Global Information Assurance Certification Paper

Security Implications of the Virtualized Data Center

GCIH Gold Certification

Author: Alan Murphy <m.alan.murphy@gmail.com>

Adviser: Jim Purcell

Accepted: September 16th, 2006

1. Introduction

"Data does not leak across virtual machines and applications can only communicate over configured network connections." - vmware.com[i]

The concepts behind application and operation system virtualization are not new concepts, they have been around long before server appliances and desktop PCs were readily available in our daily vocabulary. The recent rate of virtualization adoption however, especially that of software operating system virtualization, has grown exponentially in the past few years. According to Joe Tucci, EMC CEO, most VMware customers are planning on virtualizing 50% of their IT infrastructure within the next three years[ii]. Virtual machines have finally come into their own, and are quickly moving into the enterprise data center and becoming a universal tool for all people and groups within IT departments everywhere.

So what exactly is a virtual machine? VMware defines a virtualization as "an abstraction layer that decouples the physical hardware from the operating system..."[iii] Today, we commonly think of virtual machines within the scope of one hardware platform running multiple software operating systems. Most often this concept is implemented in the form of one operating system on one hardware box (the host platform) running multiple independent operating systems on virtual hardware platforms in tandem (the guests). Platform virtualization usually relies on full hardware segmentation: allowing individual guest platforms to use specific portions of the physical host hardware without conflicting or impacting the host platform. For example, even though guest operating systems require the same CPU and RAM access that the host system requires, the guests use different hardware locations

and addresses than the host. This allows the host and guest(s) to run in tandem without stepping on top of each other.

There are two primary types of platform virtualization: transparent and host-aware (often referred to as paravirtualization). Transparent virtualization is implemented so that the guest is not aware that it's running in a virtualized state. The guest consumes resources as if it were natively running on the hardware platform, oblivious to the fact that it's being managed by an additional component, called the VMM (Virtual Machine Monitor), or hypervisor. The more standard forms of virtualization today, such as those by VMware, implement transparent hypervisor systems. These systems can be thought of as proxies: the hypervisor will transparently proxy all communication between the guest and the host hardware, hiding its existence from the guest so the guest believes it's the only system running on that hardware.

Host-aware implementations differ in that the guest has some form of virtualized knowledge built into the kernel; these can be considered "virtual self-aware" environments. There is some portion of the guest operating system kernel that knows about the existence of the hypervisor and communicates with it directly. Rather than transparent proxying of all communication, the guest OS will call the hypervisor directly, which will in turn manage the communication to the hardware. Xen (pronounced 'zen'), a popular virtualization implementation for Linux, uses a host-aware architecture, requiring special hypervisor command code actively running in both the host and all running virtualized guests. Each form of virtualization comes with pros and cons, but both work equally as well. Transparent systems are the most portable for the guest, but sacrifice speed and

are typically designed around much heavier hypervisors; host-aware systems are faster and more lightweight, but require guest modifications and can introduce security issues that transparent systems may not suffer from.

One of the driving factors in virtualization adoption is the open nature of hardware support for VMMs; Hardware platforms, which run and manage the primary host operating system, and the VMM are not specialized devices or appliances. Virtual host platforms can be any type of hardware that we use today: single CPU desktop machines; laptops; x86 servers; SPARC servers; rack mounted appliances; etc. A normal user running Windows XP Professional on their laptop can run multiple virtual instances of other operating systems-such as Linux, BSD, or Windows Vista-using any number of freely available VMM software implementations. This flexibility, the move of virtualization software to everyday hardware, has allowed everyone direct and inexpensive access to run virtualized environments. While at first this access was relegated to technology professionals, such as Unix users who were required to run Windows as their base OS, it has quickly become the topic of IT managers. Platform virtualization provides an inexpensive mechanism to substantially expand server farms and data centers. Virtualization allows a company to purchase one high-end hardware device to run 20 virtual operating systems instead of purchasing 20 commoditized lower-end devices, one for each single operating platform.

2. Virtualized Threat Vectors

The benefits of virtualization are obvious: more bang for your buck. But everything has a pro/con list, and virtualization is no exception. The pro column

is a large one, allowing progress and flexibility for multiple projects and environments. But the con list isn't so obvious. What could be bad about running 20 servers for the price of one? Although by no means considered to be a large threat today, security of virtual machines and environments is typically not considered, not because the security of these implementations is a technological mystery, but because it is generally an unknown vector by the groups that are implementing wide-spread virtualization. In other words, virtualization is usually implemented with no specific regard to the new security risks it brings. On the surface, a virtualized BSD guest carries with it the same security threats and issues that a real, single-device copy has, and this is mostly correct. However the glaring difference is that extra management layer: the hypervisor. The hypervisor, is in effect, another operating system that manages the communication between the host OS and the guest OS. Rather than worrying about a single BSD implementation on a single device, administrators now have to worry about the security of three operating environments. For example, if you harden the virtual BSD guest installation, but leave the host and VMM unaddressed, then you're ignoring critical components. If the host is compromised, then security on the guest devices is irrelevant.

