JANUS: The Protector of Virtual Machine Networks

Insiya Javed, Amit Kumar Singh, Brajesh Kumar, Shailesh Mhaisekar

Abstract: Developing a virtual machine security tool manually is generally believed to be a tedious, time consuming, and error-prone process because of the large semantic gap. Recent advances in the field of virtualization show that we can largely narrow this semantic gap. But still there is no completely automated tool for VM security. In this paper, we present JANUS, an entirely new technique for VM security that can automatically bridge the semantic gap. The key idea is that, through oursystem, we will automatically redirect all the requests coming into the virtual machine network to our monitoring machine which will identify the valid or authentic requests and redirect them to the respective machines after they have been logged in the database. This system acts as a gateway for the VM network and offers a number of new features and capabilities. Particularly, it automatically enables inspection of every single request coming into the network and hence ensures high security. We have tested our system on several commonly used utilities on top of different kernels. The experimental results show that our technique is general (largely OS-agnostic), and it introduces 9.3X overhead on average for the redirected requests as compared to the native non-redirected ones.

Index Terms— Janus, monitoring machine, viClient, vCenter server, intrusion detection, introspection, target OS, default gateway, request handler, rule executor, rule manager, rule configurator, packet analyser.

1 INTRODUCTION

In context with its renewed popularity, researchers are identifying many new applications based on the abstractions and isolation provided by virtualization. One area that has received significant attention is security. Security applications benefit from virtualization by running in isolated virtual machines (VMs) and building smaller trusted computing bases (TCBs). This technique is being widely used in the projects now. For the purpose of security of a virtual network, our idea is to setup a secure VM that is used to monitor the other VMs running in the network. These monitors can be used in intrusion detection systems (IDS), integrity checking, honeypot systems and forensic analysis, among others. While the previous efforts in this space have focused more on the applications of introspection, our aim is to build a proper architecture to support the technique.

- *The authors are currently pursuing bachelors degree program in computer engineering from D.Y.Patil College of engineering in PuneUniversity, India.*
- *Insiya Javed. E-mail: insiyamian@yahoo.com*
- *Amit Kumar Singh. E-mail: amit.k_singh@yahoo.com*
- *Brajesh Kumar. E-mail: shyamazad@gmail.com*
- *Shailesh Mhaisekar. E-mail: mhaisekars@gmail.com*

Our experience designing and implementing this system has shown that implementing security system in a secure and efficient manner is non-trivial. However, our architecture demonstrates how one can achieve these goals without losing monitoring functionality. Monitoring with Janus requires no changes to the VMM or to the VM being monitored. In addition no changes are required to the OS being monitored, so Janus is not restricted to monitoring open source OSes. It can be extended to monitor any OS that runs on the VM. Janus incurs a minimal performance penalty for typical monitor applications.

We designed the Janus architecture based on six high-level requirements. In a general sense, these requirements can be seen as typical good programming guidelines, or good security guidelines. We identify the following six requirements for monitoring VMs:

1. No superfluous modifications to the VMM. TheVMM should remain as small and simple as possible since it is part of the TCB. If a VMM includes the necessary primitives to support the monitoring architecture, then it should not be modified. If a VMM lacks the necessary primitives, then the modifications made should be what are minimally required to support the monitoring architecture.

2. No modifications to the VM or the target OS. Modifications to the target OS (i.e., the OS being monitored), are problematic. One of the key reasons why virtualization is attractive for monitoring is the isolation between VMs. Placing monitoring code within the same OS that is being monitored bypasses this isolation, negating this key benefit. Therefore, this requirement encourages all monitoring code to remain in an isolated VM.

3. Small performance impact. An excessive performance impact can render a monitoring architecture worthless. The performance impact is measured as any reduction in performance of an application caused by the monitoring software. Ideally this impact is both small and consistent, but some initialization costs may be required.

4. Rapid development of new rules. New rules may be needed to address new types of attacks. Furthermore, it is advantageous to keep the monitor code simple to limit the opportunity for introducing errors into the monitors. The monitoring architecture should provide APIs that are used to develop new rules.Therefore, satisfaction of this requirement means that the APIs should be designed in a way that simplifies the job of the monitor developer.

5. Ability to monitor any data on target OS. The monitoring machine should have a full view into the target OS. It should not be limited to providing information about a small part of the target OS. While this ideal may not always be possible, the more information a monitor can view, the harder it is for an attacker to evade detection.

