VASP: Virtualization assisted Security Monitor for Cross-Platform Protection

Min Zhu[†], Miao Yu[†], Mingyuan Xia[§], Bingyu Li[†], Peijie Yu[†],
Shang Gao[†], Zhengwei Qi[†], Liang Liu[‡], Ying Chen[‡], Haibing Guan[§]
[†]School of Software [§]School of Electronic Information and Electrical Engineering
Shanghai Key Laboratory of Scalable Computing and Systems
Shanghai Jiao Tong University
[‡] IBM Research China
[†§] { carit, superymk, kenmark, justasmallfish, yupiwang, chillygs, qizhwei, hbguan } @ sjtu.edu.cn
[‡] { liuliang yingch }@cn.ibm.com

ABSTRACT

Numerous operating systems have been designed to manage and control system resources with largeness and complexity features, so they need high security protection. However, the security applications always can not provide adequate protection due to the untrusted execution environment. Furthermore, these security strategies cannot support a universal cross-platform system protection. This paper presents VASP, a hypervisor based monitor which allows a trusted execution environment to monitor various malicious behaviors in the operating system. This is achieved by taking advantage of x86 hardware virtualization and self-transparency technology, and providing a unified security protection to operating systems such as Linux and Windows, which can run without any modification. Our design is targeted at establishing a security monitor which resides completely outside of the target OS environment with a negligible overhead. According to the security analysis and performance experiment result, our approach can effectively protect applications and the kernel by the cost of only 0.9% average overhead in Windows XP and 2.6% average overhead in Linux.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection

General Terms

Design, Virtualization technology, Software Protection

Keywords

cross-platform, hypervisor, security, hardware virtualization

1. INTRODUCTION

Virtualization was first applied to the operating system (OS) as IBM System/370 Extended Architecture in 1970 [9] [6], utilizing virtual machine (VM) to enable binary support for the legacy code. As time goes on, both commodity and open source operating systems increasingly make full use of virtualization capabilities of the hardware. For the purpose of security, the operating system always needs other applications' help, such as anti-virus, code protection, anti-debugging and so on. Though they are proved to be safe and effective, most of these protections are established within the operating system. As a result, the protection strategies may have the same privilege level as operating system kernel or even lower than that of the kernel, which can not guarantee them away from the attackers. This paper presents VASP, a lightweight, transparent, cross-platform and high performance security monitor, which enables unified security protection inside the virtual machine monitor based on hardware virtualization technology.

Successfully monitoring and protecting the operating system to realize whether malicious code attacks when the system is running poses several challenges. Firstly, the monitor must have an isolated execution environment with the operating system. It is not acceptable for the execution of the malware to detect or attack the security monitor. Secondly, it is necessary to support a unified security protection in different operating system architecture. Nowadays, operating systems have various architecture, so the system protection need adapt to the system features. Thirdly, the performance overhead introduced by virtualization should be small. Each additional protection will cause a decline in performance, and significant performance degradation makes the original system operation blocked.

VASP is established in the virtual machine monitor (VMM) which is an effective environment to prevent from malicious behaviors [4]. What's more, taking advantage of hardware virtualization technology, VASP possesses a higher privilege level, named root mode, than the OS kernel's to manage the hardware resources. As a result, it executes in the isolated and privileged environment to ensure that there is no impact from other softwares.

VASP enables users to build the monitor not only in Windows but also in Linux, and both of them are typical operating systems. In VASP project, we deploy that the hypervisor is transparent to the existing operating system and doesn't care which architecture of operating system it runs on, as long as the hardware supports virtualization technology.

There are a number of ways to build the protection, but virtualization is always considered to cause high overhead.

SAC'11 March 21-25, 2011, TaiChung, Taiwan.

VASP makes use of few hardware resources, only for processor and memory demand, and the effect is slight relative to the traditional full virtualization. On the other hand, there is no need to modify the original OS kernel.

To the best of our knowledge, this paper is the first to address the issue of the universal security protection on different architecture of operating systems based on hardware virtualization. As we will demonstrate, instead of relying on operating system's space resources, the virtual machine monitor behaviors are safe and effective with the help of hardware virtualization.

