VNIDA: Building an IDS Architecture Using VMM-based Non-intrusive Approach

Xiantao Zhang[1 2], Qi Li[2,3], Sihan Qing[4], Huanguo Zhang[1,5]
[1]*School of Computer, Wuhan University, Wuhan, Hubei, 430079, China*
[2] *Intel Open source Technology Center, Shanghai, 200240, China*
[3]*Department of Computer Science, Tsinghua University, Beijing 100084, China*
[4]*Institute of Software, Chinese Academy of Sciences, Beijing 100080, China*
[5]*State Key lab of Software Engineering, Wuhan, Hubei, 430079, China*
E-mail: xiantao.zhang@gmail.com Phone: +86-13817663035

Abstract

Intrusion detection system (IDS) has been introduced and broadly applied to prevent unauthorized access to system resource and data for several years. However, many problems are still not well resolved in most of IDS, such as detection evasion, intrusion containment. In order to resolve these problems, we propose a novel flexible architecture VNIDA which is based on virtual machine monitor (VMM) and has no-intrusive behavior to target system after studying popular IDS architectures. In this architecture, a separate intrusion detection domain (IDD) is added to provide intrusion detection services for all virtual machines. Specially, an IDD helper is introduced to take response to the intrusions according to the security policies. Moreover, event sensors and IDS stub, as the core components of IDS, are separately isolated from target systems, so strong reliability is also achieved in this architecture. To show the feasibility of the VNIDA, we implement a prototype based on the proposed architecture. Based on the prototype, we employed some rootkits to evaluate our VNIDA, and the results shows that VNIDA has the ability to detect them efficiently, even some potential intrusions. In addition, system performance evaluation also shows that VNIDA only introduce less than 1.25% extra overhead.

1. Introduction

The pioneering work of intrusion detection models is proposed by Anderson [1] and Denning [2] in the 1980s. In their papers, they provide a general intrusion detection framework and establish the theoretical foundation for intrusion detection system (IDS), and they also present formal definitions of IDS. According to their definition, as an expert system, IDS is able to infer from event logs occurring on a system that has been or is being compromised. Currently, since computer systems become more and more complex, there are a variety of places where intrusion detection is possible. For example, the analysis of network traffic may indicate an attack in progress, a compromised daemon may be detected by its abnormal behavior, and subsequent attacks may be prevented by backdoor detection [19] and stepping stones [20].

Generally, there are two types of intrusion detection systems, network-based intrusion detection system (NIDS) and host-based intrusion system (HIDS). NIDS is implemented and applied at any place of the whole network and has much resistance to attacks, since it is separated with target system. Under such condition, they are not affected and compromised by security vulnerabilities on monitored target systems. However, this architecture still has some important shortcomings stemming from its intrinsic design, such as poor view of events occurring inside the target systems, limited data for intrusion analysis, disability to detect local root attacks. Oppositely, Host-based architecture is deployed as a monitor in target system, to collect data and use them to identify potential intrusions. HIDS is enabled to gain good view of what is happening in target system where it is deployed. However, several problems might occur due to the easy access to attack. First of all, Garfinkel [3] describes the difficulties encountered by security tools that depend on system call interposition when a HIDS monitors the target system. This implementation technique may introduce new potential vulnerabilities and enable some potential

* This work was supported by the National Natural Science Foundation of China under Grant No. 60673071, No. 60573042, National High Technology Development 863 Program of China under Grant No. 2006AA01Z442 and the National Grand Fundamental Research 973 Program of China under Grant No.G1999035802.

