ELECTRONICS AND ELECTRICAL ENGINEERING
ISSN 1392 – 1215 *2010. No. 8(104)*
ELEKTRONIKA IR ELEKTROTECHNIKA

SYSTEM ENGINEERING, COMPUTER TECHNOLOGY
T 120
SISTEMŲ INŽINERIJA, KOMPIUTERINĖS TECHNOLOGIJOS

Rootkit Detection Experiment within a Virtual Environment

J. Toldinas, D. Rudzika
Computer Department, Kaunas University of Technology,
Studentų 50, LT-51368, Kaunas, Lithuania, e-mails: eugenijus.toldinas@ktu.lt, darius.rudzika@stud.ktu.lt
V. Štuikys, G. Ziberkas
Software Engineering Department, Kaunas University of Technology,
Studentų 50, LT-51368, Kaunas, Lithuania, phone: +370 37 300399, e-mails: vytautas.stuikys@ktu.lt, ziber@soften.ktu.lt

Introduction

The expansion of Integrated Electronic Systems (IES) [1] and Information Communication Technology (ICT) resulted in a global virtualization, i.e., now we are moving from a single work station to virtual machine and towards cloud computing [2], where humans are working using mobile devices [3].

Due to the battery energy issues [4-7], it is reasonable to transfer energy-aggressive computations to virtual environments. However, they are operating under conditions that are constantly affected by threats and danger of virus attacks. In such a case, we must take care to the information security [8] and protection against the attacks.

In general, computer viruses are products of special software called malware. Malware, as described in [9], is classified into four classes: type 0, type I, type II and type III. According to statistics gathered from Microsoft's Malicious Software Removal Tool, a significant fraction of the malware it encounters consists of stealth rootkits [10]. With respect to the mentioned classification, rootkits are malware of type I and type II [9].

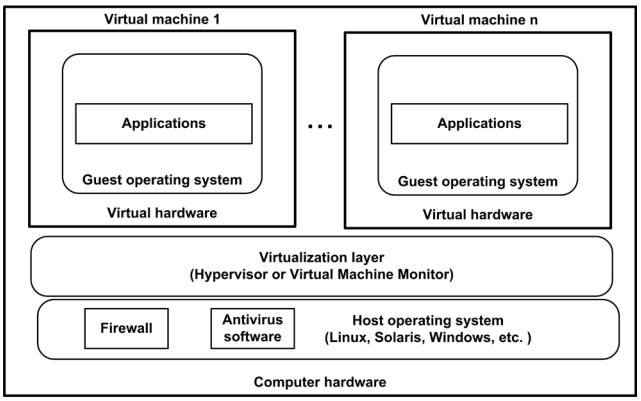
A rootkit is a small computer program that stealthily invades an operating system (OS) or its kernel and takes control of the computer [10]. Rootkits are receiving more attention now as they are becoming serious security threats to all classes of computing, including embedded devices, desktop users, and server farm machines. As rootkits hide malware activities, i.e. may run as hidden processes, gain accesses as administrator to system resources and exploit kernel vulnerabilities, it is difficult to detect them.

Basically, rootkits can be categorized into two groups: user-level (aka application-level) rootkits and kernel-level rootkits [11, 12]. If user-level rootkits are quite easy to uncover, because they do not modify the OS kernel, the second group poses a lot of problems because the rootkits modify the OS kernel to provide the faked information.

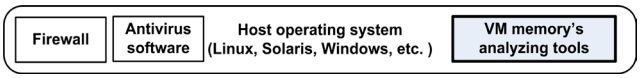
The aim of the paper is to present some framework to investigate the behavior of the kernel-level rootkits and describe the results of experiments we have carried out aiming to model the processes of detection of such kind of malware. The basic result we have identified is the rootkits detection time dependencies upon the virtual machine memory size.

