# Secure Sharing of an ICT Infrastructure through Vinci

Fabrizio Baiardi[1] and Daniele Sgandurra[2]

[1] Polo G. Marconi, La Spezia
[2] Dipartimento di Informatica
Università di Pisa
{baiardi, daniele}@di.unipi.it

**Abstract.** *Virtual Interacting Network CommunIty* (Vinci) is a software architecture that exploits virtualization to share in a secure way an information and communication technology infrastructure among a set of users with distinct security levels and reliability requirements. To this purpose, Vinci decomposes users into *communities*, each consisting of a set of users, their applications, a set of services and of shared resources. Users with distinct privileges and applications with distinct trust levels belong to distinct communities. Each community is supported by a virtual network, i.e. a structured and highly parallel overlay that interconnects virtual machines (VMs), each built by instantiating one of a predefined set of VM templates. Some VMs of a virtual network run user applications, some protect shared resources, and some others control traffic among communities to discover malware or worms. Further VMs manage the infrastructure resources and configure the VMs at start-up. The adoption of several VM templates enables Vinci to minimize the complexity of each VM and increases the robustness of both the VMs and of the overall infrastructure. Moreover, the security policy that a VM applies depends upon the community a user belongs to. As an example, discretionary access control policies may protect files shared within a community, whereas mandatory policies may rule access to files shared among communities. After describing the overall architecture of Vinci, we present the VM templates and the performance results of a first prototype.

## 1 Introduction

Among the benefits of virtualization, the most well known one is the cost saving achieved by consolidating several servers on a single physical machine [1]. We believe that a further noticeable advantage is an increase of system robustness because we can include in a virtual architecture components that check and control the other ones in a transparent way. As an example, a virtual network can include nodes, i.e. virtual machines (VMs), which run the applications and other nodes that monitor the previous ones in a completely unobtrusive way [2]. Furthermore, the ability of accessing any component of a virtual node enables the definition of more rigorous and complete checks to detect anomalies or intrusions, as when special purpose hardware units are available. Finally, an architecture composed of a large number of virtual nodes can increase the robustness of each node, and of the overall system, by minimizing the software each node runs.

D. Hausheer and J. Schönwälder (Eds.): AIMS 2008, LNCS 5127, pp. 65–78, 2008.

These considerations have led to the definition of *Virtual Interacting Network CommunIty* (Vinci), a software architecture that aims to exploit at best virtualization technologies to share in a secure way an information and communication technology (ICT) infrastructure. To this purpose, Vinci adopts a two-tier approach where several virtual networks, or overlays, are introduced and each overlay is highly parallel because it composes a large number of VMs. To increase the robustness of each overlay, Vinci minimizes the functionalities of each VM by defining several *VM templates*. As an example, Vinci instantiates *Application VMs* to run user applications, according to the applications trust level and to the user privileges, i.e. user security levels, so that each Application VM only runs the smallest number of software packages and libraries to support the considered applications. Other VM templates are introduced to control resources shared among Application VMs of the same overlay or of distinct ones, or information flowing among VMs. In Vinci, each physical node of the infrastructure runs a virtual machine monitor (VMM) [3] on top of the hardware-firmware level to multiplex the node physical resources among VMs and strongly confine them.

The number of overlays that share the infrastructure depends upon user *communities*, because a distinct overlay, or *virtual community network* (VCN), is introduced for each community. A community consists of a set of users that execute applications and of services that these applications exploit. The users and applications in a community can be handled in a uniform way because they have homogeneous security and reliability requirements. Communities can also cooperate and exchange information. Proper consistency and security checks are applied within a community, while more severe checks are enforced to cross the community border. When defining a community, an administrator pairs it with a *global level*, which defines the set of users that can join the community, the applications they can run and the resources they can access. In this way, the global level is the same for all the VMs in a community and they can be homogeneously managed because they have similar requirements. Hence, the notion of community simplifies the management of the VMs, because VMs of the same community require the same reliability level and the data they exchange can be protected through the same mechanisms.

The rest of the paper is organized as follows. Section 2 presents the overall architecture of Vinci and discusses the various VM templates introduced to run user applications, to build the overlays and to support the correct sharing of the infrastructure among VMs and among communities. Section 3 presents a first set of performance results. Section 4 reviews some related works. Finally, Sect. 5 draws some conclusions.

## 2 Vinci Overall Architecture

An example of an infrastructure where Vinci can be applied is the one of a hospital that is shared, at least, among the doctor community, the nurse community and the administrative community. Since each community manages its private information but also shares some information with the other ones, a community should be able to define its own security policy, its reliability requirements and to control information to be shared with the other ones. As an example, users in a doctor community can update the information about prescriptions whereas those in the nurse community can read but

not update the same information. The nurse community and the doctor one share some other information with the administrative community, which has to bill the patient insurances. In the most general case, each user belongs to several communities according to the applications she needs to run and the data she wants to access. Consider a doctor that is the head of the hospital: as a doctor she belongs to the doctor community but, because of her administrative duties, she belongs to the administrative community as well. Furthermore, the community the doctor joins to access critical health information differs from that she joins when surfing the Internet.

