Protecting Applications from Highly Privileged Malware Using Bare-metal Hypervisor

Kurbanmagomed Mallachiev
Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russian Federation
mallachiev@ispras.ru

Nikolay Pakulin
Institute for System Programming
of the Russian Academy of Sciences
Moscow, Russian Federation
npak@ispras.ru

***Abstract*—The paper presents a work-in-progress project on construction of a security facility that protects trusted application from malware residing at any privilege level of an OS, including OS kernel. The approach is based on the Sevigator project that used KVM to protect applications running in QEMU. The presented project is a port of Sevigator to much smaller trusted computing base of a bare-metal hypervisor.**

Keywords—security, virtualization, confidentiality, hypervisor, protection, virtual machine monitor, Sevigator

I. INTRODUCTION

The purpose of the project is to develop a security facility, that protects data confidentiality on a computer connected to the Internet and managed by an untrusted operating system. We assume that malicious code can get unlimited access to all hardware and software system resources through vulnerability or backdoors in system software.

Modern widespread operating systems (such as Linux or Windows) are based on monolithic kernel, where all components of kernel have equal privileges. When malicious code penetrates OS kernel there is a risk of losing control over any OS resources including application in-memory data, confidential information in file storage, etc. Integrity and confidentiality of data transmitted over the network are also threatened, even in the case when cryptography is used.

There are several channels for malicious code to penetrate OS kernel. It could be vulnerability of the system applications and kernel vulnerabilities, and backdoors in the drivers. Also there is a risk of theft of private keys from companies, supplying software or hardware, to sign malicious code; as a result OS trusts the signature and installs such code in the kernel.

Multiple approaches to securing workstations were proposed, including new more secure operating systems, specific hardware extensions, new application architectures. Still those approaches require massive investments in new products and significant changes in the user experience.

The question is whether it is possible to protect unmodified applications that run under unmodified commodity OS like Windows or Linux on a commodity workstation with x86 CPU. Protection systems located in kernel, such as antivirus, firewall, intrusion detection, can themselves be attacked by privileged malicious code. Possible way of protection from those attacks is the transfer of protection to more privileged level.

The answer is “probably yes”: a prototype called Sevigator [3, 4, 5, 6] protects applications in Linux from malware and comprised kernel. It uses hardware-assisted virtualization [1] to secure operating memory of applications and control access to communication hardware (network interface card). It allows to launch OS under control of virtual machine monitor (VMM, also called hypervisor). Hypervisor is much smaller than OS, fully isolated from it, and has higher privilege than OS. Hardware virtualization is supported by most modern processors, which suggests the possibility of widespread use of security systems based on hypervisors

One of the first examples of the use of virtualization to protect against untrusted OS is Overshadow project of memory protection developed by researchers in Stanford and Princeton Universities, MIT и VMware, Inc [2]. This technology does not require modification of the operating system or application. All memory of running processes is encrypted when a context switches. So, if the operating system or another program tries to read data from the memory of the process, they will receive only encrypted data, while the trusted process, referring to own data, receives it in the original form. However this approach limits cases when trusted application needs to pass some data to other processes by means, for example, of shared memory. Also in this approach all data are encrypted, even those that require no protection, and that is overkill.

Another reliable way to prevent data leakage, under the assumption that malicious code in the OS kernel, is the physical isolation of the computer from the network connection. However in this case all legitimate applications, which require access to the network, would suffer.

Sevigator isolates untrusted OS from network, but keeps operability of trusted application. For them, and only for them, an access to network resources is granted. An important feature of this approach is that there is no need to recompile any applications or OS

Within Sevigator approach OS resides in a virtual machine, while protection system is located in type 2 (hosted) hypervisor. It provides facilities to isolate untrusted applications from network access; to prevent data leaks due to

code intrusion or memory attacks it controls memory integrity of the applications under protection. Description of security algorithms can be found in [3, 4, 5, 6]. Sevigator system is based on hypervisor KVM (Kernel-based Virtual Machine)

Hypervisor KVM is type 2 hypervisor. Type 2(hosted) hypervisors runs like a module inside the host OS kernel, which handles interrupts, provides an abstraction of hardware and management of computer resources. Virtual machines with guest OS run like application in the host OS. Implementation based on this hypervisor is relatively simple, since such hypervisor allows you to develop and test on the same machine without rebooting, provides an opportunity to use debuggers and monitoring tools to find errors.