In this paper, we make the following contributions.

- We establish a lightweight security monitor with least temporal and spatial overhead to protect operating system as our experiment results revealed. And the hypervisor is designed without any modification to the protected OS.
- We construct a universal cross-platform security architecture. Both Windows and Linux operating systems can be protected under our security platform.
- This architecture is easily extensible for application. Our approach can be utilized in many different security requirements with few changes and corresponding extensions.

The rest of the paper is organized as follows. Section 2 presents the related work in this research field. Section 3 illustrates the design and implementation of our security monitoring. Section 4 evaluates the performance test in two different operating systems, Windows XP and Linux. At last, we conclude the paper in Section 5.

2. RELATED WORK

Hardware virtualization has been applied to operating systems both commercially and in research for several years. Intel VT and AMD SVM Technology make the hardware suitable for virtualizaiton and accelerate the development of it. VMware [12] [15] and KVM [7] both take advantage of hardware virtualization, allowing their virtual machines to work more quickly and smoothly on a single host. However, VMware still use the full virtualization technology which is used to virtualize most hardware resources, such as networks, I/O devices, hard disks and so on, except CPU and memory resources nowadays. KVM is a linux built-in hypervisor which has been combined with the linux kernel tree form 2.6.20 kernel version. When the linux kernel loads the special module, KVM, the whole kernel plays the role of hypervisor itself. Though KVM is small, the whole hypervisor is large with the addition of the kernel which will be entitled to occupy the same high privilege level with KVM, which makes hypervisor at risk.

Xen [1] is another kind of hypervisor which uses paravirtualzation technology. The hypervisor also places a higher privilege than the guest OS, which makes Xen hypervisor directly access to the hardware instead of the general OS. In Xen architecture, we can see that each guest OS is contained in guest domain and there is a special guest domain, termed domain0, which manages and controls the access from guest virtual machine to system hardware. As a result, once the domain0 is at risk by malicious codes, other guest domains will also face security threat.

Blue Pill [10] is a root-kit based on x86 hardware virtualization extensions like AMD-V and Intel VT-x. Its concept is to trap a running instance of the operating system by starting a thin hypervisor and virtualizing the rest of the machine under it. The infected operating system would still maintain its existing references to all devices and files. Our approach is designed based on Blue Pill project and modifies it to be used as an operating system security monitor ported to Windows and Linux. There is an assumption that the resistance to the attack similar to Blue Pill root-kit is not our work, which needs to take advantage of nested virtualization mentioned in Muli Ben-Yehuda's paper [2].

Overshadow [5] which also can be implemented by VASP provides protection of the privacy and integrity of sensitive application data. Overshadow presents an application with a normal view of its resources, but the OS kernel and other programs with an encrypted view [5]. Overshadow has to intercept all the possible entries of kernel, context switches and hardware interrupts, which induces inevitable TCB growth and unnecessary overhead comparatively.

BitVisor [11] is a thin hypervisor for enforcing I/O device security. It uses a hypervisor architecture, called parapass-through, designed to minimize the code size of hypervisors by allowing most of the I/O access from the guest operating system (OS) to pass-through the hypervisor, while the minimum access necessary to implement security functionalities is completely mediated by the hypervisor. Although this approach is effective for protecting system security, it only aims at the security functionalities of I/O access.

Some previous researches, like Proxos [13] and iKernel [14], provide a high secure execution environment and enable secure code running in the separate VMs. Proxos allows applications to configure their trust in the OS by partitioning the system call interfaces into trusted and untrusted components [3], but needs code modification to both application and kernel. For the same purpose, iKernel isolates the malicious device drivers running on separate VMs to make the operating system more secure and reliable. Focusing on the design goal of making hypervisor based monitoring on malicious behaviors handle some sensitive instructions, our approach also possesses the features of high performance and little effect on the original OS kernel with the optimized design.