While the hypervisor is the "Master Controller Program" of virtualization, it isn't the only virtualized abstraction layer that carries with it implied security risks. Another critical piece to any virtualization system is the networking layer. Different virtualization software platforms have different methods for building and handling networking between the guest and the host, ranging from the more basic, generic network emulation of Qemu[iv] through VMware's

extremely sophisticated, and proprietary, software switching platform. Often accepted as "just working," virtualized networking should be considered one of the most critical areas of security research for virtual machines and environments. Unlike hardware network devices, a software-based network brings with it security issues that typically aren't seen in hardware. For example, until recently VMware's guest networking subsystem functioned in "hub only" mode, meaning multiple guest instances on the same physical host within the same software network domain had open and free access to all network data shared within that domain. It was a standard hub implemented via software. If two guests were both a member of the same host-only network and shared the same virtual interface, both machines could see all traffic between the host and guest. This is still true today for VMware's bridge mode configuration, where all guests map to the same physical network devices. And although there are barriers for more robust configurations, such as using multiple VMware software networking devices (known as vmnet0, vmnet1, vmnet2, etc.), their entire network management subsystem still exists in software. Using multiple virtual network interfaces to segment traffic equates to plugging two machines into the same switch, each with unique subnets yet neither segmented by VLANs.

VMware is also a great example of where software network security development is rapidly moving ahead. Having created a very robust software switching network in more recent enterprise releases, VWware is moving towards full software-only segmentation of virtual switching from layer 2 VLANs through layer 4 routing. Figure 1 shows a VMware ESX Virtual Switch (vSwitch0) with multiple virtual VLAN segments. As the diagram shows, 'vm_oracle' and 'vm_sharepoint' are on

separate VLANs, but both networks are part of the same virtual switch and traffic flows through the same physical network card. But even with these security-minded advances, software switching still carries a greater risk than hardware. Unlike a real networked environment where proximity is usually a substantial barrier to attack, all guests on one host share the same software stack across all other guests and the hosts. Network stack sharing is a key issue in virtualization security; if all guests and the host share the same network stack, then an attacker only has to exploit one to gain access to the entire stack. A simple exploit in VMware's networking subsystem-for example, one that takes advantage of a flaw in the way the host software switching network driver handles VLAN tags and layer 2 frame sizes-could allow an attacker on a guest to see all network data as it passes from guest->guest, guest->host. Even data that's exchanged between the host and the physical network, whether or not there is software segmentation between the various guests, could be exposed.

