Virtual machine vs Real Machine: Security Systems

P.Felacy Silvia[1],R.Karthiha[2],R.Aarthy[3] and Dr. C. Suresh Gnana Das [4]

Final year Comp. Sci. Engg. [1,2&3] and Professor,Comp Sci.Engg [4]

Vel Tech Multi Tech Dr.Rangarajan Dr.Sakunthala Engg College

Chennai,India [1,2,3&4]

felcysilvia@gmail.com [1],rkarthiha@gmail.com [2],aarthyraviz@gmail.com [3],sureshc.be@gmail.com [4]

Abstract : **This paper argues that the operating system and applications currently running on a real machine should relocate into a virtual machine. This structure enables services to be added below the operating system and to do so without trusting or modifying the operating system or applications. To demonstrate the usefulness of this structure, we describe three services that take advantage of it: secure logging, intrusion prevention and detection, and environment migration. In particular, we can provide services below the guest operating system without trusting or modifying it. We believe providing services at this layer are especially useful for enhancing security and mobility. This position paper describes the general benefits and challenges that arise from running most applications in a virtual machine, and then describes some example services and alternative ways to provide those services.**

Keywords: Virtual machine, secure logging, Security services, intrusion prevention and detection, and Environment Migration.

1. Introduction

First proposed and used in the 1960s, virtual machines are experiencing a revival in the commercial and research communities. Recent commercial products such as VMware and Virtual faithfully emulate complete x86- based computers. These products are widely used (e.g. VMware has more than 500,000 registered users) for purposes such as running Windows applications on Linux and testing software compatibility on different operating systems. At least two recent research projects also use virtual machines: Disco uses virtual machines to run multiple commodity operating systems on large-scale multiprocessors [4]; Hypervisor uses virtual machines to replicate the execution of one computer onto a backup [3]. Our position is that the operating system and applications that currently run directly on real machines should

relocate into a virtual machine running on a real machine (Figure 1). The only programs that run directly on the real machine would be the host operating system, the virtual machine monitor, programs that provide local administration, and additional services enabled by this virtual machine-centric structure. Most network services would run in the virtual machine; the real machine would merely forward network packets for the virtual machine. This virtual-machine-centric model allows us to provide services below most code running on the computer, similar to providing services in the hardware of a real machine. Because these services are implemented in a layer of software (the virtual machine monitor or the host operating system), they can be provided more easily and flexibly than they could if they were implemented by modifying the hardware.
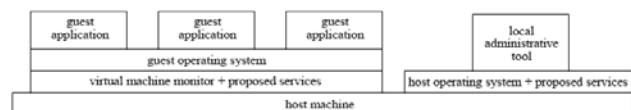

Figure 1: Virtual-machine structure. In this model, most applications that currently run on real machines relocate into a virtual machine running on the host machine. The virtual machine monitor and local administrative programs run directly on the host machine. In VMware, the virtual machine monitor issues I/O through the host operating system, so services that manipulate I/O events can be implemented in the host operating system [2].

2. Why virtual machine

The main advantages of system VMs are:

- Multiple OS environments can co-exist on the same computer, in strong isolation from each other
- The virtual machine can provide an instruction set architecture (ISA) that is somewhat different from that of the real machine
- application provisioning, maintenance, high availability and disaster recovery[2] .
- Where all these factors are not efficiently supported by real machines.

3. Services provided by Virtual machine

Providing services by modifying a virtual machine has similar benefits to providing services by modifying a real machine. These services run separately from all processes in the virtual machine, including the guest

operating system. This separation benefits security and portability.

Security is enhanced because the services do not have totrust the guest operating system; they have only to trust the virtual machine monitor, which is considerably smallerand simpler. Trusting the virtual machine monitor is akin to trusting a real processor; both expose a narrow interface (the instruction set architecture). In contrast, services in an operating system are more vulnerable to malicious and random faults, because operating systems are larger and more prone to security and reliability holes. Separating the services from the guest operating system also enhances portability. We can implement the services without needing to change the operating system, so they can work across multiple operating system vendors and versions. While providing services in a virtual machine gains similar benefits to providing services in a real machine, virtual machines have some advantages over the physical machines they emulate. First, a virtual machine can be modified more easily than a physical machine, because the virtual machine monitor that creates the virtual machine.