In the general case, we assume that the infrastructure architecture is a private network that spans several locations, it includes a rather large number of physical nodes, and it is centrally managed by a set of administrators. We also assume that most of the nodes of the infrastructure are personal computers that are only accessed by one person at time and that the infrastructure includes a set of server nodes, which store shared data and execute server applications. Vinci requires that each node runs a virtual machine monitor (VMM), a thin software layer on top of the bare machine that creates and manages several concurrent emulation environments, the VMs. The VMM is responsible of the confinement among the VMs and guarantees a fair access to the node resources.

One of the main advantages of virtualization is the ability of choosing the appropriate combination of OS and applications for each VM to minimize the overall complexity. To exploit at best this feature, Vinci defines a set of highly specialized and simple VM templates that are dynamically instantiated and connected into overlays, i.e. virtual community networks (VCNs). A Vinci VCN includes both VMs that run applications and VMs that support and monitor the previous ones. A VCN strongly resembles a virtual private network (VPN) but an important difference lies in the granularity of the computation because we are interested in minimizing the complexity of the services each VM implements. As an example, some VMs are introduced in a VCN just to apply consistency and security checks to the overall computation.

In Vinci, each VCN is built by composing VMs that are instances of the following templates:

1. *Application VM*: it runs a set of applications on behalf of a single user;
2. *Community VM*: it manages the private resources of a community by enforcing mandatory and/or discretionary access control (MAC/DAC) policies;
3. *File System VM*: it belongs to several VCNs to protect files shared among the corresponding communities. It can implement MAC and Multi-Level Security policies and a tainting mechanism to prevent illegal information flows across communities;
4. *Communication and Control VM*: it implements and monitors information flows among communities, i.e. flows among Communication and Control VMs of distinct communities, or private flows among VMs of the same community;
5. *Assurance VM*: it checks that Application VMs only run authorized software and attests the software of a VM.

Moreover, Vinci introduces *Infrastructure VMs* that do not belong to any VCN and extend the VMMs with new functionalities to manage the overall infrastructure. As shown in Fig. 1, since VMs that are instances of the same template have homogeneous requirements and system configurations, they are easy-to-deploy virtual appliances created on

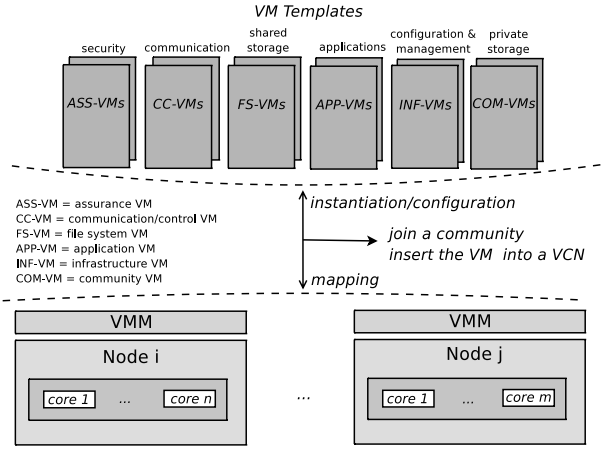


**Fig. 1.** VM Templates

demand from a generic baseline image. Moreover, when a VM is instantiated, its runtime environment is highly customized [4] according to the user and the community of interest through parameters such as the amount of memory, the running kernel modules, and the OS and applications versions. This results in the ability of strengthening each VM by tailoring its configuration to minimize the software it runs and avoid useless functionalities.

In the following, we describe the current implementation of Vinci that exploits Xen [5] to create the VMs and connect them into VCNs. NFSv3 [6] and Security-Enhanced Linux (SELinux) [7] [8] have been modified to apply security policies based upon the security levels of users or the global levels of communities. Finally, interconnections among VMs are handled through iptables [9] and OpenVPN [10].

### 2.1 Community and VCN

The key concepts of Vinci are those of community and of virtual community network (VCN). A community is composed by users and applications having the same security and reliability requirements. Users run their applications in Application VMs and, therefore, a community can also be seen as built around a set of Application VMs that share the same *global level*, i.e. the community level. This level constrains the security and reliability requirements of users that can join a community and the applications they can run. The number of communities reflects the distinct classes of users that share the private infrastructure. As an example, an administrator of a corporate enterprise can configure Vinci to support the following communities: sales, marketing, management, finance, R&D, engineering, customer support, Internet, etc. The adopted definition stresses the notion of a community as a collaborative environment where sharing of information among applications does not result in a loss of security or reliability. A VCN includes both the set of Application VMs of a community and further VMs that manage the resources of a community and interaction among communities.