However, in this case, the hypervisor reliability depends on the operating system, which runs the hypervisor; in the KVM case it is Linux. The OS architecture is based on the principle of a monolithic kernel, so the hypervisor is vulnerable to attack by drivers and models of devices in the OS kernel. These defects do not exist in the decision based on type 1 hypervisor.

Type 1 hypervisors (native hypervisor, bare-metal hypervisor) run directly on the host's hardware. This hypervisor contain microkernel for interrupt processing, memory management, input-output, etc. Bare-metal hypervisors run at a higher privilege level than the OS kernel. A guest operating-system runs on the same privilege level as in the absence of the hypervisor.

Building protection systems based on type 1 hypervisor requires considerably less trusted computing base, than in the case of type 2 hypervisor. In addition, the microkernel allows you to split device drivers, virtual machines and memory manager. Thereby compromising individual component will not lead to compromise the entire system.

In this paper we present adaptation of part of Sevigator's protection algorithms, implemented in the type 2 hypervisor KVM, for type 1 hypervisor. Functionality of isolation OS and untrusted applications from network was adapted; currently being adapted security algorithms protecting process address space from unauthorized modification through the mechanism of direct memory access

II. CHOISE OF HYPERVISOR

When designing an adaptation to the type 1 hypervisor the idea to develop a hypervisor from scratch was immediately rejected: the development of a hypervisor is a very laborious task. It was necessary to compare existing type 1 hypervisors for x86 and select one to adapt functionality in it.

There are several requirements to hypervisors:

1. Open source. It is the base requirement to implement security mechanisms in the hypervisor code.

2. Support AMD x86 architecture, because, when we start adaptation, Sevigator used AMD virtualization

3. The presence of a virtual machine monitor to create and manage virtual machines and support of arbitrary unmodified guest operating systems.

4. Support for multiple virtual machines. Sevigator architecture assumes that at least two virtual machines run simultaneously.

5. Virtualization of hardware resources to separate the hardware between multiple virtual machines.

6. Small source code to allow verification.

The following hypervisors were considered: BitVisor[7], NOVA[8], Xen[9], XtratuM[10]. All of them are distributed under open source licenses and don't require existence of a host operating system.

BitVisor is hypervisor and virtual machine monitor, designed to ensure security of computer systems. BitVisor provides encryption of network connections and data on disk. Ensuring confidentiality of network and disk data is transparent to the operating system. BitVisor designed to create minimal overhead on encryption and decryption of data. BitVisor distributed under an open source license.

Virtual machine monitor is integrated into the hypervisor and performed at the same privilege level as the hypervisor. BitVisor supports exactly one virtual machine - this is done in order to minimize the overhead on the interaction of the guest OS with the devices, primarily input and output devices. BitVisor intercepts access to certain devices (eg, SATA controller, ie, hard disk), while the rest of the devices OS accesses directly.

BitVisor was rejected because it does not support multiple virtual machines.

NOVA is a hypervisor, built on microkernel architecture. Microkernel is performed at the highest level of privileges, and the environment, including resource monitor, device drivers and monitors virtual machines run at lower privilege levels. Thanks to microkernel architecture NOVA has well isolated code: components communicate with each other via messages, and with the kernel through hypercalls, only microkernel is performed with the highest privilege level, this provides improved security system as a whole.

Strictly speaking, the abbreviation NOVA used to refer to NOVA microkernel. In addition to the kernel running guest operating systems requires additional components developed in the project NUL (NOVA UserLand). NUL includes a virtual machine monitor Vancouver, memory and hardware resources monitor Sigma, external devices' drivers. Further in the text of this paper we will refer to NOVA bundled with NUL environment as just NOVA.

Originally the microkernel was developed at the Dresden University of Technology, now the main development of the kernel is in the research center of Intel.

NOVA is developed in C++, distributed under open source license. Using of microkernel architecture allows for simultaneous execution of an arbitrary number of virtual machines that can run unmodified guest operating systems. NOVA supports virtualization devices: Vancouver provides to guest OS virtual devices, which are served in the NUL. NOVA currently provides limited support for direct access to the

computer hardware, and limited support through separation devices IOMMU.

Xen is a very popular virtualization platform, which is widely used to build cloud services.