6. Target OS cannot tamper with monitors. If the target OS can tamper with the monitoring machine, then the possibility exists for malicious code to tamper with the monitors. For this reason, the monitoring machine should be isolated or protected from the target OS. If all monitor code is in an isolated VM, then this is not difficult.

Our main contribution is the Janus monitoring architecture that satisfies the above requirements. The remainder of this paper focuses on the Janus architecture, its implementation, and some example applications that demonstrate the performance and flexibility of the system. Section 2 discusses the related work. Section 3 provides background information on the components in the system. Section 4 presents the architecture and implementation details for Janus. Section 5 discusses future scope in this research space and we conclude with Section6.

2 RELATED WORK

VMMs first came into use over 35 years ago. While Madnick and Donovan identified the security benefits of VMMs in the early 70s, research that explicitly leveraged these benefits did not take place until nearly 20 years late. More recently, virtualization is being used in different ways to address a variety of systems management, and security problems. In the security space, we have seen innovative work in intrusion detection systems, workload isolation, attack investigation and debugging, and system monitoring. Each of these applications have one thing in common: they each require the ability to monitor data from a target OS. However, the mechanics of how to properly do such monitoring have not been adequately addressed in the literature. Through the details provided in this paper, and by making Janus an open source project, we are exposing these mechanics for the benefit of future work in this space.

Joshi et al presented a system called IntroVirt that uses introspection and replay to test if a system was previously attacked through a known vulnerability. Similar to the first effort, only limited details were given regarding the introspection mechanism. More recently, several projects have provided details about their introspection techniques, only to reveal suboptimal security decisions in their architecture. The Hyperspector project is a virtual distributed monitoring environment used for intrusion detection. The Hyperspector approach to introspection is to provide access to a few specific pieces of information (processes, sockets, etc). This limited view into the target OS violates property (5) of our requirements for a robust monitoring solution, and Hyperspector also violates property (1) by extensively modifying the VMM, and (6) bysharing OS kernels between VMs.

Asrigo et al presented a system for monitoring honeypots, but they violate property(2) by requiring hooks in the target OS kernel, property(3) by causing a substantial performance impact, and property(4) by incorporating kernel code in new monitor hooks. Finally, the Antfarm system tracks only OS-level processes, violating property (5), and performs the monitoring from within the VMM, violating property (1). Each of these virtual memory introspection systems were built to provide monitoring capabilities for a security system. However, none of these systems meet our six requirements for a monitoring solution, making it much more likely for an

intruder to compromise, evade or disable the monitors.

Monitoring in a virtualized environment is not the only approach. Petroni et al developed Copilot, a secure coprocessor used to monitor the memory of a host. In practice, this approach is very similar to virtual memory introspection from a VM, but it requires extra hardware and cannot be generalized to monitoring other data such as disk I/O.Looking into the commercial world, many monitoring applications sold today simply run within the target OS. Forexample, anti-virus software typically runs in the same OSthat it is protecting. However, this architecture is flawed because malicious software can simply disable the anti-virus software.

Monitoring at the disk level has traditionally taken place as part of a research trend focused on creating smarter, more semantically-aware devices. This has applications in both systems optimization and security. Researchers at Carnegie-Mellon University have leveraged the physical isolation of such systems to enable intrusion detection and recovery capabilities. These systems are able to perform their function in a tamper-resistant manner, regardless of an OS compromise. This approach, however, has the obvious downside of requiring additional hardware support and the need for a special infrastructure for communication between the management tools insidethe OS and the disk IDS. XenAccess leverages virtualization to provide the same level of monitoring functionality without either of these limitations.

Hyper-Spector's approach is to mount a shadow version of the monitored filesystem and execute integrity checkers (e.g.tripwire). Not only does this require significant modifications to the VMM, violating property (1) and increasing the chances of a VMM compromise; it also limits access tothe disk data by providing an exclusively static and high level view of it, violating property (5) and making it very easy to evade the monitor. Elango et al and Jones et al have applied some of the principles of semantically smartdisk systems and gray-boxing to the performance improvement of Xen virtual machines. Their results show how monitoring and active control of virtual machines can have a wide variety of applications outside the security area.

