Remote Service of System Calls in Microkernel Hypervisor

[1]Kurbanmagomed Mallachiev <mallachiev@ispras.ru>,
[2]Nikolay Pakulin <npak@ispras.ru>
[1]Lomonosov Moscow State University,
Faculty of Computational Mathematics and Cybernetics,
119991, Leninskie Gory, 1, Moscow, Russia
[2] Institute for System Programming of the Russian Academy of Sciences,
109004, A. Solzhenitsina, 25, Moscow, Russia

Abstract. This paper presents further development of Sevigator hypervisor-based security system. Original design of Sevigator confines users' applications in a separate virtual machine that has no network interfaces. For trusted applications Sevigator intercepts network-related system calls and routes them to the dedicated virtual machine that services those calls. This design allows Sevigator protect networking from malicious applications including high-level intruders residing in the kernel.

Modern microkernel-based hypervisors opened the door to redesign of Sevigator. Those hypervisors are small operating systems by nature, where management of virtual machines as well as most of hardware operations are isolated in processes with low priority level. Compromising such a process does not result in compromising the whole hypervisor. In this paper we present an experimental design of Sevigator based on NOVA hypervisor where system calls of trusted applications are serviced by a dedicated process in the hypervisor rather than a separate VM. The experiment shows about 25% performance gain due to reduced number of context switches.

Keywords: virtualization, hypervisor, security, microkernel

DOI: 10.15514/ISPRAS-2015-27(3)-18

For citation: Mallachiev Kurbanmagomed, Pakulin Nikolay. Remote Service of System Calls in Microkernel Hypervisor. Trudy ISP RAN/Proc. ISP RAS, vol. 27, issue 3, 2015, pp. 267-278. DOI: 10.15514/ISPRAS-2015-27(3)-18.

1. Introduction

The main 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 vulnerabilities or backdoors in system software.

Today popular modern operating systems (such as Linux or Windows) are based on monolithic kernel, where all components of kernel have equally high privileges. In this case if malicious code penetrates OS kernel, then 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.

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] protects applications in Linux from malware and comprised kernel. It uses hardware-assisted virtualization [1, 2] 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, making the widespread use of security systems based on hypervisors possible.

Sevigator provides isolation of 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 modify or recompile any applications or OS.

Within Sevigator approach OS resides in a virtual machine, while protection system is located in 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. The hypervisor provides simultaneous execution of two completely isolated from each other virtual machines. The first one called *user* is the primary one, user interacts with it, and it believes that network adapter is physically absent. The second VM called *service* is service system which has unlimited access to network. Network support for trusted processes in user machine is provided by hypervisor through remote execution of required (limited) set of system calls in the service virtual machine. Full description of security algorithms can be found in [3, 4, 5].

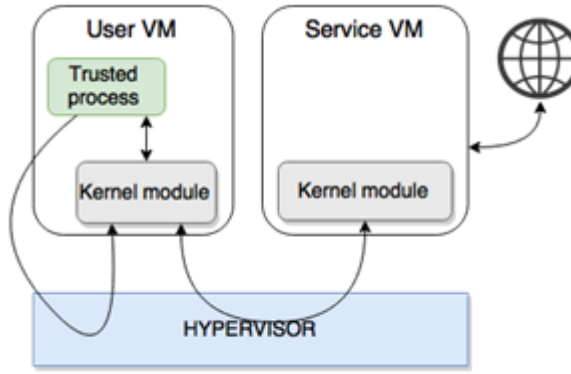

Fig 1. Sevigator architecture

We refer to this scheme as *remote servicing of system calls* since the hypervisor intercepts parameters of a system call in the user VM and transfer them to the service VM, where the actual code is executed.

The scheme with two VMs was motivated by the following considerations: isolation networking operations from user machine and minimization the risk of hypervisor compromise in the case of compromised network component. Isolation makes network access possible only for trusted application. Execution within service VM means that compromise of the VM will not lead to compromise of hypervisor kernel.