Second, it is much easier to manipulate the state of a virtual machine than the state of a physical machine. The state of the virtual machine can be saved, cloned, encrypted, moved, or restored, none of which is easy to do with physical machines. Third, a virtual machine has a very fast connection to another computing system, that is, the host machine on which the virtual machine monitor is running. In contrast, physical machines are separated by physical networks, which are slower than the memory bus that connects a virtual

machine with its host.

The three services that can be provided at the virtual-machine level.

Others have used virtual machines for many other purposes, such as preventing one server from monopolizing machine resources, education, easing the development of privileged software, and software development for different operating systems[10].

3.1. Secure logging

Most operating systems log interesting events as part of their security strategy. For example, a system might keep a record of login attempts and received/sent mail. System administrators use the logged information for a variety of purposes. For example, the log may help administrators
understand how a network intruder gained access to the system, or it may help administrators know what damage the intruder inflicted after he gained access. Unfortunately, the logging used in current systems has two important shortcomings: integrity and

completeness. First, an attacker can easily turn off logging after he takes over the system; thus the contents of the log cannot be trusted after the point of compromise. Second, it is difficult to anticipate what information may be needed during the post attack analysis; thus the log may lack information needed to discern how the intruder gained access or what actions he took after gaining access.

The virtual machine monitor is much smaller and simpler than the guest operating system and hence is less vulnerable to attack. To improve the completeness of logging, we propose logging enough data to replay the complete execution of the virtual machine [3]. The information needed to accomplish a faithful replay is limited to a checkpoint with which to initialize the replaying virtual machine, plus the non-deterministic events that affected the original execution of the virtual machine since the time of the saved checkpoint. These non-deterministic events fall into two categories: external input and time. External input refers to data sent by a non- logged entity, such as a human user oran external computer (e.g. a web server). Time refers to the exact point in the execution stream at which an event takes place. For example, to replay the interleaving pattern between threads, we must log which instruction is preempted by a timer interrupt [17] (we assume the virtualmachine monitor is not running on a multi-processor). Note that most instructions executed by the virtual machine do not need to be logged; only the relatively infrequent non-deterministic events need to be logged.

We can reduce thevolume of logged network data greatly by using messagelogging techniques developed in the fault-tolerance community. For example, there is no need to log message data received from computers that are themselves being logged, because these computers can be replayed to reproduce the sent message data [11]. For an important class of servers (e.g. web servers), the volume of data received in messages is relatively small (HTTP GET and POST requests). Last, as disk prices continue to plummet, more computers (especially servers worthy of being logged) will be able to devote many gigabytes to store log data [20].

3.2. Intrusion prevention and detection

Another important component to a security strategy is detecting and thwarting intruders. Ideally, these systems *prevent* intrusions by identifying intruders as they attack the system [9]. These systems also try to *detect* intrusions after the fact by monitoring the events and state of the computer for signs that a computer has been compromised [8, 12]. Virtual machines offer the potential for improving both intrusion prevention and

intrusion detection. Intrusion preventers work by monitoring events that enter or occur on the system, such as incoming network packets. Signature-based preventers match these input events against a database of known attacks; anomalybased preventers look for input events that differ from the norm. Both these types of intrusion preventers have flaws, however. Signature - based systems can only thwart attacks that have occurred in the past, been analyzed, and been integrated into the attack database. Anomaly-based systems can raise too many false alarms and may be susceptible to re-training attacks. A more trustworthy method of recognizing an attack is to simply run the input event on the real system and seeing how the system responds. Of course, running suspicious events on the real system risks compromising the system.