In the past, many researchers choose to work with User Mode Linux (UML), a virtualization solution that allows you to boot a Linux kernel as a process in a running version of Linux. The earliest work with introspection used VMWare, a full featured commercial virtualization product. Looking forward, interest is now growing in the kernel-based virtualization driver (KVM) that is built into the Linux kernel startingwith version 2.6.20. While our techniques are viable on any of these platforms, a virtualization solution designed as an independent and lightweight software layer running directlyon the hardware offers a solid foundation to asecurity-oriented solution.

3 BACKGROUND

Three capabilities of virtual machines make them particularly attractive for building an intrusion detection system. The first capability is *isolation.* Software running in a virtual machine cannot access or modify the software running in the VMM or in a different VM. Isolation ensures that even if an intruder has completely subverted a guest virtual machine, he or she still cannot tamper with the IDS.

The second capability is *inspection.* In a virtual machine system, guest VMs run on emulated hardware and the virtual machine monitor has access to the entire state of each guest VM. Being able to directly inspect the virtual machine makes it particularly difficult to evade a VM-based IDS because there is no state in the VM that the IDS cannot see.

The third capability is *interposition.* The presence of privileged instructions forces the VMM to trap and emulate these instructions, which incur extra overhead that would not exist in a conventional system. However, these privileged instructions also provide hooks to allow a VM-based IDS to record or modify privileged instruction parameters and other virtual machine state.

It is not only helpful to detect attacks as they happen, but also to harden systems ahead of time so that they are more resilient to intrusions. Intrusion prevention systems do just this. With the aid of virtual machine technology, the systems described in this section enforce policies that help protect critical resources so that they are safe from potential attackers.

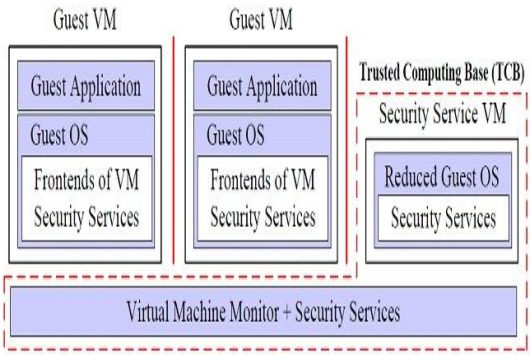

Fig. 1. Virtual Machine Security

Some of the features of VM which have been explored to provide security to virtual machine networks in the past are:-

1. Snapshotting: A snapshot is the state of a virtual machine, and, generally, its storage devices, at an exact point in time. Snapshots are "taken" by simply giving an order to do so at a given time, and can be "reverted" to on demand, with the effect that the VM appears (ideally) exactly as it did when the snapshot was taken.

The capability is, for example, useful as an extremely rapid backup technique, prior to a risky operation. It also provides the foundation for other advanced capabilities.

2. Teleportation: The snapshots described above can be moved to another host machine with its own hypervisor; when the VM is temporarily stopped, snapshotted, moved, and then resumed on the new host, this is known as teleportation. If the older snapshots are kept in sync regularly, this operation can be quite fast, and allow the VM to provide uninterrupted service while its prior physical host is, for example, taken down for physical maintenance.

3. Failover: Similar to teleportation above, failover allows the VM to continue operations if the host fails. However, in this case, the VM continues operation from the last-known coherent state, rather than the current state, based on whatever materials the backup server was last provided with.

Safety Requirements:

A VMM can have different kinds of relationships with the host operating system. It may be an unprivileged application, but it may also be require cooperation from inside the kernel. A thirdpossibility is that no host operating system is present. In this case one could say the VMM itself acts as a minimal operating system. When used for debugging purposes, it is desirable that the VMM can run as an application. This avoids restarting and allows one to run other applications besides the VMM while debugging. It is preferable that kernel cooperation is not required, because one would typically want to avoid installing kernel-mode drivers since these may make the operating system less stable. Another problem is that installing such a driver is normally only allowed for the root user. The same reasoning goes when virtual machines are used for running untrusted applications.The intrusion detection rules should be defined in such a manner that it shouldn't block any necessary system process.

Security Requirements

Virtual machines should provide absolute isolation and, as such, perfect security. The only means for communication between virtual machines should be the virtual network connection. Unfortunately, perfection is hard to come by. Both the VMM and the host kernel are pieces of software and, as such, we can expect them to contain bugs. We will therefore evaluate the robustness of the security in the presence of bugs. We will find that in some cases the VMM can increase robustness, while in others the isolation entirely depends on the ability of the kernel to isolate applications. Secure isolation is the core feature when virtual machines are applied to isolate untrusted programs or to secure honeypots. It is also of utmost importance when used for server consolidation, since a breach of security would mean down-time. In each of these cases we can expect malicious attempts to break security. For debugging purposes we rely less on the robustness of VMMs. Although crashes of the virtual machine are possible and should not effect the physical machine, we do not expect any malice. Only system admin should have control to add or modify the intrusion detection rules of the policy module.