### 2.2 Application VMs

Each Application VM runs applications of a single user and is paired with the global level inherited from the corresponding community. In general, the resources and services an Application VM can access depend upon the user security level and the global level of the community the VM belongs to. Since a user can join distinct communities through distinct Application VMs, she can access distinct resources/services according to the global level of each community. In some cases, there may exist some resources a user can access regardless of the community she currently belongs to. As an example, each user can always access its private files. While the global level of a community constrains the users that can join the community and the applications they can run, an administrator can also dictate which resources a community can access and/or share with other communities. Thus, when a user wishes to join a community, and she has the rights to do so, the community statically defines the set of applications that the Application VM can run and the resources it can access.

In Vinci, during the login phase, a user chooses the community she wants to join. Then, the Infrastructure VM of the local node configures and starts up a corresponding Application VM to run those applications that the community policy allows and it inserts the Application VM into the proper VCN. A user can run several Application VMs on the same node concurrently, each belonging to the same community or to distinct ones.

In the current prototype, each Application VM is associated with a minimal partition on one of the disks in the physical node, which stores the OS kernel loaded during the boot-up of the Application VM. Other files may be stored either locally, in a Community VM in the same VCN, or in a File System VM shared with other VCNs. To simplify the implementation of security policies, at boot time an IP address is statically assigned to an Application VM and both the VM global level and the user security level are paired with this address. Since the IP address uniquely determines the resources the VM and the user can access, Communication and Control VMs implement proper checks to detect any spoofed traffic in a VCN.

### 2.3 Community VMs

A Vinci VCN always includes at least one Community VM to manage and control the resources shared within the corresponding community, i.e. among its Application VMs only. This VM stores the community private files, which includes configuration files, system binaries, shared libraries and user home directories.

In the current prototype, files shared through Community VMs are protected by MAC and DAC security policies where the subject of the policy is the user security level, which is deduced from the IP address of the requesting Application VM. To this purpose, we have extended SELinux and NFS so that the NFS server enforces a policy where the subject is the user paired with the IP address of the requesting Application VM. In more detail, SELinux labeling and access rules have been extended to introduce a new subject that corresponds to an Application VM, and to define the operations it can invoke. We have also extended the object class of SELinux network object (*node*) to insert the operations that the NFS server executes on behalf of the requesting Application VM such as read, write or create files or directories. A node object is used to control

the network traffic, for example to grant or deny a process the permissions to exchange data with a specific IP address and it is associated with the IP address of an Application VM. In this way, the administrator can define a distinct protection domain for each Application VM by dynamically pairing the NFS server process with the security context of the Application VM currently requesting the operation on a file.

SELinux stores most runtime security information about the running processes in the `task_security_struct` structure. We have extended this structure to include the security identifier bound to the node type and, therefore, to the corresponding subject. Moreover, we have introduced a function, `map_ip`, which maps the Application VM IP address into a security identifier (SID) according to the current policy in the SELinux security policy database. In this way, the security context can be deduced from the IP address of the Application VM that is trying to access a shared file. Every time the NFS server processes a request, if the policy pairs the requesting IP address with a node type, `map_ip` returns the SID of the requesting Application VM. Otherwise, `map_ip` returns a default unprivileged SID. Before invoking the file system operation, the Application VM SID is copied into the `task_security_struct` paired with the NFS daemon process serving the request. In this way, anytime the NFS server invokes an operation on a shared file system on behalf of Application VMs, the Community VM kernel triggers a Linux Security Module (LSM) hook to delegate the security controls to SELinux. Accordingly, we have modified the LSM hooks paired with file system operations so that the subject of the security policy is the SID paired with the IP address of the Application VM that has invoked the operation rather than the NFS server.

### 2.4 File System VMs

A community can share some files with other communities through a file system implemented by a File System VM that belongs to several VCNs. The security policies that this VM enforces extend those of a Community VM by considering both the user security level and the community each user belongs to. Furthermore, File System VMs exploit the capabilities of a *Tainting module* to prevent the flowing of information among predefined communities. This module has been introduced to: (i) confine information flows among communities; (ii) increase the robustness with respect to contamination attacks; (iii) log the actual flow of information among communities. To this end, the Tainting module pairs each community with a bit mask, i.e. the community mask, with exactly one bit set to 1 that represents the community. Furthermore, it pairs each file with a mask that represents the communities that either have interacted with the file or have exchanged some information through the file. Anytime a user tries to apply an operation on a file, the module computes an `OR` of the masks of the file and of the user community. If the result shows an illegal information flow among communities, then the operation is forbidden, otherwise the result becomes the new mask of the file, in the case of a write operation, or of the community, in the case of a read operation. In any case, the Tainting module logs into a file the operation type, the name of the community and of the file and the original and the new masks.

