Hypervisor Malware

Honors Thesis

Fionnbharr Davies

October 2007

Supervisor:
Richard Buckland

THE UNIVERSITY OF NEW SOUTH WALES
SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

Contents

Abstract

By abusing the unintended use of virtualisation extensions in modern processors from Intel and AMD an attacker is able virtualise a running operating system without needing a restart, placing itself as the hypervisor between the operating system and the hardware. This thesis presents the design and implementation of one such piece of software and discusses the prevention / detection issues that are unique to these types of malware.

Acknowledgements

Thank you to my supervisor Richard Buckland for letting me to do something this awesome for my honors thesis. To my parents for creating me. To Bob Krouse for helping make my first venture into the kernel world less scary. To Joanna Rutkowska and Matasano Chargen (Especially Thomas Ptacek and Dino Dai Zovi) for inspiring me to work on this. Finally but not least - to all my friends for pretending to be interested in my thesis while I bored them with it.

Chapter 1

Introduction

Virtualisation is an increasingly growing industry with it being deployed much more widely than ever before. With processor speed being able to deal with having multiple virtualised hosts on a single machine it has allowed even desktop machines to be accessible to this technology. Virtualisation has become so important that CPU vendors such as Intel and AMD have added in extensions to their processors that assist in writing and using hypervisors. This has been done through an extended instruction set which allows for the loading and unloading of guest virtual machines all controlled from a hypervisor.

Rootkits have existed for a long time and are an important area of security that should be studied. Detection and prevention of rootkits is essential as another line of defence against attackers and to prevent any more harm from occurring. They are used to help an attacker keep control over a system once they have left, by hiding files/processes/connections and granting root access when the attacker returns. Once this was only an tool that skilled attackers would use against compromised hosts but rootkit techniques are now being used by other malicious programmers such in commercial botnet programs like the Storm Worm [1].

This paper shall be exploring recent research into rootkit design that utilises these new virtualisation extensions to assist the rootkit in staying hidden. Essentially they have taken on the role of a Virtual Machine Monitor, which sits outside of the operating system, making it extremely difficult for the guest to discover that they are indeed in a virtual environment. One of the principal developers of this new rootkit type has even gone so far to say that it would be ‘100% undetectable’ [2, 30]. An introduction into rootkits and their history shall also be presented to further show how this new type differs from previous designs. There is a detailed account of known attacks to defeat these types of rootkits, and how the rootkit can thwart these very attacks. Finally I present my own incarnation of this type of rootkit and protective hypervisor.

> ‘Know your enemy and know yourself, find naught in fear for 100 battles. Know yourself but not your enemy, find level of loss and victory. Know thy enemy but not yourself, wallow in defeat every time.’

– Sun Tzu

Chapter 2

Literature Review

Since my topic has its background in two previously disparate fields, virtualisation and rootkits, this literary revue will cover both of them as individuals and then in how they are now more recently intertwined. There is quite a noticeable difference in the literature for the two, with virtualisation being quite well documented by large companies who pioneered it such as IBM who were the first to coin the term virtual machine with the M44/44X [3]. Compare this to rootkits where a lot of the best references come from 'underground' magazines such as Phrack [4].

2.1 Virtualisation

Virtualisation is a well documented field tracing its lineage back to the old time share computers of the 60's, the most notable of this era being IBM's CPCMS [5] Operating System (OS) which later became the base for VM370. These OS's were able to produce stand alone copies, or virtual machines, of the computer itself each with their own operating system [6]. The type of virtualisation used by these specific examples is particularly relevant in that they were hypervisors [7] or Virtual Machine Monitors (VMM) using full virtualisation. Full virtualisation is a technique used to implement a virtual environment in which it has a complete simulation of the underlying hardware. This means that all software that can normally be run on the unvirtualised hardware can now run in the VM without any modification of the guest OS or the software [8].

There are two types of hypervisors, the type described above where the virtualised software runs directly on the physical hardware and the second that runs within an operating environment [9]. The focus shall be on type I.

Since these beginnings virtualisation has become commonplace in many fields with its uses such as for having multiple servers on one physical machine [10], malware analysis in an environment that can be sandboxed and reset [11], and efficient access to multiple operating systems for testing software [12]. This has pushed hardware developers such as AMD and Intel to add virtualisation extensions to the x86 architecture that helps negate performance loss due to the virtualised state. The extensions are called Pacifica/AMD-V [13] and IVT [14] respectfully. With antivirus labs using virtual machines as a place to examine and analyse potentially malicious code and servers being run as virtual machines the security of virtualisation is under scrutiny. The idea that an attacker on one guest (virtualised OS) could attack another guest on the same machine is a potential problem [15]. Malware have the potential to add checks in their code to see if they are in a virtualised host and then take actions against this [16]. These actions could be destroying themselves or even go as far as trying to break out of the VM into the real system. A paper released recently from Symantec researcher Peter Ferrie went into depth describing how to detect if you're in a VMware or VirtualPC sandbox, two popular virtual machine emulators, and included assembly sourecode proof of concepts [17]. It is interesting to note that AMD and Intel are in a marketing and feature based war at the moment [18], which is one reason for these virtualisation extensions. For the first time it's no longer just about the gigahertz that a processor runs at but rather the features of the CPU too. This is important because with every new feature there is the possibility of bugs being introduced into an instruction set that has been reasonably static for a number of years. The fact that they're competing to get out as many features as possible also implies that testing is not as thorough as possible meaning a higher chance of bugs or other errata.

2.2 Rootkits

Rootkits first started to appear in 1994 on SunOS [19] and linux in 1996 [20]. These types of rootkits were basically backdoored daemons and command line utilities that allowed an attacker back into the system when they wanted to. It usually came bundled with a modified version of tcpdump specialising in password sniffing. A computer would be compromised through a vulnerability or a known password stolen from another system, the computer rooted through another local vulnerability and then the kit installed. The sniffer would be left on collecting passwords while the attacker cracked the passwords from other accounts on the system. This would be then repeated on a new computer using the new passwords garnered from the current host. Since then they have become much more complex with the introduction of abusing Loadable Kernel Modules to help avoid detection, which was first proposed in Phrack 50 in 1997 [21]. The presented techniques such as syscall redirection would become the basis of later rootkits such as SuckIT (released in 2001) [22]. Despite the publication of these new techniques more primative rootkits were still being widely used at the time such as T0rn [23].