Sevigator system originally was based on hypervisor KVM (Kernel-based Virtual Machine), and using the second VM was the only possible solution to satisfy the constraints. Later Sevigator without changes of its architecture was ported to NOVA microkernel hypervisor [6].

Our work shows that using hypervisor based on the microkernel architecture allows us to replace the second virtual machine with a process in hypervisor with the same functionality. This is possible because microkernel isolates processes and executes them at lower privilege level than the microkernel. And this change significantly reduces overhead of having dedicated OS only for remote execution of service calls.

2. Hypervisors Overview

There is a lot of hypervisors and they use different ideas. We chose NOVA [7] to port Sevigator because it was the only one that satisfied own requirements for original Sevigator design (requirements and hypervisor comparison can be found in [6]). And when we ported Sevigator, NOVA architecture gave us idea how we can redesign Sevigator to reduce overhead but keeping security.

With new design of Sevigator, where dedicated process is responsible for servicing system call, we again looked if it can be implemented in different hypervisor besides NOVA. The following hypervisors were considered: BitVisor[8], SecVisor[9], Xen[10], Qubes OS [11]. 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 (VMM), 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 doesn't separate VMM and kernel of the hypervisor, so performed at the same privilege level. 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 based on parapass-through architecture: hypervisor intercepted memory access and I/O access, and pass-through anything else. Bitvisor intercept accesses to protect hypervisors from the guest OS, and enforce security functionalities. Bitvisor cannot execute processes at lower privilege level. Therefore Bitvisor didn’t satisfy the requirements.

SecVisoris a very small hypervisor (about 10 times smaller than NOVA) which goal is protecting OS kernel against an attacker who controls everything but the CPU, the memory controller, and system memory chips.

SecVisor provides a lifetime guarantee of the integrity of the code executing with kernel privilege. In other words, SecVisor prevents an attacker from either modifying existing code in a kernel or from executing injected code with kernel privilege, over the lifetime of the system. SecVisor ensures that only code approved by the user can execute with kernel privilege. SecVisor also executes all its parts at the same privilege level.

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, creation and deletion virtual machines, moving VMs between servers in the cloud, etc. is implemented in a dedicated virtual machine dom0.

All functions related to network, disk drives, video cards emulation and other devices are 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 architecture requires using dedicated virtual machine for servicing network-releated system calls and this is a big overhead. Furthermore, Xen codebase is large and nearly impossible for thorough security analisys.

Qubes OS is a hypevisor based on Xen. Qubes implements a security–by-isolation approach. In Qubes, the isolation is provided in two dimensions: hardware (separated network domain, storage domain, GUI) and software (domain with different levels of trust e.g. work domain – most trusted, shopping domain, random domain – less trusted). Domains are separated by executing within different virtual machines.

3. Original Sevigator Design

3.1 General Architecture

Among the applications running in the OS, the protection system identifies several applications that are considered as trusted. All others applications are considered as untrusted. The security problem is to prevent the leakage or compromising of confidential data of trusted applications. Trusted applications for the normal functioning may require access to the public network. This network connection can be used by malicious code in the OS kernel for the leakage of sensitive data.

The solution is based on use of hardware virtualization technology, execution of an OS in the virtual machine (VM), 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 OS. The first VM, we will call it *user*, is the primary one. It is there where critical data resides and applications (both trusted and untrusted) are executed processing those data. Hypervisor blocks access to the network interface for user VM and its guest OS believes that the network adapter is physically absent. Thus, even if malicious code managed to gain access to critical data, it will not be able to transfer them to the outer world.

Network access for trusted applications is supplied by the second VM called *service*. It has free access to the network. However, due to VMs isolation provided by the hypervisor the software in the service VM (including OS kernel) cannot gain access to data residing within the user VM.