4 ARCHITECTURE AND IMPLEMENTATION

4.1. Architecture:

The system architecture shows that the system consists of the following components, each performing its own set of functions:-

1. **The Central Management Server:** The CMS is the most important part of the

system. It controls the entire network. It acts as a mediator between the client and the virtual machine. It is a central body which is connected to the clients as well as to the VMs. The client request first goes to the CMS from where it is directed to the corresponding virtual machine.

2. **The Client:** The client is the requesting body. There may be multiple clients connected to the virtual network at a time. These clients are assigned some authorities by the administrator of the system. Some clients may have certain access rights while others may not. This record is kept by the system admin. The client issues requests to access particular VMs.
3. **The Router:** The request issued by the client proceeds towards the Central Management Server but it is interrupted by a router in its path. It prevents the request from hitting the CMS and redirects it to the Monitoring Machine where it will be scrutinized and further action will be decided.
4. **The Monitoring Machine:** The monitoring machine first identifies the incoming request and makes an entry in the log. Secondly, processing of the request is done in which the authority rights of the requesting client are checked and it is decided whether the request is authentic or not. Lastly, the necessary action is taken which may be one of the following:-
   - If the request is authentic then it will be forwarded to the central management server which will send it to the corresponding VM.
   - If the request is unauthentic then it will not be allowed to hit the CMS and will be terminated by the monitoring machine. Its entry will be made in the log, which will indicate to the admin that some client tried to intrude into his system by performing an unauthorized action.
5. **ESX:** The ESX acts as a service console for the virtual machines. When we try to mount a virtual machine on a VMware hypervisor, first the ESX is installed. It is a kind of Linux kernel which provides a platform to install the necessary files of a VM. There are multiple ESX kernels connected to the CMS and each ESX may carry one or more VMs.
6. **Virtual Machines:** Virtual machines are the target machines. These machines are subjectively a complete machine just like a physical one but objectively they are merely a set of files installed on the hypervisor. The CMS directs the client request to the corresponding VM by sending it to the ESX where the VM is installed and finally the request hits the VM and the task is performed.