Because of the nature of rootkits it's very hard to have any statistical data on how many were infected because they have to be found first. In 2004 a vulnerability in CVS was reported that allowed attackers to have arbitrary code executed on a host running a vulnerable CVS server. The proof of concept exploit was very unstable and only worked with one distribution. An anonymous email a day later posted a full working exploit for any Linux / FreeBDS / Solaris box, stating that they had known about this vulnerability for years and had compromised all the cvs machines out there loading up SuckIT onto many. This list included names like apache.org, perl.org and kernel.org [24]. This showed that even in high security servers rootkits can persist for many years undetected. Rootkits became more widely known with Sony employing it's own rootkit to help stop copyright infringement in 2005 [25]. They included Extended Copy Protection (XCP) and MediaMac CD-3 software on a number of music CD's that when played in a computer installed the software automatically, even before the End User License Agreement was presented. A researcher discovered this rootkit while running antirootkit software [26] and reported this to the media with a full technical rundown of the rootkit. Sony eventually recalled the CD's and offered a DRM free version and had a number of class actions brought against them [27]. Rootkits and other malware such as worms started to use Sony's rootkit as another method of staying undetected, and have been detected in the wild [28]. Microsoft released statistics stating that it had treated over 530,000 machines with the Sony DRM rootkit on it [29].

2.3 Future Trends - Rootkits and Virtualisation

Experimental rootkits have started appearing in research groups that use these new virtualisation extensions to hide completely outside of an operating system as a malicious virtual machine monitor that has claimed to be '100% undetectable' [2, 30]. This obviously has created quite a lot of controversy mainly targeted at the researcher Joanna Rutkowska who created this rootkit code named 'Blue Pill'. It has prompted AMD [31] and a senior Xen developer [32] to come out with statements trying to 'debunk' her claims. Unfortunately reading through these statements shows a reasonable lack of understanding of what Blue Pill does. This could be partially attributed to when it first came into the limelight it was used to bypass protective mechanisms in Windows Vista with the press misreporting that it was akin to a root exploit [33]. Since then another researcher, Dino Dai Zovi, has written a similar rootkit but for Mac OSX first demonstrated at BlackHat Briefings 06' [34]. Rutkowska gave a 2 day course on understanding stealth malware at Blackhat Briefings '07 [35] where she dicussed currently rootkit techniques and released the sourcecode to the new version of Blue Pill.

Chapter 3

Malware and Rootkit Analysis

3.1 Type 0

The simplest type of rootkit, a userland rootkit, is more akin to a backdoor than anything resembling a rootkit but for completeness it shall be listed here. This is usually a backdoored commonly used executable, most often suid root, that is either replaced or changed by the attacker in some way through patches, hooks or injected code into the running process (such as a daemon) that will allow them access to the system when executed in a certain way. Common executable targets like this include passwd, login, su, sudo, and can be any executable with suid privileges. These are often bundled with other modified software that might aid the attacker in staying undetected. An example of this would be an attackers ls binary that would have a range of functions such as hiding the attackers files and displaying modified access dates/times. These were the first sorts of rootkits to be created and eventually found in the wild, but also the easiest to detect. The best way to detect these sorts of rootkits is with software like Tripwire [36], which in essence records the size of all the executables on the system and will alert you when these change.

This type of rootkit is quite basic and reasonably uninteresting from a current research standpoint. Current infection and hiding techniques are all seem to use documented API's making them not only easy to write but also easy to detect.

3.2 Type I

The next type of rootkit is much more advanced than the previously described rootkit and starts to subvert the actual operating system instead of userland applications. Rutkowska's taxonomy [37] groups this type of rootkit into two distinction areas; malware that modify relatively constant parts of an OS and those that modify parts that are changing all the time. An example of the former would be modifying parts of the kernel such as an interrupt handler to use the rootkits syscall table, giving our rootkit control in the kernel or even changing the syscall table itself. An example of the latter type would be modifying data sections of running processes and the kernel. These two types are called Type I and II malware respectively.

Type I malware is easily the most common type of rootkit that would be found on a linux system if it was compromised by an attacker. They are generally installed using one of two techniques. The simplest and most common is just inserting a Loadable Kernel Module (LKM) into the running kernel using insmod. This is a extremely easy way to get code running in the kernel without much effort, but can be easily defeated by turning off the ability to insert modules which can be done while compiling the kernel. The most widely known rootkit that does this is adore written by Team Teso's 'stealth'.

The second method of getting rootkit code to run in the kernel is to use runtime kernel patching of /dev/kmem. This is not as widely used because it's technically more advanced and not usually needed as security isn't often that much of a concern to remove the ability to insert a LKM. This technique was known to be theoretically possible for a long time but never actually implemented until Silvio Cesare published his paper [38] in 1998. The first documented rootkit that used this technique is SucKIT presented in Phrack 3 years later [22].

One technique that can be used to detect this sort of malware that is similar to Tripwires technique is to take hashes or signatures of parts of the kernel that are often changed, but shouldn't be. A difference in the hash would indicate that a rootkit has modified part of the kernel that should never be changed under normal circumstances. But for this to happen we do need to have a original untainted hash in which to compare with, which would have to be kept secure.

3.3 Type II

Type II malware is different from Type I in that it modifies only dynamic resources to get its code run, such as function pointers in the kernel. The benefit of this is that it these changes are hard to distinguish from legitimate changes that occur in normal operation. This makes it much more difficult to automatically verify a system's integrity as described previously by taking hashes since this data is changing all the time. Not nearly as much research has gone into detection of this type of rootkit as we still don't have any sort of reliable way to detect type I rootkits.

A novel example is prrf [39] that is reasonably outdated now as it was initially publicly released in 2002 but still an interesting example. It shows how it is possible to backdoor a kernel through modification of the proc file system without changing any system calls. This rootkit was able to do everything a normal rootkit could do such as process/file/connection hiding and giving root access to certain processes all through manipulation of proc entries.

3.4 Type III

Type III malware is something entirely different to the previously described types of rootkits and is closely related to the growing field of hardware assisted hypervisors. This type of malware shall be the focus for the rest of this document. While the other types of malware either live in user or kernel land, type III malware lives completely outside of the operating system and does not change any visible registers or memory of the running OS. This is all achieved through new hardware assisted virtualisation additions on most of the newer CPUs such as the Core 2 Duo's from Intel with their virtualisation technology VT-x and AMD's SVM.

