On the Impossibility of Detecting Virtual Machine Monitors

Shay Gueron[1] and Jean-Pierre Seifert[2]

[1] Department of Mathematics, Faculty of Science, University of Haifa, Haifa 31905, Israel and Intel Corporation, Israel Development Center, Haifa, Israel
shay@math.haifa.ac.il
[2] Technische Universität Berlin and Deutsche Telekom Laboratories, 10587 Berlin, Germany
jeanpierreseifert@yahoo.com

Abstract. Virtualization based upon Virtual Machines is a central building block of Trusted Computing, and it is believed to offer isolation and confinement of privileged instructions among other security benefits. However, it is not necessarily bullet-proof — some recent publications have shown that Virtual Machine technology could potentially allow the installation of undetectable malware root kits. As a result, it was suggested that such virtualization attacks could be mitigated by checking if a threatened system runs in a virtualized or in a native environment. This naturally raises the following problem: *Can a program determine whether it is running in a virtualized environment, or in a native machine environment?* We prove here that, under a classical VM model, this problem is not decidable. Further, although our result seems to be quite theoretic, we also show that it has practical implications on related virtualization problems.

1 Introduction

The concept of Virtual Machines (VM) has been closely coupled with Trusted Computing from its early days, cf. [20,2,31,4,36,40,43]. Clearly, isolation through a robust Virtual Machine Monitor implementation provides a very powerful security ingredient. This concept has already been extensively used for security reasons in past computer generations cf. [25,26,36]. Furthermore, recent mass-market oriented Trusted Computing efforts (driven by the Trusted Computing Group (TCG) [44,34]) also capitalize on this excellent domain isolation concept provided by Virtual Machines, cf. [1,6,10,23].

Along with the TCG efforts, a vibrant academic research community has addressed to solve numerous computer security problems through Virtualization. We mention here only a small sample from the many publications on this topic. The early influential works of [17,18] offer a broad view of various security applications enabled by the concept of Virtualization, such as Digital Rights Management (RDM) systems and the Rootkit installation problem. An excellent overview is given by [28].

However, even more recently, some interesting publications [39,27,46] have shown how the VM technology could be misused for allowing the installation of undetectable Rootkits. Finally, [19] described even a simple and convincing method for circumventing DRM schemes along the idea of misusing the Virtualization concept. To mitigate such "virtualization attacks", it is obvious to simply check whether or not a threatened

D. Gritzalis and J. Lopez (Eds.): SEC 2009, IFIP AICT 297, pp. 143–151, 2009.

application runs in a virtualized environment. Consequently, the following theoretical problem arises very naturally:

> *Can a program distinguish whether it is running in a virtualized environment or in a native machine environment?*

This problem and its derivatives are the focus of our paper. We formally prove that, under a classical VM model, it is impossible to design an algorithm that would allow a program to determine whether it is running in a virtualized environment or in a native machine environment. Note that it makes no sense at all to ask the question of detecting "virtualization" for a specific fixed Virtual Machine Monitor and a specific fixed native machine.

Although our results seem theoretical at first sight, they bear practical security ramifications as pointed out by Microsoft and Intel. Our results especially show that Bill Gates' requirement, cf. [15], that "Microsoft must be careful that the VM does not become a security weakness through which an attacker could insert a VM under the operating system, negating Windows security protections" is a fundamental key issue. In light of our results it seems quite reasonable that Microsoft recommended in its public analysis of these rootkit issues, cf. [32], some strongly protected default settings for the Enabling/Disabling of the new hardware Virtualization extensions offered by AMD and Intel. Also Pat Gelsinger from Intel explained that there are gaps to be plugged around virtual security. "As virtual machine migrations become popular, they become vulnerable," he said, warning that a "whole new set" of attacks will emerge focused on VMs, cf. [38].

One may argue, cf. [5,12,14,13,11,16,18,21,37], that there may always be ways to discover whether or not the execution is taking place in a virtualized environment, and that these methods are not considered in our purely theoretical impossibility result that is offered here.

However, later in this paper we will explain why we expect that for security reasons, the reality would converge to our ideal model of virtualization. This assumed convergence of ideal virtualization is also nicely explained in [14], describing it there as a VMM detection arms race. And recent concerns, cf. [8], about the "Risk of Virtualization" fit very well into Microsoft's and Intel's concerns.