Problem motivation

The typical structure of virtual environment consists of hardware with host operating system (OS), virtualization layer and series (pool) of virtual machines (VM) with guest OS as it is presented in Fig. 1, a).



a)



b)

Fig. 1. Typical virtual architecture with firewall and antivirus software (a) and extra tools to support security (b)

The benefits of virtualization are numerous and include, for example, portability, manageability and efficiency in using of computational resources. But the

benefits are not for free: we must consider and evaluate security problems that arrive with virtualization. The typical solution of the problems is the use of the firewall and antivirus software installed in the host operating system (see Fig.1, a)). Though the solution is simple enough, however, it cannot ensure entirely the security of virtual machines from the rootkit attacks (for details, see [13]).

To protect virtual machines from that kind of virus, it is possible to install the firewall and antivirus software in each VM. However, such a solution is rather too complex and lacks of flexibility in terms of efficiency and costs. First, there are extra memory losses in each virtual machine. And next, licenses are need for each virtual application, i.e. in each guest OS. As virtual machines are usually created dynamically the number of required licenses should be anticipated for the maximum meaning that not all of them will be in use.

A question that is often asked is: "Can a virtual machine be used to compromise its host server?" Although no known compromises exist today which could be used to attack a host server from within a virtual machine, it is conceivable that it could be accomplished through the virtualization platform's communication mechanisms between host server and virtual machines used by the platform's guest OS enhancement tools [14, see page.98].

What we propose for the problem solution in this paper is the use of VM memory's scanning tools from outside, e.g. from the host OS (see Fig. 1, b). The tools are storied into the host computer and are operating under host OS control. The tools perform scanning of VM memory aiming to identify the existence of the virus.

Problem representation domain

Depending on the level of exploitation, a rootkit can operate in the user space and the kernel space. Kernel mode rootkits are more detrimental than user mode rootkits as they can obtain unrestricted accesses at the root privilege level and thus can freely manipulate any component of the system via the compromised OS.

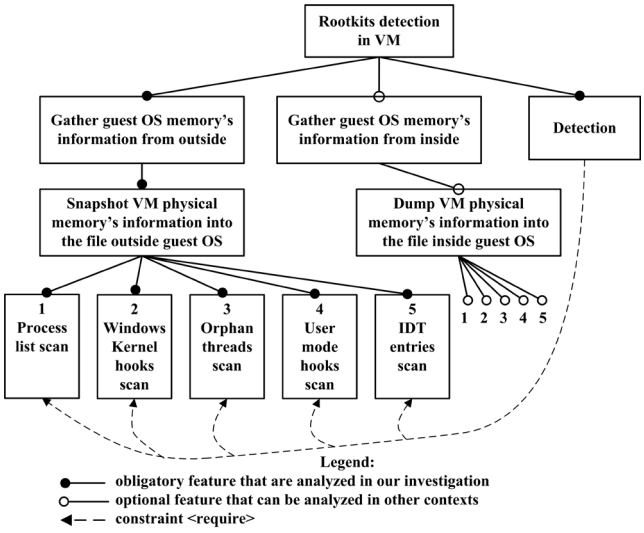


Fig. 2. Problem representation using feature diagram notation

When a virtual machine is running it is possible through the use of existing tools (e.g., memory dump to file) to gather information about memory's (e.g., process list, interrupt description table, kernel mode, user mode, etc.) from both inside and outside of the virtual machine OS. Fig. 2 explains the features of the problem and solution domains. Note that black circles denote obligatory features that were taken into account in our investigation. White circles denote optional features that can be analyzed in other contexts.

Rootkits detection may be made by the memory scan for known anomalies in the process list, in the kernel mode, in the orphan threads list, in the user mode and in the interrupt descriptor table (IDT).

A virtual machine in the host computer is presented as a continuous file, where the guest OS structure and all information are saved. We provide a brief overview of tools used as follows.