3. DESIGN AND IMPLEMENTATION

VASP implements hypervisor based monitoring for popular operating systems, such as Windows XP and Linux, with some challenges of no modification to OS, universal protection strategy, self-transparency and Multi-core Support. Our approach is tailored from Blue Pill project as the base for implementation, inheriting its features of lightweight, reliability, easily extensibility and configurability. In this section, we describe the architecture of VASP hypervisor at first, then introduce the details of implementations, and show two case studies at last.

3.1 VASP Architecture

Intercepting the malicious behavior with configurable function is the basic and key point to protect and monitor the whole system. VASP leverages this mechanism to intercept the sensitive behaviors, like CR registers change, debugging interrupts, special memory access, etc., which common attackers use to harm the operating system.

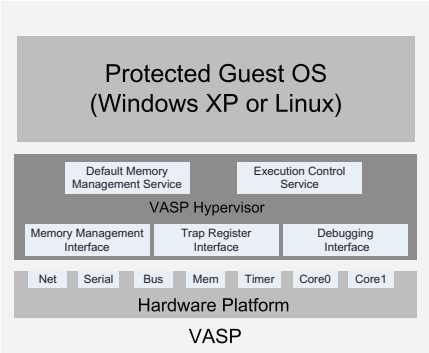

Figure 1: VASP Architecture: An overview of VASP architecture and it consists of three layers.

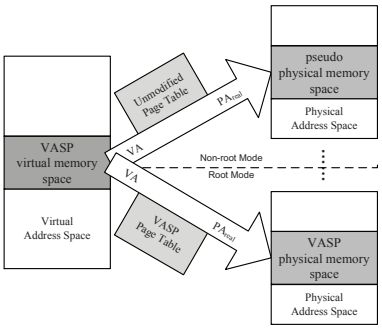

Figure 2: Translation with VASP Memory Strategy: The different memory translation with memory self-transparency strategy between non-root and root mode. The upper one is for guest OS, and the lower one is for VASP hypervisor.

Figure 1 provides an overview of VASP architecture. VASP is divided into three layers: hardware platform layer, VASP hypervisor layer and guest operating system layer.

Hardware Platform Layer. This layer is the platform where the hypervisor is established on. This platform needs to support hardware virtualization technology, such as Inte VT-x and AMD SVM, though they have some differences in hardware design. These differences, involving virtualization-related instructions, checking platform, VMCS configurations, etc., can be masked by platform-related codes to provide unique interfaces to the hypervisor.

VASP Hypervisor Layer. This layer is the core of the whole platform. There is a set of interfaces exported to the future developers, such as memory management interface, trap register interface and debugging interface. Memory management interface is used to realize its own memory management of VASP. Trap register interface supports the extension usage of VASP and provides the configuration of intercepting behavior. Debugging interface is used for dynamic analysis when developing the hypervisor. With the help of these three interfaces, we also implement two basic services for the hypervisor to protect the guest: default memory management service and execution control service. This memory management service realizes the memory self-transparency strategy for the self-protection of the hypervisor. Execution control service contains all the trap event handlers, and it handles the corresponding event according to the indicated #VMEXIT reason.

Guest Operating System Layer. This layer contains the protected operating system, Windows XP or Linux. With the hypervisor established, original operating system executes in the guest virtual machine monitored by the hypervisor. VASP is realized to support only one guest machine at one moment currently.

3.2 Implementation

Next, we will introduce the detailed implementation of VASP, such as its control flow, memeory self-transparency and Multi-core support.

3.2.1 VASP Control Flow

According to the VASP architecture described in the last subsection, VASP hypervisor plays a role as virtual machine monitor between the guest operating system and hardware environment. There is a flow of control for handling a sensitive system behavior interception, involving guest application or guest kernel, hardware and the VASP hypervisor. The trap occurs in step 1, and the control is transferred to hardware with the generation of a #VMEXIT event, such as cpuid instruction. In step 2, the hardware stores the guest execution states in virtual machine control structure (VMCS) and finds out the handler of all the traps by VMCS which is configured when initializing VASP hypervisor. In step 3, VASP hypervisor saves the contents of all applications or kernel registers at trap point, then selects the trap handler routine to handle the corresponding behavior properly. After the trap handling, hypervisor passes the control back to hardware by executing the #VMRESUME instruction and restoring the saved registers. In the last step, the return instruction pointer (IP) and stack pointer (SP) registers are modified by hardware in accordance with the return state stored in step 2, and hardware transfers the control to the intercepted point finally.

