Towards Designing Secure Virtualized Systems

Hedi Benzina
LSV, ENS Cachan, CNRS, INRIA
61 avenue du Président Wilson, 94230 CACHAN, France
Email: benzina@lsv.ens-cachan.fr

***Abstract*—Virtual machine technology is rapidly gaining acceptance as a fundamental building block in enterprise data centers. It is most known for improving efficiency and ease of management. However, it also provides a compelling approach to enhancing system security, offering new ways to rearchitect todays systems and opening the door for a wide range of future security technologies. While this technology is meant to enhance the security of computer systems, some recent attacks show that virtual machine technology has many weaknesses and becomes exposed to many security threats. In this paper we present some of these threats and show how we protect these systems through intrusion detection and security policies mechanisms.**

Keywords: Virtualized systems, security, virtual networks., security policies.

I. Introduction

Virtualization refers to the creation of a virtual machine that acts like a real computer with an operating system. Software executed on these virtual machines is separated from the underlying hardware resources. For example, a computer that is running Microsoft Windows may host a virtual machine that looks like a computer with Ubuntu Linux operating system; Ubuntu-based software can be run on the virtual machine. A hypervisor, also called virtual machine manager (VMM), is the software doing virtualization, e.g. Xen [1], VirtualBox [2], and VMWare [3].

In recent years, virtualization has been seen by several as an opportunity for enforcing better security. The fact that two distinct VMs indeed run in separate sandboxes was indeed brought forward as an argument in this direction. However, many security threats apperead recently since

A virtual network can be built between VMs, this allows them to communicate by simple network primitives. This kind of networks can be seen as a solution to the complexity of building physical networks, on the other hand, most of the security threats we face in a non-virtualized environment exist in virtualized environments as well. Furthermore, virtual networks have other security weaknesses related to the the architecture of the network, since everything is located in the same machine. This needs serious defence and rigourous security policies. We propose in this paper a multi-level security policy that covers common network operations and administrative actions. We take into consideration the constraints that must be satisfied during the communication between VMs and propose the policy model and discuss its implementation.

This work was supported by grant DIGITEO N°2009-41D from Région Ile-de-France.

II. Related work

A body of existing work has already examined the issues arised by virtualized architectures [5][6][7]. The introduction of the Xen Security Modules (XSM) framework enables the enforcement of comprehensive control over the resources of the hypervisor. The XSM policy model is based on SELinux [8], so VMM policies will be comprehensive, but determining whether a security goal is enforced correctly seems to be non-trivial for beginning users due to the complexity of policy rules organization.

sHype [10] is one of the best-known security architecture for hypervisors : its primary goal was to control the information flows between VMs. sHype is based on the Xen hypervisor and does not protect other virtualized architecture.

In [11] [12], a role-based access control policy was introduced to VMMs by Hirano et al. This policy focuses only on the access between guest VMs and the VMM layer, and does not treat inter-VM communication.

III. Securing Virtual machines

A. An autoprotection mechanism for administrators

In [16] we present an approach where we give the administrator of the virtualized system the mean to write security policies and compile them into attack signatures that can be fed up to our Intrusion Prevention System (IPS) called Orchids [13] [14]. Our point is that $signatures$, i.e., specifications of attack patterns, are best expressed in a logic including temporal connectives to express ordering of events. This allows one to describe attacks in a declarative way, free of implementation decisions. As in programming languages, using a $declarative$ language allows one to focus on $what$ to monitor instead of how to monitor. This caters for easier writing and easier understanding of signatures, and improves maintainability of signature files. We have implemented a tool : *RuleGen* [15], which takes as input a security policy written in temporal logic and translates it into Orchids' automata *i.e.* attack signature[16]. This approach is efficient against Denial of Service and local attacks executed in an isolated virtual machine, but shows its limits when the system faces complex network attacks or remote attacks, hence the need of a security policy taking into consideration all possible actions performed by system users. In the rest of the paper, we present our security policy model dedicated to inter-VMs communications. We will present the model and briefly give some security requirements and constaraints that must be respected by users.