- *Volatility framework* – the Volatility Framework is a completely open collection of tools, implemented in Python under the GNU (General Public License), for the extraction of digital artifacts from volatile memory (RAM) samples (www.volatilesystems.com/default/volatility). The extraction techniques are performed completely independent of the system being investigated but offer unprecedented visibility into the runtime state of the system.
- *Volatility plug-ins* – list of the published plug-ins for the Volatility framework.
- *Python* – is an object-oriented, interpreted, and interactive programming language.
- *MinGW* – a port of the GNU Compiler Collection (GCC), and GNU Binutils, for use in the development of native Microsoft Windows applications. In our case the tool was used for developing Volatility plug-in.
- *Flypaper.exe* – HBGary Flypaper is loaded as a device driver and it blocks all attempts to exit a process, end a thread, or delete the memory. All components used by the malware remain as a resident in the process list and stay in the physical memory. The entire execution chain is reported so that you can follow each step. HBGary Flypaper is free for non-commercial use.
- *VMware Workstation v6.5 and v7* – desktop oriented virtualization platform.

Strategy of the methodology for the given platform

The proposed methodology contains three stages: initialization, snapshot and analysis. The first stage needs both the guest OS and host OS initial initialization. Each guest OS needs to be initialized with:

- Rootkit for simulation process (we use FU - open source rootkit with well known code and behavior (www.securityfocus.com/columnists/358/1).
- Network services (such as, tfpd, ftpd, syslogd) for communication with host OS.

Host OS needs to be initialized with: *Python* runtime v2.6 serves to enable volatility framework run because it is

entirely written in that language; *Volatility framework; Volatility framework plug-ins; Microsoft ® PowerShell*.

The second stage serves for taking guest OS memory's information snapshots periodically and saving that information into the file VMEM (meaning 'virtual memory' for short) outside guest OS for the next stage. Actions of the snapshot stage are detailed in Fig.3.

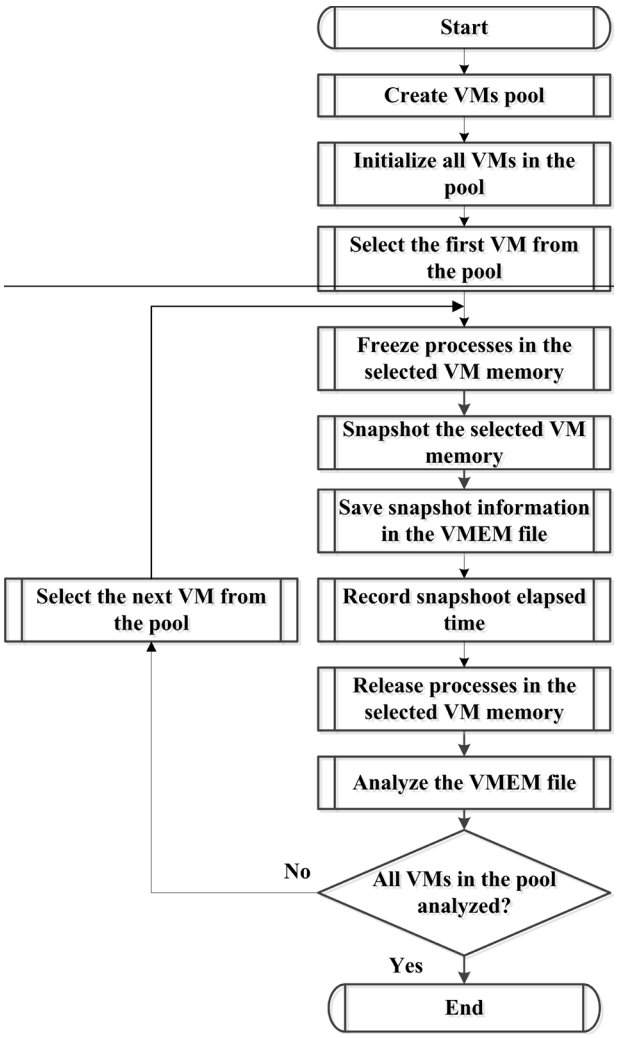


Fig. 3. Algorithm that models ROOTKIT detection functionality within the VM memory's analyzing tools (see Fig.1. b))

It is important to state that we need to freeze the processes within the guest OS for taking the snapshot. At the end of snapshot we record the elapsed time and, than the guest OS processes are released.