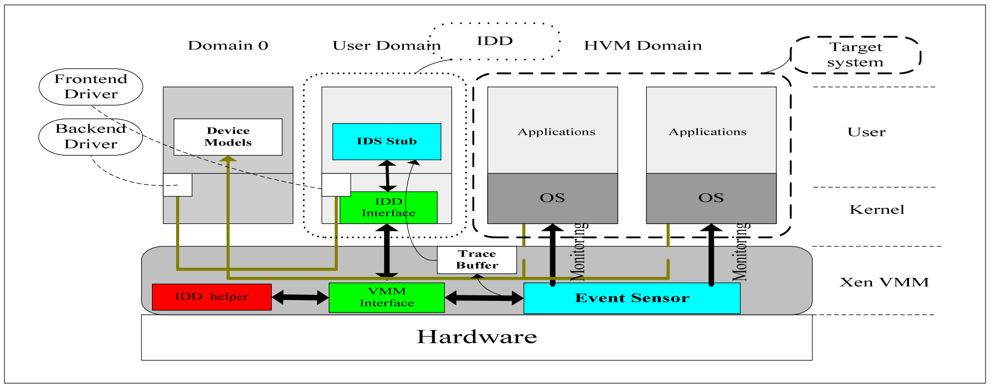

Figure 1: Overview of *VNIDA* in Xen Environment

attacks. In order to address these problems emerging from Host-based and Network-based architectures, we proposed a VMM-based [8, 11, 12] non-intrusive architecture called *VNIDA* in this paper. Our *VNIDA* architecture not only offers a flexible and easy way to detect, and even make active response to attacks, but also has some outstanding features such as good visibility, strong reliability and isolation, high efficiency and no intrusion to target system that can not be achieved in traditional IDS architecture. Furthermore, many existing intrusion detection solutions can be easily integrated into *VNIDA*. To show the feasibility of *VNIDA*, we implemented a *VNIDA* prototype and the evaluation result shows that *VNIDA* can detect the intrusions effectively. Meantime, the performance study demonstrates that *VNIDA* only introduces less than 1.25% extra overhead.

The remainder of the paper is structured as follows. In Section 2, we illustrate the new architecture and explain the mechanisms of its main components. Section 3 lists the advantages which are presented by our *VNIDA*. During the discussion, the limitations existing in Host-based and Network-based architectures are addressed respectively. In Section 4, we first present one implementation for *VNIDA*, and then the performance impact of our *VNIDA* is evaluated through the prototype system we developed. Section 5 discusses the related work. Finally, Section 6 concludes the paper.

2. *VNIDA* Architecture

In order to fully understand VMM-based Non-intrusive Intrusion Detection Architecture (*VNIDA*), we implement a proof-of-concept IDS based on *VNIDA* using Xen VMM [12, 16]. The illustration of the entire architecture and the presentation of its main components are made in Figure 1 according to Xen VMM architecture.

As Figure 1 illustrated, the *VNIDA* is composed of 5 components VMM, IDD, Event Sensor (ES), IDS Stub (IS) and IDD Helper (IH). In order to better understand the work flow of the *VNIDA* architecture, we will describe each component in details in this section.

2.1 VMM

In recent years, with the advancement of virtualization, the concept of VMM is becoming more and more popular. In the field of security research, VMM is also involved in many research projects [4, 14, 17], and plays an important role with its benefits such as isolation, flexibility, and efficiency. As for its definition, a virtual machine monitor [7, 9, 12] is a thin software layer for a computer system that creates efficient, isolated virtual machine environments that are identical to underneath real hardware machine. Thus, a physical machine can be converted to a set of virtual machines and different virtual machines load different copies of operating system, and each copy is totally isolated from the others. Since VMM runs at the most privileged ring and owns entire real machine, it is capable of inspecting arbitrary behavior of virtual machine.

In our *VNIDA*, as the basis of the entire architecture, VMM is responsible for managing virtual machines, providing a communication channel for virtual machines. In addition, it is also the container of event sensors and IDD helper.

2.2 Event Sensor

As shown in Figure 1, Event Sensor is described as a separate module in VMM, so it has same privileged properties with VMM, such as controlling virtual machine, inspecting the memory of virtual machine. Typically, it is implemented as a set of sensors that are designed to monitor the states of target system. For instance, a system call sensor can be implemented and deployed to acquire the sequence of system calls occurring in target system. Here, we generally design different sensors to collect various data for detecting intrusions or to acquire information about attackers. Moreover, Event Sensor also need expose some interfaces to communicate with other components. In addition, since event sensor is entirely implemented in VMM, our *VNIDA* has no intrusive behaviors to target system for collecting states of guest system, unlike Host-based architecture that generally needs to implement hooks or interceptor inside kernel of target system.

2.3 Intrusion Detection Domain (IDD)

In our *VNIDA*, we use a dedicated virtual machine to implement IDD. In the Xen environment, a para-virtualized user domain [12] is used to achieve this goal. With an eye to security, IDD should be implemented as a simple system with short network support, minimal kernel and root file system.