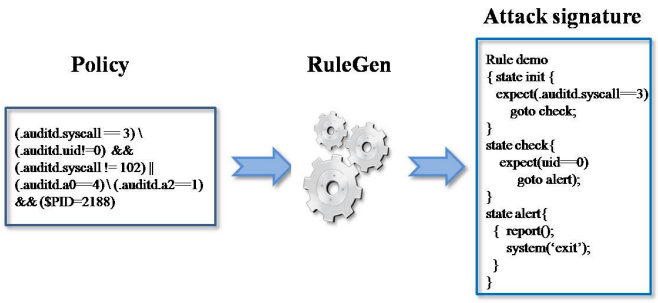

Fig. 1. The RuleGen tool

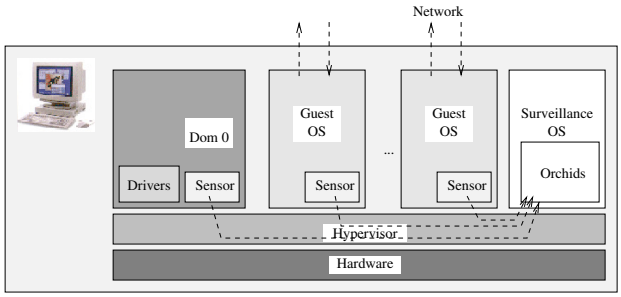

Fig. 2. Decentralized Supervision based on a virtual network

IV. Communication in Virtual Environments

In [16] we showed that virtual networks can be very useful for intrusion detection by proposing a decentralized supervision architecture on a single physical host based on the Xen hypervisor. This architecture is based on a virtual network allowing the communication between ordinary VMs, the surveillance VM and the administration VM called $domain0$. See Figure 2, which is perhaps more typical of Xen than other hypervisors.

On the other hand, the rapid scaling in virtual networks can tax the security system. In fact, the fast and unpredictable growth that can occur with VMs can exacerbate management tasks and significantly multiply the impact of catastrophic events, e.g. worm attacks where all machines should be patched, scanned for vulnerabilities, and purged of malicious code.

Collections of specialized VMs give rise to a phenomenon in which large numbers of machines appear and disappear from the network sporadically. While conventional networks can rapidly anneal into a known good configuration state, with many transient machines getting the network to converge to a known state can be nearly impossible.

For example, when worms hit conventional networks they will typically infect all vulnerable machines fairly quickly. Once this happens, administrators can usually identify which machines are infected, then cleanup infected machines and patch them to prevent re-infection, rapidly bringing the network back into a steady state. Besides, in an unregulated virtual environment, such a steady state is often never reached. Infected machines appear briefly, infect other machines, and disappear before they can be detected, their owner identified, etc. Vulnerable machines appear briefly and either become infected or reappear in a vulnerable state at a later time. Also, new and potentially vulnerable virtual machines are created on an ongoing basis, due to copying, sharing, etc. As a result, worm infections tend to persist at a low level indefinitely, periodically flaring up again when conditions are right. The requirement that machines be online in conventional approaches to patch management, virus and vulnerability scanning, and machine configuration also creates a conflict between security and usability. VMs that have been long dormant can require significant time and effort to patch and maintain. This results in users either forgoing regular maintenance of their VMs, thus increasing the number of vulnerable machines at a site, or losing the ability to spontaneously create and use machines, thus eliminating a major virtue of VMs.

For instance, rolling back a machine by the checkpoint and restore mechanism can re-expose patched vulnerabilities, reactivate vulnerable services, re-enable previously disabled accounts or passwords, use previously retired encryption keys, and change firewalls to expose vulnerabilities. It can also reintroduce worms, viruses, and other malicious code that had previously been removed.

A subtler issue can break many existing security protocols. Simply put, the problem is that while VMs may be rolled back, an attacker's memory of what has already been seen cannot. For example, with a one-time password system like S/KEY where a user's real password is combined in an offline device with a short set of characters and a decrementing counter to form a single-use password. In this system passwords are transmitted in the clear and security is entirely reliant on the attacker not having seen previous sessions. If a machine running S/KEY is rolled back, an attacker can simply replay previously sniffed passwords.

A more subtle problem arises in protocols that rely on the freshness of their random number source e.g. for generating session keys or nonces. Consider a virtual machine that has been rolled back to a point after a random number has been chosen, but before it has been used, then resumes execution. In this case, randomness that must be fresh for security purposes is reused.

V. Multilevel Network Security

Multi-level security was formalized by Bell and La- Padula [17] in order to control how information is allowed to flow between subjects in a system. These subjects are given a sensitivity level, or security clearance, and objects are also given a similar security classification. MLS policies attempt to restrict how information may flow between designated sensitivities. As an example, consider a military application with 4 sensitivities, ordered from least to most sensitive: Unclassified (UC), Confidential (CO), Secret (S), and Top Secret (TS). In this case, TS dominates S. Note that in this example the sensitivites form a total ordering; each sensitivity is either higher, lower, or equal to another. This is not always the case.

However, constraining how information may flow within a system is at the heart of many protection mechanisms and many security policies have direct interpretations in terms of multilevel security style controls. These include: Chinese Walls [18][19]; separation of duties and well formed transactions [20][21] and Role-Based Access Control [22].

Let us assume that we have a collection of trusted and untrusted VMs and we would like to connect them to form a secure virtual network. A network is said to be multilevel secure if it is able to protect multilevel information and users. That is the information handled by the network can have different classifications and the network users may have varying clearance levels.