The snapshot what was saved into the file identified as VMEM is used at the last stage: analysis for rootkit detection. Fig. 4 presents main processes of the last stage.

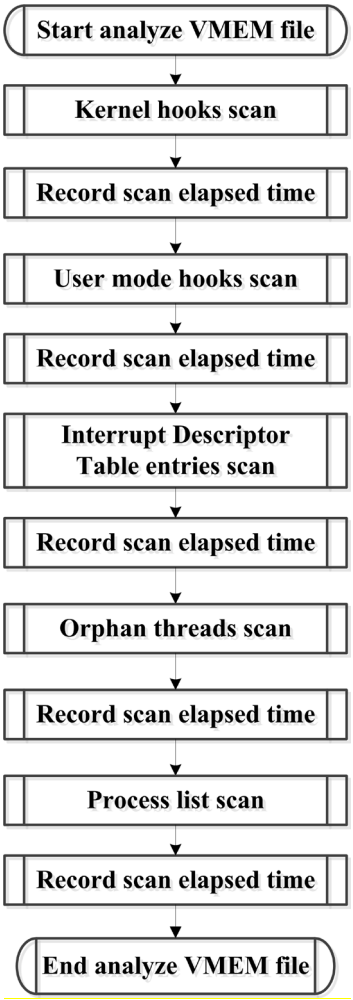


Fig. 4. Analyzing processes for communication and elapsed time recording

We applied five scans for rootkit detection and every scan elapsed time was recorded. The task of mentioned measures was to evaluate and detect how match time takes every scan period and how scan times depends upon guest OS memory's size.

Scripts were written for each scan processes and scan times recording. For all collected snapshots volatility framework was run from Microsoft ® *PowerShell*.

The number of command lines used for getting experiment results is shown in Table 1.

Table 1. Scripts for snapshot scanning

Analyze purpose	Script
Process list scan	python volatility.py pslist -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'
User mode hooks scan	python volatility.py usermode_hooks -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM' –d C:\temp
Orphan threads scan	python volatility.py orphan_threads -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'
Kernel hooks scan	python volatility.py kernel_hooks -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM' –d C:\temp
IDT entries scan	python volatility.py idt_entries -f '$FULL_PATH_TO_VM_MEMORY_IMAGE.VMEM'

Experiments

The aim of our experiment was to collect experimental data of real VM and to evaluate the proposed methodology by recording the scan time's dependencies upon the VM memory size.

In our experiments, the host machine is Intel P9400 running host *Windows 7* 32-bit OS. The VM hypervisor is *VMware Workstation v6.5*.

The pool of the guest OS was implemented using *Windows XP SP3* 32-bit with the different sizes of random access memory (RAM). The first VM RAM size was 128MB, the second 256 MB, and the next 384MB and so on till the last 2048MB.

In this section we present the results of using our verification function for kernel rootkits detection and give a brief evaluation of verification performance.

The experiment results for each increment of the guest OS memory size, i.e. VM, are presented in Table 2.

Table 2. Experiment result details

VM memory size, MB	Snapshot time, s	Orphan threads scan, s	Process list scan, s	Kernel hooks scan, s	User mode hooks scan, s	IDT entries scan, s	Total time, s
128	1	3,88	0,14	2,65	127,10	0,77	135,55
256	1	1,12	0,14	2,59	119,72	0,33	124,91
384	1	4,63	0,14	2,57	116,88	0,36	125,59
512	2	6,17	0,14	2,58	116,21	0,42	127,52
640	2	6,70	0,16	2,56	100,38	0,49	112,28
768	2	7,79	0,14	2,58	109,60	0,38	122,50
896	2	8,79	0,14	2,53	108,99	0,50	122,96
1024	19	22,34	0,14	2,56	101,38	0,54	145,97
1152	22	25,93	0,14	2,60	97,35	0,79	148,80
1280	26	27,86	0,14	2,59	108,73	0,56	165,88
1408	26	28,58	0,14	2,53	113,36	0,52	171,14
1536	33	30,72	0,14	2,60	116,85	0,53	183,83
1664	27	33,05	0,14	2,58	121,85	0,56	185,19
1792	33	39,82	0,14	2,55	108,97	0,48	184,96
1920	44	43,66	0,15	2,59	113,93	0,55	204,87
2048	55	46,58	0,14	2,58	117,96	0,40	222,67