In the IDD, IDS stub and well-defined IDD interfaces are presented. The IDD interfaces works with VMM interfaces together to provide specific communication channels between VMM and IDD. In general, the interfaces have two functionalities. The first one is that, when event sensor acquires some data from target system and stores them into the trace buffer, it needs to send a notification to IDD through interfaces and let IDD fetch them from trace buffer for further analysis. The notification mechanism relies on these well-defined interfaces. The other one is that, if IDS stub wants to adopt some prevention actions to control compromised system, it may notify IDD helper in VMM to dominate the target system through these interfaces.

2.4 IDS Stub

IDS stub, as the core component of intrusion detection systems, is responsible for interpreting the data of target system, logging intrusion report, reporting potential attacks, and responding to the attacks. Since IDS stub is deployed in the dedicated IDD, strong isolation is provided between it and target systems. Furthermore another big advantage is the response mechanism provided by it. Generally, if target systems have been compromised, IDS can not adopt preventive rules to control target system except for issuing alarms. But it is possible to carry out prevention policies to control the compromised target system through response mechanism in virtual machine environment. For example, IDS stub can pause or reboot target system to prevent the attacks once the system is compromised. Besides, flexible policies can be developed to provide fine-grained intrusion detections in IDS stub.

2.5 IDD helper

As to IDD helper, it is deployed in VMM as a separate module. Its responsibility is to help IDS stub to complete prevention actions. When IDS stub detects potential attacks, system administrators often hope that the helper will adopt some prevention actions to avoid leaking data or resource from the security viewpoint automatically. In this case, IDS has the ability to freeze or reboot the compromised target system with the help of IDD helper according to relevant policies.

To sum up, these components described above need work together with the entire virtual machine environment. In the Xen environment, the other components of Xen infrastructure, such as the domain0, HVM domain and User domain[1], are used to build the holistic architecture.

3. Discussion

As described in Section 1, it is very hard to conquer or mitigate the disadvantages in traditional IDS architectures, because these shortcomings stem from inherent function limits of architectures. Our *VNIDA* architecture not only resolves the problems in traditional IDSes, but also provides many attractive features which can not be achieved in previous architectures.

- ***Good visibility.*** Network-based architecture has generally limited functions because of its poor view of target systems. *VNIDA* is based on virtual machine monitor, and VMM has the ability to access whole real machine system, including CPU, memory, I/O devices etc. Therefore, it is easy to inspect the arbitrary states of the monitored target system which is run in guest virtual machine. In addition, the platform hosting virtual machine is entirely virtualized and managed by VMM, so all states of target virtual machine can be acquired by VMM at any moments. Thus, *VNIDA* has a good view of what is happening in target system.
- ***Strong reliability and isolation.*** Because Host-based architecture is based on host system, it often suffers the reliability and isolation issues. As the basis in *VNIDA*, VMM has higher

[1] HVM domain and User doman are the Xen terms, and are used to differential two virtualziation approaches.

assurance to protect components in different systems, because it is just a relatively simple kernel with limited functionality and a narrowly well-defined interface to the software run on it. Unlike traditional operating systems, which must support file system, network protocol stacks, etc., a VMM only need present relatively simple abstractions, such as a virtual CPU and memory. Thus, strong reliability of *VNIDA* is achieved due to the reliable basis provided by VMM. In addition, *VNIDA* clearly isolate target system from intrusion detection system which is implemented in a separated virtual machine system, and target system can not access or control the IDS components in any way.

- ***High efficiency.*** Traditional Host-based architecture generally undergoes large performance decrease because it adds many extra mechanisms to trace system states. Although *VNIDA* needs some additional efforts to implement the virtual machine environment, but experience with virtual machine monitors over the past 30 years shows the overhead introduced by virtualization is negligible [6]. Moreover, the dedicated monitor mechanism of event sensor in VMM is more efficient than complex purposeless trace approach, and the total overhead introduced by *VNIDA* is much less than Host-based IDS architecture.
- ***No intrusion to target system.*** In Host-based architecture, in order to acquire more information from system, intrusion detection system relies on inserting modules into OS kernel, but this will damage the integrity of target system. In the *VNIDA* side, it utilizes VMM to monitor target systems and acquire the states of target system and VMM has the ability to access any resource of real and virtual system, so it is not necessary to add portions to target system. From the viewpoint of system integrity, non-intrusive implementation in *VNIDA* makes more sense than that in Host-based architecture.