Periodically, the Tainting module parses the log file and updates in an incremental way the dependency graph [11] that represents the information flows among communities and files. Each node in the graph represents either a file or a community and it is

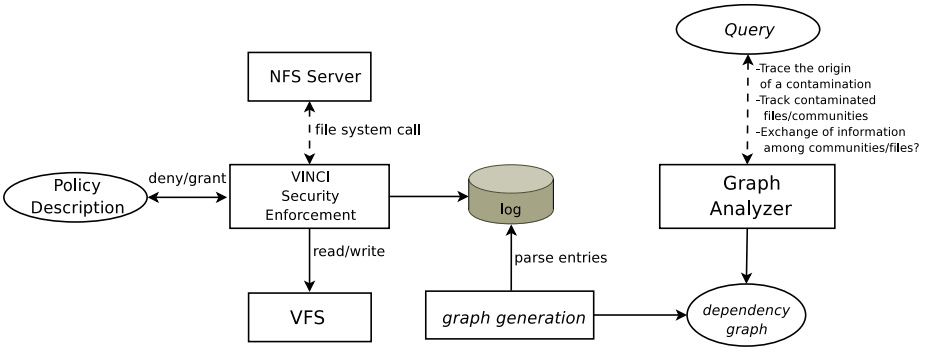


**Fig. 2.** File System VM Policy Enforcement and Query Generation

paired with a unique identifier of the community or of the file as well as with the corresponding mask. A node that represents a file is created the first time the file is involved in an operation. An arc represents an operation and is associated with the information about the requested operation. To identify how information flows, a read operation is represented by an arc from a node that represents a file to one that represents a community, while a write operation by an arc from a community to a file. As shown in Fig. 2, an administrator can query the module to analyze the dependency graph to discover those communities that have exchanged some information, to trace the source of a contamination and track all the files/communities that may have been contaminated by a community/file.

Since the implementation of a File System VM generalizes that of a Community VM, the same extensions of Community VMs have been applied to File System VMs as well. Moreover, we have extended the File System VM Linux kernel to insert the Tainting module in-between the NFS server and the Virtual File System (see Fig. 2). To this purpose, we have modified the `nfsd_permission` function, which verifies file requests, and `nfsd_vfs_read` and `nfsd_vfs_write` to check, respectively, read and write requests.

### 2.5 Communication and Control VMs

A Communication and Control VM protects and monitors information flows by implementing and managing a local virtual switch that supports communications among VMs. Communication and Control VMs can be further specialized in: (i) *Virtual Private Network (VPN) VMs*, which create an authenticated and protected communication channel among VMs of the same community on distinct physical nodes of the infrastructure; (ii) *Firewall VMs*, which filter information flows so that only authorized Application VMs can access the infrastructure and interact with other communities; (iii) *IDS VMs*, which monitor information flowing among VMs in the same community or in distinct ones. The introduction of a Firewall VM enables the administrators to define which communities can interact and, hence, to define the legal interconnections among distinct VCNs. As an example, a community can be isolated or communities with a lower global level may not be allowed to communicate with those with higher

global levels. Firewall VMs can decide whether to forward a packet, based on the source and destination communities. Furthermore, Firewall VMs support the authentication of shared resource requests through the IP address of the originating VM because they control that Application VMs do not spoof traffic on the virtual switches interconnecting the VMs. Finally, IDS VMs monitor information flows among VMs in the same or in distinct communities to discover attacks. An IDS VM can also retrieve and correlate partial information from other IDS VMs in the same VCN or in distinct ones to minimize the time to detect a distributed attack [12].

### 2.6 Assurance VMs

Assurance VMs exemplify the advantages of virtualization to build a robust system. In fact, these VMs have been introduced just to monitor critical Application VMs and the VMs of a VCN that manage critical components, to attest their integrity and to authenticate their configuration as well. Furthermore, to define severe tests, Assurance VMs use virtual machine introspection [13] to retrieve critical data structures in the VM memory and evaluate assertions on these structures. Each assertion is an invariant for the original application and it is false only if the application has been attacked.

The software attestation implemented by Assurance VMs enables a VM to verify the integrity of critical code in another VM. As an example, an Assurance VM can compare a hash of the software of a VM against a value computed offline to discover anomalous updates of the configuration of the VM. Consider a critical server in an Application VM that is remotely accessed by other Application VMs. In this case, the server Application VM may require not only the client authentication but also have some assurance that the software stack of the client VM is not compromised and that the client version is correct.

