Intrusion Detection in Virtual Machine Environments

Marcos Laureano, Carlos Maziero, Edgard Jamhour
Graduate Program in Applied Computer Science
Pontifical Catholic University of Paraná - Brazil
{laureano, maziero, jamhour}@ppgia.pucpr.br

Abstract

A virtual machine is a software replica of an underlying real machine. Multiple virtual machines can operate on the same host machine concurrently, without interfere each other. Such concept is becoming valuable in production computing systems, due to its benefits in terms of costs and portability. As they provide a strong isolation between the virtual environment and the underlying real system, virtual machines can also be used to improve the security of a computer system in face of attacks to its network services. This paper presents a new approach to achieve this goal, by applying intrusion detection techniques to virtual machine based systems, thus keeping the intrusion detection system out of reach from intruders. The results obtained from a prototype implementation confirm the usefulness of this approach.

1. Introduction

A central problem in system security is the difficulty in getting reliable information from a compromised system. Once an intrusion has occurred, the monitoring data coming from such system is no more reliable, as the intruder can disable or modify the system monitoring tools in order to hide his/her presence.

Virtual machines can be used to improve the security of a computing system against attacks to its services [6]. The virtual machine concept was defined in the 1960s: in the IBM VM/370 environment, a virtual machine created an exclusive environment for each user [11]. The use of virtual machines is becoming interesting also in modern computing systems, because of their advantages in terms of cost and portability [5]. Examples of currently used virtual machines environments are *VMware* [18] and UML – *User-Mode Linux* [7]. A frequent use of virtual machine –based systems is the so-called *server consolidation*: instead of using several physical equipments, one can use a single (and more robust) hardware equipment, in which several distinct, isolated virtual machines host distinct operating systems, applications, and services.

This work presents a proposal to increase the trustworthiness of computing systems using virtual-machine technology. It proposes the application of intrusion detection mechanisms in order to detect and block attacks against services running on virtual machines. The main benefit of this approach is to monitor the virtual machine from outside (from the real underlying system), thus keeping the intrusion detection system safe, out of reach from intruders. This article is structured as follows: section 2 recalls virtual machine concepts used in this work; section 3 introduces some intrusion detection techniques used here; section 4 details the proposal, section 5 presents implementation results, and section 6 discusses related work.

2. Virtual machines

A virtual machine (VM) is defined in [16] as an efficient and isolated duplicate of a real machine. Typical uses for virtual machine systems include the development and testing of new operating systems, simultaneously running distinct operating systems on the same hardware, and server consolidation [17].

A virtual machine environment is created by a *Virtual Machine Monitor* (VMM), also called an "operating system for operating systems" [13]. The monitor creates one or more virtual machines on top of a single real machine. Each VM provides facilities for an application or a "guest system" that believes to be executing on a normal hardware environment.

There are two classical approaches to build virtual machine systems. In *type I* environments, the virtual machine monitor is implemented between the hardware and the guest systems, as shown in Figure 1; *Xen* [2] and *VMware ESX Server* [18] are good examples of such approach. On the other hand, in *type II* environments, the monitor is implemented as a normal process of an underlying real operating system, called the *host system* (Figure 2). Both *VMware Workstation* [18] and *User-*

Mode Linux [7] fit in this category. This article considers the application of type II virtual machine environments in system security.