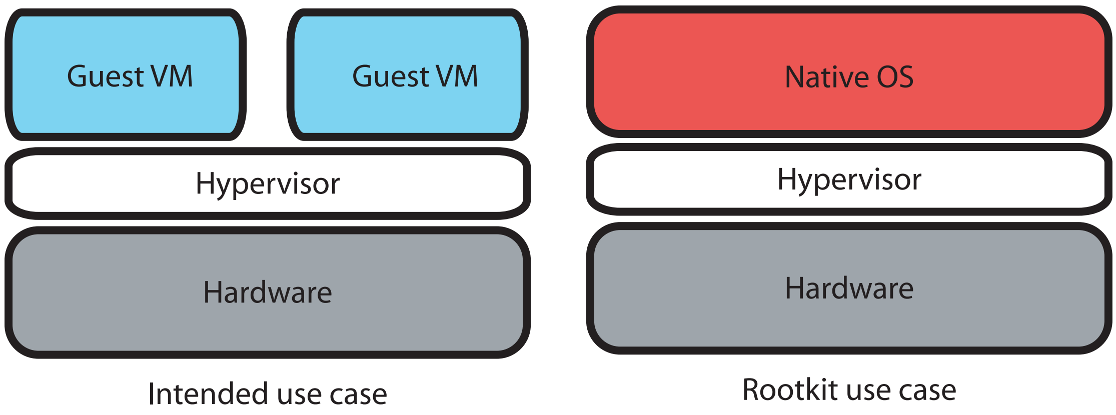


Image based from presentation from Lawson, Goldsmith and Ptacek at Blackhat Briefings 07' [49]

Through the addition of new opcodes that allow the setup, modification and launch of Virtual Machines (VMs) an operating system can be put in a virtualised state with a malicious hypervisor running in a protection level that is above ring 0 with the normal ring 0 becoming the VMX non-root state. This is the essence of the new additions in that it carves out a new protection level called VMX root, which has been dubbed ring -1 by some, where the Virtual Machine Monitor (VMM) resides. The VMM is generally not very intrusive and only executes in response to certain instructions and events in the VMX non-root operation. A VM-Exit is when the execution is given to the VMM because of an event in the VM, and similarly a VM-Entry is when a VMM gives back control to a VM. When setting up a VM certain fields in the Virtual Machine Control Structure (VMCS), which is used to set up the environment for a VM about to be launched/resumed, can be set which allows VM exits to occur under different circumstances than normal [40].

The VMCS is extremely important as it manages the transition in and out of VMX non-root operating as well as how the processor behaves in VMX non-root operation. From Intel's Software Development Manual 3B:

The VMCS data are organized into six logical groups:

- **Guest-state area.** Processor state is saved into the guest-state area on VM exits and loaded from there on VM entries.
- **Host-state area.** Processor state is loaded from the host-state area on VM exits.
- **VM-execution control fields.** These fields control processor behaviour in VMX non-root operation. They determine in part the causes of VM exits.
- **VM-exit control fields.** These fields control VM exits.
- **VM-entry control fields.** These fields control VM entries.
- **VM-exit information fields.** These fields receive information on VM exits and describe the cause and the nature of VM exits. They are read-only.

The most important part of what is described above is that there is no real change in the way the OS executes as far as it can tell. There is no hooking, patching, or modification of any part of the OS as the rootkits code is completely disconnected from the systems own code. While it is still resident in physical memory somewhere it is not particularly obvious, but we shall further expand on that point later in this document. In addition to this a host can be virtualised on the fly without need of a restart.

From the Intel VT-x specification, 'There is no software-visible bit whose setting indicates whether a logical processor is in VMX non-root operation. This fact may allow a VMM to prevent guest software from determining that it is running in a virtual machine.'. One of the goals of Intel and AMD with these new additions is to make being inside a VM as undetectable as possible. Type III rootkits are not exploiting a bug but instead abusing a feature and features are rarely ever removed unlike bugs, which can be fixed. In addition Intel or AMD would never have a hypothetical CHECK instruction that would return true or false depending on whether you are in a VM or not since this would defeat one of their main aims of not being able to detect if you're virtualised or not.

This is an ever-changing subject with researchers discovering detection methods and attackers patching against them. It's a traditional arms race. What is interesting is how the tables are turned once the attacker gets inside a machine. Usually all an attacker has to do is find one weak link in the fence and they are in, but if they want to cover their tracks they have to put up their own fences to stop an admin discovering the malware, an ironic reversal on system security.

3.5 Public Implementations of Type III

This section will be for discussing known VM rootkits that will be quite brief; with the rest focusing on currently known public attacks on these types of malware. There is very little literature outside of blog posts and PowerPoint presentations discussing these types of malware which is representative of how new and closed off this research has been so far. There are some public implementations of type III malware as mentioned before with the two highest profile implementations being Blue Pill 1/2 written by Joanna Rutkowska and Alex Tereshkin for Windows Vista, and Vitriol for Mac OSX written by Dino Dai Zovi. The source code for Blue Pill has been publicly released.

3.6 Weaknesses and current known attacks

3.6.1 Nested VMs

One of the first questions that come to mind with these VM rootkits is 'can they be nested', as in can multiple instances of them be run on the same box living inside each other. The answer is yes, a VM rootkit can be programmed to fully emulate VMX instructions allowing VM within a VM ad infinitum. But any VM malware that isn't programmed to emulate these new instructions can be potentially detected by trying to load up a simple VM that will do some benign instructions. Failures would imply the host may be within a VM.

Blue Pill is an example of a VMM that can emulate these instructions and can have multiple copies of itself running on the same host while still maintaining integrity 'how many Blue Pills inside each other can you run and still be able to play your favorite 3D game smoothly?' [41].

3.6.2 Latency

Another basic attack is real world timing of events that would potentially cause a VM exit, such as the CPUID instruction. If this was called repeatedly over and over in loop timing the overall instructions with RDTSC, with a log of the result kept, it could be used much like a low tech tripwire setup to baseline the known good host to the future potentially compromised self. The time difference would come because the trapped instruction would cause a VM exit every time it's called, in this case causing the CPUID return value being changed, adding on only a very tiny time slice to the whole instruction but is more noticeable when added together over time.

This was forseen since 1974 by Popek and Goldberg's paper on formal requirements for virtualisable architectures [42]. Both Intel and AMD have included the 'TSC Delta' field in the Virtual Machine Control Structure (VMCS) designed to offset the TSC results given to the guest. This was essentially created to stop the described attack above. While the TSC field is the easiest way to see how many clock cycles have occured between two different instructions, processors have many more counters beyond the simple TSC field that could be used to betray interference by a hypervisor. These include HPET counters, performance counters, ACPI timers and MSRS that can indicate timing and latency [49].