### 2.7 Infrastructure VMs

Infrastructure VMs extend the VMMs to configure and manage the VCNs. In particular, distinct Infrastructure VMs cooperate to monitor the overall infrastructure and update the topology of the virtual overlays and their mapping. The adoption of these VMs simplifies the implementation of some functionality too complex to be applied at the VMM level, thus minimizing the VMM size. An Infrastructure VM runs a minimal kernel, it does not run any Internet service and the functionalities it implements cannot be directly accessed by any user but the administrators.

As shown in Fig. 3, all the Infrastructure VMs, one per node, belong to a *Management Community* that does not interact with any other community. During the creation of the Management Community, one Infrastructure VM is designated as the *Master Infrastructure VM* that contacts the other Infrastructure VMs to set up proper communication channels to support cooperation in the Management Community. To properly configure the Vinci runtime environment, Infrastructure VMs can: (i) create/kill, freeze/resume any VM in their node or request this operation to another Infrastructure VM; (ii) configure a VM through specific parameters such as network configuration, amount of memory, the number of VCPUs; (iii) retrieve information about the current mapping of VMs and the resulting resource usage; (iv) update the mapping by migrating VMs, which requires an interaction with some Communication and Control VMs

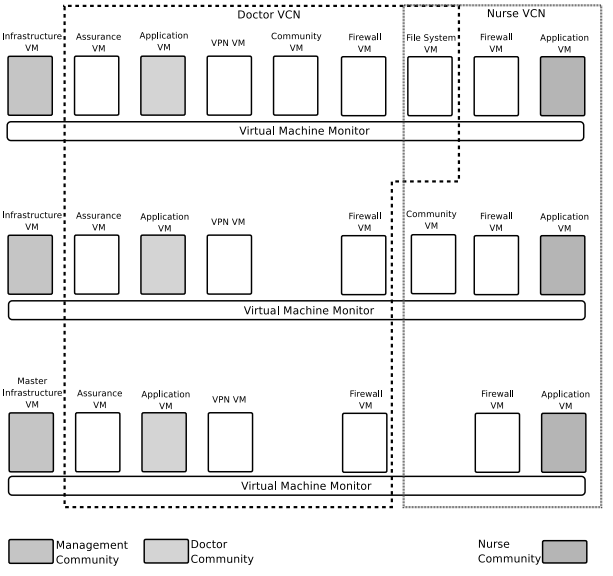


**Fig. 3.** Example of Communities and VCNs

to manage the resulting communications; (v) setup, compile and deliver to each File System and Community VM the general security policy it has to enforce.

Among the challenges that the Management Community has to face, one is concerned with data management issues, to enable a fast access of a community to its data [14], or with VMs mapping. Alternative mapping strategies may evenly distribute Application VMs running server applications on the available nodes, or map Community VMs onto physical nodes directly connected to those that run the Application VMs of the corresponding community. The Management Community may migrate VMs among physical nodes to handle errors and faults, to reduce the communication latency or to balance the computational load.

Infrastructure VMs also authenticate users through a centralized authentication protocol, so that users can log on Application VMs with the same combination of username and password anywhere in the infrastructure. In this way, the association among users and privileges is managed in a centralized way. The set of users of all the communities that share the infrastructure is globally known so that Vinci can uniquely identify users through their user-name or their associated user identifier (UID). The UID is paired with the privileges of the user, i.e. with its security level. Whenever a user has been authenticated and has chosen the community she wants to join, the Infrastructure VM starts up an Application VM, which includes only those applications that the community policy allows and with the proper global level. After the login and boot-up phases, the local Infrastructure VM contacts the proper Communication and Control VMs to update the topology of the VCN to insert this Application VM into a VCN and to add communication rules to handle the corresponding information flows. Finally, the security policies of Community VMs and File System VMs may be updated. Figure 4

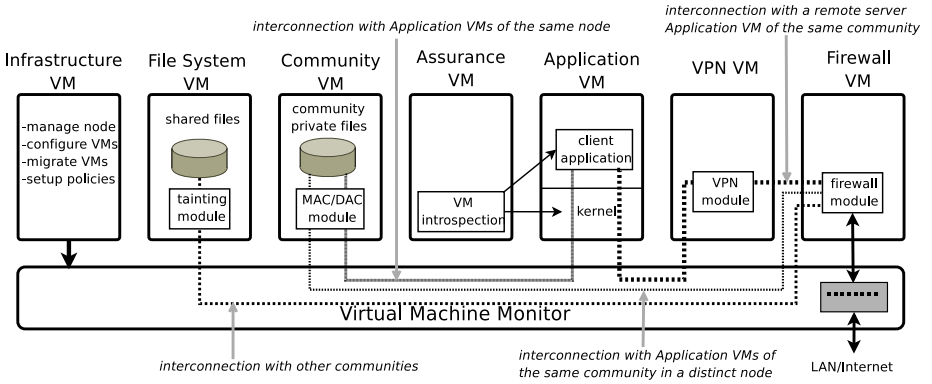


**Fig. 4.** Interactions among Communities

shows the various interactions among the VMs running on a physical node. Currently, Infrastructure VMs are assigned to Xen Domain 0 VMs, i.e. they are privileged VMs that can access the control interface to manage a physical node and the VMs that the node runs.