Through above discussion, we can see that *VNIDA* benefits more than Host-based architecture and Network-based architecture.

4. Implementation and Evaluation of Prototype System

In this section, we implemented a proof-of-concept system of the proposed *VNIDA* based on Xen/IA64 project. There are three reasons for us to choose Xen VMM as the experiment system: (1) Xen is a popular VMM, and has been developed and applied to more and more areas in the real world. Leveraging Xen to implement the *VNIDA* is closer to practicality, which a target system normally comes up against. (2) Xen VMM provides enough visibility to virtual machines, and is capable of monitoring all the status of virtual machines through well-defined interfaces. For example, the extension domain of Xen VMM, domain0, has the ability to access the whole memory range of all virtual machines through a libxc library provided by the Xen framework. (3) In addition, since Xen is an open source virtual machine, it is very convenient for us to implement all the components of *VNIDA*, log system events occurring in the monitored target VM system, and dump the context information related to the corresponding events.

4.1 Xen VMM

In our proof-of-concept system, Xen hypervisor is used as VMM. Thus, we first briefly introduce the Xen project in this section. Xen is an open source virtual machine monitor, and initiated by Cambridge Computer lab, and targets for supporting execution of multiple guest operating systems with unprecedented levels of performance and resource isolation. So far, Xen VMM supports many platforms, such as x86, IA64, Power PC, Arm and so on. Xen also provides secure partitioning between virtual machines. Xen supports one privileged virtual machine, called domain 0, which has access to a control interface provided by Xen. In addition, Xen also supports some User domains, which have less privileged than domain 0. In our prototype system, we directly use a user domain to implement the intrusion detection domain for analyzing intrusions, and deploying flexible policy engine. But considering the fact that IDD require restricted security requirement, we tailored a user domain kernel for implementing IDD. The kernel of IDD only keeps the minimal requirement for run, without network core architecture support, and device drivers except VBD front-end driver, and so on.

4.2 Intercepting System calls

In this section, we will detail the implementation of event sensor, as the core component of *VNIDA*, and study how it works to perform collection for system information in our prototype system. Confessedly, through analyzing the sequences of system call of a special process, possible intrusions can be identified out. This approach has been invited in and applied to a lot of research projects [3, 5, 13]. In our *VNIDA* prototype, we leverage this technique for IDS stub. Therefore, the major work of our implementation is to intercept dynamic behaviors (the sequences of system call) of processes in the guest virtual machines.

To better understand how we achieve the system call sequence of processes without any intrusion to target system, we have to firstly understand the details of system call mechanism implemented in IA64 Linux OS. Here, we briefly summarize it as follows:

(1) In IA64 Linux System, user application can leverage two paths to request system calls. The first one is using the *break* instruction which traps to break vector of system IVT table with a special *immediate value* 0x100000, and then OS can identify out the system calls according to its arguments passed by user application. The second one approach is also using a special instruction *epc* which is capable of promoting system privilege level smoothly form ring 3 to rings 0 without breaking in the pipeline of CPU. More specifically, this instruction can be executed in ring 3, but the page including this instruction must have special access rights for execution. Therefore, user-level application or library should cooperate with OS kernel to achieve this goal. In IA64 Linux system, they are implemented through a gate page with a fixed address for *epc* instruction.

(2) In Xen/IA64, considering the fact that HVM domains are running in the guest mode, we can insert some callback functions in special paths to get corresponding system states in VMM. To intercept system calls of OS in HVM systems, VMM can identify them by tracing *break* and *epc* instructions.

(3) Intuitively, we can maintain a single data structure for each HVM domain in VMM to trace system call sequence. Unfortunately, it is not feasible to modern OS kernels, because the multitasking and SMP support are provided and can run tasks simultaneously. Specially, in our *VNIDA* prototype system, *multi*-HVM domains can run in the same time. As such, we have to provide a mechanism to handle this case. IA64 Linux OS use the kernel register 6 (kr6) to store the pointer which specifies which process running on the current processor. In our implementation, we use this information provided by kr6 to identify system calls from different processors.