In Fig. 5, we present a graphical relationship between the VM memory size and the total time consumed for the snapshot and all types of the VM memory scans.

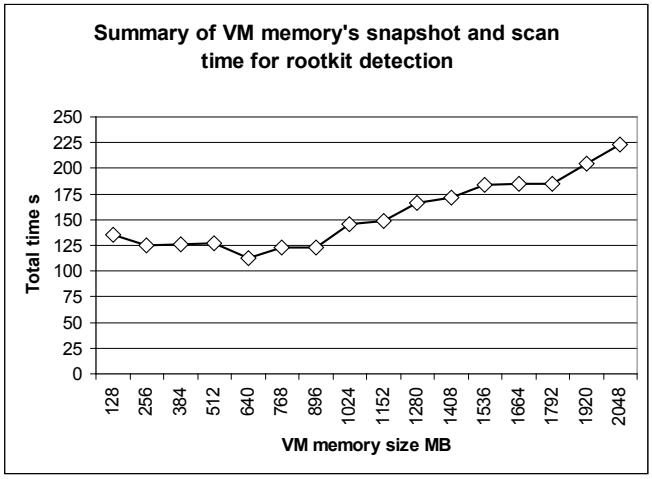


Fig. 5. Total analyze time in VM with various memory sizes

We have also identified that there is a difference in time when we are repeating the scanning process, for example, for the orphan threads scan and for the user mode hooks scan.

As it is depicted in Fig. 6, the second orphan threads scan requires much less time then the first We explain the phenomena by the memory data cashing effect.

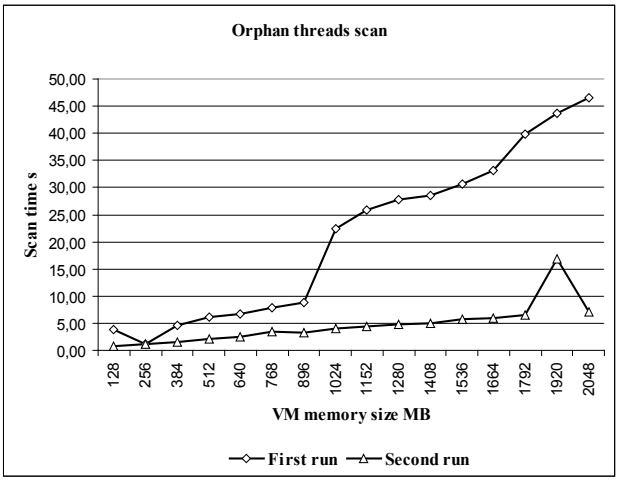


Fig. 6. Two runs of the orphan threads scan

As it is depicted in Fig. 7, the second user mode hooks scan requires also less time then the first.