If a high degree of assurance is required, Vinci may require that some nodes of the infrastructure are equipped with a Trusted Platform Module (TPM) subsystem [15], which can also be virtualized on the VMs the node runs [16]. The TPM acts as a *root of trust* to build and setup the software environment and its mechanisms may be exploited to ensure that a platform has loaded its software properly and that the VMM and the Infrastructure VM of the local node are started in a safe state. Moreover, the TPM can protect secrets such as the asymmetric and symmetric keys to establish secure communications among VMs, to authenticate the VMs or to remotely attest the software that a VM runs. If a high degree of assurance is required but a TPM is not available, administrator should guarantee that each node cannot be physically tampered with so that the VMM and the Infrastructure VM can be safely initialized. In this case, the Infrastructure VM may emulate the features of the TPM and export it to other VMs through the virtual TPM.

## 3 Performance Results

This section shows a preliminary performance evaluation of the current Vinci prototype. The tests were performed on several machines equipped with Intel Core 2 Duo E6550 2.33GHz CPUs. A first experiment evaluated in an integrated way the performance of file sharing through File System VMs and Communication and Control VMs. An Application VM on a node ran the IOzone [17] NFS test while a File System VM, on a distinct physical node, stored the requested files. Requests were transmitted along a communication channel implemented by two Firewall VMs and the physical nodes were connected through a 100MB Ethernet. Fig. 5 compares the throughput of the `write` test against the one of the insecure version that does not apply the security checks. The overhead due to the enforcement of the security policies, in the average case, is lower

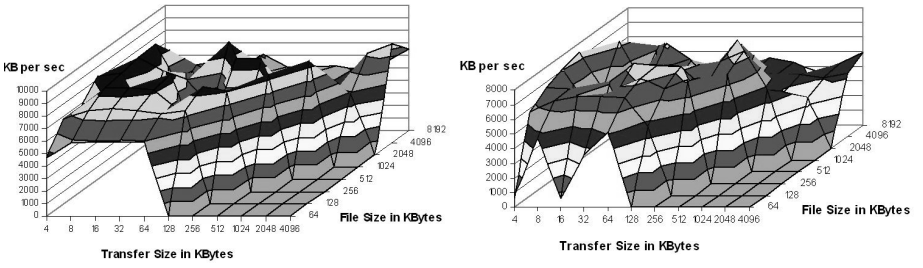


**Fig. 5.** IOzone NFS Read Performance without (left) and with (right) Policy Enforcement

than 9%. Instead, the tests on the enforcement of security policies by a Community VM resulted in a reduction of the final throughput lower than 5%. The same tests executed on an Application VM connected to a remote File System VM through two VPN VMs resulted in an overhead that, in the worst case, is lower than 13%.

We also evaluated the overhead due to the security checks enforced by an Assurance VM on the kernel code of an Application VM. To this purpose, an Application VM executed the command `tar -xjf linux-2.6.20.tar.bz2` while an Assurance VM, which runs on the same node, checked an increasing number of Application VM kernel pages by computing their hashes and comparing them against their original values. The period of time between two consecutive executions of the consistency checks was set to 2 seconds. The relative overhead was lower than 8% in the worst case. The coverage of these checks is rather satisfactory, because they quickly detect, for instance, any attempt to install a rootkit.

## 4 Related Works

[18] proposes an architecture where computing services can be offloaded into *Trusted Virtual Domains*, i.e. execution environments that meet a set of security requirements. *Trusted Grid Architecture* [19] is a framework to build a trustworthy grid architecture by using a combination of Trusted Computing and virtualization technologies. The proposed approach allows a user to check that a selected provider is in a trusted state before accessing a submitted grid job. Both the previous architectures consist of a grid of nodes where clients require services, using some form of negotiation to locate a trustworthy provider. Vinci, on the other hand, is more resemblant of a traditional client/server approach, where services and applications are mostly statically bound to physical nodes. Moreover, in Vinci users are managed by a central authority that also administers the whole architecture. [20] proposes an access control architecture that enables corporations to verify client integrity properties using a TPM, and to establish trust upon the capability of the client to enforce the policy before allowing the client to access the corporate Intranet. This framework could be easily integrated into Vinci, for example when a remote user tries to access the corporate infrastructure. *SVGrid* [21] introduces distinct execution environments for the applications and the storage areas, to protect file systems and network services from untrusted grid applications. In addition to these features,