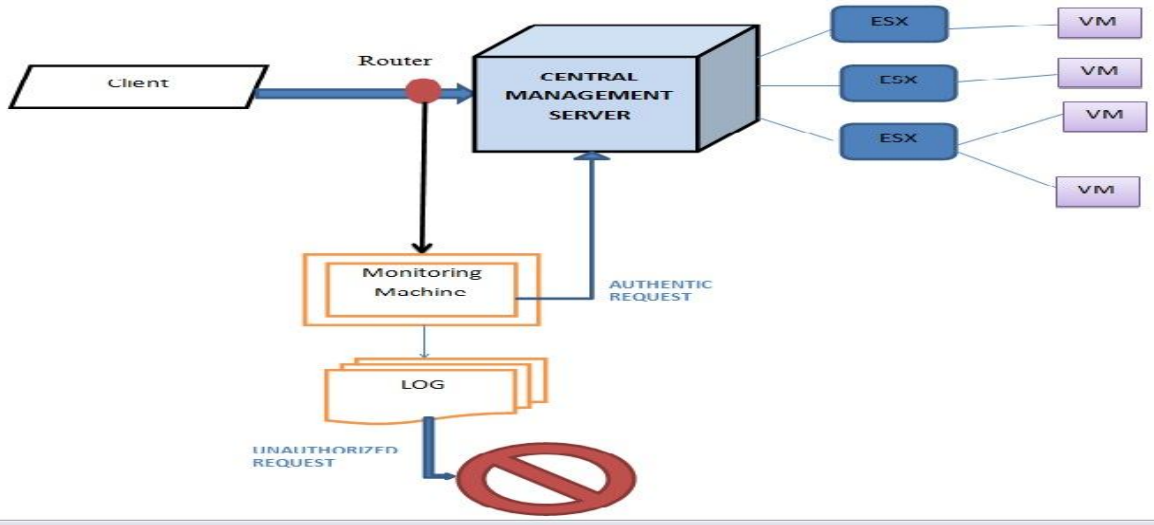

Fig. 2. JANUS Architecture

4.2 Implementation:

Our system introspects into low-level disk traffic, just as it is able to map raw memory pages. It therefore satisfies property by providing full and complete access to data. Our system also includes an inference engine which is able to dynamically infer the high-level file system operations executed inside a domain based on the intercepted low-level disk traffic. To this end, we have decided to leverage the architecture described earlier since it simplifies the implementation of the interception mechanism and avoids making modifications to the VMM, which is encouraged by property. The biggest challenge, however, is faced by the inference engine which must somehow overcome the semantic gap between the low-level view and the desired higher-level, file system-oriented view that will be given as output. It does this combining pre-programmed file system structure knowledge with dynamic inference techniques. Whereas the interception mechanism (which is roughly equivalent to the introspection memory-mapping) is independent of the current OS and file system by only providing raw access to disk traffic, the inference engine is dependent on knowledge of the file system in use. So far, knowledge has been included in the inference engine to be able to determine only file/directory creation/removal operations under the ext2 file system, although knowledge about other file systems can be incorporated.

Our system has preliminary support to perform memory introspection on fully virtualized (HVM) VMs. In HVM VMs, physical addresses and machine addresses are the same. Therefore, system will automatically detect HVM domains and not attempt to perform this translation in those cases. In practice, the P2M translation is a simple table lookup, so omitting this step does not measurably improve performance. Since memory introspection support for HVM VMs is in its early stages, there is some reduced functionality. This reduced functionality is the reason why there is no HVM performance data available for the user address function.

Every request that comes into the VM network is governed by a rule. Every rule constitutes of a condition and a respective action that must be taken if the condition is satisfied. Thus, the Janus web application will have two interfaces as shown in the figure:

1. For the application administrator to configure the rules
2. For monitoring the incoming vmware client request.

The administrator will create and configure rules in the rule configurator which will have a set conditions mapped to a set of actions.

The vmware client request will go to the request handler which will store all the incoming requests in a queue for further processing.

The request handler will forward the first request from the queue to the rule executer. The rule executer is the module where the condition in the request is compared to the condition stored in the configurator and then the necessary action is taken. The rule executer is the part responsible for taking the desired action. It is here that the client request will be logged into the database and will either be blocked or forwarded to the VM network, based on the action that is mapped to the condition satisfied by the client request.

When request reaches the rule executer, it demands the rule manager to select the appropriate rule i.e. condition→action from the rule configurator and supplies it to the rule executer for further action. The rule manager supplies the most appropriate rule to the rule executer. The rule executer implements this rule on the client request and sends a response to the request handler. This response will either be a response to intimate that the request has been successfully forwarded into the VM network or that the request was not authentic and hence has been blocked.

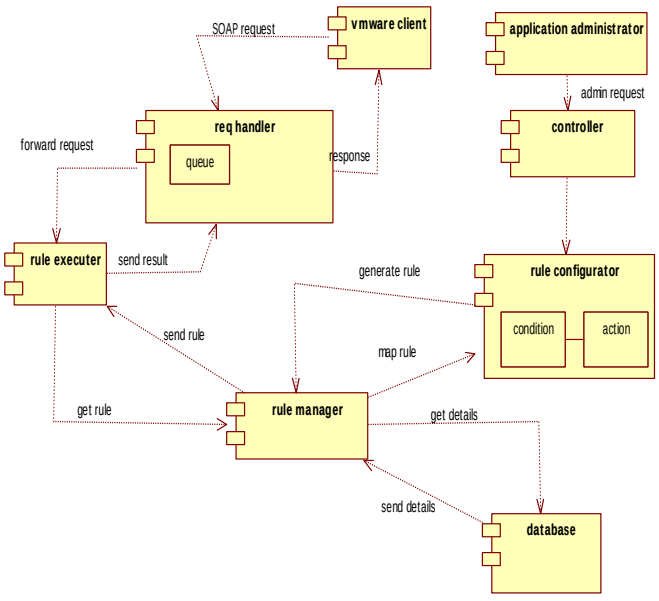

Fig. 3. Rule Execution System

The request handler will send a response message to the VM client in accordance with the response received from the rule executer. This response will intimate the client whether his request was successfully executed or abolished. At the same time the rule executer will also send a message to the administrator to intimate him that a particular client tried to access his system.