Another type of latency attack that is effective in detecting virtualisation is to exploit the different cores that are in most modern processors. Assigning two concurrent threads to two separate cores, the first executing a instruction that would cause a VM exit such as CPUID and the second incrementing a counter until the first thread finishes. If the first thread takes longer than it should as shown by the seconds threads counter then that would be an indication that a hypervisor is present.

To give an indication of how much an impact having a hypervisor, even a thin one, can on a sensitive instruction data from Ptacek and Goldsmiths blackhat talk is presented below.

- **1.** RDTSC
- **2.** CPUID is executed 1,000,000 times
- **3.** RDTSC

On a clean computer this series of instructions typically takes approximately 200 cycles. On an infected computer it took approximately 40,000 cycles. [49]

3.6.3 TLB and CPUID

The CPUID opcode can be used to determine processor type and certain features such as checking for VMX support, and is a instruction that would be trapped by a hypervisor if it was being run inside a VM. What is particularly interesting about this instruction is that other than causing a VM exit it is not destructive to the TLB cache. This makes it perfect for an attack described in Ferrier's paper [17], taking advantage of cache timing in the TLB.

If the TLBs are all flushed then the time it takes to access a new page can be recorded by reading the TSC before and after the access. This value can then be averaged out across all the TLBs, which are then filled with random data. The reading process is then repeated, yielding the average time it takes to read a cached page, which is again averaged out across all TLBs. Armed with these two averages we can attempt to see if we are inside a VM.

Then the CPUID opcode is executed, causing a VM exit switching to the hypervisor. This switch will flush out some of the TLBs, which the hypervisor could not possibly stop as it happens before the hypervisor gains control. When the control is returned back to the VM the TLBs are all reread and timed. If any of the access times matches that of reading a newly paged piece of memory instead of what should be there, a cached page, then a VM exit must have occurred meaning the host is a VM.

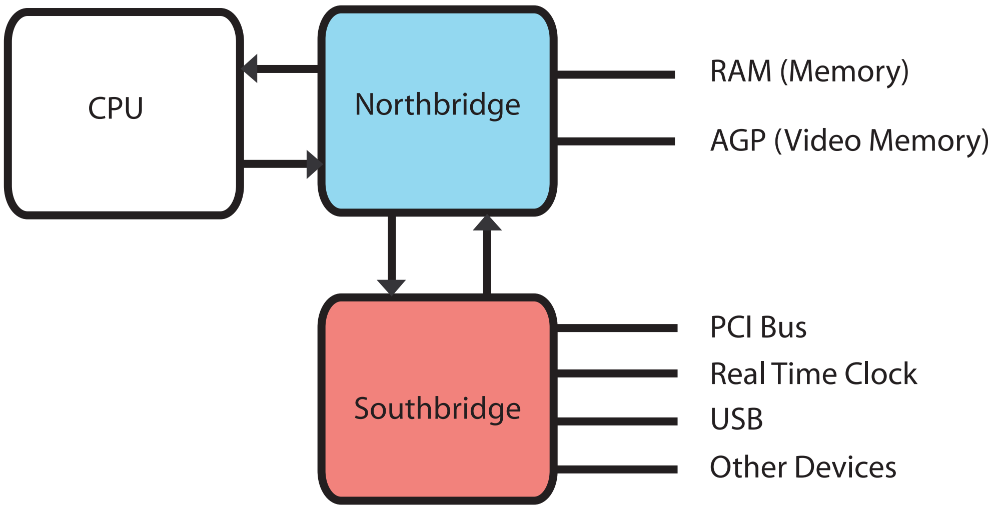
AMD's SVM differs from Intels VT-x in this regard because they have a system called Address Space ID (ASID) [43]. This prevents a TLB flush from occurring when there is a context switch between a VM and its hypervisor. This was actually designed to help performance but has the extra feature that it avoids detection by this technique. Intel will most likely follow suit and bring in something similar to this for the VT-x platform.

3.6.4 Direct Memory Access (DMA)

DMA can be potentially used to completely dump out the physical memory of the computer without the intervention of the CPU as the hardware interfaces directly to the memory controller. The simplest way to do this is over the firewire bus, first brought into the limelight at Cansec West [44] then at Ruxcon where it was shown this technique also applies to Microsoft Windows [45]. This could potentially be used to pull a VM hypervisor out of physical memory for analysis without the VMM being able to stop or know about it.

Unfortunately AMD64 PCI/HyperTransport Northbridge (the interface between the CPU, memory controller, and PCI) has control/status registers (CSRs) 'that map physical address ranges to I/O address space (that is, the range of memory addresses that are serviced by peripherals, not DRAM). This mapping is implemented in between the bus and the memory controller, and not mediated by the CPU. Which is to say, you can program the AMD64 Northbridge to remap physical addresses for devices on the PCI bus. In particular, you can map any physical address back to I/O space, so that when your Firewire OHCI controller tries to read it, the Northbridge bounces the request back out to the PCI bus.' [46] This was originaly presented at Blackhat [47].

Input/output Memory Management Units (MMU) are MMU's that connects DMA a capable I/O bus to the primary storage memory and maps



virtual addresses to physical addresses in the RAM [50]. Intel, AMD, Sun and IBM have all published specifications on their implementations that will be shipping on future processors. Abusing IOMMU would make DMA from a device like Firewire impossible which has similar implications as described above. Once IOMMU's are on processors there will be no true DMA available anymore potentially making it more difficult for forensics teams to get an unobtrusive snapshot of a system.

3.6.5 External Hardware Timing

There is little a VMM can do to stop people outside the computer hooking up a piece of hardware that records the number of CPU cycles or how long it takes to do a single instruction and check that against a previously taken baseline. The fact is that while it is vulnerable to this attack, it's not very feasible to do on large scale, such as in an enterprise situation, or in most users homes and will not be covered past this point.

Chapter 4

Design and Implementation

This section presents the design and implementation of my own Type III malware the Flying Spaghetii Hypervisor (FSH), including the reasons chosen for such design decisions. FSH is both a rootkit and a protective hypervisor (a Supervisor) with both modes of operation discussed.

4.1 Conceptual Design

One of the goals of FSH is to 'Keep It Simple, Stupid' or KISS. As FSH is a Proof Of Concept (POC) it is not designed to have any advanced features that a real rootkit might have such as the ability to hide itself in memory. It only has a basic ability to virtualise the running OS, grant specific process's root privileges and if needed unload itself to allow another hypervisor to be loaded up.