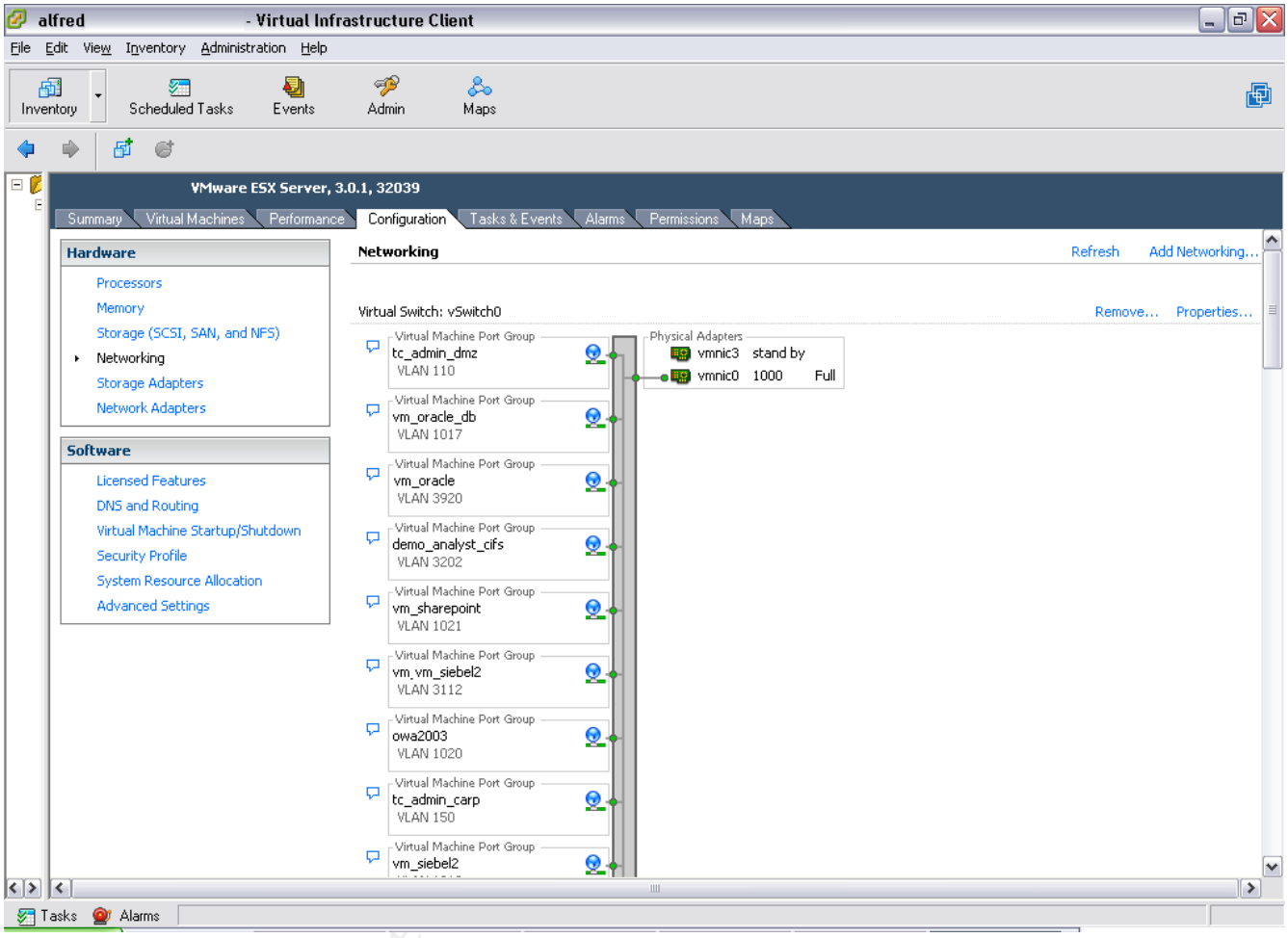

Figure 1 - VMware ESX Virtual Switch with multiple segmented VLANs

More than anything, the platform virtualization paradigm needs to lead us to new types of administrators: system, network, and security administrators should expand their knowledge set to include new concepts introduced by virtualization. Network administrators spend years honing their craft, but those years are spent in the physical world, with solid-state switches, cables, CAM tables, and VLANs protected by proximity. In the virtualized world, not only do all of these concepts move from hardware to software (and many times from software kernel space to user space), but now they've been obfuscated and "closed," unavailable to

traditional access methods. System administrators that manage virtual switched networks no longer have easy tools at their disposal for monitoring and trouble-shooting. An admin can't walk up to a virtual switch and plug in their laptop, add a network tap, reliably mirror a port, or usually even review statistics for the virtual device. All of that power and knowledge has been taken from the finely-tuned administrator and abstracted behind software controllers, management GUIs, proprietary kernel modules and binaries, and most importantly, moved into a realm where only the designers and developers know how to truly administer the devices. Imagine if the only people that could troubleshoot a CheckPoint firewall or Cisco switch were the 10 architects and designers that built them. This is a scary situation.

3. New Security Order

We've established that virtualization should not be taken for granted in the security realm, and should instead be treated as more of a daily threat than physical and single-purpose operating systems and appliances. But what are some specific concerns with "going virtual"? Let's look at a few examples of where virtual security has real-world issues:

Attacking Management Interfaces: Host

All virtual operating system environments have some type of software management or control interface responsible for managing IPC (Inter-Process Communication) calls between the guests and the host. This is typically managed by a local process on the host OS and controlled via the hypervisor. These management

applications are what allow processes on the host to monitor and interact directly with guests; otherwise there would be no way for the host to manage (or control) the guests. The best examples of this type of architecture are VMware's vmware-tools and Xen's guest kernel modules, both of which allow their respective host applications to control the guest machines and environments via the hypervisor.