3.2.2 Memory Self-Transparency

Our approach is in the form of a driver or module which possesses the highest privilege to execute the special virtualization instructions, so we need the corresponding APIs of memory management to build the memory space of hypervisor. As a result, the hypervisor can be accessed by kernel, and it is vulnerable with no self-protection strategy. To improve the security of VASP itself, the memory self-transparency strategy is essential to conceal the hypervisor. Figure 2 depicts our memory self-transparency strategy for VASP hypervisor.

After the hypervisor is built, it applies for another page of memory and clones the kernel page table to this allocated memory space, which will be provided as the page table when executing in the hypervisor. Next, we need a piece of spare memory space, used as pseudo memory space of hypervisor, and modify the physical address of hypervisor in the original kernel page table to point to the physical address of pseudo memory. As a result, the application or kernel in the guest operating system can only access the physical address of pseudo memory when using the virtual address of hypervisor, but hypervisor can access its real physical address using the same virtual address when it executes in the root mode.

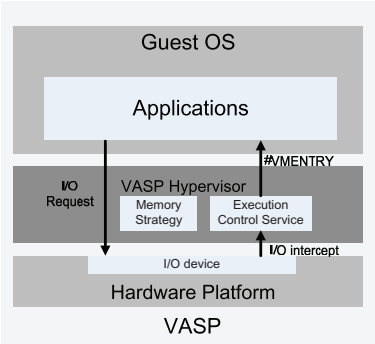

Figure 3: VASP I/O Monitoring Case: VASP intercepts I/O input behavior and protects its data.

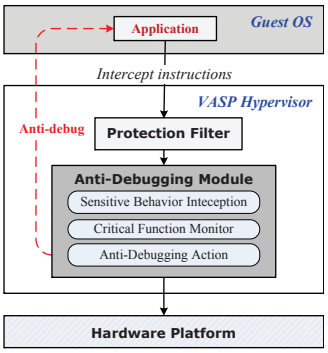

Figure 4: VASP Anti-Debugging Case: VASP prevents anti-debugging behavior

3.2.3 *Multi-core Implementation*

Multi-core processor becomes more and more popular in the current computer world, and occupies most part of marketplace, so realizing the hypervisor to monitor the multi-core system is significant and necessary. We use the affinity APIs, like KeSetSystemAffinityThread(), to establish the hypervisor in each core of processor and unify the memory space of two hypervisor which can work in coordination with each other in monitoring behaviors. Although the multi-core virtualization in Linux operating system has not been implemented yet, it is just a engineering work similar to implementation of Windows. As a result, we use only one core of processor to test the performance.

3.3 Case Study

In order to verify the efficiency and usability of VASP security platform, we show the two case studies of monitoring the system with the help of VASP hypervisor. One is to monitor the I/O resources, and the other is an example of anti-debugging protection.

3.3.1 *I/O monitoring*

VASP also supports monitoring the guest machine on accessing some physical resources such as I/O related instructions. Taking the password input protection for an example, the malicious applications always modify or steal the passwords through hooking some important kernel APIs of I/O related when users are inputting them. Our protection goal is: to intercept the I/O operations from the keyboard when inputting user's password, store them in a protected memory space, and finally give the pseudo data to the I/O buffer.

Figure 3 illustrates the design and usage model of VASP for I/O monitoring. The memory self-transparency strategy is still an important module used to conceal the code and data segment of hypersisor. By reconfiguring the virtual machine control structure of VASP, the hypervisor can intercept each data input of keyboard device before it is transferred to the guest OS. During the hypervisor handling the trap, it will verify the data whether it is a password data for the protected application and then copy it to the safe memory space monitored by hypervisor. Furthermore, VASP will fill old buffer with other pseudo data which can cheat the malicious software to protect the application. At last, with a #VMENTRY event the hypervisor transfers control to the guest. When this application wants to catch this password data, there will be another interception by hypervisor which can give the right password to application. The details of this case is not the key research in this paper.