5 FUTURE SCOPE

Stepping back to look at the six requirements given for a robust monitoring solution, we note that JANUS satisfies each of these requirements. (1) The system uses an unmodified version of ESX as a VMM platform.(2) Using the capabilities provided by this system, no special code needs to be inserted into the target OS. This is especially useful as it allows the monitoring machine to work with both open and closed source target OSes. (3) Our performance testing shows that our address translation, memory copying, and disk I/O monitoring functions have small overheads, making these capabilities effective for a variety of monitoring applications. (4) Developing monitors with Janus is straight forward, with a minimal learning curve. (5) The Janus architecture is easily extensible to collect any type of data from the target OS. (6) Finally, leveraging the protections provided by the VMM, Janus is sufficiently isolated from the target OS and any possibility of tampering by malicious software.

Janus currently provides a solid foundation for monitoring in a virtualized environment. Yet, our experiences working with virtual machine monitoring highlighted some areas that would benefit from additional research. Introspection requires use of OS-specific information. This means that it is possible for an OS upgrade, hotfix, or patch to break the monitors. Ideally, JANUS should provide an abstraction layer that dynamically adapts to these changes and provides a consistent interface to monitor applications. Finding techniques to enable this approach is still an open research problem.

For reasons of backwards compatibility, changes in file system structure and layout are very rare. So monitoring is not prone to the types of problems discussed above for introspection. Instead, we envision the future work in
this space to focus on scalability, functionality, and HVM support.

6 CONCLUSION

This paper described Janus, a monitoring system for virtual machine network. Janus' development was guided by a set of design principles aimed at providing a solid foundation for secure and flexible virtual machine monitoring. Janus implements virtual machine security by comparing the condition in the incoming request to those present in the rule configurator and then taking the necessary action that is mapped to that particular condition. Our evaluation revealed that Janus imposes a minimal performance overhead to the target OS and disk operation.

BIBLIOGRAPHY

1. Books

i. Virtualization Essentials by Mathews.
ii. VMware ESX and VMware ESXi.
iii. Hypervisor: High-impact Technology by Kevin Roebuck.
iv. VMware and CPU virtualization technology by Jack Lo.

2. Papers

[1] W. A. Arbaugh, D. J. Farber, and J. M. Smith.A secure and reliable bootstrap architecture.IEEE Computer Society, Washington, DC, USA, 1997.

[2] P. M. Chen and B. D. Noble.When virtual is better than real. In HOTOS '01: Proceedings of the Eighth Workshop on Hot Topics in Operating Systems, page 133. IEEE Computer Society, Washington, DC, USA, 2001.

[3] R. Goldberg. Survey of virtual machine research. IEEE Computer Magazine, 7:34–45, June 1974.

[4] S. King and P. Chen. Subvirt: Implementing malware with virtual machines. In IEEE Symposium on Security and Privacy. Oakland, California, May 2006.

[5] X. Zhao, K. Borders, and A. Prakash.Towards protecting sensitive files in a compromised system.In 3rd International IEEE Security in Storage Workshop. 2005.

[6] R. Goldberg. Survey of virtual machine research. IEEE Computer Magazine, 7:34-45, June 1974.

[7] WU Qingbo, WANG Chunguang and TAN Yusong. System Monitoring and Controlling Mechanism based on Hypervisor. In 2009 IEEE International Symposium on Parallel and Distributed Processing with Applications.

[8] Tal Garfinkel and Mendel Rosenblum. A Virtual Machine Introspection Based Architecture for Intrusion Detection. In VMSec '09 Proceedings of the 1st ACM workshop on Virtual machine security. In Proceedings of the 2003 Network and Distributed System Symposium (2003)

3. Links

i. http: / / en.wikipedia.org / wiki / Hypervisor
ii. http: / / xen.org / products / xenhyp.html
iii. http: / / www.computer.org / portal / web / csdl / doi / 10.1109 / ISPA.2009.99
iv. http: / / www.usenix.org / event / deter07 / tech / full_papers / duchamp / duchamp_html
v. KVM: Kernel-based Virtual Machine. http: / / kvm.sourceforge.net / .
vi. Intel Virtualization Technology. http: / / www.intel.com / technology / virtualization.
vii. http: / / pubs.vmware.com
viii. http: / / microsoft_technet / libraries
ix. http: / / Nas-san.com
x. http: / / vim.datastore.html