We also want to mention here that the appearance of our impossibility results is quite similar to the early theoretical work of Cohen on Computer Virus Detection, cf. [7]. It tackles a real-world computer security problem within a theoretical model, and derives an impossibility result which looks at first sight simple and not very relevant from the practical point of view. Nevertheless, it is the definitive and undisputable answer from the theory of computation aspect and confirms the VMM detection arms race of [14] which describe that there is a chance that VMMs eventually and successfully evade detection.

The paper is organized as follows. Section 2 gives a high-level introduction to the concepts Virtual Machine, Virtual Machine Monitor and even Virtualization, and develops the tools that allow a formalized proof of our main Theorem. This Theorem along with some important implications is presented in Section 3. In Section 4 we discuss the relevance of our theoretical result to the proposed heuristic methods to detect

Virtualization environments. Finally, Section 5 presents our conclusions and further directions of interesting research topics.

2 Definitions and Preliminaries

In this paper, we follow the classical formalism of Popek and Goldberg [37] to define the concepts of Virtualization, Virtual Machine, and Virtual Machine Monitor, and will use the following definitions.

Definition 1. A *Virtual Machine Monitor (VMM)* is any control program that satisfies the three properties: efficiency, resource control, and equivalence.

Definition 2. The functional environment that any program experiences when running with a VMM present is called a *Virtual Machine (VM)*. The VM consists of the original machine and the VMM.

Intuitively, a VM is an efficient, isolated duplicate of the real machine. A VMM is a piece of software that

1. Provides an environment for other programs, which is essentially identical to the environment provided by the original machine. Programs that run in this environment show, at most, a minor decrease in their running speed.
2. Has complete control of the system's resources. Here, "essentially identical" implies that only minor exceptions in the availability of system resources and only minor timing differences are tolerated.

Under this view, a classical Operating System that provides quasi-parallel execution of different processes is not considered as a VMM. "Efficient" requires that a statistically significant part of the machine's instructions are executed directly on the original machine, with no VMM intervention. Under this view, emulators and software interpreters are also not considered as a VMM. "Complete control" implies that programs

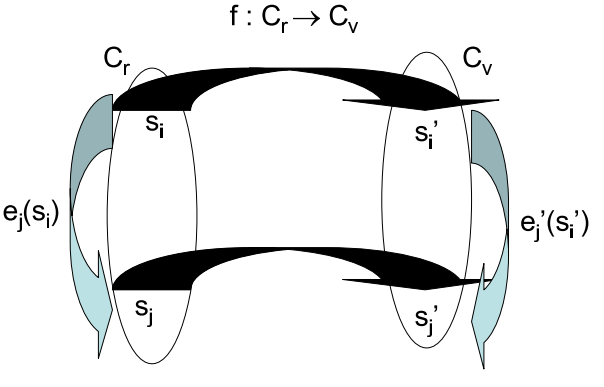

Fig. 1. The Virtual Machine map

running under a VMM control cannot gain system resources that were not explicitly assigned to them. However, on the other hand, under certain circumstances, a VMM itself can regain control of resources that have been already allocated.

The equivalence property can be defined in terms of a homomorphism on the possible machine states. To this end, we partition the set $\mathcal{C}$ of all possible states of the native machine into $\mathcal{C} = \mathcal{C}_r \cup \mathcal{C}_v$ where $\mathcal{C}_v$ contains the states for which the VMM is present in memory, and $\mathcal{C}_r$ consists of the remaining states. Each instruction i in the native machine can be viewed as a unitary operator $i : s_j \mapsto s_k$ operating on $\mathcal{C}$. Similarly, each sequence $i_1 \circ \cdots \circ i_n(s_1) = e_n(s_1) = s_2$ of n successive instructions can be viewed as an unitary operator operating on $\mathcal{C}$. Let the set of all sequences of instructions having a finite length be $\mathcal{I}$. We define a *Virtual Machine map* (VM map) $f : \mathcal{C}_r \to \mathcal{C}_v$ as a one-one homomorphism with respect to all of the operators e_i in $\mathcal{I}$. That is (see Figure 1), for any state $s_i \in \mathcal{C}_r$ and any instruction sequence e_i, there exists an instruction sequence e'_i such that $f(e_i(s_i))) = e'_i(f(s_i))$. We also require that f is one-one (i.e., it has a left inverse) and that for each e_i there is a way to find the appropriate e'_i and to execute it. We can now define "equivalence" and "essentially identical" as follows:

Definition 3. Let $\mathcal{M}$ be a real machine and $\mathcal{V}$ a virtual machine defined via the Virtual Machine map $f : \mathcal{C}_r \to \mathcal{C}_v$. Let s_1 be any starting state leading to a halting state s_2 of the real machine $\mathcal{M}$, where $f(s_1) = s'_1$. Suppose thet the starting state s'_1 leads to a halting state s'_2 in $\mathcal{V}$. The corresponding VMM is said to have the equivalence property, if $f(s_2) = s'_2$.

