resource controls onto the device and moving the device access into GuestOS (and eliminating shared device access path) can be easily simulated and verified for its effectiveness. In order to evaluate this aspect, we generated a layered queuing network model of the entire end-to-end architecture and present the results in Figure 3**.**

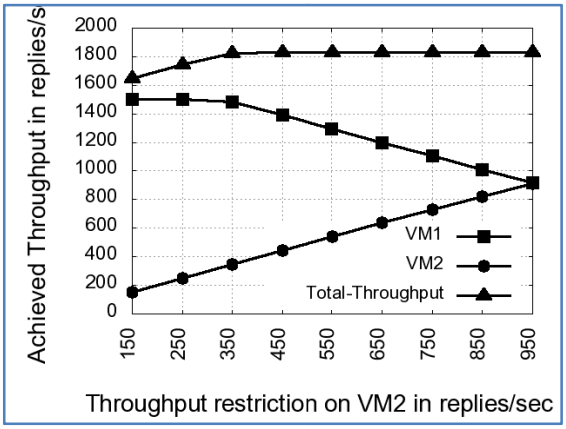

Figure 3: Improved availability of device bandwidth with device based resource controls.

As can be observed from the graphs, we notice that by eliminating the device access path sharing, we achieve better application throughputs. And, by enabling the resource controls on the device we eliminate bandwidth wastage, which is reflected in the improved throughput availability for VM1 depending on the applied constraint on VM2. For reader's who are interested in details of the simulation environment and validation of the simulation results with that of experimental observations, they may refer to [7][12].

5. Related work

Recent advances, in adopting virtualization, have resulted in looking closely at the security issues with virtualization technologies on offer [3]-[6]. Largely it is believed that, the nature of security threats to virtualized systems is no different from the ones afflicting non-virtualized servers [10][11]. What gets added to the long list is low-level hypervisor attacks and deployment of malicious virtual machines. Literature on system security has cited that faulty or malicious device drivers are one of the major causes of security threats. In this regard, prevalent I/O virtualization techniques have

only increased this threat. I/O handling in traditional OSes has always been through the privileged OS kernel. Prevalent virtualization architectures have merely extended these OSes to support I/O device virtualization using a software layer. Such architectures force the device virtualization layer to be shared amongst all the VMs accessing the I/O device and thus make it the vulnerable component of the system. Any failures or crashes in this layer, which includes the device driver, leads to failure of all the VMs using the device. Both hosted and para-virtualized technologies are examples of such systems [3]. Also, due to the software abstraction for virtual devices, these architectures are susceptible to denial of service attacks [10]. This is particularly due to unrestricted use of I/O device bandwidth on the incoming device queue [7]. Since on virtualized servers, single physical system is shared amongst multiple VMs, it is mandatory to have resource usage controls closer to the physical resource, rather than within the OS abstraction layers as is the case with current technologies. Doing so leads to effective control on monopolizing the resource and limits denial of service attacks. We believe that virtualization can be used as a mechanism to provide the desired isolation [8] and security [9] if architected correctly. For this, the hardware micro-architecture must be modified to support safe concurrent device access to the multiple VMs sharing it and the virtualization architecture should permit the virtual devices to be defined within the VM's address space. With these inclusions, the proposed I/O virtualization architecture addresses both the issues of vulnerabilities identified in the prevalent architectures.

6. Conclusion

Most of the prevalent virtualization architectures extend the single-OS single-hardware model to support virtual I/O devices. As a consequence, all shared devices also cause device access path to be shared. This model of sharing makes the virtualization software the most vulnerable component of the system. In the paper we argued that virtualization has the natural ability to support simple and efficient isolation provided it is done the right way. We identified the security issues in the prevalent I/O virtualization architectures and proposed an enhancement to overcome the issues, taking NIC virtualization as an example.

References

[1] Smith, J. E. and Nair, R. 2005. The Architecture of Virtual Machines. Computer 38, 5 (May. 2005

[2] M. Welsh, D. Culler, Virtualization considered harmful: OS design directions for well conditioned services, Hot Topics in OS 8th Workshop, 2001.

[3] Karger, P. A. 2009. Securing virtual machine monitors: what is needed?. In Proceedings of the 4th international Symposium on information, Computer, and Communications Security (Sydney, Australia, March 10 - 12, 2009). ASIACCS '09. ACM, New York, NY, 1-2

[4] Karger, P.A.; Safford, D.R.; , "I/O for Virtual Machine Monitors: Security and Performance Issues," Security & Privacy, IEEE , vol.6, no.5, pp.16-23, Sept.-Oct. 2008

[5] T. Ormandy, An Empirical Study into the Security Exposure to Host of Hostile Virtualized Environments, Available online http://taviso.decsystem.org/virtsec.pdf.

[6] Steven J. Vaughan-Nichols, "Virtualization Sparks Security Concerns," Computer, pp. 13-15, August, 2008

[7] J. Lakshmi, S. K. Nandy, I/O Device virtualization in Multi-core era, a QoS Pespective, Workshop on Grids, Clouds and Virtualization, Conference on Grids and Pervasive computing:128–135, 2009.

[8] R. Sailer, et. Al., Building a MAC-based security architecture for the Xen opensource hypervisor. In *Proceedings of the Annual Computer Security Applications Conference (ACSAC)*, December 2005.

[9] Andre van Cleeff, Wolter Pieters, Roel J. Wieringa, "Security Implications of Virtualization: A Literature Study," Computational Science and Engineering, IEEE Int. Conf on, pp. 353-358, 2009

[10] T. Garfinkel and M. Rosenblum, When virtual is harder than real: Security challenges in virtual machine based computing environments, In 10th Workshop on Hot Topics in Operating Systems, May 2005.

[11] J.S. Reuben. A Survey on Virtual Machine Security. Helsinki University of Technology, 2007. Available online at http://www.tml.tkk.fi/Publications/C/25/papers/Reuben_final.pdf. Last accessed, 20 March 2010.

[12] J.Lakshmi, S.K.Nandy, Modeling Architecture-OS interactions using Layered Queuing Network Models, International Conference Proceedings of HPC Asia:382–389, 2009.