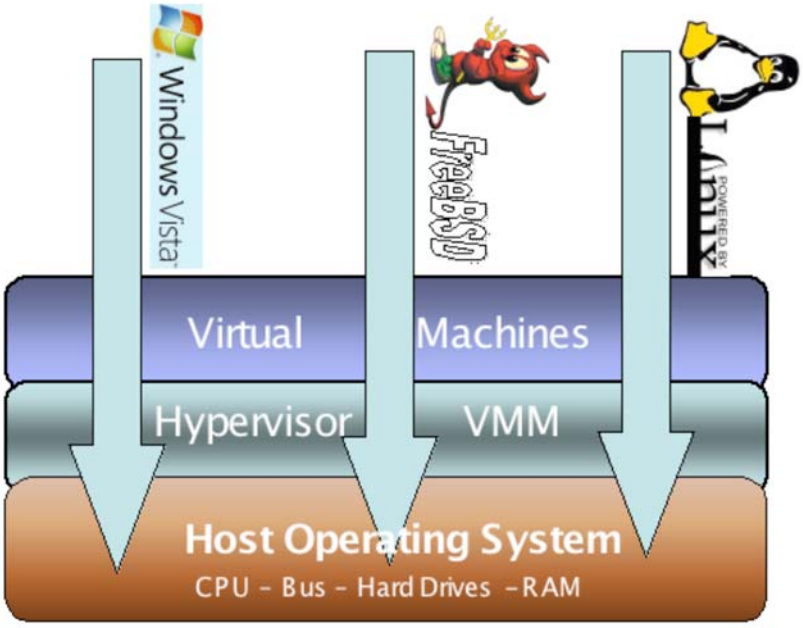

Figure 2 - Virtual machine Hypervisor architecture

Anyone that's ever run VMware has used and understands vmware-tools. This application provides VMware direct access from the host to the guest via kernel-level IPC applications and services. Typically vmware-tools is used for non-emulated hardware access from the host to the guest, such as mounting CD-ROMs (or ISO files) from the host to the guest, and providing accelerated hardware access for peripherals such as USB devices and mice. The same communication model is also

used by the VMM/hypervisor to manage user interactions between the host and the guest, such as controlling the mouse as it moves over a virtual instance in the management GUI.

Let's look at using vmware-tools to mount an ISO file from the host as a virtual CD on the guest. First, the VMware process on the host must be able to access the ISO file. This normally isn't an issue because VMware processes run as an escalated user with escalated privileges ('root' on a Linux machine or 'system' on a Windows machine). Next, there has to be a process on the guest that knows how to communicate with the process on the host. This requires software to be installed on the guest that communicates, via the command calls through the hypervisor, with the host VMM. The guest management subsystem makes a call to the host binary, typically via a named pipe, asking the host binary to initiate a hardware call on behalf of the guest, "tricking" the guest into thinking it's accessing a physical CD via a physical drive. This is a very basic example of how VMware implements vmware-tools to mount a CD from the host onto the guest. Simple yes, but extremely important for security, because we've created an unchecked system running as super-user at a kernel level on both the host and the guest, which can be manipulated and exploited by a malicious attacker.

Let's look at a real world example, a Linux VMM host that's running 10 virtual instances of Windows 2003. Each of the virtual guests is part of a redundant HTTP server farm running the finance department's accounting web application. The content and data on each of the virtual guests is replicated between virtual drives and is always identical. Regardless of which HTTP server the end user accesses, the back-end data will always be the same. The user has no

idea that they are accessing more than one server at any given time, or that all of these web servers are virtual. Example Co. hasn't built out a fully redundant database back-end for this data yet, so they've built a very inexpensive virtual infrastructure to manage distribution of this financial data.

Once a week, a cron job on the Linux host machine mounts a new CD locally containing that week's financial statements for Example Co. A job is scheduled on each of the Windows virtual guests to copy these financial statements off of the virtual mounted CD (mounted via vmware-tools, remember) allowing replication of the data between each of the financial servers. Alice, the malicious attacker, has compromised the Linux host by using a weak password attack and her favorite local escalation exploit, the ELF 'uselib' kernel exploit, granting her full root-level remote access to this host. Although her primary goal is to steal the weekly financial data for extortion, she quickly realizes that she has an opportunity to threaten both the data and the applications hosting the data.

Alice has a plethora of tools and options available to her to attack the guest systems. For example, she could replace the ISO with one that contained an AutoRun function, forcing the contents of the virtual CD to be run immediately on the Windows guest as soon as it was mounted. She could then simply wait for the ISO to be mounted on the guests at the scheduled time, running any arbitrary application she wished on the guest, such as attack tools, viruses, sniffers, or maybe netcat. With netcat, she could then access the guest systems directly (and remotely) any time she wanted. This delayed access allows her to covertly wreak havoc on the guest webservers, sniffing out passwords of all the executives that log in to request the financial statements, and then funneling the credential data

back to her hideout in the remote woods of West Virginia. What started as a simple data hijacking attack can quickly morph into a full assault on an entire virtualized data infrastructure.

Attacking the Hypervisor via the Guest

Escaping Virtual Machines

Let's look at the same architecture scenario above, but this time Alice has compromised a VMware guest instead of the host. Without even realizing it, Alice attacked a virtual system instead of a physical one. On the surface, the attack vectors are very similar. Once on a guest HTTP server, Alice has access to the financial data that she's looking for, but in this scenario she's sandboxed to just this one guest; she doesn't yet realize that she's on just one virtual machine in a virtual server farm. Before she can mount an attack on the entire virtual infrastructure, she 1) needs to realize that she is inside a virtual machine managed by a host VMM, and 2) escape that virtual machine. Once she escapes the virtual sandbox, then she can access the rest of the infrastructure and continue on with her master extortion plan.