Network support for trusted processes is implemented through remote servicing of required set of system calls in the service VM. The hypervisor intercepts network-related system calls invoked by a trusted process, analyzes the data and, when necessary, transmits them to the service VM. Note that the remote service of the system call is made transparent for a trusted process and an OS.

3.2 NOVA based architecture

NOVA is a microkernel for hypervisor. NOVA itself is only a kernel, for running virtual machines you should use one of the environments, built atop of it: NUL, NRE or Genode. We use NUL because NRE still misses some NUL features, and Genode is much larger.

Because of microkernel design, only the NOVA kernel runs with the highest priority and every process of NUL is executed as user space process with priority level CPL3 (lowest on Intel IA-32 architecture).

NUL is an experimental operational environment and it is still work in progress. It contains a number of simplified components, e.g. direct access to host PCI devices works unstable. As a result VMM (Virtual Machine Monitor) has to emulate hardware devices for the guest virtual machine. And if the emulated model needs access to a host device, than a driver in NUL is required for that device. For networking NUL provides a small number of drivers, most notable is the classic NE2000 network card. For our experiment we used NE2000-compatible network card RTL8029AS, for which NUL has a driver.

The port of Sevigator architecture to NOVA hypervisor uses two virtual machines [13] to service network-related system calls of trusted users' applications. As an example Fig. 2 shows how servicing *send* system call works.

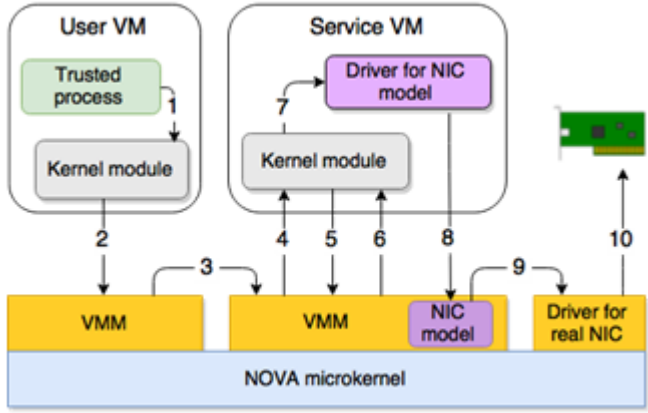

Fig 2. Path of send message in original design Sevigator

Yellow colored boxes are processes in NOVA. Interaction with and between processes always imply calling NOVA kernel, but for simplicity we don't show them on the figure.

When trusted process executes *send* system call the Sevigator module in OS kernel intercepts it (1), forms special fixed size message and free size vault and executes the hypercall (2). VMM passes (3) the message and the vault to another VMM. This VMM sends (4) the message to service VM kernel module. Module finds vault size

in message, allocates memory, asks (5) for vault and receives (6) it. Module forms a new message and sends it to Linux kernel, which calls (7) network driver for network card emulated by VMM. The driver sends (8) bytes to the network card model, which passes (9) them to driver of the actual card. And finally the driver in the hypervisor sends bytes to the network card.

As we can see the path that passes network messages is really long. In the next chapter we will show how to achieve a shorter pass.

4. New Sevigator Design

Microkernel based hypervisor allows us to redesign Sevigator. Those hypervisors have well isolated parts. Only a small kernel has highest priority level. Most of hardware operations as well as management of virtual machines are isolated in processes with low priority level.

The idea of the redesign is to move servicing system calls of trusted applications to hypervisor applications. Having dedicated processes in hypervisor we keep all pluses of using dedicated virtual machine such as isolation of servicing system calls in code and securing the risk of compromise the system by reduction of priority level. It means that compromising such a code doesn't mean compromising the whole hypervisor. But redesigning gives more: it reduces trusted code base from millions of lines of code (LoC) for service VM to tens of thousands LoC for dedicated applications in hypervisor. And also we reduce overhead of context switching: redesigned system doesn't need at least context switching between VMM and service VM; so we increase performance of the whole system.