To capture all system call events in all target systems, *VNIDA* leverages Xen domain ID (VM-id) to distinguish HVM domains. Here, we propose a 5-Tuple (VM-id, Pid, P-name, Syscall-Num, Args) to wrap all system calls information from event sensor.

- ***VM Identity (VM-id)***: It is a unique identifier in Xen VMM, and is used to identify different virtual machines in Xen world. In Xen execution, Xen VMM is responsible for allocating and de-allocating VM-id at VM creation time. So, we can leverage it to mark different virtual machines.
- ***Process ID (Pid)***: We use this field to differential processes. In our *VNIDA*, it can be achieved through the task_struct pointer stored in kr6. More specifically, Linux consolidates three different data structure (the basic task struct, thread_info, and process kernel stack) into a 32K-aligned storage area for each process, and stores the base address in kr6. As described in Figure 2, in task_struct data structure, one field called pid is marked as the processor id. Therefore, Xen VMM can get pid information through its interface *copy_from_guest* which use value of kr6 as argument.

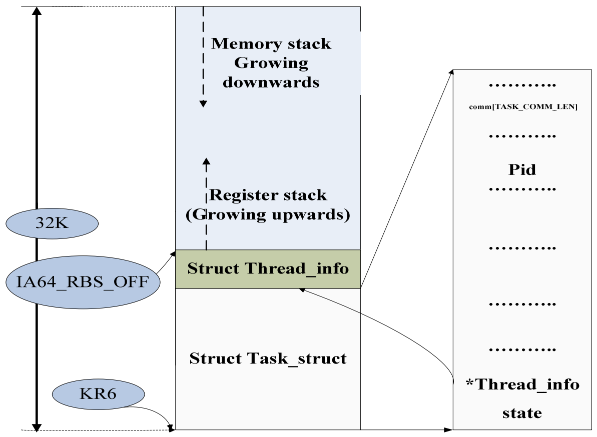

Figure 2: Task_struct data structure of process in IA64 Linux OS

- **Process name (P-name)**: It is used to generate the final detection report. Process name is also gotten from task_struct's field (char comm[TASK_COMM_LEN]). Similar to process identity, process name is also gotten from the interface *copy_from_guest*.
- ***Syscall Number (Syscall-Num)***: System call number. In Linux OS, system call number is used to identify system calls, and defines all system call numbers in the head file asm/unistd.h. In IA64 Linux OS, software call convention uses scratch register r15 to encode system call number. In the *VNIDA* system, when event sensor detects possible syscall execution through *break* and *epc* instruction tracing, it can get the system call num by reading general register r15.
- ***Arguments (Args)***: System call arguments. Since we just care about the sequence of system call, we do not use arguments to detect possible intrusion. But we need put them as important audit log information stored in IDD stub. System administrators can use these information to confirm detected intrusions. In *VNIDA*, Xen VMM gets these arguments from the register stack (r32,

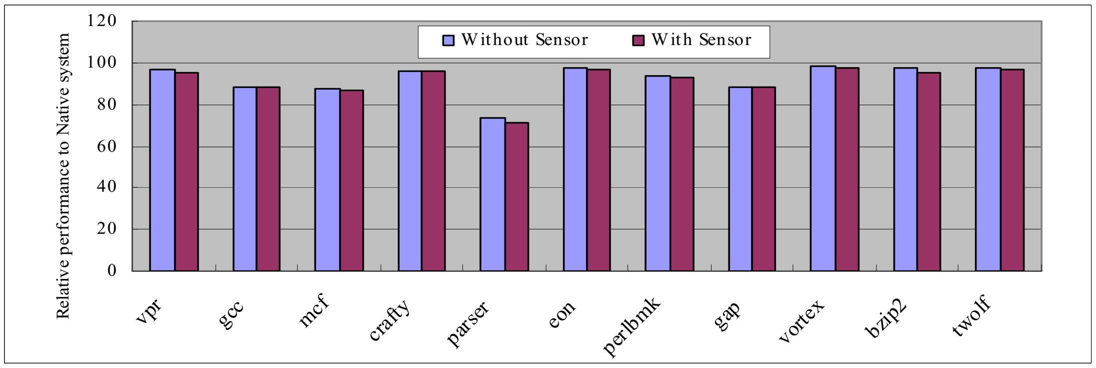

Figure 3: Overhead imposed by the runtime event sensor. All results are relative performance to native system in SPEC INT2000 benchmark results

r33, r34 …..) through a well-defined interface get_rse_reg.