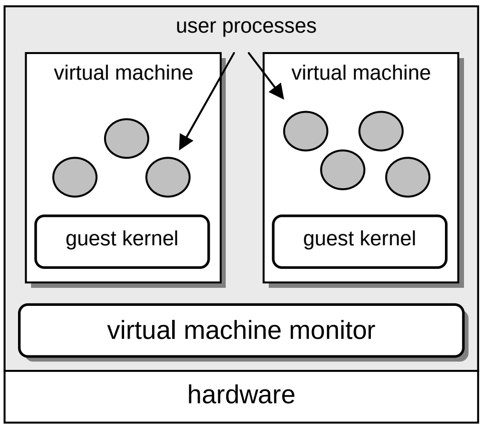

Figure 1. Type I virtual machine environment

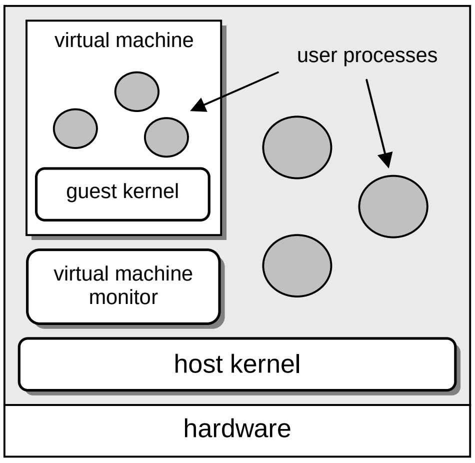

Figure 2. Type II virtual machine environment

Common Intel-like PC processors provide no adequate support for virtualization. Consequently, virtualization overhead can be as high as 50% of total computing time [5, 7, 18]. However, recent research significantly reduced this cost, achieving overhead levels under 10%, as shown in [14, 15, 19]. For instance, VMware [18] adopts a technique of code rewriting that consists of dynamically rewriting parts of the code being loaded by the guest kernel, in order to adapt it to the virtual machine environment and thus obtain a better performance. Recently, the *Xen project* [2] proposed and built a type I virtual machine environment in which average costs remain under 3% for virtualizing Linux, FreeBSD, and Windows XP. These works open many perspectives on the effective use of virtual machines in production environments.

3. Intrusion detection

An *Intrusion Detection System* (IDS) continuously collects and analyzes data from a computing system, aiming to detect intrusive actions. With respect to the origin of analyzed data, there are two main approaches for intrusion detection [1]: *Network-based IDS* (NIDS) – based on watching the network traffic flowing through the systems to monitor, and *Host-based IDS* (HIDS) – based on watching local activity on a host, like processes, network connections, system calls, logs, etc. The main weakness of host-based intrusion detection is its relative fragility: in order to collect system activity data, a HIDS agent should be installed in the machine to monitor. This agent can be deactivated or tampered by a successful intruder, in order to mask his/her presence, turning the detection system useless.

Techniques used for analyzing collected data in order to detect intrusions can be classified in: *signature detection*, when collected data is compared to a base of previously known attacks patterns (signatures), and *anomaly detection*, when collected data are compared to previously collected data representing the normal activity of the system. Normality deviations are then signaled as threats.

Several papers describe techniques for anomaly-based intrusion detection which uses the sequences of system calls generated by processes. In the proposal presented in [9, 12], the system calls issued by a process are recorded in sequence, without their parameters. This *execution history* is then transformed in sequences of system calls of length *k*. The collection of all possible sequences of length *k* defines the normal behavior of that process. Any sequence of *k* system calls issued by that process and not present in its normal behavior is considered an anomaly, or a threat.

To illustrate that technique, let us consider a process which issued the following system calls during its execution:

```
[open read mmap mmap open read mmap]
```

Adopting *k=3*, the following set of sequences is obtained:

```
(open read mmap) (read mmap mmap)
(mmap mmap open) (mmap open read)
```

If the process issues a different sequence, like (`open open read`), it should be placed under suspicion. Despite the set of *system calls* to be system-dependent and the capture of the complete behavior of a process to be potentially laborious, this method presents good detection efficiency, as shown by their authors.

The papers [3, 4] present a secure operating system proposal, called *Remus*, which is based in classifying *system calls* and controlling access to them by processes. The authors classify the UNIX *system calls* in some functionality groups (communication, file system and memory management are some examples) and four levels of threat. S*ystem calls* classified in threat level 1 can be used to get full access to the operating system; level 2 contains *system calls* that can be used for denial of service attacks; *system calls* able to compromise processes are classed in threat level 3; finally, *system calls* in level 4 are harmless for system security. This classification is being used in this work.

4. Intrusion detection in virtual machines

As shown, host-based IDS are vulnerable to local attacks, because the intruder can disable or tamper them. Thus, monitoring data coming from a compromised system cannot be considered reliable [1]. The use of virtual machines provides a solution to this problem. The proposal presented here allows building more reliable host-based intrusion detection systems.

The proposal's main idea is to encapsulate the system to monitor inside a virtual machine, which is monitored from outside. The intrusion detection and response mechanisms are implemented outside the virtual machine, i.e. out of reach of intruders. The proposal considers a type II virtual machine monitor, so the detection and response system can be implemented as host system processes. Figure 3 illustrates the main components of the proposed architecture.