3.3.2 *Anti-debugging*

Another case study is the anti-debugging protection, which can be used to defend the malicious behaviors with the help of debugging related technology. Debugging is a method which usually facilitates the dynamic program analysis of run-time application for software development. However, as everything has two sides, debugging could also be adopted by attackers. Our protection goal is: intercepting the debugging related instructions, such as INT1 instruction and INT3 instruction, and verifying the malicious behavior, then making this debugging behavior invalid.

Figure 4 shows the design and implementation of VASP used for anti-debugging. Besides the significant and necessary memory self-transparency module, we will use the additional anti-debugging module. The hypervisor first intercepts the sensitive behavior including INT3, INT1 exceptions, CPUID instruction and CR3 conversion which are the features of debugging behaviors. Then it will trace the debugging routines in the system and verify whether debugging is used by other illegal applications. If the result is true, the hypervisor could stop this debugging behavior and pass it to a normal execution flow. The detailed implementation is published as our previous work which shows the anti-debugging protection only in Windows platform.

4. EVALUATION

In this section we present a thorough performance evaluation and analysis of VASP. We begin by benchmarking VASP with macro benchmarks that represent real-life workloads. Next, we evaluate the overhead of virtualization as measured by micro benchmarks which show micro behavior effects. Most of our experiments are executed both in Windows XP and Linux operating system with the similar hardware environment.

4.1 Experiment Setup

We utilize both microbenchmarks and application benchmarks to do the performance test with the desktop computer hardware platform. Our setup consists of two different configurations. One is configured with a dual-core Intel Core2 Duo E6320 and with 2GB DDR2 of memory for Windows

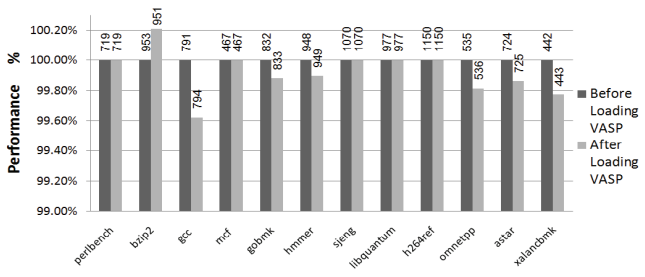

Figure 5: SPEC CINT 2006 Benchmarks for Windows XP. It depicts the percentage of performance overhead relative to the native Windows system, and the number upon each column shows the seconds of run time.

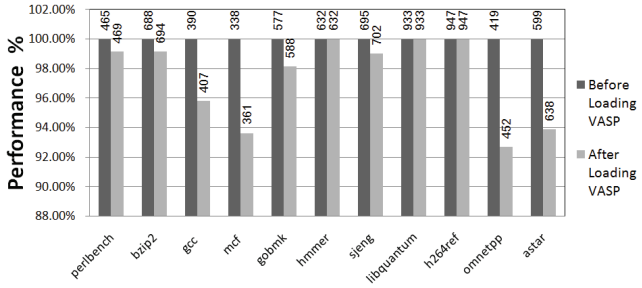

Figure 6: SPEC CINT 2006 Benchmarks for Fedora 12. It depicts the percentage of performance overhead relative to the native Linux system, and the number upon each column shows the seconds of run time.

XP platform. And the other is configured with a single-core Intel Core2 Duo E8400 and with 1GB DDR2 of memory for Fedora 12 (Linux version 2.6.31) platform. Although the hardware environment for each OS is different, our approach shows the same trend of performance overhead. What's more, the distinction of memory size is due to the SPEC requirement that each core need at least 1GB.

4.2 Application Benchmark

We have performed a set of experiments in order to evaluate the overhead of the various operating system relative to running on VASP platform. The SPEC CPU suite contains a series of long-running computationally-intensive applications intended to measure the performance of a system's processor, memory system and compiler quality, but performs little I/O related qualities.