However, we can safely conduct this type of test on a *clone* of the real system. A potential obstacle to using clone-based intrusion prevention is the effect of clone creation or maintenance on the processing of innocent events. To avoid blocking the processing of innocent events, an intrusion preventer would ideally run the clone in the background. Allowing innocent events to go forward while evaluating suspicious events implies that these events have loose ordering constraints. For example, a clone-based preventer could be used to test e-mail messages for viruses, because ordering constraints between e-mail messages are very loose. Intrusion detectors try to detect the actions of intruders after they have compromised a system. Signs of an intruder might include bursts of outgoing network packets (perhaps indicating a compromised computer launching a denial-of-service attack), modified system files [12], or abnormal system-call patterns from utility programs [8]. As with system logging, these intrusion detectors fall short in integrity or completeness. Host-based intrusion detectors (such as those that monitor system calls) may be turned off by intruders after they compromise the system, so they are primarily useful only for detecting the act of an intruder breaking into a system. If an intruder evades detection at the time of entry, he can often disarm a host based
intrusion detector to avoid detection in the future. Network-based intrusion detectors can provide better integrity by being separate from the host operating system (e.g. in a standalone network router), but they suffer from a lack of completeness. Network intrusion detectors can see only network packets; they cannot see the myriad other events occurring in a computer system, such as disk traffic, keyboard events, memory usage, and CPU usage.

Implementing post-intrusion detection at the level of a virtual machine offers the potential for providing both integrity and completeness. Like a network-based intrusion detector, virtual machine-based intrusion detectors are separate from the guest operating system and applications.
Unlike network intrusion detectors, however, virtual-machine intrusion detectors can see all events occurring in the virtual machine they monitor. Virtual machine intrusion detectors can use this additional information to implement new detection policies. For example, it could detect if the virtual machine reads certain disk blocks (e.g. containing passwords), then issues a burst of CPU activity (e.g. cracking the passwords). Or it could detect if the virtual machine has intense CPU activity with no corresponding keyboard activity. As with secure logging, a key challenge in post-intrusion detection in a virtual machine is how to bridge the semantic gap between virtual machine events and operating system events. This challenge is similar to that encountered by network-based intrusion detectors, which must parse the contents of IP packets.

3.3. Environment migration

Process migration has been a topic of interest from the early days of distributed computing. Migration allows one to package a running computation—either a process or collection of processes—and move it to a different physical machine. Using migration, a user's computations can move as he does, taking advantage of hardware that is more convenient to the user's current location.

The earliest systems, including Butler [15], Condor [14], and Sprite [6], focused on load sharing across machines rather than supporting mobile users. These load sharing systems typically left residual dependencies on the source machine for transparency, and considered an individual process as the unit of migration. This view differs from that of mobile users, who consider the unit of migration to be the collection of all applications running on their current machine.

Recently, migration systems have begun to address the needs of mobile users. Examples of systems supporting mobility include the Teleporting system [16] and SLIM [18]. These systems migrate the user interface of a machine, leaving the entire set of applications to run on their host machine. In the limit, the display device can be a stateless, thin client. This approach provides a better match to the expectations of a migrating user, and need not deal with residual dependencies. However, these systems are intolerant of even moderate latency between the interface device and the cycle server, and thus support only a limited form of user mobility.

There are several challenges that must be overcome to provide migration at the virtual-machine level. The first is that a machine has substantial state that must move with it. It would be infeasible to move this state synchronously on migration. Fortunately, most of this state is not needed immediately, and much may never

be needed at all. We can predict which state is needed soon by taking advantage of temporal locality in disk and memory accesses. This prediction is complicated by the guest operating system's virtual memory abstraction, because the physical addresses seen by a virtual machine monitor are related only indirectly to accesses issued by applications. We can reconstruct information about virtual to physical mapping by observing manipulation of virtualized hardware elements such as the TLB.

After identifying the state likely to be needed soon, we need a mechanism to support migration of that state to the new virtual machine. If migration times are exposed, one can take advantage of efficient, wide-area consistency control schemes, such as that provided by Fluid Replication [5]. It depends on typical file system access patterns, in particular a low incidence of concurrent data sharing.