To have the ability to run a number of privileged instructions needed to successfully virtualise a running computer it is necessary to be at kernel level or ring 0. As such FSH is designed as a Loadable Kernel Module (LKM) that is loaded into the kernel through the command 'insmod'. Another potential approach considered was using the rootkit technique of runtime kernel patching through /dev/kmem (or /dev/mem which is all memory including kernel memory) but instead opted for writing a LKM which is the simplest way to get in this privileged mode. In a real rootkit it would most like use the latter technique, as not allowing loadable kernel modules is prevalent in hardened systems.

FSH is only memory resident and does not have any infection techniques that would allow it to survive a reboot. This can be an advantage to malware as there are no files in the file-system that have to be hidden. In addition it becomes harder to get a copy of the rootkit once it has been installed as it is in control of the host OS. This coupled with anti-DMA techniques discussed previously makes it nigh impossible to examine even in a controlled circumstance if installed previously. During the development phase this was a boon as the development machine was a clean slate upon every reboot.

Most of the design of FSH is already laid out by Intel as it is mostly launching a virtual machine described in their software manuals, the fact that it is the running OS is irrelevant.

4.2 Implementation

FSH has gone through two iterations with the first being discarded because of too much complexity. It was originally based off Xen, an open source hypervisor, but because of the way that Xen had structured its code as a high-level production hypervisor FSH's complexity started to grow. The code was much more complex than it needed to be as a POC and as such the development on that code was scrapped. This was because Xen is designed to support multiple VM's and CPU's, neither of which FSH was going to and a large array of other features [51]. Basing the original code off Xen allowed me to understand in greater depth how larger hypervisors work but ultimately was not the route that should have been taken. The second, and current, iteration is based heavily off 'mobydefrags' VMM framework [52] designed for the Windows OS and small select parts of Xen.

Currently the code is only designed for a single core system to simplify development as this means there is no chance of the other core doing any actions during particularly sensitive times when the hypervisor thinks that it is atomic. It has been tested on Linux kernel 2.6.22. This is especially important in keeping the state of the virtualised OS in the same state between

VM Exits/Entries and the initial setup of the VMCS. Windows, Mac OSX and Linux have the option to configure the number of running cores on the fly.

FSH is quite simple in that it only had two logical parts; the setup phase and the VMM handler code.

4.2.1 VM Setup

The initial step is to allocate a number of areas of memory for the VMXON and VMCS regions. A VMCS is needed for each VM while the VMXON region is required for the VMM to run. The VMCS region is created in non-pageable memory by the VMM of the size reported by the MSR IA32_VMX_BASIC. The first 32 bits of the VMCS is then initialised to the VMCS revision identifier. From this point FSH can enter VMX root operation by executing a VMXON with the address of the VMXON region.

The VMCS region is cleared with the VMCLEAR instruction supplied with the guest's VMCS address. This initialises the VMCS region in memory and sets the launch state of the VMCS to 'clear'. Before issuing any VMWRITES a VMPTRLD instruction is used to set the active VMCS to our VMCS.

At this stage the VMM can start to fill the VMCS structure with data gleamed from the running operating system that will control the execution of the VM once it is running. A short list of items needed to be loaded into the VMCS is

- Control registers (CR0, CR3 and CR4)
- Selector fields for the segment registers (CS, SS, DS, ES, FS, GS and TR)
- Base address fields (for FS, GS, TR, GDTR and IDTR; RSP, RIP and the MSRs that control fast sytem calls)
- VM-exit control fields
- VM-entry control fields

- VM-execution control fields

Most values can be read directly from MSRS and written with little to no modification.

```
rdmsrl(MSR_IA32_SYSENTER_ESP, msrVar64);
printk("HOST_IA32_SYSENTER_ESP %lx, msrVar64);
__vmwrite(HOST_IA32_SYSENTER_ESP, msrVar64);
```

Reading from a MSR and writing to the VMCS

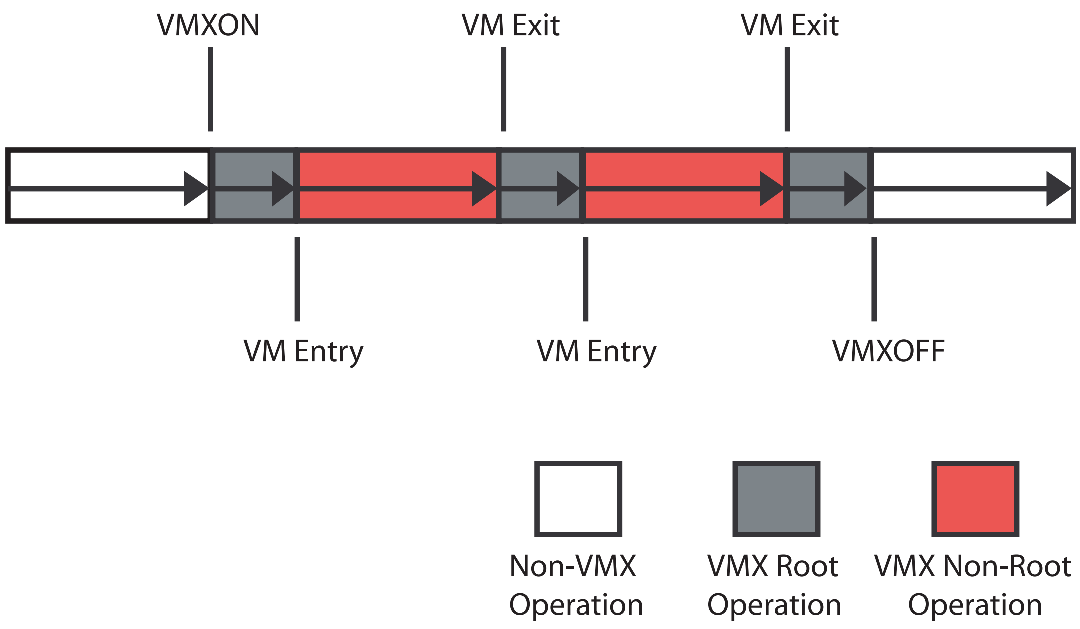
Inline assembly is used throughout the code to move values from and to specific registers and other instructions that are not accessible through normal C code.

```
asm("mov %%fs, %0\n\t" : "=r"(sr));
printk("HOST_FS_SELECTOR %x\n", sr);
__vmwrite(HOST_FS_SELECTOR, sr);
//Store Task Register
asm("str %0\n\t" : "=r"(sr));
printk("HOST_TR_SELECTOR %x\n", sr);
__vmwrite(HOST_TR_SELECTOR, sr);
```

Example inline assembly usage

Once this structure is filled and set up properly a VM launch can occur allowing the hardware to successfully virtualise the OS described within the VMCS.