Figure 5 presents the results of benchmarking VASP against monitoring Windows XP from the SPEC CINT2006 suite. The black bar shows the native performance before loading VASP hypervisor as the base of this test. The gray one indicates the real-life workload result after loading VASP relative to the native. Most of the results demonstrate that the overhead of VASP protection is negligible and is increased only about 0.9% in average, because VASP intervenes few operating system behaviors which are configured sensitive instructions and proceedings. Furthermore, the handling process for each trap in the hypervisor is always very quickly. But only bzip2 test shows a lower overhead than the native system, and regrettably the reason of this phenomenon can not be explained by us currently.

Another figure (Figure 6) illustrates the results of performance test when VASP monitors Fedora Linux system from the SPEC CINT2006 suite as well. The black bar and the gray one still represent the same as what in the last figure. The results show that running the hypervisor only brings in a little overhead about 2.6% in average, even if it is based on Linux. More overhead proportion than that in Windows XP is due to the different design architecture, like process scheduling management, memory management and so on. But total time spent in the test is less than the time cost in Windows, because the system processor is more powerful.

4.3 Microbenchmark

To more precisely measure the areas of overhead in Windows and Linux operating system with VASP protection,

	Before Loading VASP	After Loading VASP
Cycles	218	2573

Table 1: Microbenchmarks. Clock cycles of execution CPUID instruction before and after loading VASP hypervisor.

we perform a number of smaller experiments targeting particular subsystems. We compute the CPU execution cycles of CPUID instruction to measure the overhead of processor intercepting in Windows XP. On the other hand, we use McVoy's lmbench program [8] of version 3.0, as this addresses many testing issues, to examine the overhead of VASP protecting Linux system. We present tables for both native Linux (Native) and loading VASP (VASP) results.

In Table 1 we show the results of microbenchmarks running in Windows XP. In this experiment, the benchmark exhibits low performance on executing intercepted instructions on guest machine, nearly 11 times more cycles needed to handle the interception and relevant events after loading VASP. The reason is that trapping into hypervisor introduces extra CPU cycles overhead due to accessing VMCS region, so does invoking the proper callback function.

In the process microbenchmarks (Table 2), VASP exhibits slower fork, exec and sh performance than native Linux's and others are very close. This is expected, since these operations require large number of page table updates which will cause a bit of traps during CR3 conversion. Table 3 shows context switch times between different numbers of processes with different working set sizes. VASP incurs almost a twice larger overhead than native Linux in each test. That is also because context switch accompanied by CR3 switch will be intercepted by hypervisor. The mmap la-

Config	null call	null I/O	stat	open clos	slct TCP	sig inst	sig hndl	fork proc	exec proc	sh proc
Native	0.29	0.45	1.78	2.72	2.76	0.51	1.28	88.9	297	1188
VASP	0.31	0.46	1.80	2.76	2.77	0.53	1.28	95.4	323	1244

Table 2: lmbench: Processes - times in μs.

Config	2p 0K	2p 16K	2p 64K	8p 16K	8p 64K	16p 16K	16p 64K
Native	0.72	1.10	0.85	1.53	1.06	1.52	1.12
VASP	1.91	2.36	2.17	2.98	2.53	2.95	2.46

Table 3: lmbench: Context switching times in μs.

Config	0K File Create	0K File Delete	10K File Create	10K File Delete	Mmap Latency	Prot Fault	Page Fault
Native	11.4	7.1611	18.3	12.7	298.0	0.354	0.65
VASP	11.5	7.1783	18.4	13.0	303.0	0.357	0.67

Table 4: lmbench:File & VM system latencies in μs.

tency and page fault latency results shown in Table 4 are very close. Because VASP protection will not affect the file system and the additional latency caused by hypervisor interception is very small relative to the original overhead.

5. FUTURE WORK AND CONCLUSION

We have presented the architecture and design of VASP, which hosts privilege to implement hardware virtualization based hypervisor running transparently under the guest machine and supporting cross-platform protection to the guest OS without any modification to the existing OS.