As mentioned above, the equivalence property [37] allows for two possible minor exceptions in timing and resource availability.

Timing
Due to occasional intervention of the control program, certain instructions of an executed program may take longer than expected. Thus, one cannot make assumptions about exact execution times. In our model, we first ignore such timing differences, and later comment on how this assumption could be ensured in certain hardware-supported virtualization environments.

Resources Availability
Note that, resources-wise, an actually constructed VM can be smaller than the real mother machine (a smaller version of the real machine, but logically still the same hardware). Thus, we require the equivalence to be guaranteed between the artificially smaller VM and the smaller version of the real machine

3 The Main Theorem

Unlike [12,14,13] we do not try check if another remotely-connected machine is a virtual machine or a native machine. The problem that we tackle here is whether some algorithm can distinguish between a native and a virtualized environment during its run-time, while it is running on this potentially "hostile" environment (about which it has to decide). In other words, the distinguishing algorithm cannot use any "trusted"

or reliable external help like an oracle. For this environment, we prove the following result.

Theorem 1. *There exists no algorithm that can decide, for every real machine and for every VMM, whether it is running under control of a VMM or on a real machine without a VMM present.*

Proof. Consider real machines virtualizable along the above definitions from [37]. Assume by contradiction that there exists an algorithm A which for every real machine $\mathcal{M}$ and for every virtual machine $\mathcal{V}$ given by its virtual machine map $f : \mathcal{C}_r \to \mathcal{C}_v$ satisfies the following:

- When A is executed on a real machine $\mathcal{M}$, it leads $\mathcal{M}$ to a halting state $s_{\text{no VMM present}}$.
- When A is executed on a virtual machine $\mathcal{V}$ controlled by its corresponding VMM, it results in a halting state $s'_{\text{VMM present}}$ of $\mathcal{V}$.
- $f(s_{\text{no VMM present}}) = s'_{\text{VMM present}}$ with $s'_{\text{VMM present}} \neq s_{\text{no VMM present}}$, to have a unique distinction between the two cases.

Now, fix one real machine $\mathcal{M}$ and one virtual machine $\mathcal{V}$ for this fixed real machine. We define a new machine $\hat{\mathcal{M}}$, state-wise, to be identical to $\mathcal{V}$, where the VMM is hidden inside its finite state control thus being invisible for programs running on $\hat{\mathcal{M}}$. Intuitively, $\hat{\mathcal{M}}$ can be viewed as the process of moving the effects of the VMM (and especially the VMM program itself) directly into the finite state control of $\hat{\mathcal{M}}$. This process can be seen as realizing $\mathcal{V}$ "directly in hardware" where the effects of the VMM and its control behaviour are still present in $\hat{\mathcal{M}}$, but now without explicitly having the VMM present in memory any more.

By our assumption, algorithm A works for all real machines, i.e., in particular also on $\hat{\mathcal{M}}$. Since $\hat{\mathcal{M}}$ is defined to follow, state-wise, the actions of $\mathcal{V}$, the execution of A on $\hat{\mathcal{M}}$ must lead $\hat{\mathcal{M}}$ into the halting state

$$s'_{\text{VMM present}}. \tag{1}$$

However, on the other hand, executing A on $\hat{\mathcal{M}}$ is essentially equivalent to following the steps of the machine $\mathcal{M}$ without having the corresponding VMM present in memory. Thus, algorithm A executed on $\hat{\mathcal{M}}$ must end in the halting state

$$s_{\text{no VMM present}}. \tag{2}$$

The contradiction between the halting states (1) and (2) for executing A on $\hat{\mathcal{M}}$ shows that no universal distinguishing algorithm can exist. □

3.1 Remark

Although the above proof looks simple or almost trivial, we would like to mention that the core of the above proof implicitly rests upon the ideas of the so called *S-m-n* Theorem and the *Recursion* Theorem of Kleene, (cf. [24]), which are among the two deepest theorems in the "Theory of Computing". However, following the advice from

one of the most respected theoretical computer science theorist, cf. [22], that *"the very simple facts and the basic approaches are the ones that have most impact,"* we chose a simple and basic approach for our proof and explicitly avoided the former two heavy approaches.