Event senor is used to collect all above information organized as the 5-tuple, and send to the trace buffer through pre-defined interfaces. Subsequently, IDD can fetch them from the trace buffer and analyze the possible intrusions in time.

4.3 Evaluation

To evaluate the effectiveness and performance overhead of our proposed architecture, we implemented a *VNIDA* prototype using Xen/IA64 [8, 16]. As described in Figure 1, we use HVM domains supported by Intel ® Virtualization Technology [15] to run target systems, and one para-virtualized user domain to host the IDD.

Firstly, we employed some rootkits, such as *ARK*, *Adore toolkits*, and *Knack* [22], to test its effectiveness. The experiment result demonstrates that it gets the similar effectiveness with traditional sys-call tracing approach. In addition, we also implement a kernel monitor sensor to scout the modification of kernel sys-call table, and our *VNIDA* also efficiently detect the intrusions. Thus, our *VNIDA* has enough strong adaptability as implemented through different detection approaches.

Secondly, according to the architecture of *VNIDA*, the system overhead stems mainly from event sensors for target system, we deployed the micro benchmark SPEC CPU2000, which is compute-intensive to show the performance impact in two situations (with or without event sensor) in general computation environment. The detailed performance data in both situations is illustrated in Figure 3[1].

Under each situation, their performance data is compared with that of native system using same configurations. As Figure 3 shows, the extra overheads introduced by the system call sensor in *VNIDA* basically range from 0.1% to 1.25% in every test suite of SPEC INT2000 [21]. This performance decrease mainly stems from the overhead of monitoring system call sequence of target system and communication between target system and IDD. It shows that the VNIDA architecture has little side impact for performance, and is acceptable for real applications. Our experiment environment is described as follows. Native system and HVM system are all hosted in a Linux RHEL4U3 system and with Intel Tiger4 Platform, configuring 1.6G processors X 4 and 20G main memory. In addition, Cset12014 of Xen/IA64 source code [19] is used to build the virtualization environment.

5. Related Work

There are a number of documented studies [4, 14, 17] that investigate virtual machine monitor for security research. Dunlap *et al.* conduct a study in *ReVirt* [4] which is based on a virtual machine monitor and logs sufficient information to replay exactly the execution in a virtual machine. This allows fine-grained examination and leads to a better understanding of security issues. *Revirt* is not used to detect potential intrusions, but we can integrate its idea into *VNIDA* to provide a complete intrusion detection solution with fine-grained examination. Garfinkel *et al.* describe Terra, a Trusted Virtual Machine Monitor (TVMM) prototype [17], which focuses on security, assurance, and attestation in relation to virtualization. Thus, their work presents a trusted computing base (TCB) model for virtual environment, and all researches based on VMM may leverage it to enhance their security bases. In addition, Garfinkel *et al.* propose Livewire [14] which is also a VMM-based IDS. Livewire has the similar architecture with *VNIDA*. The major difference between these two architectures is that they adopt different approaches to implement

[1] In oder to compare the performance in different cases clearly, we assume that native system gains 100% of performance.

IDS stub (policy engine in Livewire). *VNIDA* leverages a dedicated virtual machine to build IDS stub, and this virtual machine is generally implemented with strict security policies to ensure its own security, while Livewire provides its policy engine in the host system. From this point of view, *VNIDA* provide stronger isolation between target system and IDS stub. In addition, Zhang *et al.* [18] also examine the effectiveness of secure coprocessor-based intrusion detection. In their study, IDS is directly run on a coprocessor other than host system, and this architecture provide many similar features with our architecture, such as good visibility, isolation. Our *VNIDA* achieves the same ability through the virtual machine monitor.

6. Conclusion

In this paper, we proposed a VMM-based intrusion detection architecture called *VNIDA* which is intended to address the shortcoming of traditional IDS architectures. Subsequently, we examine the components of *VNIDA*, and analyze benefits achieved in *VNIDA*, such as good visibility, strong reliability and isolation, high efficiency and no intrusion to target system. In addition, in order to further evaluate the performance impact to target system in *VNIDA*, we implement a prototype based on proposed *VNIDA* and analyze the performance in guest domain in *VNIDA*, and the result shows *VNIDA* introduces less than 1.25% extra overhead in the experiment of SPEC INT 2000 with highly efficient detection.