Xen virtualization platform includes a hypervisor, virtual machine monitor for guest OS, dedicated virtual machine dom0 to work with devices and specialized drivers to access the device via the dom0. These drivers are called paravirtualized as they "know" that the OS is running under Xen and effectively interact with the hypervisor and dom0.

Xen hypervisor implements the minimum set of operations: management of RAM, processor status, real time clock, interrupt processing and control of DMA (IOMMU). All other functions, such as the implementation of virtual devices, create and delete virtual machines, moving VMs between servers in the cloud, etc. is implemented in a dedicated virtual machine dom0.

All functions related to ensuring network performance, disk drives, video cards emulation and other devices placed outside the hypervisor. Typically, the request handling devices consist of two parts. Driver in the guest operating system translates requests from the OS to program handler in dom0. To increase the security of the system servers, virtualize devices run as separate processes in OS dom0. Failure in such a program leads to a denial of only one virtual device in one VM and does not affect the work of other copies of the server.

Xen hypervisor supports virtualization even on platforms where there is no hardware virtualization. As a result, the hypervisor code is quite large - on the order more microkernel NOVA - and convoluted. In addition, Xen does not support running unmodified guest OS: it requires specialized drivers to run the OS under Xen supervision.

XtratuM is hypervisor to separate computer resources into multiple virtual real time machines. XtratuM hypervisor provides real-time guarantees for the service interruptions hypercall, memory operations. XtratuM provides mechanisms isolation of virtual machines, the minimum software interface to run real-time applications without the guest OS, the means of communication between VMs. Developers claim support of x86 architecture, but the official website of the project distributes documentation for LEON processor family only – specialized clones if SPARC architecture.

XtratuM hypervisor is supported by several real time operating systems. Guest OS requires paravirtualized drivers, XtratuM does not support execution of arbitrary unmodified guest OS.

On the basis of requirements to the hypervisor we selected NOVA as the platform for bare metal hypervisor with security functions ported from Sevigator hypervisor.

III. SEVIGATOR ARCHITECTURE

Among the applications running in the operating system, the protection system identifies several applications that are considered trusted. The specific mode of functioning is provided to these applications. All other applications are considered as untrusted, the security problem is to prevent the leakage or compromising of confidential data of trusted applications. In particular, trusted applications for the normal functioning may require access to the public network. This network connection in the absence of external control can be used by malicious code in the kernel of the operating system for the leakage of sensitive data. The task of the security is to prevent data leakage.

The solution is based on use of hardware virtualization technology, execution of an operating system and all software in the virtual machine, and implementation protection system in the body of a virtual machine monitor (hypervisor) [3]. The hypervisor provides simultaneous execution of two completely isolated from each other virtual machines (fig. 1). Both are running the same untrusted operating system. The first virtual machine, we will call it *private*, is the primary one. It is there where critical data resides, applications are executed (both trusted and untrusted), processing those data. When the private virtual machine starts hypervisor blocks access to the network interface. The operating system, which runs in the VM, believes that the network adapter is physically absent. Therefore, any attempt to establish a network connection from within the virtual machine and transmit the data to a remote computer will inevitably lead to error. Thus, the malicious code running on any hardware privileges inside the private virtual machine, even if it managed to gain access to critical data, will not be able to transfer them to the outer world.

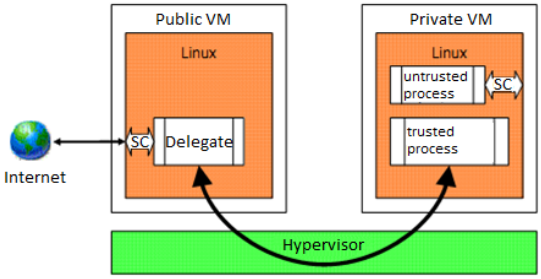

Fig. 1. Sevigator architecture

Network access for trusted applications is supplied by the second virtual machine. From here on we will refer to it as *public*. Public virtual machine has free access to the network interface, and any program in this virtual machine can interact with remote computers on the network. However, due to virtual machines isolation provided by the hypervisor the software in the public virtual machine (including Linux kernel) cannot gain access to data residing within the private virtual machine.

Network support for trusted processes is implemented through remote execution of required (limited) set of system calls to the public virtual machine. The hypervisor intercepts system calls invoked by a trusted process, analyzes the data and, when necessary, transmits them to the public virtual. System calls of other processes as well as the rest of the system calls of trusted processes are serviced locally in the private virtual machine.