3.2 Consequences of the Main Theorem

Here, we address the more practical question of constructing an algorithm for detecting a Virtualization attack when a specific machine and an arbitrary VMM for that machine are given. The following corollary states the result.

Corollary 1. *Suppose that a specific machine $\mathcal{M}$ and an arbitrary VMM for that machine are given. There exists no algorithm that is able to determine whether it is running on $\mathcal{M}$ under control of a VMM or on $\mathcal{M}$ without a VMM present in memory.*

Proof. First, we have to assume here that the given machine $\mathcal{M}$ is indeed virtualizable. We follow the proof of Theorem 1 for the given machine $\mathcal{M}$, constructing the new machine $\hat{\mathcal{M}}$ for an arbitrary but fixed virtual machine map f. By definition of the equivalence property (which of course holds for the map f) we get that $\hat{\mathcal{M}}$ and $\mathcal{M}$ are functionally equivalent. Thus, we are not changing the fixed and specific machine functionality implied by $\mathcal{M}$. Thus, when considering $\hat{\mathcal{M}}$ or $\mathcal{M}$ we still have the same fixed machine "functionality", we can simply follow the above proof and conclude the corollary. □

4 Practical Detection of Virtualization

We comment here on the relation between our theoretical impossibility result and the common perception that in practice, there might be always methods to discover whether execution is happening in a virtualized or in a native environment (e.g., [5,12,14,13,11,16,18,21,37]). In this context, Lauradoux [30] suggested that a so called "hard-clock" and the cache timing behavior of the machine could be used for detecting whether an execution environment is virtualized or not. Similar artifacts of the underlying CPU microarchitecture were also subsequently used in [12,14,13]. Also, we would like to stress that the distinguishing algorithms of [12,14,13] rely on an external "trusted clock" which is not part of our model. Thus, their practical "remote detection" scenario doesn't compare at all with our classical detection problem which allows no external help. Indeed, [14] agree that this "trusted clock" is easy to circumvent by a malicious VMM by just disabling or masking the TCP timestamps, thus leaving the distinguisher in the dark. Moreover, [14] also briefly discuss getting rid off their "trusted external clock" but conclude that this an open and difficult question.

Timing artifacts which presumably provide a VMM detection mechanism, rely on the possibility to precisely measure the execution time, a task which is known to be a technical challenge for a full and secure virtualization (cf. [41,42]). Particularly, such precise timing is based on the non-privileged Read Time Stamp Counter (`RDTSC` for the x86 architecture) instruction being a critical instructions.

The recent Intel initiative to provide full secure Virtualization support using the IA-32 architecture (cf. [45]) followed the path outlined in [41]. Here, all instructions are classified as either critical or non-critical instructions, where the functionality of the critical instructions (including `RDTSC`) is "somehow" confined to a VMM level class, so the VMM itself has the freedom to decide how to handle the functionality of those critical instructions. The emerging problem in this case is that a VMM could give the programs running under its control the "precise timing illusion" of a real machine, by simply subtracting the extra cycles spent inside the VMM, from the Time Stamp Counter. Thus, as pointed out in [14], having the possibility to let the VMM directly control sensitive machine instructions like `RDTSC` complicates heuristic VMM detection algorithms relying on the discovery of timing artifacts, cf. [39,46].

5 Conclusions

We showed here that under the classical VM model, it is impossible to design an algorithm that would allow a program to decide whether it is running in a virtualized environment or in a native machine environment. We also proved here a stronger and more practical impossibility result. This result states that is even for specific and fixed machine type impossible to design an VMM detection algorithm.

This theoretical result has also practical implications. Noting Lampson [29] and some recent publications [3,33,35] that illustrate the security threats of indirect information leaks caused by imperfect isolation, we expect that closing all potential information leaks (including timing) would become a necessary requirement for future hardware virtualization technologies like [45]. Thus, we expect the reality converging (for security reasons) in the future to our ideal model of virtualization. On the other side, closing these holes may pose a serious challenge for heuristic virtualization detection mechanisms, cf. [5,11,30]. This confirms again the hypothesis of [14] that there is an VMM detection arms race which might result in that VMMs eventually and successfully evade detection.