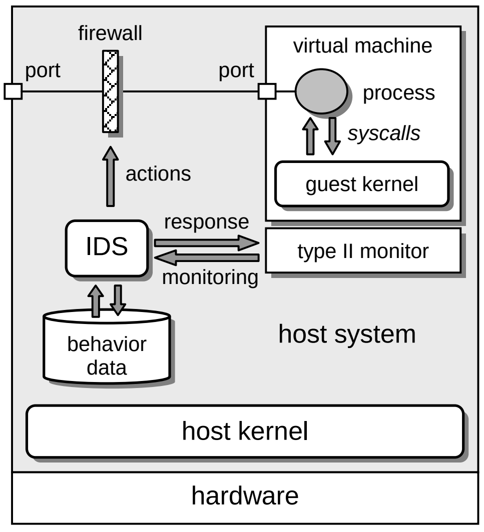

Figure 3. Architecture proposal

The interaction of guest system processes with the outside world is done only through the network, using a software *firewall* managed by the host system. Under the guest system's viewpoint, it is an external *firewall*, therefore inaccessible to intruders. The interaction between the guest system and the intrusion detection and response system is done through the virtual machine monitor. Two kinds of interactions are defined: a) *monitoring*, in which guest system data is extracted from the guest system (through the virtual machine monitor) for external analysis (for example, the system calls generated by guest processes), and b) *response*, as the intrusion detection system can act on the guest system through the monitor, in order to respond to intrusions. Beyond actions on the guest system, the response system can also interact with the software firewall used by the guest system, blocking ports and connections as needed.

The architecture presented here keeps the detection and response system out of reach of intruders. However, to guarantee the system security it is important to observe that interactions with the guest system always must be done through the virtual machine monitor, the virtual machine monitor must be inaccessible to guest system processes, and all the network services must be provided by guest system processes. Network access to the underlying host system should be avoided.

Our current implementation adopted an anomaly-based approach for intrusion detection. It uses the system call sequence analysis algorithm described in section 3. The detection and response program implemented in the host system is responsible for recording and analyzing the information sent by the VM monitor.

The system has two operation modes: a *learning mode* and a *monitoring mode*. When executing the learning mode, all the processes executing in the guest system and their respective users are recorded as authorized processes and users, thus generating an access-control list (ACL). The system also stores the sequences of system calls for specific processes. The learning mode allows, therefore, recording the "normal behavior" of the system, collecting essential data for intrusion detection.

When in monitoring mode, the system receives data from the virtual machine monitor and compares it to the previously stored "normal" data. The current prototype analyzes sequences of system calls issued by guest processes, using a window of length k=3. If a system call sequence issued by a given process is not found in the stored data, an anomalous situation is signaled and that process is declared *suspect*. Also, users and processes not found in the generated ACL are also declared *suspect*.

Suspect processes are restricted in their access to the guest system, to prevent harmful actions. Thus, all the system calls which can be used to gain full access to the guest operating system (classified as threat level 1 in [3, 4] and shown in Table 1) are denied for suspect processes.

Using this, the guest operating system can isolate a suspect process without causing severe impact to its other processes.

Table 1. *System calls* denied to suspect processes

Group	System Calls
file system and devices	`open link unlink chmod lchown rename fchown chown mknod mount symlink fchmod`
Process mgmt	`execve setgid setreuid setregid setgroups setfsuid setfsgid setresuid setresgid setuid`
Module mgmt	`init_module`

5. Implementation and results

A prototype was implemented in a Linux platform, using the virtual User-Mode Linux (UML) monitor [7], under kernel 2.4.20. Although UML does not have an acceptable performance for production systems, its source code is open and publicly available, allowing us to implement the prototype. The UML code was modified in order to allow extracting detailed data from the guest system, like the system calls issued by guest processes. The communication between the UML monitor and the monitoring process is done through *named pipes*. This way, the host operating system synchronizes the data flow between them.

Some time measurements were carried out on the execution of basic user commands, in order to evaluate the performance impact of the proposal. The hardware used in the experiments was a standard PC system (AMD Athlon XP 1600 CPU, 512 MBytes RAM). The host operating system was *Suse Linux Professional 8.2*, and as guest operating system we used *Linux Debian 2.2*.

The execution time of commands `find`, `ls` and `ps` were measured in four situations: a) in the host system, b) in the original guest system, c) in the guest system (learning mode), and d) in the guest system (monitoring mode). One should notice that such programs are *system utilities*, and that their execution life cycle consists of mainly to execute system calls. User application programs use system calls less intensively, so performance figures are expected to be better than those got from system utilities.