In our paper we present a proof of concept of the new approach to servicing system calls of trusted applications in dedicated environment.

We selected networking system calls for study. Fig. 3 presents the idea: networked system calls are serviced in the dedicated process over NOVA microkernel. The application is based on popular embedded TCP/IP stack called lwIP[12]. The application is a wrapper around lwIP that parses the parameters of remote system calls and invokes corresponding lwIP operations. In the following text we will refer to this application as "lwIP".

Fig.3 shows servicing of send message in redesigned system. Here we will only discuss difference of redesigned system. Steps (1) and (2) are the same as in the original design. VMM sends (3) message and vault to LwIP process, which analyses the message, understands what system call was called, and forms a packet, that will be sent (4) to driver. Driver sends bytes to the real network card.

We can see that in the new design the path is much shorter, and one can expect that the new design should work faster. We present the performance study in the in the next section. In order to support the concept of socket used by trusted application we implemented a small glue layer over lwIP. The prototype implementation supports socket *create* and *close*, socket *bind* and *connect*, *send* and *recv* for TCP and UDP. Raw sockets (e.g. for ICMP messages) are not supported yet.

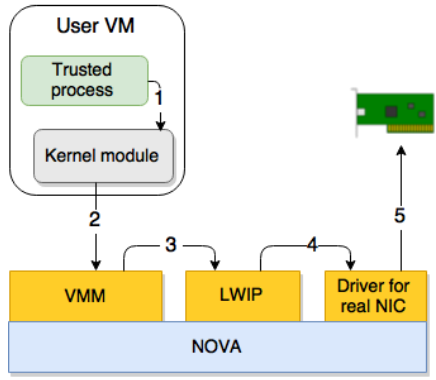

Fig 3. Path of send message in redesigned Sevigator

5. Performance

We conducted an experiment to measure network performance of the redesigned system. During experiment we compared performance of the original design with two VMs, and the new design with the dedicated process. As the reference point we used native Linux running on hardware without hypervisor and ran hypervisor with pure lwIP application without VMM.

All measurements were performed on the same machine with AMD Phenom II x4 980 3.7 GHz CPU, 16 GB RAM. As network card we used once popular RTL8029AS card. It is ne2000 compatible and is one of the few cards supported by NOVA/NUL. The card is 10Mbit/s. We use this old card because other cards supported by NOVA turned out to be much harder to find.

For testing, we run test application in Linux, which executes 1000 times *sendto* system call, sending UDP packets to the network. We were sending short 60 bytes message. The destination workstation received the packets, identified lost packets and measured time between the first and the last packets. We did not measure time at the guest virtual machine because return from *sendto* call does not mean that the corresponding packet was actually sent.

Fig.4 shows the test performance difference between original and new architectures and pure Linux.

The experiments showed that replacing the virtual machine with a dedicated application increased performance by 26%. The overhead compared to the native Linux execution was reduced from almost 100% to 29%.

Comparing with pure lwIP case shows that current overhead for transfer system call in lwIP is only 1.4 µs. For 10 Mbit/s network this is insensitive. The bottleneck of current realization is lwIP and NE2000 driver. The NE2000 driver in NOVA is far from perfection and careful queuing of pending packets may reduce the total overhead even more.