To escape a virtual machine, a detour into hardware and operating system design is in order. In x86-based CPUs, ring 0, often referred to as kernel mode, is part of the CPU where kernel-level processes are managed. Likewise, ring 3, or user mode, is where user-level processes are allocated processing space. Virtualized operating systems require ring 0 access to the CPU just like a real, locally installed operating system. Physical and virtual operating systems both expect certain pieces of kernel code, such as interrupt tables and maps, to run in

ring 0 and always know where to locate those pieces of code in RAM. Unlike their local physical counterparts, however, it is impossible for a virtual guest machine to place an interrupt table in the same location in RAM as it would if it were a stand-alone machine; the host's interrupt table will already occupy that location in memory. So while the guest believes that it's interrupt table may be located at 0x0000ffff (where the table is always stored in this particular operating system) in it's virtual RAM, the host has actually mapped this location, via the transparent hypervisor, to location 0x1234abcd in the physical RAM, obscuring the real location from the virtual system. Every time the virtual guest operating system needs to reference the interrupt table at 0x0000ffff, the hypervisor transparently translates that call to the real location at 0x1234abcd, and then back again as the instruction result is returned to the guest. The guest is unaware that the hypervisor has manipulated the location of the interrupt table. This is kernel-level space, however, so even though the host hypervisor has manipulated the actual location of the interrupt table for the guest, the virtual table must still run from (and reside in) privileged ring 0 on the real CPU.

For the most part, any attack that would target the guest's virtual CPU would most likely result in nothing more than crashing the guest; a virtual DoS in essence. This is not much different than exploiting CPU microcode in a physical CPU from a host operating system, but reading and writing bits directly to CPU registers is not a trivial task. But with a virtualized environment, the situation is a bit different. Not only are the CPU registers now themselves virtualized, but there are more pieces to this puzzle, and as discussed above, most of these pieces are hidden from the user. VMware, for example, doesn't open its source code for

vmware-tools or the hypervisor, so there's no way for the general public to know what happens in VMware's virtual ring 0 when the guest makes a request to hypervisor for data in its interrupt table. We know that the VMM translates and delivers that call via IPC to the super-user binaries running on host, but that's about it. If an attacker can find a way to exploit virtual microcode, then they may be able to manipulate the host kernel and CPU. This is called virtual machine escaping: jumping out of the confines of the virtual environment and into the physical environment.

Virtual Machine Detection

Before an attacker can launch targeted attacks against a virtual environment, the attacker must know where the virtual machines are and if they are currently on one. If local access to the platform is available, such as a command terminal accessible via SSH for example, there are tell-tale signs that a machine is virtualized. Easily detectible items, such as the MAC address, the process list, files installed on the machine, drivers, etc. are just a few keystrokes away. Beyond the basics, however, there are a few tools that will aid the attacker in detecting if a machine is running in a virtual environment, and many times even return critical information such as the current location of the guest's virtualized interrupt table. Two popular tools for virtual detection are redpill[v] and scoopy do[vi], each of which can be run silently in the background via the CLI as to not create suspicion, and then delivered back to an attacker via a network tunnel with tools such as netcat. An attacker may utilize these tools on every machine that they have compromised, creating a list of known virtual environments for targeting with specific attacks. There's no need wasting cycles trying to

attack a virtual CPU with destructive microcode before knowing if the machine even has a virtual CPU. Tools like redpill are the first step in mass virtual machine attack harvesting.

Attacking the guest and escaping the virtual machine really becomes the Holy Grail in targeted attacks against a virtualized environment. If, or when, attacks focused on virtual machine become readily available, the attacker only has to spend time attacking one virtual machine, which could lead to compromising other virtual machines over a closed network, and eventually escaping the virtual VMM environment and accessing the host. Remember, virtual machines must think that their kernels are running in protected and privileged space on the CPU and in RAM, and likewise have to be granted access to these physical locations on the host by the hypervisor. But instead of one kernel running against one CPU platform, virtualization requires multiple kernels to share access and interact together, even though they don't realize it. If an attacker's goal is to attack as many machines as possible, and they are aware that a particular system farm is all virtual, hypervisor-based attacks will provide the most lucrative attack pool. This protected-level access shared across multiple virtual kernels could open the door for all types of attacks, such as:

- Passing destructive microcode through the virtual CPU down to the physical CPU

- Adding Trojans to the virtual machine that are passed down through the hypervisor to the host machine, installing and running resident in ring 0 on the host

- Manipulating the virtual machine management interface itself to bypass authentication between the guests and host

- Using the hypervisor to gain access to the software network subsystem, and placing both the guest and host network interfaces into promiscuous mode allowing sniffing of the host network transparently via the guest