As assumed by [15] and [38] this conflict hints that hardware-assisted Virtualization could become a double-edge sword, especially when considering the recent VM attacks from [39,27,46] and [19]. Thus, similarly to the topic considered in [9], the following new research vector naturally arises: *What kind of cryptographic protection against the VM adversary can we achieve by new constructions, if we have to assume running under the control of a potentially malicious VMM which has full control of what we are doing?*

References

1. Advanced Micro Devices, Pacifica — AMD Secure Virtual Machine Architecture Reference Manual, AMD (2005)
2. Attanasio, C.R.: Virtual Machines and Data Security. In: Proceedings of the Workshop on Virtual Computer Systems, pp. 206–209 (1973)
3. Bernstein, D.J.: Cache-timing attacks on AES, 37 pages (2005), `http://cr.yp.to/papers.html/cachetiming`

4. Bishop, M.: Computer Security: Art and Science. Addison Wesley Professional, Reading (2003)
5. Carpenter, M., Liston, T., Skoudis, E.: Hiding Virtualization from Attackers and Malware. IEEE Security and Privacy 5(3), 62–65 (2007)
6. Chen, Y., England, P., Peinado, M., Willman, B.: High Assurance Computing on Open Hardware Architectures, Microsoft Technical Report, MSR-TR-2003-20 (March 2003)
7. Cohen, F.B.: Computer Viruses: Theory and Experiments. Computers and Security 6, 22–35 (1987)
8. Dignan, L.: Virtualization: What are the security risks? ZDNet.com (January 22, 2008)
9. Dinda, P.A.: Addressing the trust asymmetry problem in grid computing with encrypted computation. In: Proceedings of the 7th Workshop on Languages, Compilers, and Run-time support for scalable systems, pp. 1–7 (2004)
10. England, P., Lampson, B., Manferdelli, J., Peinado, M., Willman, B.: A Trusted Open Platform. IEEE Computer 36(7), 55–62 (2003)
11. Ferrie, P.: Attacks on Virtual Machine Emulators. In: AVAR 2006, Auckland, New Zealand, December 3-5 (2006)
12. Franklin, J., Luk, M., McCune, J., Seshadri, A., Perrig, A., van Doorn, L.: Remote Virtual Machine Monitor Detection. In: ARO-DARPA-DHS Special Workshop on Botnets, Arlington, VA (June 2006)
13. Franklin, J., Luk, M., McCune, J., Seshadri, A., Perrig, A., van Doorn, L.: Remote Detection of Virtual Machine Monitors with Fuzzy Benchmarking. ACM SIGOPS Operating System Review (Special Issue on Computer Forensics) (April 2008)
14. Franklin, J., Luk, M., McCune, J., Seshadri, A., Perrig, A., van Doorn, L.: Towards Sound Detection of Virtual Machines. In: Lee, W., Wang, C., Dagon, D. (eds.) Botnet Detection: Countering the Largest Security Threat, November 2007. Springer, Heidelberg (2007)
15. Galli, P.: Microsoft puts IE enhancements of fast track. Interview with Bill Gates in *eWeek* 22(18) (2006)
16. Garfinkel, T., Adams, K., Warfield, A., Franklin, J.: Compatibility is Not Transparency: VMM Detection Myths and Realities. In: Proceedings of the 11th Workshop on Hot Topics in Operating Systems (HotOS-XI), (May 2007)
17. Garfinkel, T., Pfaff, B., Chow, J., Rosenblum, M., Boneh, D.: Terra: a virtual machine-based platform for trusted computing. In: Proceedings of the 19th ACM Symposium on Operating Systems Principles (2003)
18. Garfinkel, T., Rosenblum, M.: A virtual machine introspection-based architecture for intrusion detection. In: Proceedings of the 2003 Network and Distributed Systems Symposium (NDSS 2003) (2003)
19. Ghodke, N., Figueiredo, R.J.: On the Implications of Machine Virtualization for DRM and Fair Use: A Case Study of a Virtual Audio Device Driver. In: Proceedings of 4th ACM DRM Workshop (2004)
20. Goldberg, R.P.: Architecture of virtual machines. In: Proceedings of the Workshop on Virtual Computer Systems, pp. 74–112 (1973)
21. Goldberg, R.P.: Survey of virtual machine research. IEEE Computer Magazine 7, 34–45 (1974)
22. Goldreich, O., Rosenberg, A.L., Selman, A.L. (eds.): Theoretical Computer Science. LNCS, vol. 3895. Springer, Heidelberg (2006)
23. Grawrock, D.: The Intel Safer Computing Initiative: Building Blocks for Trusted Computing. Intel Press (2006)
24. Hopcroft, J.E., Ullman, J.D.: Introduction to Automata Theory, Languages, and Computation. Addison-Wesley Publishing Company, Reading (1979)