The VMM entry code is invoked whenever the active VM does an action that the VMM is trapping or needs to know about. This allows the hypervisor to step in and tweak anything that it might need. FSH tries to trap on the least amount of actions possible to have the smallest impact on the running VM.



VMX Operation and VMX Transitions

The reason for the VMM to start executing (in the VMX root operation) is stored as a field inside the VMCS as the ‘exit reason’. If the read exit reason is valid the exit-qualification field provides more details on the VM-exit. Depending on this other information other information is read from the VMCS relevant to the exit reason and qualification.

Below is a table giving an outline of what FSH traps on and how it handles them.

- **VM* Instructions** - If FSH was going to support nested hypervisors such as multiple versions of itself running inside each other, this is where the command emulation would be handled. Currently this isn't supported so all the VMX commands except for VMCALL are ignored. These instructions effectively do nothing other than cause a VM exit and subsequent entry. These include VMCLEAR, VMLAUNCH, VMRESUME, VMPTRLD, VMPTRST, VMREAD, VMWRITE, VMXOFF and VMXON.

- **VMCALL** - See Supervisor Mode section.

- **INVD (Invalidate Internal Caches)** - The INVD instruction is trapped but not tweaked.
- **ReadMSR** - ReadMSR instruction is trapped but not tweaked.
- **WriteMSR** - WriteMSR instruction is trapped but not tweaked.
- **CPUID** - CPUID instruction is tweaked by entering in dummy values into EBX, ECX and EDX unless the EAX register is not 0x0. This is done to allow us put any value in there before doing a CPUID that will show us the non-tampered values.
- **MOV to/from CR3** - This is a more complex operation that has to be trapped as VMM has to find whether CR3 is being written to or read from and into which register the value should be returned to. The final CR3 value is not changed.

Finally after the reason for the VM-exit is handled, interrupts are turned back on through an 'STI' instruction and the guest resumes execution. If there are any problems handling the exit, such as an unhandled VM-exit reason, VMX operation is turned off and the guest returns to normal operation non-virtualised.

Initially FSH was envisioned separate from the protective hypervisor or 'Supervisor' as they were both had opposite jobs and having them as separate spin offs from a shared code base was logical. During development it became clearer that this would not have to be the case because of that exact logical reason - they both did opposite jobs. It is enough to have a simple if-then-else statement in two parts of FSH to separate the distinct roles each play as the Supervisor ignores all requests for a root UID and FSH ignores all the VMX instructions. The differences in execution of the two are described here.

Rootkit Mode

One of the main features of a rootkit is to give root priveledges to a non-root account through a backdoor. There are a large number of different ways this could be accomplished such as manipulating /proc entries or scanning through memory to change a processes UID to 0. The simplest method to implement a backdoor in the kernel is through syscall hooking which is a tried and true technique. The syscall table has to be found in kernel memory, as it is not exported in 2.6.* kernels, and then patched to call the hooking function instead of the original. FSH hooks 'setuid' and when a magic UID is requested, such as 1337 in this case, it gives the process root privileges instead of the requested UID.

```
int
main(void) {
setuid(1337);
system("/bin/sh");
return 0;
}
```

Example userland program using the backdoor

Since this modifies the kernel in a basic way it would be simple to spot by most rootkit detection code as the syscall table should not be modified in normal operation. If this were a non-POC a stealthier technique could be used. One example of this is would be finding the process's task_info struct in memory and changing it's UID to 0.

Supervisor Mode

The Supervisor is meant to be the chief hypervisor allowing other hypervisors to be loaded up inside it. The supervisor only lets authorised software access to the VMX instructions through signed VMs or passwords. But emulating all the VMX instructions completely is beyond the scope of this code. FSH in this mode instead removes itself from the kernel and unvirtualises the OS allowing a new hypervisor to replace it. This is achieved by passing a password established at compile time to FSH through a VMCALL, with the password being loaded into the EBX register.

This would stop any other rootkit like FSH for working while FSH in supervisor mode is currently loaded.

```
int
main(void) {
asm( "mov $0x13371337, %ebx\n\t"
".byte 0x0f,0x01,0xc1" //vmcall
);
return 0;
}
```

Example userland program using VMCALL to trigger a backdoor with the magic number 0x13371337

Chapter 5

Discussion

This section contains a brief discourse on virtualisation security and issues.

5.1 Hypervisors in Operating Systems

I do not believe this is the direction that next generation malware and rootkits are going to take. While it can have a number of advantages over a traditional kernel rootkit it is not necessary to go to these lengths to still achieve around the same level of stealth. The current size of the Microsoft Windows, Linux and Mac OSX kernels means there a huge array of different places a rootkit can potentially hide. While it may be prudent to include the ability to ‘hyperjack’ a running OS within a normal kernel rootkit this will not be the norm for a long time as it isn’t really necessary. Examining the history of rootkits again shows rootkit authors took a number of years to start using techniques such as runtime kernel patching after they had been publicly known.

What is happening though is the proliferation of hypervisors included inside operating systems for not only server architectures but desktops as well. Sun Solaris has had Logical Domains, their hypervisor, since 2001. The Linux kernel has Kernel-based Virtual Machine (KVM) as of 2.6.20. Microsoft has released Windows Server 2008 RC0 to TAP customers [53], which features their Viridian hypervisor in a server role. Mac OSX has no plans to include any virtualisation built into it rather opting for 3rd party solutions such as Parallels [54]. The future of the hypervisor rootkits is, in my opinion, the hijacking of already running VMM's or infecting the loading scheme allowing a malicious hypervisor to be loaded instead of the correct one.

Hypervisors have been used for security in the past such as for the Xbox 360. Their hypervisor was designed to only allow code signed by Microsoft's private key allowing them control on what ran on their hardware. But their software was not without problems. In early 2007 an 'anonymous hacker' posted details for a privilege escalation vulnerability to hypervisor level in the Xbox 360. This coupled with another method to inject data into non-privileged memory areas allows arbitrary code to be run, such as another operating system, with the full privileges and hardware access.