- Exploiting virtual hardware drivers to manipulate drivers on the host

- …and possibly most destructive and lucrative, attacking and manipulating the hypervisor itself. If an attacker can manipulate and control the hypervisor on a multi-platform virtualized host, then the attacker can control ***all*** hardware and software commands for ***every*** guest accessing the hypervisor. This type of attack would equate to owning every server in a data center, down to the CPU and bus level. It would be a rootkit on a data center sized scale, translating core kernel operations between the host and every guest

Real-World Attack Examples

While all of these may sound extreme, sophisticated, and highly theoretical, the most basic security threat imposed by any virtualization system is the law of large numbers. In a typical attack scenario, an attacker has to focus their attacks on one machine at a time, regardless of their intent. Sure, there are methods to go after multiple machines at the same time, utilizing zombies and bot armies, but these attacks are still a one-to-one relationship. Attack one machine to inflict harm on that one machine. Virtualized environments remove that

restriction and create a one-to-many attack scenario: attack the host, own the guests. Or even attack one guest, possibly own them all.

Attacking Virtual Disks from the Host

Let's use our example from above where an attacker, Alice, has compromised a Linux host running VMware with 10 Windows 2003 virtualized guests. Alice's ultimate target is to destroy the financial data stored on all the virtual webservers hosted on the single Linux host system. Alice knows that she'll need to destroy this data within each virtual machine because, like our previous example, each virtual guest writes to its own local storage and then propagates the data out to each redundant virtual storage device. Alice needs to remove both the shared storage and the localized virtual storage devices in the guests to eliminate all traces of the critical financial data. In a typical single-box scenario, Alice would have to gain access to each box individually in order to electronically shred every local data partition, which would mean mounting a tedious 1:1 box attack. Since all of Alice's targets for this attack are virtual and hosted on one physical host, she can take advantage of this virtual infrastructure. All of the guests use virtual hard drives: flat files accessible from each guest and from the host.

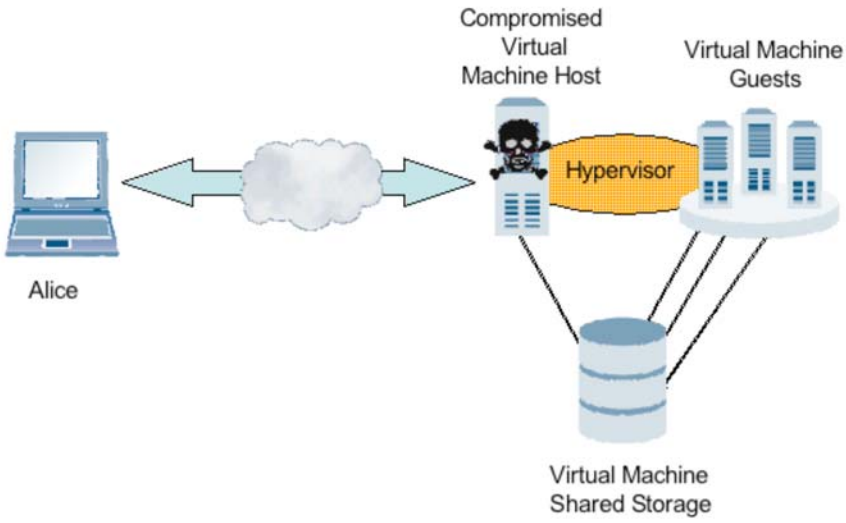

Figure 3 - Virtual shared storage architecture

By now, you're probably thinking "Of course! On the Linux host, Alice has access to vmnet*, which include the management interfaces and VLANs for the HTTP servers, so she can simply attack the guest from the virtual host network, or easier, just sniff the SMB Admin password as it flows from eth1 on the host to vmnet3 on the guest!", and you would be correct. However, Alice is only intent on destroying data, so she's going to take the path of least resistance. She already has root access to the Linux host, so she simply schedules a cron job to run at 2:57 a.m. the next day, which runs:

```
[root@vmhost:/] # for vmdisk in `find . -name "*.vmdk"`; do dd bs=1024
count=10 if=/dev/zero of=$vmdisk; done
```

And that's it. Within seconds, Alice will have overwritten the first 10k in each VMDK file (the flat file VMware uses as a virtual physical hard disk), rendering them all unreadable. Unlike a physical disk, where portions of the disk are allocated for things like the boot sector and master boot record, and just deleting those usually won't render the data on the rest of the disk inaccessible, VMware typically isn't as resilient. If it can't understand the beginning of the flat file virtual drive, then the rest of the file is just random data, and lost forever.

Alice was able to attack the physical data across 10 critical Windows web servers simply by mounting ***one*** attack against the Linux host machine. She doesn't need to spend her time working on each machine individually, and she doesn't need to know anything about how to attack a Windows box. Example Co.'s own virtual infrastructure environment allowed Alice to accomplish her destructive goal with one attack.