4. Challenges Hold By Virtual Machine

Providing services at the virtual-machine level holds two challenges.

The first is performance. Running all applications above the virtual machine hurts performance due to virtualization overhead. For example, system calls in a virtual machine must be trapped by the virtual machine monitor and re-directed to the guest operating system. Hardware operations issued by the guest must be trapped by the virtual machine monitor, translated, and reissued. Some overhead is unavoidable in a virtual machine; the services enabled by that machine must outweigh this performance cost. Virtualizing an x86-based machine incurs additional overheads because x86 processors don't trap on some instructions that must be virtualized (e.g. reads of certain system registers). One way to implement a virtual machine in the presence of these "non-virtualizable" instructions is to re-write the binaries at run time to force these instructions to trap [13], but this incurs significant overhead. The second challenge of virtual-machine services is the semantic gap between the virtual machine and the service. Services in the virtual machine operate below the abstractions provided by the guest operating system and applications. This can make it difficult to provide services. For example, it is difficult to provide a service that checks file system integrity without knowledge of on-disk structures. Some services do not need any operating system abstractions; secure logging (Section 4.1) is an example of such a service. For services that require higher-level information, one must re-create this information in some form. Full semantic information requires re-implementing guest OS abstractions in or below the virtual machine. However, there are several abstractions—virtual address spaces, threads of control, network protocols, and file system

formats— that are shared across many operating systems. By observing manipulations of virtualized hardware, one can reconstruct these *generic* abstractions, enabling services that require semantic information.

5. Alternative Approach For Implementing Services Of Virtual Machine

Each of the above services can be implemented in other ways. One alternative is to include these services in the operating system. This structure makes it easier for the service to access information in terms of operating system abstractions. For example, an intrusion detector at the operating system level may be able to detect when one user modifies files owned by another user. A virtual machine service, in contrast, operates below the notions of users and files and would have to reconstruct these abstractions. In addition, including these services in the operating system reduces the number of layers and redirections, which will likely improve performance relative to a virtual machine.

However, including services in the operating system has some disadvantages. First, such services are limited to a single operating system (and perhaps a single operating system version), whereas virtual-machine services can support multiple operating systems. For example, a secure logging service in a virtual machine can replay any operating system. Second, for security services such as secure logging and intrusion detection, including the service in the operating system depends critically on the integrity of the operating system. Because operating systems are typically large, complex, and monolithic, they usually contain security and reliability vulnerabilities. For example, the Linux 2.2.16 kernel contained at least 7 security holes [1]. In particular, secure logging is challenging to provide in the operating system, because an intruder may try to crash the system to prevent the log tail from being written to stablestorage.

Some of the disadvantages of including services in the operating system can be mitigated by re-structuring the operating system into multiple protection domains [19] and placing security-related services in the most-privileged ring. This approach is similar to kernels that include only the minimum set of services [7]. However, this approach requires re-writing the entire operating system, and frequent crossings between multiple protection domains degrade performance. A different approach is to add services to a language specific virtual machine such as Java. However, these services would be available only for applications written in the target language. For the system wide services described

above, the entire system would have be written in the target language.

6. Conclusions

Running an operating system and most applications inside a virtual machine enables a system designer to add services below the guest operating system. This structure enables services to be provided without trusting or modifying the guest operating system or the applications. We have described three services that take advantage of this structure: secure logging, intrusion prevention and detection, and environment migration. Adding services via a virtual machine is analogous to adding network services via a firewall. Both virtual machines and firewalls intercept actions at a universal, low-level interface, and both must overcome performance and semantic-gap problems. Jut as network firewalls have proven useful for adding network services, we believe virtual machines will prove useful for adding services for the entire computer.

7. References

[1] Linux Kernel Version 2.2.16 Security Fixes, 2000.http://www.linuxsecurity.com/advisories/slackware_advisory-481.html.