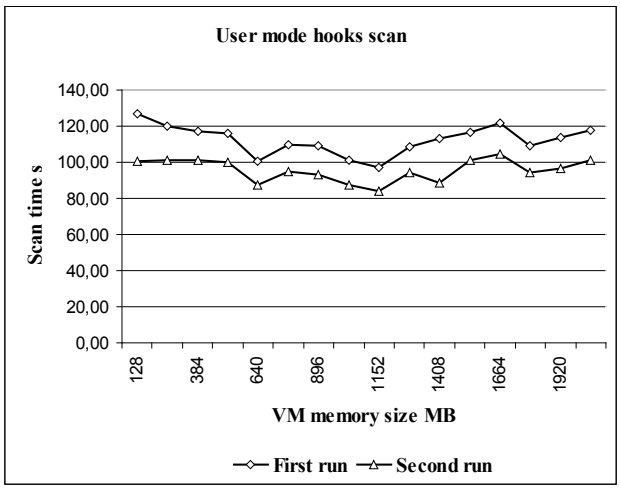


Fig. 7. Two runs of the user mode hooks scan

Comments on the modelling results given in Table 2 are as follows. Process list, Kernel hooks and IDT entries scans are approximately constant because they are independent upon VM memory size (they depend on guest OS initial configuration only). Small discrepancies are due to the measurement inaccuracy.

User mode hooks scan varies depending on the VM workload. In the rest scans times vary depending on the VM memory size, but there is a noticeable jump when the VM memory size reaches 1024 MB. The jump is due to the hypervisor's specificity: when the VM memory size is less than 1024 MB VM are still under operation, and when the size of VM is more or equal to 1024 MB VM are interrupted until the end of the scans.

Conclusions

Our experiment shows that the VM memory size is influential on the total elapsed time for rootkits detection in the following manner: 1) with the increase of memory size the snap shot time grows largely but there is no evident relationship (e.g., if the size of memory changes by factor 2-3 the time increases by factor 15-20); 2) the memory size is most influential on scanning time of Orphan threads, again there is no clearly identifiable expression of the relationship; 3) Process list and Kernel hooks scan times are practically independent upon the memory size of virtual machines; 4) user mode hooks and IDT scan times not much depend on the memory size, though one can observe some discrepancies in time values. Our experiment results also show that the difference between the minimal amount of time and the maximal amount of time can be expressed by factor 2.

References

1. **Keras E., Balaišis P., Dzingus N., Eidukas D., Valinevičius A.** Electronic Plant Leaf Testing System // Electronics and Electrical Engineering. – Kaunas: Technologija, 2008. – No. 8(88). – P. 99–102.
2. **Pokharel M., Park J.** Cloud computing: future solution for e–governance // Proc. of the 3rd International Conference on Theory and Practice of Electronic Governance. – Bogota, Columbia, 2009. – P. 409–410.
3. **Batkauskas V., Kajackas A.** Quality of Heterogeneous Mobile Data Services: Capabilities and End–user Achievements // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 5(101). – P. 43–46.
4. **Tiliute D.** Battery Management in Wireless Sensor Networks // Electronics and Electrical Engineering. – Kaunas: Technologija, 2007. – No. 4(76). – P. 9–12.
5. **Damaševičius R., Štuikys V., Toldinas J.** Embedded Program Specialization for Multiple Criteria Trade–offs // Electronics and Electrical Engineering. Kaunas: Technologija, 2008. – No. 8(88). – P. 9–14.
6. **Toldinas J., Štuikys V., Damaševičius R., Ziberkas G.** Application–Level Energy Consumption in Communication Models for Handhelds // Electronics and Electrical Engineering. – Kaunas: Technologija, 2009. – No. 6(94). – P. 73–76.
7. **Miedzinski B., Rutecki K., Habrych M.** Autonomous Monitoring System of Environment Conditions // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 5(101). – P. 63–66.
8. **Toldinas J., Štuikys V., Ziberkas G., Naunikas D.** Power Awareness Experiment for Crypto Service–Based Algorithms // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 5(101). – P. 57–62.
9. **Rutkowska J.** Introducing Stealth Malware Taxonomy // COSEINC Advanced Malware Labs, Version 1.01, 2006.
10. **Bowman M., Brown H., Pitt P.** An Undergraduate Rootkit Research Project: How Available? How Hard? How Dangerous? // Proc. of Information Security Curriculum Development Conference. – Kennesaw, Georgia, USA, 2007. – P. 43–48.
11. **Jones S., Arpaci–Dusseau A., Arpaci–Dusseau R.** VMM–based Hidden Process Detection and Identification using Lycosid // Proc. of VEE '08. – Seattle, Washington, USA, 2008. – P. 91–100.
12. **Quinh N., Takefuji Y.** Towards a Tamper–Resistant Kernel Rootkit Detector // Proc. of SAC'07. – Seoul, Korea, 2007. – P. 276–283.
13. **Vasisht V., Lee H.** SHARK: Architectural Support for Autonomic Protection against Stealth by Rootkit Exploits // IEEE, 2008. – P. 106–116.
14. **Marshall D., Reynolds W., McCrory D.** VMware® and Microsoft® Platforms in the Virtual Data Center // Auerbach Publications, Taylor & Francis Group, 2006.