Planting Attack Seeds: Virtual Library Check-Out

The malicious data storage attack is one example where virtualization itself introduces new attack vectors for the intelligent attacker…but what about the more standard virus outbreak, where a virus is unintentionally introduced into the virtual network? This type of anonymous attack is much more common, and one that enterprise networks are faced with every day. Virtual networks tend to give a false sense of mitigated security risk because most administrators consider these environments to be locked-down and isolated. Let's examine a scenario where this will end up hurting Example Co., not helping it. In this example, an entire

network was defined and created in a virtual environment just to help stop these types of attacks.

Example Co. is building a proof-of-concept (PoC) database environment for a future CRM system. Once completed, the production environment will consist of multiple database platforms, web application front-ends to allow access to the data, and client systems that will be deployed in kiosks around the world, all virtualized. Before Example Co. spends tens of millions of dollars on an entire global system, they want to see a small virtual version in action. To mimic this production network, 10 virtual systems will be created in the staged development network. They hire a consulting company to come on-site and build the virtualized database PoC lab.

Work progresses as expected for weeks, as consultants bring their laptops in and out of the quarantined corporate network. While working on this CRM PoC environment, the consultants routinely "check out" the images they are working on from the virtual infrastructure library and work on these images off-site for days and weeks at a time. When they return to the corporate office, the contractors will "check in" the updated virtual image back into the infrastructure library, saving Example Co. on-site consulting and travel costs. These mobile images are allowed access to only the virtual library PoC environment and network they are building which is further removed from the corporate network via segmented VLANs on a /27 network segment. Example Co.'s IT department doesn't trust the consulting laptops-rightly so-and does not grant them, or these virtual images, access to any part of the private corporate network. Eventually the virtualized PoC database lab is complete and ready for the IT group to take over and begin testing.

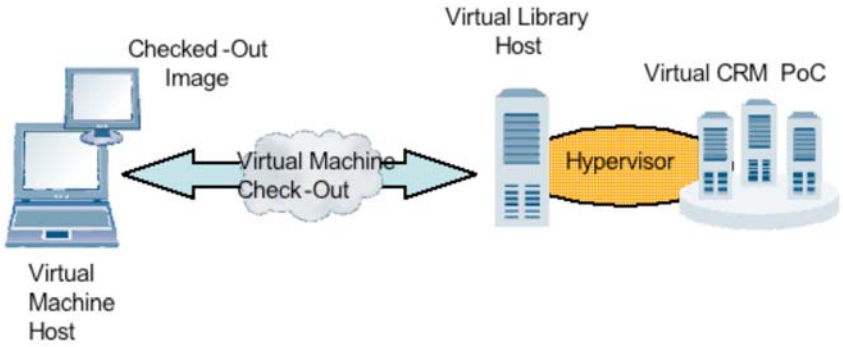

Figure 4 - Example Co.'s virtual library check-out system

Days progress and Example Co. is very happy with their virtual PoC library, which has allowed them to fully test an entire CRM system at a fraction of the cost, ironing out bugs and logistical issues that would have been show-stoppers for a live environment. Unfortunately, this blissful state is short-lived. Over the weekend, all of the new virtual images crashed unexpectedly, taking the entire virtualized lab off-line. The IT staff spends all of Monday trying to bring the images back up without much success; most of the images, except for one, are completely dead and aren't bootable; they're virtual drive files have been corrupted. Debugging has taken much longer than expected because standard forensic investigation of the hard drive and filesystem is unavailable; these systems are virtualized and the hard drives are now just large corrupt binary files.

The sole remaining image is a haggard state, however, and although it will boot enough to recognize a boot loader, the operating system won't boot and no applications will run. Thankfully, Example Co. has a resident virtual machine expert, and she is able to mount the corrupt filesystem into a new image and repair it enough for some basic investigation. She is also able to determine that this image was the last image to be checked in by the consultants before they finished their assignment. The news is grim. This image contained a time bomb: a piece of malicious code installed and programmed to "go off" at a later date. Once auto-detonated, this particular bomb spread from machine to machine via a self-replicating worm that contained a very nasty payload. The payload consisted of tools specifically looking for virtual environments and exploits against a very well-known virtual management machine. This particular exploit targeted the host hypervisor via an infected guest image, allowing the worm to locate and corrupt the hypervisor via the guest and take down the entire virtual library in one fell swoop. Destroy the hypervisor, destroy the virtual infrastructure. The only flaw in this particular worm was its inability to remove itself and the image it was carried in on; it was unable to completely remove its carrier image because the binary had to remain resident to execute the attack.