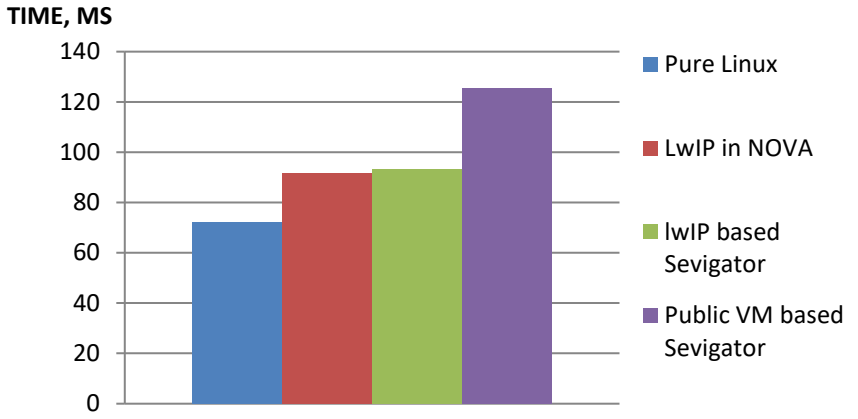

Fig 4. Time for sending 1000 UDP packets

Servicing of system calls in an application compared to a dedicated VM simplifies the flow control. Removing the second VM resulted in omitting:

interrupt injection in the service VM (required to notify the VM that there are packets pending);

VM exit to pass frames from service OS to NIC model in the VMM;

IPC calls between VMM and NIC driver in the hypervisor.

Another important gain is significant reduction of the trusted code base required for servicing network-related system calls. The design with two virtual machines implied that we have to trust the whole Linux kernel, i.e. millions lines of code due to the monolithic nature of that kernel. When system calls are serviced by the lwIP application, the trusted computing base shrinks to about 70,000 LoC, the size of lwIP.

6. Future Work

In future we want to develop NUL drivers for modern network cards and make experiments on them. Also because NOVA UserLand was made as a test project and is not fully stable for now, we have encountered problems with memory management, and have errors while working with big packets. We want to find the causes the revealed problems and fix it.

Finally, we will port guest modules to modern Linux kernel and see if there are any changes in performance.

7. Conclusion

Our work shows that using microkernel-based hypervisors opens new perspectives and facilitates new approach to servicing OS system calls in hypervisor.

Using microkernel hypervisor allow us to redesign system by moving system call servicing in hypervisor application. Those applications are executed as processes with low priority, so compromising of an application doesn't lead to compromising of the whole hypervisor.

We were able to move servicing of network-related system calls to such a process. It significantly reduces overhead for servicing network-assisted system calls and speeds up execution: new design makes network connection 30% faster. Furthermore, it reduced trusted code base by two orders of magnitude, and this is very important for security system, because it makes audit or verification of system simpler.

References

[1]. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, and 3C: System Programming Guide.

[2]. AMD64 Architecture Programmer's Manual Volume 2: System Programming PDF, 2011

[3]. I. Burdonov, A. Kosachev, P. Iakovenko 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. Silakov. Using Hardware-assisted Virtualization in the Information Security Area. pp. 25-36. Proceedings of ISP 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 ISP RAS, volume 18, 2010. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)

[6]. K. Mallachiev, N. Pakulin. Protecting Applications from Highly Privileged Malware Using Bare-metal Hypervisor. DOI: 10.15514/SYRCOSE-2014-8-10.

[7]. U. Steinberg and B. 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.

[8]. T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. 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.

[9]. A. Seshadri, M., Ning Qu, and A. Perrig. 2007. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes.*SIGOPS Oper. Syst. Rev.*41, 6, 335-350. DOI=10.1145/1323293.1294294 C. Takemura and L. S. Crawford. The Book of Xen. 2009, 312 pp. ISBN-13 978-1-59327-186-2,

[10]. J. Rutkowska. Software compartmentalization vs. physical separation. Invisible Things Lab, 2014
http://www.invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf

[11]. A. Dunkels lwIP, a small independent implementation of the TCP/IP protocol suite. http://www.nongnu.org/lwip

Удаленное обслуживание системных вызовов в микроядерном гипервизоре

[1]К. Маллачиев <mallachiev@ispras.ru> ,
[2]Н. Пакулин <npak@ispras.ru>
[1]Московский государственный университет имени М.В.Ломоносова, факультет вычислительной математики и кибернетики
119991, Россия, г. Москва, Ленинские горы, д. 1
[2]Институт Системного Программирования РАН,
109004, Россия, г. Москва, ул. А. Солженицына, д. 25.