Vinci aims also to provide assurance on the running software on critical VMs. Moreover, Vinci introduces a large number of VMs templates, so that each VM only runs a small amount of software. *Poly*$^2$ [22] is a framework aimed at segregating applications and networks and at minimizing OSes. The proposed approach separates network services onto different systems and it isolates specific types of network traffic. To this purpose, administrative and application-specific traffic are mapped onto distinct networks. Moreover, minimized OSes should only provide the services required by a specific network application. Vinci shares with this framework the minimization of OSes and applications but it introduces distinct overlay networks for each community and dynamically manages the configuration of these overlays. Moreover, Vinci applies introspection to provide assurance on the running software. [23] considers VMs as sandboxes that simplify the deployment of collaborative environments over wide-area networks. Each VM sandbox can be seen as a virtual appliance made available to several users by the administrator, so that new nodes can easily join and be integrated into the virtual network. A feature that characterizes Vinci with respect to the previous framework is the concept of community, which simplifies the management of users with similar security and reliability requirements. PlanetLab [24] is a global overlay network that runs concurrently multiple services in *slices*, i.e. networks of VMs that include some amount of processing, memory, storage and network resources. A slice is conceptually similar to a Vinci VCN, but in Vinci resources are those of a private infrastructure and their allocation is mostly static, whereas PlanetLab exploits the concept of an open grid of machines where resources can be dynamically allocated and discovered. Furthermore, Vinci introduces the concept of community, which allows administrators to define flexible security policies. Another important difference is that PlanetLab exploits OS-level virtualization, while Vinci exploits hardware-level virtualization and, therefore, it introduces a VMM that results in a better confinement among the VMs of the same node.

## 5 Conclusion

The focus of Vinci is on the definition of highly parallel overlays, i.e. VCNs, which simplify management and sharing of a private ICT infrastructure. Moreover, VCNs increase the overall robustness due to the introduction of specialized VMs that enforce security checks within and across VCNs. Each VCN supports a user community, i.e. a set of users with similar security and reliability requirements. This approach requires that each physical node runs a virtual machine monitor that manages and protects the physical resources and it results in the definition of several VM templates. Distinct templates are used to support the execution of user applications, to enforce consistency checks or to apply security policies that protect resources shared within or across distinct communities. We have shown that proper components can be introduced to protect the communities from contamination as well as to trace any contamination that can arise. Moreover, a specialized community includes a set of VMs that manage the configuration of the other VMs to achieve the required level of reliability and security. Preliminary results of our prototype show that Vinci can be adopted in a real-world scenario, such as the one of a hospital, a bank, or a corporate enterprise, which need to guarantee high security requirements and confine critical resources.

Concerning the shortcomings of state-of-the-art virtualization, we note that managing and configuring a node to support virtualization and to run a set of VMs, each with a customized OS, requires more effort than managing a node that only runs a standard OS, especially in the case of a para-virtualization approach that requires the OSes to be modified to run inside the virtual environment. However, since virtualization is becoming increasing popular, there is a large amount of tools that help the administrator to set-up and manage virtual nodes. Another counterpart of the advantages of virtual, highly parallel and secure overlays is the overhead due to the context switching that the VMM applies to multiplex the physical resources. A multi-core architecture [25] can strongly reduce this overhead because of the native support for multiplexing. Moreover, it can run several VMs in consolidation and assign a dedicated core to some VMs. This guarantees that VMs that implement critical tasks, such as management and protection of other VMs, are never delayed. Moreover, the virtualization overhead can be strongly reduced because of the extension of hardware instruction sets to efficiently support virtualization technology [26].

## Acknowledgments

We would like to thank the anonymous reviewers for their valuable comments and suggestions.