Table 2 presents the average execution times for each command (average execution time for 10 executions; observed variances are under 5% in all measurements). Execution times observed in the guest system are far superior to those observed in the host system. This is due to the high virtualization overhead presented by the current version of UML, as discussed in section 2.1. Table 3 presents the overhead imposed by modifications in the virtual machine monitor to interact with the external learning, detection, and response system.

Some intrusion detection tests have been carried out, using popular *rootkits* (described in table 4). These rootkits modify commands of the original operating system to prevent their detection (occulting the intruder's processes, files, network connections and so) and to steal typed information like logins and passwords (through modifications in commands like `telnet`, `sshd` and `login`). The modifications inserted by those rootkits were detected in all the performed tests. The *rootkits* used in this work are available in *http://www.antiserver.it/Backdoor-Rootkit/*.

Table 2. Average execution time (in seconds)

Command	host system	guest system		
		original	learning	monitoring
`ps –ef`	0.020	0.110	0.126	0.166
`find / >/dev/null 2>&1`	0.016	0.360	0.541	0.960
`ls -laR / >/dev/null 2>&1`	0.058	0.659	0.974	1.361

Table 3. Time overhead

Command	Original guest wrt. host system	Learning mode wrt. original guest system	Monitoring mode wrt. original guest system
`ps –ef`	450.0%	14.5%	3.2%
`find / >/dev/null 2>&1`	2150.0%	50.3%	77.4%
`ls -laR / >/dev/null 2>&1`	1036.2%	47.8%	39.7%

Table 4. Rootkits used to test the prototype

Name	Description
FK 0.4	Linux Kernel Module *rootkit* and Trojan SSH.
Adore	Hides files, directories, processes, network traffic. It installs *a backdoor* and a control program.
ARK 1.0	*Ambient's Rootkit for Linux*. It includes backdoor versions of commands `syslogd`, `login`, `sshd`, `ls`, `du`, `ps`, `pstree`, `killall`, and `netstat`.
Knark v.2.4.3	Hides files, network traffic, processes and redirects program execution.
hhp-trosniff	Complete set of modifications of `ssh`, `ssh2m`, `sshd2`, and `openssh`, to extract and to register connection origin, destination, host name, user name, and password.
ulogin.c	*Universal login Trojan* - Used to record login names and passwords.

The tests evidenced the effectiveness and complementarity of both mechanisms implemented in the system: the IDS mechanism detects and hinders the execution of known but tampered binary files, while the access control list hinders the execution of unknown binary files.

6. Related work

The paper [6] cited some benefits the use of virtual machines can bring to the security and compatibility of systems, as the capture and processing of log messages, intrusion detection through the control of virtual machine internal state) or system migration easiness. However, the article does not demonstrate how these situations should be structured and implemented, nor analyzes their impact on system performance.

The article [8] describes an experience of use of virtual machines for the security of systems. The proposal defines an intermediate layer between the monitor and the host system, called *Revirt*. This layer captures the data sent through the *syslog* process (the standard UNIX logging *daemon*) of the virtual machine and sends it to the host system for recording and later analysis. However, if the virtual system is compromised, the log messages can be manipulated by the invader and consequently are no more reliable.

The work described in [10] is close to our approach. It defines an architecture for intrusion detection in virtual machines called VMI-IDS (*Virtual Machine Introspection Intrusion Detection System*). Their approach considers the use of a type I VMM, executing directly on top of the hardware. The IDS executes in a privileged virtual machine and scans data extracted from the other VMs, searching for intrusion evidences. Only the low-level internal state of each virtual machine is analyzed, without taking in account the activities carried out by its guest processes. Also, the system response ability is limited: in case of intrusion suspicion, the suspect virtual machine is suspended for deeper analysis. If the intrusion is confirmed, the virtual machine should be restarted from a (previously stored) safe state.

That approach differs from our proposal in several aspects, like collected data granularity, intrusion detection methods, access control, and intrusion response. Our proposal allows analyzing processes separately, detecting anomalous activities and hindering intrusions from compromised processes. This way, perturbations on valid guest processes are minimized. Moreover, there is no need to suspend the virtual machine for intrusion confirmation. Another unique feature in our proposal is the use of an authorization model for users and processes, automatically generated during the learning phase.

7. Conclusion

This paper describes a proposal to increase the security of computing systems using virtual machines. The basis of the proposal is to monitor guest processes actions through an intrusion detection system, external to the virtual machine. The data used in intrusion detection is obtained from the virtual machine monitor and analyzed by an IDS process in the underlying real machine. The detection system is inaccessible to virtual machine processes and cannot be subverted by intruders.