Received 2010 03 05

J. Toldinas, D. Rudzika, V. Štuikys, G. Ziberkas. Rootkit Detection Experiment within a Virtual Environment // Electronics and Electrical Engineering. – Kaunas: Technologija, 2010. – No. 8(104). – P. 63–68.

In the context of virtual environments, the security problems are highly important. The paper presents some results of experiments we have carried out within the real virtual machine environment through modeling aiming to identify dependencies between the virus, called Rootkits, detection time and the virtual machine memory size. Rootkits exploit kernel vulnerabilities and gain privileges (popularity) within any system, virtual or not. The basic result of the paper is as follows: 1) the Rootkits detection methodology for the virtual environment when the memory size of a virtual machine is changing; 2) dependences between the virtual machine memory size and Rootkit detection time. Ill. 7, bibl. 14, tabl. 2 (in English; abstracts in English, Russian and Lithuanian).

Е. Толдинас, Д. Рудзика, В. Штуйкис, Г. Зиберкас. Эксперимент определения Rootkit в виртуальной среде // Электроника и электротехника. – Каунас: Технология, 2010. – № 8(104). – С. 63–68.

В среде виртуальных машин очень важна проблема безопасности. В статье представлены некоторые результаты эксперимента, используя моделирование, проведенное нами в конкретной среде виртуальных машин, с целью определения зависимости времени обнаружения вирусов типа Rootkits от размера оперативной памяти виртуальной машины. Данные вирусы поражают функциональные свойства ядра операционной системы, широко распространены в различных операционных системах, как в стационарных, так и в виртуальных. Основные результаты: 1) предложена методика обнаружения вирусов типа Rootkits в среде виртуальных машин, при измененяющемся объеме оперативной памяти; 2) установлена связь между временем обнаружения вируса и размером памяти виртуальных машин. Ил. 7, библ. 14, табл. 2 (на английском языке; рефераты на английском, русском и литовском яз.).

J. Toldinas, D. Rudzika, V. Štuikys, G. Ziberkas. Rootkit virusų aptikimo eksperimentas virtualioje aplinkoje // Elektronika ir elektrotechnika. – Kaunas: Technologija, 2010. – Nr. 8(104). – P. 63–68.

Virtualių mašinų aplinkoje labai didelę reikšmę turi saugumas. Šiame straipsnyje pateikiami kai kurie eksperimentiniai rezultatai, gauti konkrečioje virtualių mašinų aplinkoje modeliavimo būdu siekiant nustatyti virusų, žinomų Rootkits vardu, aptikimo trukmę priklausomai nuo virtualios mašinos atminties dydžio. Šie virusai klastingai pažeidžia operacinės sistemos branduolio funkcionalumą, yra labai paplitę bet kurioje sistemoje, virtualioje arba stacionarioje. Pagrindinis straipsnio rezultatas toks: 1) pasiūlyta virusų Rootkits aptikimo metodika virtualių mašinų aplinkoje, kai keičiasi virtualių mašinų operatyviosios atminties dydis; 2) nustatytos viruso aptikimo laiko priklausomybės nuo virtualios mašinos atminties dydžio. Il. 7, bibl. 14, lent. 2 (anglų kalba; santraukos anglų, rusų ir lietuvių k.).