While this vulnerability wasn't a security risk as such for an end user it did demonstrate that an attacker could subvert a hardened close source hypervisor in a difficult environment. In recent months there have been a number of high profile security bugs found in different virtualisation software, most notably VMware [55]. With hypervisors starting to be shipped in operating systems by default they too will come under scrutiny for security flaws similar to the Xbox 360 hack. A flaw could potentially allow an attacker to inject their own code into a running hypervisor or 'hop' to the host OS / other VM's running along side the compromised one. Exploits for a VMM such as VMWare might be even more worthwhile to an attacker than a kernel bug in the future.

> 'You are absolutely deluded, if not stupid, if you think that a worldwide collection of software engineers who can't write operating systems or applications without security holes, can then turn around and suddenly write virtualization layers without security holes.'

Theo de Raadt in an October 24th, 2007 message on the OpenBSD -misc mailing list.

5.2 Detection Issues

In the past year there has been a lot of research by different parties into seeing if an operating system can find out that it is actually inside a virtual machine. A number of new techniques have been developed, some of which are discussed earlier in this paper, which would make it very difficult for a hypervisor to continue to fool the operating system. At the end of the day the hypervisor must be taking some resources away from the host computer, no matter how small that might be, it is still having an impact which can be detected. This is no longer being argued by anyone. But one of Rutkowska's main points in her talk at Blackhat 07' was on the ubiquitous nature of virtualisation detecting that something is virtualised will soon be expected.

> 'As hardware virtualization technology gets more and more widespread, many machines will be running with virtualization mode enabled, no matter whether blue pilled or not. In that case blue pill-like malware doesn't need to cheat that virtualization is not enabled, as it's actually expected that virtualization is being used for some legitimate purposes. In that case using a 'blue pill detector', that in fact is just a generic virtualization detector is completely pointless.'

Joanna Rutkowska [56]

Displaying that there is virtualisation occuring does not immediately imply that there is a malicious hypervisor. This coupled with infecting an already running hypervisor makes for an extremely difficult detection arms race for the defenders. A number of the detection methods put forward by researchers have been 'hacks' and exploiting errata, neither of which are reliable in the long run and may even differ from different generations of the same processor. I share Rutkowska's opinion that the security industry should not be based on such unreliable techniques. If we have to use such things as this then the computer industry is in a bad state for the future.

5.3 Hypervisor Shims

When trying to detect a rootkit the winner of the battle is the one that can get in ‘lowest and first’. That is the closer the software is to the hardware, the harder it is to detect. This is because the detection software is still relying on parts of the operating system to give it information and tell the truth, which can be altered by malicious software. For example it is hard for a userland program to check the kernel as it is relying on infomation from the compromised kernel.

Having a small hypervisor shim, that would have only a small impact on the performance of the computer, is the easiest way to defend against another hypervisor rootkit. This is exactly what my Supervisor is. A simple Supervisor would only allow verified VMs or hypervisors to be loaded up in its place through either passwords or through code signing. The Supervisor could then be thoroughly checked for any vulnerabilities more effectively than it’s larger cousins because of its small size. Unfortunately no VMM has any support for this sort of protection and could potentially load up FSH without any complaints. This has not being tested though, but Blue Pill is reported to work within VirtualPC.

As mentioned earlier in this chapter most operating systems are shipping their own hypervisors built into the kernel itself. It is not infeasible for them to have a ‘shim’ mode where they would act as I described above. I believe it is an easy solution to malware getting in ‘lowest and first’.

Chapter 6

Conclusion

This thesis has addressed the issues around type III malware and virtualisation in general. I have presented the first public hypervisor rootkit/supervisor, the Flying Spaghetti Hypervisor, designed for the Linux kernel. The source code to which will be publicly accessible under the GPL.

In addition to this virtualisation techniques and prevention have been discussed with the idea of a pre-installed hypervisor shim.

Bibliography

[1] ‘Storm Worm‘, http://antivirus.about.com/od/virusdescriptions/p/storm.htm

[2] ‘’Blue Pill’ Prototype Creates 100% Undetectable Malware’, Ryan Naraine (2006), http://www.eweek.com/article2/0,1895,1983037,00.asp

[3] ‘Performance Modeling: Experimental Computer Science at its Best’, Peter J. Denning (1981), http://www.cs.gmu.edu/cne/pjd/PUBS/ecs.pdf

[4] ‘Phrack straddles the world of hackers’, http://72.14.253.104/search?q=cache:r_LAbVI0C8IJ:www.landfield.com/isn/mail-archive/1998/Sep/0091.html

[5] ‘CP/CMS’, http://en.wikipedia.org/wiki/CP/CMS

[6] ‘The Origin of the VM/370 Time-sharing System’, R. J. Creasy (1981), http://researchweb.watson.ibm.com/journal/rd/255/ibmrd2505M.pdf

[7] ‘Hypervisors’, http://en.wikipedia.org/wiki/Hypervisor

[8] ‘An Overview of Xen Virtualization’, Tim Abels, Puneet Dhawan and Balasubramanian Chandrasekaran (2005), http://www.dell.com/downloads/global/power/ps3q05-20050191-Abels.pdf

[9] ‘Virtual Machine Monitors’, http://cisr.nps.navy.mil/projects/vmm.html

[10] ‘Xen in action: Deploying multiple servers’, Justin Korelc and Ed Tittel (2005), http://searchenterpriselinux.techtarget.com/tip/0,289483,sid39_gci1183361,00.html

[11] 'Practical Malware Analysis', Kris Kendal and Chad McMillan (2007), https://www.blackhat.com/presentations/bh-dc-07/Kendall_McMillan/Presentation/bh-dc-07-Kendall_McMillan.pdf

[12] 'Virtualisation approach for testing', 'Seryph' (2007), http://seryph.wordpress.com/2007/02/16/virtualisation-approach-for-testing

[13] 'Processor Based Virtualisation, AMD64 Style Part 1', Alan Zeichick (2006), http://developer.amd.com/articles.jsp?id=14&num=1

[14] 'Intel Virtualization Architecture Overview', http://www.intel.com/technology/itj/2006/v10i3/1-hardware/5-architecture.htm

[15] 'VMs Create Potential Risks', Kelly J. Higgins (2006), http://www.darkreading.com/document.asp?doc_id=117908

[16] 'Make A Virtual Machine Your Safe Internet-Browsing Sandbox', John P. Cook, http://www.devx.com/vmspecialreport/Article/30377/1954?pf=true

[17] 'Attacks on Virtual Machine Emulators', Peter Ferrie, http://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf

[18] 'AMD/Intel War Rumbles On', Gordon Kelly (2005), http://www.trustedreviews.com/cpu-memory/news/2005/04/14/AMD-Intel-War-Rumbles-On/p1