The main objective of the project, to hinder the execution of suspect process in the virtual machine and consequently avoid the system compromise, was reached with the current prototype. However, complementary work must be done to diminish the virtualization overhead and to improve the performance of the current intrusion detection and response mechanism. We are also studying more flexible ways to interact with the guest kernel, allowing killing or suspending specific suspect processes. Also, the interactions between the IDS and the host system firewall, to block suspect network traffic, need to be refined. Other questions to be studied include to implement monitoring mechanisms based on other relevant data, like the network traffic generated by the virtual machine, and the behavior of guest users on their processes. More sophisticated algorithms for intrusion detection can be implemented based of such information, helping to reduce the occurrence of false results (both positive and negative).

References

[1] Allen J., Christie A., Fithen W., McHugh J., Pickel J., Stoner E. (1999) “State of the Practice of Intrusion Detection Technologies”, Technical Report CMU/SEI-99-TR028. Carnegie Mellon University.

[2] Barham P., Dragovic B., Fraser K., Hand S., Harris T., Ho A., Neugebauer R., Pratt I., Warfield A. (2003) “Xen and the Art of Virtualization”, 19th ACM Symposium on Operating Systems Principles – SOSP 2003.

[3] Bernaschi, M., Grabrielli, E., Mancini, L. (2000) “Operating System Enhancements to Prevent the Misuse of System Calls”, Proceedings of the ACM Conference on Computer and Communications Security. Pgs 174-183.

[4] Bernaschi, M., Grabrielli, E., Mancini, L. (2002) “REMUS: A Security-Enhanced Operating System”, ACM Transactions on Information and System Security, Vol 5, 01, pgs 36-61.

[5] Blunden, B. (2002) “Virtual Machine Design and Implementation in C/C++”, Wordware Publ. Plano, Texas – USA.

[6] Chen, P., Noble, B. (2001) “When Virtual Is Better Than Real”, Proceedings of the 2001 Workshop on Hot Topics in Operating Systems (HotOS).

[7] Dike, J. (2000) “A User-mode port of the Linux Kernel”, Proceedings of the 4th Annual Linux Showcase & Conference. Atlanta – USA.

[8] Dunlap, G., King, S., Cinar, S., Basrai, M., Chen, P. (2002) “ReVirt: Enabling Intrusion Analysis through Virtual-Machine Logging and Replay”, Proceedings of the 2002 Symposium on Operating Systems Design and Implementation (OSDI).

[9] Forrest, S., Hofmeyr, S., Somayaji, A. (1996) “A sense of self for Unix processes”, Proceedings IEEE Symposium on Research in Security and Privacy.

[10] Garfinkel, T., Rosenblum, M. (2003) “A Virtual Machine Introspection Based Architecture for Intrusion Detection”, Proceedings of the Network and Distributed System Security Symposium (NDSS).

[11] Goldberg, R. (1973) “Architecture of Virtual Machines”, AFIPS National Computer Conference. New York – NY – USA.

[12] Hofmeyr, S., Forrest, S., Somayaji, A. (1998) “Intrusion Detection using Sequences of System Calls”, Journal of Computer Security, 6:151–180.

[13] Kelem, N., Feiertag, R. (1991) “A Separation Model for Virtual Machine Monitors”, Research in Security and Privacy. IEEE Computer Society Symposium, pages 78-86.

[14] King, S., Chen, P. (2002) “Operating System Extensions to Support Host Based Virtual Machines”, Technical Report CSE-TR-465-02, University of Michigan.

[15] King, S., Dunlap, G., Chen, P. (2003) “Operating System Support for Virtual Machines”, Proceedings of the 2003 USENIX Technical Conference.

[16] Popek, G., Goldberg, R. (1974) “Formal Requirements for Virtualizable Third Generation Architectures”, Communications of the ACM. Volume 17, number 7, pages 412-421.

[17] Sugerman, J., Ganesh, V., Beng-Hong L. (2001). *Virtualizing I/O Devices on VMware Workstation’s Hosted Virtual Machine Monitor*. Proceedings of the 2001 USENIX Annual Technical Conference.

[18] VMware Inc. (1999) “VMware Technical White Paper”, Palo Alto – CA - USA.

[19] Whitaker, A., Shaw, M. e Gribble, S. (2002) “Denali: A Scalable Isolation Kernel”, Proceedings of the 10th ACM SIGOPS European Workshop, Saint-Emilion – France.