5.1 Future Work

We believe that VASP is useful and efficient to monitor and protect the target system. After the initial release we plan a number of extensions and improvements to VASP. We will popularize this platform to a heterogeneous multi-core system which supports a special designed guest OS to protect other normal guest machines. What's more, we can extend our approach to realize more security applications, besides I/O monitoring and Anti-debugging protection mentioned above. In addition, we reserve the memory management interface, so the hypervisor can build its own memory space and manage page table by itself, or add EPT support for memory protection.

5.2 Conclusion

VASP provides an excellent platform based on hardware virtualizaiton technology for the cross-platform security protection to common operating system, with the features of lightweight, transparent and extension capability. VASP supports to intercept the sensitive instruction and behavior to judge whether they are harmful to operating system, and implements memory self-transparency strategy to protect hypervisor itself. As our experimental results show in Section 4, the performance overhead of VASP hypervisor is practically equivalent to the performance of the native operating system on both Windows and Linux platforms.

6. ACKNOWLEDGEMENT

This work is supported by National Natural Science Foundation of China (Grant No.60773093, 60873209, 60970107), the Key Program for Basic Research of Shanghai (Grant No.09JC1407900, 09510701600, 10511500100), and IBM SUR Funding and IBM Research-China JP Funding.

7. REFERENCES

[1] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. L. Harris, A. Ho, R. Neugebauer, I. Pratt, and A. Warfield. Xen and the art of virtualization. In *SOSP*, pages 164–177, 2003.

[2] M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The turtles project: Design and implementation of nested virtualization. In *USENIX Symposium on Operating Systems Design and Implementation*, Vancouver, Canada, 2010.

[3] I. Burdonov, A. Kosachev, and P. Iakovenko. Virtualizationbased separation of privilege: working with sensitive data in untrusted environment. In *In Proceedings of the 1st EuroSys Workshop on Virtualization Technology for Dependable Systems*, 2009.

[4] X. Chen, J. Andersen, Z. M. Mao, M. Bailey, and J. Nazario. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *DSN*, pages 177–186, 2008.

[5] X. Chen, T. Garfinkel, E. C. Lewis, P. Subrahmanyam, C. A. Waldspurger, D. Boneh, J. S. Dwoskin, and D. R. K. Ports. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. In *ASPLOS*, pages 2–13, 2008.

[6] P. H. Gum. System/370 extended architecture: facilities for virtual machines. *IBM Journal of Research and Development.*, 27(6):530–544, 1983.

[7] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. Kvm: the linux virtual machine monitor. In *Proceedings of the Linux Symposium*, 2007.

[8] L. W. McVoy and C. Staelin. lmbench: Portable tools for performance analysis. In *USENIX Annual Technical Conference*, pages 279–294, 1996.

[9] A. Padegs. System/370 extended architecture: design considerations. *IBM J. Res. Dev.*, 27(3):198–205, 1983.

[10] J. RUTKOWSKA. Subverting vista kernel for fun and profit. In *Blackhat*, 2006.

[11] 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. Bitvisor: a thin hypervisor for enforcing i/o device security. In *VEE*, pages 121–130, 2009.

[12] J. Sugerman, G. Venkitachalam, and B.-H. Lim. Virtualizing i/o devices on vmware workstation's hosted virtual machine monitor. In *USENIX Annual Technical Conference,General Track*, pages 1–14, 2001.

[13] R. Ta-Min, L. Litty, and D. Lie. Splitting interfaces: Making trust between applications and operating systems configurable. In *OSDI*, pages 279–292, 2006.

[14] P. Tan, E. M. Chan, P. Farivar, N. Mallick, J. C. Carlyle, F. M. David, and R. H. Campbell. ikernel: Isolating buggy and malicious device drivers using hardware virtualization support. In *In Proceedings of the Third IEEE International Symposium on Dependable, Autonomic and Secure Computing*, 2007.

[15] C. A. Waldspurger. Memory resource management in vmware esx server. In *OSDI*, 2002.