The only information transmitted outside the private virtual machine is explicitly specified by a trusted process as parameters of system calls, and besides transfer the information outside of the virtual machine is serviced by trusted code

(hypervisor). Note that the remote execution of the system call is made transparent for a trusted process and an operating system for a virtual computing machine.

Trusted processes are executing under the control of an untrusted operating system. In-memory data of trusted applications are not encrypted, stored in clear (unencrypted) form, and protection system does not restrict the access (both system and user) to these data. Since untrusted components, including the kernel, do not have access to the network, they are not able to disclose sensitive information.

However, in untrusted OS environment it is necessary to take into account the risk of injecting code into trusted applications: malicious code in the operating system kernel can load into the address space of a trusted process necessary code, then pass control to it, and trusted process on its behalf will take all necessary actions for the delivery critical information to a remote computer, which is controlled by the attacker. To prevent these harmful effects security system protects context of a trusted process against unauthorized modification by any program in private virtual machine, including privileged.

IV. ADDAPTATION FROM KVM TO NOVA

So, as mentioned above, open source NOVA hypervisor has been selected as the hypervisor. At the moment it is being actively developed in the Intel Research laboratory.

By the arguments in favor of NOVA, above, may be added that the hypervisor much less than all popular hypervisors therefore potentially more secure. Also its kernel code has been verified [11]. On the Fig. 2 you can see a comparison of the sizes of popular hypervisors and NOVA.

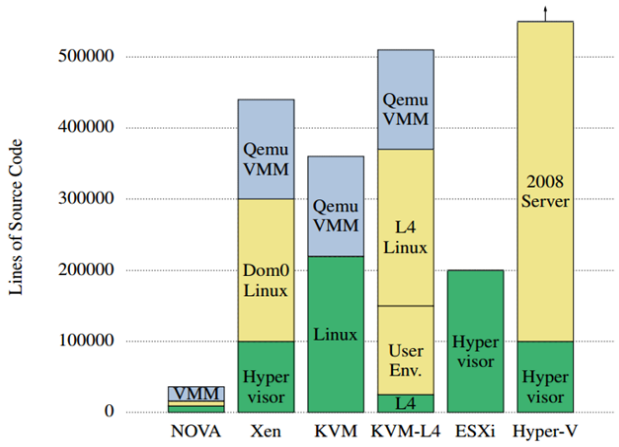

Fig. 2. comparison of the sizes of popular hypervisors

NOVA is built on microkernel architecture. Microkernel, which size is less than 10,000 lines of C ++, launches virtual machines and routes interrupts and system calls. In addition to the core NOVA includes the “Nova UserLand” NUL, which includes a virtual machine monitor (Vancouver), memory and hardware resources monitor (Sigma0) and drivers of external devices. The total size of NOVA and NUL is less than 50 thousand lines.

Sevigator was built on top of KVM kernel module that provides hardware virtualization in QEMU environment. Intercepting calls of the virtual machines, it entrusts to processing of many functions host OS kernel, under which it is launched. Moreover KVM is included as a module in the host OS kernel, and therefore has the highest level of privilege. There is no division of privilege levels in KVM. The size of KVM is over 300 thousand lines of code. In the prototype Sevigator based on KVM interaction was through the address space and virtual interrupt of pci device, which emulated by qemu.

During transferring Sevigator to NOVA platform most of Sevigator’s algorithms have been implemented in components of NUL.

This section briefly describes the changes made to the NOVA and NUL, to implement algorithms from Sevigator.

After transfer Sevigator to NOVA a remote service system call will appear as diagram in Fig. 3

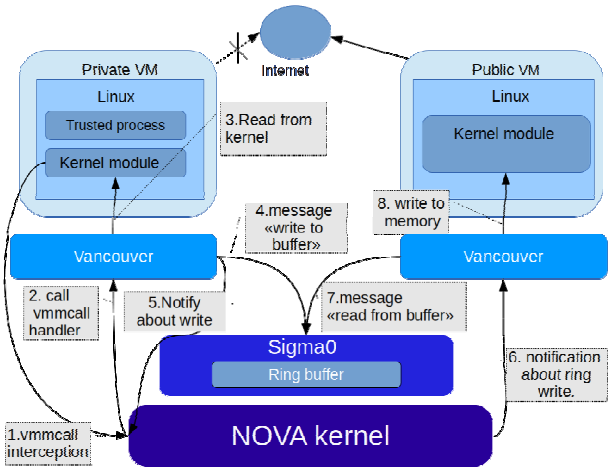

Fig. 3. Architecture of Sevigator in NOVA.