[19] 'Recognizing and Recovering from Rootkit Attacks', David O'Brien (1996), http://www.cs.wright.edu/ pmateti/Courses/499/Fortification/obrien.html

[20] 'Linux Rootkit II login has been detected', http://xforce.iss.net/xforce/xfdb/14856

[21] 'Abuse of the Linux Kernel for Fun and Profit', 'halflife' (1997), http://www.phrack.org/archives/50/P50-05

[22] ‘Linux on-the-fly kernel patching without LKM’, ‘sd’ (2001), http://www.phrack.org/archives/58/p58-0x07

[23] ‘Analysis of the T0rn Rootkit’, Toby Miller (2000), http://www.securityfocus.com/infocus/1230

[24] ‘Declaring Open Season on Open Source’, Anonymous (2004), http://lists.immunitysec.com/pipermail/dailydave/2004-May/000603.html

[25] ‘Sony DRM uses Rootkit Techniques’, Paul F. Roberts (2005), http://www.eweek.com/article2/0,1895,1880543,00.asp

[26] ‘Sony, Rootkits and Digital Rights Management Gone Too Far’, Mark Russinovich (2005), http://blogs.technet.com/markrussinovich/archive/2005/10/31/sony-rootkits-and-digital-rights-management-gone-too-far.aspx

[27] ‘2005 Sony BMG CD copy protection scandal’, http://en.wikipedia.org/wiki/2005_Sony_BMG_CD_copy_protection_scandal

[28] ‘First Trojan using Sony DRM spotted’, John Leyden (2005), http://www.theregister.co.uk/2005/11/10/sony_drm_trojan/

[29] ‘Microsoft Releases Windows Malware Stats’, Brian Krebs (2006), http://blog.washingtonpost.com/securityfix/2006/06/microsoft _releases_malware_sta.html

[30] ‘Introducing Blue Pill’, Joanna Rutkowska (2006), http://theinvisiblethings.blogspot.com/2006/06/introducing-blue-pill.html

[31] ‘Faceoff: AMD vs. Joanna Rutkowska’, Ryan Naraine (2006), http://securitywatch.eweek.com/rootkits/faceoff_amd_vs_joanna_rutkowsk.html

[32] ‘Debunking Blue Pill myth’, virtualization.info and Anthony Liguori (2006), http://www.virtualization.info/2006/08/debunking-blue-pill-myth.html

[33] 'Vista Hacking Challenge Answered', http://it.slashdot.org/article.pl?sid=06/08/07/228227&tid=201][http://weblog.infoworld.com/yager/archives/2006/06/blue_pill_is_an.html

[34] 'Hardware Virtualisation Rootkits', Dino Dai Zovi (2006), http://www.theta44.org/software/HVM_Rootkits_ddz_bh-usa-06.pdf

[35] 'Understanding Stealth Malware', Joanna Rutkowska (2007), http://theinvisiblethings.blogspot.com/2007/04/understanding-stealth-malware.html

[36] 'Intrusion Detection with Tripwire', Barry O'Donovan (2004), http://linuxgazette.net/106/odonovan.html

[37] 'Malware Taxonomy', Joanna Rutkowska (2006), http://invisiblethings.org/papers/malware-taxonomy.pdf

[38] 'Runtime kernel kmem patching', Silvio Cesare (1998), http://vx.netlux.org/lib/vsc07.html

[39] 'Sub proc_root Quando Sumus (Advances in Kernel Hacking)', 'palmers' (2002), http://phrack.org/archives/58/p58-0x06

[40] 'Intel® 64 and IA-32 Architectures Software Developer's Manual Volume 3B: System Programming Guide', http://www.intel.com/design/processor/manuals/253669.pdf

[41] 'Understanding Stealth Malware', Joanna Rutkowska (2007), http://theinvisiblethings.blogspot.com/2007/04/understanding-stealth-malware.html

[42] 'Formal requirements for virtualizable third generation architectures', Gerald J. Popek and Robert P. Goldberg (1974), http://portal.acm.org/citation.cfm?id=361073&coll=GUIDE&dl=GUIDE&CFID=22221035&CFTOKEN=53362438&ret=1#Fulltext

[43] 'Virtualisation Technology for AMD architecture', Steve McDowell and Geoffery Strongin,

http://download.microsoft.com/download/9/8/f/98f3fe47-dfc3-4e74-92a3-088782200fe7/TWAR05014_WinHEC05.ppt

[44] 'Firewire - all your memory are belong to us', Michael Becher, Maximillian Dornseif and Christian N. Klein (2005), http://cansecwest.com/core05/2005-firewire-cansecwest.pdf

[45] 'Hit by a Bus: Physical access attacks with Firewire', Adam Boileau (2006), http://www.security-assessment.com/files/presentations/ab_firewire_rux2k6-final.pdf

[46] 'Rutkowska on Cheating Physical Memory Acquisition: Details', Thomas Ptacek (2007), http://www.matasano.com/log/712/rutkowska-on-cheating-physical-memory-acquisition-details/

[47] 'Cheating Hardware Based RAM Acquisition', Joanna Rutkowska (2007), http://invisiblethings.org/papers/cheating-hardware-memory-acquisition-updated.ppt

[48] 'IPS vs IDS', Ellen Messmer (2004), http://www.networkworld.com/weblogs/security/005784.html

[49] 'Don't tell Joanna, the virtualised rootkit is dead', Goldsmith, Nate and Ptacek (2007), presented at Blackhat 07'

[50] 'IOMMU', http://en.wikipedia.org/wiki/IOMMU

[51] 'XEN Source data sheet', http://www.citrixxenserver.com/Documents/xensourcev4_datash

[52] 'Mobeydefrags VMM framework sourcecode', https://www.rootkit.com/vault/mobydefrag/vmxcpu.rar

[53] 'Microsoft releases Viridian hypervisor along with Windows Server 2008 RC0', http://www.virtualization.info/2007/09/microsoft-releases-viridian-hypervisor.html

[54] 'Apple reiterates no interest in virtualization for Leopard', Katie Marsal (2006), http://www.appleinsider.com/articles/06/12/01/apple_reiterates_no_interest_in_virtualiz

[55] ‘VMware bugs highlight virtualization security risks‘, Robert McMillan (2007), http://www.infoworld.com/article/07/09/20/VMware-bugs-highlight-virtualization-security-risks_1.html

[56] ‘Virtualization detection vs Blue Pill‘, Rutkowska (2007) http://theinvisiblethings.blogspot.com/2007/08/virtualization-detection-vs-blue-pill.html