[2] *VMware Virtual Machine Technology. Technical report,VMware, Inc., September 2000.*

[3] Thomas C. Bressoud and Fred B. Schneider. Hypervisor- Based Fault-Tolerance. In *Proceedings of the 1995 Symposiumon Operating Systems Principles*, pages 1–11, December1995.

[4] Edouard Bugnion, Scott Devine, Kinshuk Govil, and MendelRosenblum. Disco: Running Commodity Operating Systemson Scalable Multiprocessors. *ACM Transactions on ComputerSystems*, 15(4):412–447, November 1997.

[5] Landon P. Cox and Brian D. Noble. Fluid Replication. In*Proceedings of the 2001 International Conference on DistributedComputing Systems*, April 2001.

[6] Fred Douglis and John Ousterhout. Transparent Process Migration:Design Alternatives and the Sprite Implementation.*Software Practice and Experience*, 21(7), July 1991.

[7] Dawson R. Engler, M. Frans Kaashoek, and James O'TooleJr. Exokernel: an operating system architecture for application-level resource management. In *Proceedings of the 1995Symposium on Operating Systems Principles*, pages 251–266,December 1995.

[8] Stephanie Forrest, Steven A. Hofmeyr, Anil Somayaji, andThomas A. Longstaff. A sense of self for Unix processes. In*Proceedings of 1996 IEEE Symposium on Computer Security and Privacy*, 1996.

[9] Ian Goldberg, David Wagner, Randi Thomas, and Eric A.Brewer. A Secure Environment for Untrusted Helper Applications.In *Proceedings of the 1996 USENIX Technical Conference*,July 1996.

[10] Robert P. Goldberg. Survey of Virtual Machine Research. *IEEE Computer*, pages 34–45, June 1974.

[11] David B. Johnson and Willy Zwaenepoel. Sender-BasedMessage Logging. In *Proceedings of the 1987 InternationalSymposium on Fault-Tolerant Computing*, pages 14–19, July 1987.

[12] Gene H. Kim and Eugene H. Spafford. The design and implementationof Tripwire: a file system integrity checker. In*Proceedings of 1994 ACM Conference on Computer andCommunicationsSecurity*, November 1994.

[13] Kevin Lawton. Running multiple operating systems concurrentlyon an IA32 PC using virtualization techniques, 1999.http://plex86.org/research/paper.txt.

[14] M. J. Litzkow. Remote UNIX: turning idle workstations intocycle servers. In *Proceedings of the Summer 1987 USENIXTechnical Conference*, pages 381–384, June 1987.

[15] D. A. Nichols. Using idle workstations in a shared computingenvironment. In *Proceedings of the 1987 Symposium onOperating System Principles*, pages 5–12, November 1987.

[16] T. Richardson, F. Bennet, G. Mapp, and A. Hopper. Teleportingin an X window system environment. *IEEE PersonalCommunications*, 1(3):6–12, 1994.

[17] Mark Russinovich and Bryce Cogswell. Replay for concurrentnon-deterministic shared-memory applications. In *Proceedingsof the 1996 Conference on Programming LanguageDesign and Implementation (PLDI)*, pages 258–266, May1996.

[18] Brian K. Schmidt, Monica S. Lam, and J. Duane Northcutt.The interactive performance of SLIM: a stateless, thin-clientarchitecture. In *Proceedings of the 1999 Symposium on OperatingSystemsPrinciples*, pages 32–47, December 1999.

[19] Michael D. Schroeder and Jerome H. Saltzer. A hardware architecturefor implementing protection rings. *Communicationsof the ACM*, 15(3):157–170, March 1972.

[20] John D. Strunk, Garth R. Goodson, Michael L. Scheinholtz,Craig A.N. Soules, and Gregory R. Ganger. Self-securingstorage: Protecting data in compromised systems. In *Proceedingsof the 2000 Symposium on Operating Systems Designand Implementation (OSDI)*, October 2000