At system start the virtual machine monitor, controlling user (private) VM, is configured so that it does not have a network card emulator. That is, when the OS at boot enumerates PCI bus, it lacks the class of device "network device". So, operating system has no access to the network, and all untrusted components, including the kernel, cannot transmit and receive data from the outside.

However, when a trusted application performs a system call related to network access, the system call is intercepted by the hypervisor. System call parameters are copied into an internal ring buffer, which is not accessible by guest OS in virtual machines. VM monitor service receives notification of a new data, reads data from the ring buffer and transmits to the public VM for handling. Transferring the data from the public VM to the private VM is implemented similarly.

System calls are intercepted by the hypervisor, but the processing parameters of the call, the data transmission between the VM, return control to the VM implemented in virtual machine monitor. This is done to improve the security of the system: if the query processing network has vulnerability then only one VM has been compromised, all other processes

are executed as a microkernel's process and isolated from each other.

Functions that implement algorithms of Sevigator were added to the virtual machine manager Vancouver, which runs as an application process in NOVA. This protects hypervisor itself from compromising by the security system. Even if malicious code can take control over Sevigator, in a single VM, it will not be able to subdue other virtual machines.

V. MODIFICATION OF NOVA AND NUL

A number of changes to the core components of NOVA and NUL was required to properly implement the Sevigator.

1. Sevigator's components, working in the OS kernel and user space (trusted applications launcher), interact with the hypervisor through the vmmcall instruction. It was necessary to implement handlers of vmmcall in NUL. For this, interception vmmcall instruction has been activated in the NOVA's kernel and implemented handler for instruction in virtual machines manager Vancouver. On AMD platform instructions interception implemented by setting to 1 the bit, responsible for this instruction in a control block of the virtual machine (VMCB). The analysis showed that by the default NOVA set to 1 bits responsible for intercept vmload, vmsave, clgi, skinit. In order not to break code integrity the special flag has been added and the specific mask has been changed.

2. To trace action of the operating system, primarily context switching, Sevigator intercepts specific x86 instructions. These are system calls, software interrupts, returns from the interrupt, and others. To support it special handlers, embedded in KVM code, are implemented in the prototype. When moved to NOVA it is necessary to implement these instructions interception means of Sevigator. While intercepting instruction, you must be able to emulate, as the interception occurs before running the instruction [12]. Currently Vancouver does not have complete emulation necessary instructions (iret, int n, syscall, sysexit), but the implementation of the emulation is planned by the developers. Since emulation instructions was not included in the plan of this work, we decided to temporarily inserted into the core of the guest virtual machine vmmcall calls immediately after a system call and before returning from the system call. This decision violates the initial concept of protection without making changes to the OS, but it is temporary and it will be replaced as soon as Vancouver developers implement a full emulation of instructions.

3. To implement memory protection, we need to be able to access to the virtual machine memory. In Vancouver there is a subsystem, responsible for emulation instructions that work with memory. It has no external interfaces through which the other subsystem (in particular Sevigator) can work with memory. To access the virtual machine's memory by virtual address have been added a special message for reading and writing to the memory of virtual machines. Recall that the interaction between the components of NUL is given by messaging. Accordingly, new types of message for reading and writing were added, as well as handlers of these messages to the subsystem operating with the memory.

4. To implement the algorithms of the Sevigator's network subsystem should be developed mechanism and implemented software interface of communication subsystem between virtual machines NUL. We implemented a ring buffer between managers of virtual machines, and notification of arrival a new data signaled through interruption to the virtual machine. In order to do this, special messages (OP_SVG_WRITE, OP_SVG_READ) handlers were added to operation memory monitor. And for notification, during initialization both Vancouvers initialize its own portals, and send its addresses to the NOVA kernel; NOVA has handler of the specific system call, which sends a signal to the desired portal. After storing data to ring buffer Vancouver needs to send a special system call to the kernel NOVA, which will notify the other Vancouver about recording.