Many aspects of the architecture can be extended in the future. For example, we can leverage Trusted Computing technology to build Trust VMM [17] and further secure the key components in the architecture.

7. Acknowledgment

The authors gratefully acknowledge members from Intel Open-Source Technology Center. They provided us the experiment platform, and gave us lots of technical assistances. In addition, we also thank reviewers of our paper for their valuable suggestions.

8. References

[1] J.P Anderson. Computer Security Threat Monitoring and Surveillance. *Technical report*, James P Anderson Co., Fort Washington, Pennsylvania, April 1980.

[2] D.E Denning. An Intrusion Detection Model. *IEEE Transactions on Software Engineering*, Number 2, page 222, February 1987.

[3] T.Garfinkel. Traps and pitfalls: Practical problems in system call interposition based security tools. *In Proceedings of the 10th Annual Symposium on Network and Distributed System Security (NDSS 2003)*, February 2003.

[4] G.W Dunlap, S. T King, S.Cinar, M.Basrai, and P. M Chen. Revirt. Enabling intrusion analysis through virtual-machine logging and replay. *In Proceedings of the 5th Symposium on Operating Systems Design and Implementation (OSDI 2002),* December 2002.

[5] A P Kosoresow,SA Hofmeyr. Intrusion detection via system call traces. IEEE Software,1997,14 (5) :35 -42.

[6] R..Howworth. Virtualservers pay off. *ITWeek*, March 2003.

[7] R. J.Creasy. TheoriginoftheVM/370 time-sharing system. *IBM Journal of Research and Development*, 25(5), 1981.

[8] Xen Source Corp. Xen/IA64 project. http://xenbits.xensource.com/ext/xen-ia64-unstable.hg

[9] A.Whitaker, M.Shaw, and S.D. Gribble., "Denali: Lightweight virtual machines for distributed and networked applications," *Technical Report* 02-02-01, 2002.

[10] B. Mukherjee, L T.Heberlein and K.N Levitt. Network Intrusion Detection, *IEEE Network*, May/June 1994, pages 26-41

[11] P.A.Karger,M.E.Zurko,D.W.Bonin,A.H.Mason, and C. E. Kahn. A Retrospectiveon the VAX VMM Security Kernel. *IEEE.Transactions on Software Engineering*, 17(11):1147–1165, November 1991.

[12] P.Barham, B.Dragovic, K.Fraser, S. Hand, T.Harris, A. Ho, R.Neugebauer, I. Pratt, and A.Warfield. Xen and the art of virtualization. *In Proceedings of the ACM Symposium on Operating Systems Principle*s, October 2003

[13] Bry S. A. Hofmeyr, S. Forrest, and A. Somayaji. Intrusion detect using sequences of system calls. *Journal of Computer Security*, 6:151–180, 1998.

[14] T. Garfinkel and M. Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. *In Proceedings of the 2003 Network and Distributed System Security Symposium (NDSS),* February 2003.4.

[15] Intel® Corp, "Intel® Virtualization Specification for the Intel® Itanium® Architecture (VT-i)".

[16] Y. Dong, S. Li, A. Mallick; J. Nakajima, K. Tian, X. Xu, F. Yang, W. Yu, "Extending Xen* with Intel® Virtualization Technology." *Intel Technology Journal.* Oct 2006.

[17] T.Garfinkel, B. Pfaff, J.Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. *In Proceedings of the 19th ACM Symposium on Operating Systems Principles,* Bolton Landing, NY, USA, October 2003.

[18] X.Zhang, L.v Doorn, T. Jaeger, R. Perez, R.Sailer. Secure coprocessor-based intrusion detection. In Proceedings of the ACM SIGOPS European Workshop, September 2002.

[19] Y. Zhang and V. Paxson. Detecting backdoors. *In Proceedings of 9th USENIX Security Symposium,* August 2000.

[20] Y. Zhang and V. Paxson. Detecting stepping stones. *In Proceedings of 9th USENIX Security Symposium,* August 2000.

[21] Standard Performance Evaluation Corporation http://www.spec.org/cpu2000.

[22] http://www.antiserver.it/backdoor-rootkit.