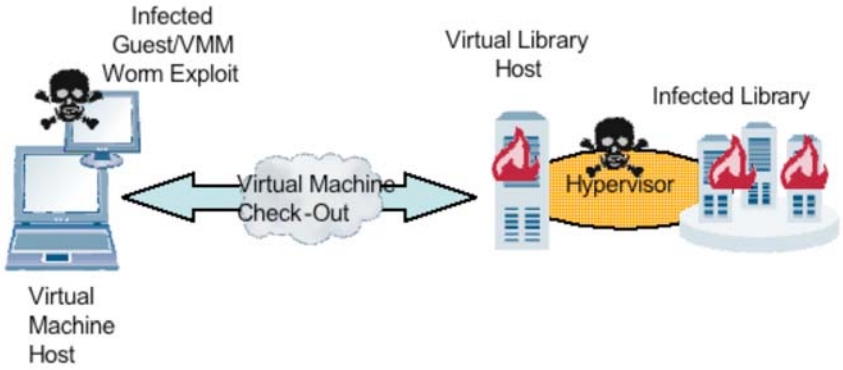

Figure 5 - Infected virtual infrastructure "guest bomb"

The IT group felt that since the network was completely segmented and isolated from the corporate network, there really wasn't that much risk. Ultimately both the host and the entire guest network had to be rebuilt from scratch, forcing the project to start over and doubling the budget. The virtual infrastructure had been a success for building a quick, isolated test environment, but the IT department had failed to realize the risks involved with allowing mobile operating systems and virtual guests. The guests were allowed be taken home and brought back without any type of screening process. IT departments routinely ban employees from brining personal machines into the corporate network, and from corporate machines going home, all in the name of security. Why should virtual machines be treated any differently? Unfortunately, it's usually due to a lack of

understanding about the destruction that a mobile platform can inflict. It's easier for us to realize these dangers with a physical mobile device, like a laptop, than it is with virtual devices, which are often viewed just as applications.

4 Conclusion

With enterprise-class virtualization software hitting the market from VMware and Microsoft (ESX Virtual Infrastructure and Virtual PC, respectively), large names and support dollars are backing the virtual data center migration. Microsoft's Longhorn platform will ship with a native hypervisor, allowing direct virtualization via the operating system without the installation of a third party VMM. CPU manufacturers are rapidly working on implementing hardware-based hypervisors, such as AMD's Pacifica project. It is now almost common-place for corporate IT divisions to replace en masse the single-purpose 1U machine hosting one operating system with a 4U machine hosting 20 independent operating systems. We accept that operating systems carry with them security flaws and thus we're wary to deploy un-patched systems on our networks, yet we blindly accept that VMMs and their guest operating systems are secure. The concept that virtual operating environments are just as secure as their physical counterparts can be a very expensive and destructive fallacy.

With this virtualized paradigm shift comes an entire new set of security issues, problems, and risks. Security administrators are familiar with phrases such as "hardened operating system," "walled garden," and "network segmentation" in the one-box-for-one-application world, but how do administrators apply these

concepts to the uncharted waters of the virtual data centers? How can we protect ourselves in new environments we don't understand? Today's system and security administrators need to begin focusing on virtual security, preparing for a new threat arena for distributed and targeted attacks.

There are many, many security risks and considerations that virtual infrastructure administrators should be aware of and prepared for, many of which were not covered in this discussion. And there are many questions that still need to be addressed before moving to a fully virtualized environment, such as:

- How will our current analysis, debugging, and forensics tools adapt themselves to virtualization?
- What new tools will security administrators be required to master between all of the virtualization platforms?
- How does patch management impact the virtual infrastructure for guests, hosts, and management subsystems?
- Will new security tools, such as hardware virtualization built into CPUs, help protect the hypervisor by moving it out of software?
- How will known security best practices, such as no-exec stacks, make a difference when fully virtualized? Will hardware virtualization pave the way to a truly secure VMM?
- Virtualization and shared storage: What happens if we virtualized all the way down to the iSCSI transport layer? Are we opening up a

floodgate which bypasses built-in SAN security?

These are all questions that need to be addressed before the enterprise world moves full-on into virtualization. More than anything, we should be thinking today about where virtualization security will take us tomorrow. We all agree that virtualization is for the better and it's here to stay, but security administrators need to make sure they keep ahead of the threats and think about virtualized threat vectors before attackers have already coded for them.

5 References

[i] VMware, (2006). Retrieved October 9, 2006, from VMware Virtualization Web site: http://www.vmware.com/virtualization

[ii] Babcock, Charles. January 23, 2007. Virtualization Vendor VMware Revenue Jumps 101% in 2006. *InformationWeek*. Retrieved March 6th, 2007 from http://www.informationweek.com/software/showArticle.jhtml?articleID=196902989

[iii] VMware, (2006). Retrieved October 9, 2006, from VMware Virtualization Web site:

http://www.vmware.com/virtualization

[iv] http://www.qemu.org/

[v] http://invisiblethings.org/papers/redpill.html

[vi] http://www.trapkit.de/research/vmm/scoopydoo/README