## References

1. Uhlig, R., Neiger, G., Rodgers, D., Santoni, A., Marting, F., Anderson, A., Bennett, S., Kagi, A., Leung, F., Smith, L.: Intel Virtualization Technology. Computer 38(5), 48–56 (2005)
2. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M.A., Chen, P.M.: Revirt: enabling intrusion analysis through virtual-machine logging and replay. SIGOPS Oper. Syst. Rev. 36(SI), 211–224 (2002)
3. Goldberg, R.P.: Survey of virtual machine research. IEEE Computer 7(6), 34–45 (1974)
4. Huang, W., Abali, B., Panda, D.: A case for high performance computing with virtual machines. In: Proc. of the 20th annual international conference on Supercomputing, pp. 125–134 (2006)
5. Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Pratt, I., Warfield, A., Barham, P., Neugebauer, R.: Xen and the art of virtualization. In: Proceedings of the ACM Symposium on Operating Systems Principles (October 2003)
6. Callaghan, B., Pawlowski, B., Staubach, P.: NFS V3 Protocol Specification. RFC 1813
7. Loscocco, P., Smalley, S.: Integrating flexible support for security policies into the linux operating system. In: Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference, Berkeley, CA, USA, pp. 29–42. USENIX Association (2001)
8. Loscocco, P.A., Smalley, S.D.: Meeting critical security objectives with security enhanced linux. In: Proceedings of the 2001 Ottawa Linux Symposium. (2001)
9. Netfilter.org: Netfilter/Iptables project, `www.netfilter.org/`
10. OpenVPN: OpenVPN - An Open Source SSL VPN Solution, `http://openvpn.net/`
11. King, S.T., Chen, P.M.: Backtracking intrusions. ACM Trans. Comput. Syst. 23(1), 51–76 (2005)
12. Cheetancheri, S.G., et al.: A distributed host-based worm detection system. In: LSAD 2006: Proc. of the 2006 SIGCOMM workshop on Large-scale attack defense, pp. 107–113. ACM Press, New York (2006)

13. Garfinkel, T., Rosenblum, M.: A virtual machine introspection based architecture for intrusion detection. In: NDSS (2003)
14. Figueiredo, R.J., Dinda, P.A., Fortes, J.A.B.: A case for grid computing on virtual machines. In: ICDCS 2003: Proceedings of the 23rd International Conference on Distributed Computing Systems, Washington, DC, USA, p. 550. IEEE Computer Society, Los Alamitos (2003)
15. Pearson, S.: Trusted Computing Platforms, the Next Security Solution. Trusted Computing Group Administration, Beaverton (2002)
16. Berger, S., Cáceres, R., Goldman, K.A., Perez, R., Sailer, R., van Doorn, L.: vTPM: virtualizing the trusted platform module. In: USENIX-SS 2006: Proceedings of the 15th conference on USENIX Security Symposium, Berkeley, CA, USA, pp. 21. USENIX Association (2006)
17. IOzone: IOzone Filesystem Benchmark, `http://www.iozone.org/`
18. Griffin, J.L., Jaeger, T., Perez, R., Sailer, R., van Doorn, L., Caceres, R.: Trusted Virtual Domains: Toward secure distributed services. In: Proc. of 1st IEEE Workshop on Hot Topics in System Dependability (HotDep) (2005)
19. Löhr, H., Ramasamy, H.V., Sadeghi, A.R., Schulz, S., Schunter, M., Stüble, C.: Enhancing Grid Security Using Trusted Virtualization. In: Xiao, B., Yang, L.T., Ma, J., Muller-Schloer, C., Hua, Y. (eds.) ATC 2007. LNCS, vol. 4610, pp. 372–384. Springer, Heidelberg (2007)
20. Sailer, R., Jaeger, T., Zhang, X., van Doorn, L.: Attestation-based policy enforcement for remote access. In: CCS 2004: Proceedings of the 11th ACM conference on Computer and communications security, pp. 308–317. ACM Press, New York (2004)
21. Zhao, X., Borders, K., Prakash, A.: SVGrid: a secure virtual environment for untrusted grid applications. In: MGC 2005: Proceedings of the 3rd international workshop on Middleware for grid computing, pp. 1–6. ACM Press, New York (2005)
22. Bryant, E., Early, J., Gopalakrishna, R., Roth, G., Spafford, E., Watson, K., William, P., Yost, S.: Poly$^2$ Paradigm: A Secure Network Service Architecture. In: Proceedings 19th Annual Computer Security Applications Conference, 2003, pp. 342–351 (2003)
23. Wolinsky, D.I., Agrawal, A., Boykin, P.O., Davis, J., Ganguly, A., Paramygin, V., Sheng, P., Figueiredo, R.J.: On the Design of Virtual Machine Sandboxes for Distributed Computing in Wide Area Overlays of Virtual Workstations. In: First Workshop on Virtualization Technologies in Distributed Computing (VTDC) (November 2006)
24. Chun, B., Culler, D., Roscoe, T., Bavier, A., Peterson, L., Wawrzoniak, M., Bowman, M.: Planetlab: an overlay testbed for broad-coverage services. SIGCOMM Comput. Commun. Rev. 33(3), 3–12 (2003)
25. Gepner, P., Kowalik, M.F.: Multi-core processors: New way to achieve high system performance. In: PARELEC 2006: International symposium on Parallel Computing in Electrical Engineering, pp. 9–13. IEEE Computer Society Press, Washington (2006)
26. Leung, F., Neiger, G., Rodgers, D., Santoni, A., Uhlig, R.: Intel Virtualization Technology: Hardware support for efficient processor virtualization. Intel Technology Journal 10(3), 167–178 (2006)