5. Sevigator assumes access to the network card. In NUL possibility of direct access to the pci-devices are implemented only partially. NOVA has a feature to be the network provider for the virtual machine, but to use it you need to write a driver for the network card. We based our driver on the driver, which was included to the NUL to work with a network card family ne2000 that emulated by qemu. This driver has been slightly modified to work with a specific physical card of the ne2000 family. The changes are minimal – we added a missing flag and reduced the size of the buffer.

VI. CONCLUSION

In this paper we presented an implementation of privacy protection of network connection of trusted Linux processes in bare-metal hypervisor. We have conducted a study of open source bare-metal hypervisors: we have formulated certain requirements to the type 1 hypervisor, analyzed the available open hypervisors and by comparing have selected hypervisor NOVA.

Porting Sevigator from KVM to NOVA is in progress, we have partly ported confidentiality protecting mechanisms. We have conducted a study of architecture and mechanisms of components interaction within NOVA. We have identified and implemented changes in NOVA, necessary for the functioning of Sevigator. This allowed:

- to port mechanisms of remote service of system calls;
- to port mechanisms to control trusted application context;
- to port mechanisms to control memory integrity of trusted applications;
- implement a prototype of network boot mechanism.

Thus, the outcome of the project is a prototype of security system that protects the integrity and confidentiality of information stored and processed on a computer connected to the network and controlled by potentially malicious operating system.

Developed prototype provides protection of information systems from the injection of malicious code by modifying the memory, images of executable files or script files, protect the integrity and confidentiality of data while transiting over the network.

[1] Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, and 3C: System Programming Guide [Online]. Available: http://www.intel.com/content/dam/www/public/us/en/documents/manuals/64-ia-32-architectures-software-developer-system-programming-manual-325384.pdf)

[2] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A.Waldspurger, D. Boneh, J. Dwoskin, and D. R. Ports, "Overshadow:a virtualization-based approach to retrofitting protection in commodityoperating systems," SIGOPS Oper. Syst. Rev., vol. 42, pp. 2–13, March 2008. [Online]. Available: http://doi.acm.org/10.1145/1353535.1346284

[3] Burdonov I., Kosachev A., Iakovenko P. Virtualization-based separation of privilege: working with sensitive data in untrusted environment. //1st Eurosys Workshop on Virtualization Technology for Dependable Systems, New York, NY, USA, ACM. 2009. P. 1-6.

[4] D.V. Silakov. Using Hardware-assisted Virtualization in the Information Security Area. pp. 25-36. Proceedings of the Institute for System Programming of RAS, volume 20, 2011. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)

[5] P. Iakovenko. Transparent mechanism for remote system call execution. pp. 221-242. Proceedings of the Institute for System Programming of RAS, volume 18, 2010. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)

[6] P. Iakovenko. Ensuring confidentiality of information processed on a computer with a network connection. Information security problems. Computer Systems. №4. 2009. pp. 23-41. (In russian)

[7] Takahiro Shinagawa, Hideki Eiraku, Kouichi Tanimoto, Kazumasa Omote, Shoichi Hasegawa, Takashi Horie, Manabu Hirano, Kenichi Kourai, Yoshihiro Oyama, Eiji Kawai, Kenji Kono, Shigeru Chiba, Yasushi Shinjo, and Kazuhiko Kato. 2009. BitVisor: a thin hypervisor for enforcing i/o device security. In Proceedings of the 2009 ACM SIGPLAN/SIGOPS international conference on Virtual execution environments (VEE '09). ACM, New York, NY, USA, 121-130.

[8] Udo Steinberg and Bernhard Kauer. 2010. NOVA: a microhypervisor-based secure virtualization architecture. In Proceedings of the 5th European conference on Computer systems (EuroSys '10). ACM, New York, NY, USA, 209-222.

[9] Chris Takemura and Luke S. Crawford. The Book of Xen. No Starch Press. October 2009, 312 pp. ISBN-13 978-1-59327-186-2,

[10] A. Crespo, I. Ripoll, and M. Masmano. 2010. Partitioned Embedded Architecture Based on Hypervisor: The XtratuM Approach. In Proceedings of the 2010 European Dependable Computing Conference (EDCC '10). IEEE Computer Society, Washington, DC, USA

[11] Nova Micro-Hypervisor Verification http://os.inf.tu-dresden.de/papers_ps/tr-tews-vnova-2008.pdf

[12] AMD64 Architecture Programmer's Manual Volume 2: System Programming http://developer.amd.com/wordpress/media/2012/10/24593_APM_v21.pdf