25. Karger, P., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A VMM security kernel for the VAX architecture. In: Proceedings of the 1990 IEEE Symposium on Security and Privacy, pp. 2–19 (1990)
26. Karger, P., Zurko, M.E., Bonin, D.W., Mason, A.H., Kahn, C.E.: A Retrospective on the VAX VMM Security Kernel. IEEE Transactions on Software Engineering 17(11), 1147–1165 (1991)
27. King, S.T., Chen, P.M., Wang, Y.-M., Verbowski, C., Wang, H.J., Lorch, J.R.: SubVirt: Implementing malware with virtual machines. In: Proceedings of the 2006 IEEE Symposium on Security and Privacy (2006)
28. King, S.T., Smith, S.W.: Virtualization and Security: Back to the Future. IEEE Security & Privacy 6(5), 15 (2008)
29. Lampson, B.W.: A note on the confinement problem. Communications of the ACM 16(10), 613–615 (1973)
30. Lauraoux, C.: Detecting virtual machines (manuscript), cedric.lauradoux@inria.fr
31. Madnick, S.E., Donovan, J.J.: Application and analysis of the virtual machine approach to information system security and isolation. In: Proceedings of the Workshop on Virtual Computer Systems, pp. 210–224 (1973)
32. Microsoft, CPU Virtualization Extensions: Analysis of Rootkit Issues (October 2006), http://www.microsoft.com/whdc/system/platform/virtual/CPUVirtExt.mspx
33. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and Countermeasures: the Case of AES, Cryptology ePrint Archive, Report 2005/271 (2005)
34. Pearson, S.: Trusted Computing Platforms: TCPA Technology in Context. Prentice Hall PTR, Englewood Cliffs (2002)
35. Percival, C.: Cache missing for fun and profit. In: Proc. of BSDCan 2005, Ottawa (manuscript, 2005), http://www.daemonology.net
36. Pfleeger, C.P., Lawrence Pfleeger, S.: Security in Computing, 3rd edn. Prentice Hall PTR, Englewood Cliffs (2002)
37. Popek, G., Goldberg, R.: Formal Requirements for Virtualizable Third Generation Architectures. Communications of the ACM 17(7), 412–421
38. Rogers, J.: Virtualization Is Key to Disaster Recovery, Byte and Switch News (September 11, 2007)
39. Rutkowska, J.: Subverting VistaTM Kernel For Fun And Profit. In: SyScan 2006, July 21st, Singapore, and Black Hat Briefings, August 3rd, Las Vegas (2006)
40. Silberschatz, A., Gagne, G., Galvin, P.B.: Operating system concepts, 7th edn. John Wiley and Sons, Chichester (2005)
41. Robin, J.S., Irvine, C.E.: Analysis of the Intel Pentium's Ability to Support a Secure Virtual Machine Monito. In: Proceedings of the 9th Usenix Security Symposium, pp. 129–144 (2000)
42. Sibert, O., Porras, P.A., Lindell, R.: The Intel 80x86 Processor Architecture: Pitfalls for Secure Systems. In: 1995 IEEE Symposium on Security and Privacy, pp. 211–223 (1995)
43. Smith, J., Nair, R.: Virtual Machines: Versatile Platforms for Systems and Proccesses. Elsevier Press, Amsterdam (2005)
44. Trusted Computing Group, http://www.trustedcomputinggroup.org
45. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A.L., Martins, F.C.M., Anderson, A.V., Bennett, S.M., Kagi, A., Leung, F.H., Smith, L.: Intel Virtualization Technology. Computer 38(5), 48–56 (2005)
46. Dai Zovi, D.A.: Hardware Virtualization Rootkits. In: Black Hat Briefings 2006, Las Vegas, August 3 (2006)