Аннотация. В данной работе описывается дальнейшая разработка системы защиты Sevigator, использующей аппаратную виртуализацию. Изначальное устройство Sevigator состоит в исполнении пользовательских приложений в отдельной виртуальной машине, у которой отсутствует сетевой интерфейс. Для доверенных приложений Sevigator перехватает системные вызовы, связанные с операциями с сетью, и перенаправляет их на обслуживание в выделенную виртуальную машину. Такое устройство позволяет системе Sevigator защищать сетевое взаимодействие от вредоносных приложений, включая злонамеренный код на самом высоком уровне привилегий в ядре ОС. Использование современных гипервизоров, построенных по микроядерной архитектуре, позволяет изменить архитектуру системы Sevigator. Такие гипервизоры по своей природе являются маленькой операционной системой, в которой большинство аппаратных операций и управление виртуальными машинами изолированно в процессы с низким уровнем приоритета. Компрометация таких процессов не приведет к компрометации всего гипервизора.

В данной работе мы предоставляем экспериментальную архитектуру Sevigator-а, основанную на гипервизоре NOVA, в рамках которой системные вызовы доверенных приложений обрабатываются в отдельном процессе в гипервизоре, а не в отдельной виртуальной машине. Этот эксперимент показал 25% прирост производительности при уменьшении количества переключений контекстов.

Ключевые слова: виртуализация, гипервизор, безопасность, микроядро

DOI: 10.15514/ISPRAS-2015-27(3)-18

Для цитирования: Маллачиев К., Пакулин Н. Удаленное обслуживание системных вызовов в микроядерном гипервизоре. Труды ИСП РАН, том 27, вып. 3, 2015 г., стр. 87-96 (на английском языке). DOI: 10.15514/ISPRAS-2015-27(3)-18.

Список литературы

[1]. Intel® 64 and IA-32 Architectures Software Developer's Manual Combined Volumes 3A, 3B, and 3C: System Programming Guide.

[2]. AMD64 Architecture Programmer’s Manual Volume 2: System Programming PDF, 2011

[3]. I. Burdonov, A. Kosachev, P. Iakovenko 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]. Д.В. Силаков. Использование аппаратной виртуализации в контексте информационной безопасности, Труды ИСП РАН том 20. 2011 г. стр.25-36. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)

[5]. П.Н. Яковенко. Прозрачный механизм удаленного обслуживания системных вызовов. Труды ИСП РАН Том 18. 2010 г. Стр. 221-242. ISSN 2220-6426 (Online), ISSN 2079-8156 (Print)

[6]. K. Mallachiev, N. Pakulin. Protecting Applications from Highly Privileged Malware Using Bare-metal Hypervisor. DOI: 10.15514/SYRCOSE-2014-8-10.

[7]. U. Steinberg and B. 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.

[8]. T. Shinagawa, H. Eiraku, K. Tanimoto, K. Omote, S. Hasegawa, T. Horie, M. Hirano, K. Kourai, Y. Oyama, E. Kawai, K. Kono, S. Chiba, Y. Shinjo, and K. 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.

[9]. A. Seshadri, M., Ning Qu, and A. Perrig. 2007. SecVisor: a tiny hypervisor to provide lifetime kernel code integrity for commodity OSes.*SIGOPS Oper. Syst. Rev.*41, 6, 335-350. DOI=10.1145/1323293.1294294 C. Takemura and L. S. Crawford. The Book of Xen. 2009, 312 pp. ISBN-13 978-1-59327-186-2,

[10]. J. Rutkowska. Software compartmentalization vs. physical separation. Invisible Things Lab, 2014
http://www.invisiblethingslab.com/resources/2014/Software_compartmentalization_vs_physical_separation.pdf

[11]. A. Dunkels lwIP, a small independent implementation of the TCP/IP protocol suite. http://www.nongnu.org/lwip