VI. The Security Policy Model

In developing the security policy, we combine certain features of some well computer security models such as the Bell-LaPadula model together with issues relevant to network security. Informally, the network discretionary and mandatory access control policy can be described as follows : we assume that the information required to provide discretionary access control resides within each network component, rather than in a centralized access control centre. The network discretionary access control policy is based on the identity of the network components, implemented in the form of an authorized connection list. This list determines whether a connection is allowed to be established between two network entities. The individual components may in addition impose their own controls over their users - e.g. the controls imposed when there is no network connection.

The network mandatory security policy requires appropriate labelling mechanisms to be present. One can either explicitly label the information transferred over the network or associate an implicit label with a virtual circuit connection. In our model we have the following scheme :
(a) Each network component is appropriately labelled. A mandatory policy based on the labels of the network components is imposed and it determines whether a requested connection between two entities is granted or not.
(b) Information transferred over the network is appropriately labelled. A mandatory security policy is used to control the flow of information between different subjects and objects, when performing different operations involving information transfer over the virtual network.

A. Modelling approach

The network security policy model we describe here is a state-machine based model. Essentially a state machine model describes a system as a collection of entities and values. At any time, these entities and values stand in a particular set of relationships. This set of relationships constitutes the state of the system. Whenever any of these relationships change the state of the system changes. The common type of analysis that can be carried out using such a model is the reachabitity graph analysis. The reachability graph analysis is used to determine whether the system will reach a given state or not. For instance, we may identify a subset of states W which represent "insecure" states and if the system reaches a state within this subset W, then the system is said to be insecure. In describing such a state machine based security model, we need to perform the following steps :

- Define security related state variables in the network system.
- Define the requirements of a secure network state.
- Define the network operations which describe the system state transitions.

We make the following assumptions :

1) Reliable user authentication exists within each VM.
2) Only a user with the role of $Admin$ can assign security classes to network subjects and network components, and assign roles to users.
3) Reliable transfer of information across the network.

B. Model Representation

In order to be generic, our model needs to take into consideration the recent development in virtualized systems area, thus we will deal with Input/Output devices as separated VMs : in fact VMware, Xen and many other hypervisors tend to dedicate a whole VM for I/O [3], and sometimes for the processor (see Figure **??**), which reduces consequently the overhead for communicating the I/O and processor commands.

We define a network security model, MODEL, as follows :

$$MODEL =< S, O, s_0 >$$

where S is the set of States, O is the set of system Operations and s_0 is the initial system state.

Let us first define the basic sets used to describe the model:

- Sub : Set of all network subjects. This includes the set of all Users (Users) and all Processes (Procs) in the network. That is : $Sub = Procs \cup Users$
- Obj : Set of all network objects. This includes both the set of Network Components (NC) and Information Units (IU). That is : $Obj = NC \cup IU$.
  Typically, the set of Network Components includes virtual machines (VMs), Input-Output Devices (IOD) and Output Devices (OD) whereas Information Units include files and messages. That is : $NC = VMs \cup IOD \cup OD$
- $SCls$: Set of Security Classes. We assume that a partial ordering relation $\geq$ is defined on the set of security classes.
- Rset : Set of user roles. This includes for instance the role *Admin* dedicated to the administrator of the network who is typically the administrator of the whole virtualized architecture.

We use the notation x_s, to denote the element x at state s.

1) System State: We only consider the security relevant state variables. Each state $s \in S$ can be regarded as a 11-tuple as follows :
$s =< Sub_s, Obj_s, authlist, connlist, accset, subcls, objcls, curcls, subrefobj, role, currole, curvm >$

Let us now briefly describe the terms involved in the state definition :
- Sub_s and Obj_s defines respectively the sets of subjects and objects at the state s.
- $authlist$ is a set of elements of the form (sub, nc) where $sub \in Sub_s$ and $nc \in Obj_s$. The existence of an element (sub_1, nc_1) in the set indicates that the subject sub_1 has an access right to connect to the network component nc_1.
- $connlist$ is again a set of elements of the form (sub, nc). This set gives the current set of authorized connections at that state.
- $accset$ is a set of elements of the form $(sub, iuobj)$, where $sub \in Sub_s$, and $iuobj \in Obj_s$. The existence of an element $(sub_1, iuobj_1)$ in the set indicates that the subject sub_1 has an access right to bind to the object $iuobj_1$.
- $subcls : Sub \rightarrow SCls$, is a function which maps each subject to a security class.
- $objcls : Obj \rightarrow SCls$, is a function which maps each object to a security class.
- $curcls : Sub \rightarrow SCls$, is a function which determines the current security class of a subject.
- $subrefobj : Sub \rightarrow PS(Obj)$, is a mapping which indicates the set of objects referenced by a subject at that state.
- $role : Users \rightarrow PS(Rset)$, gives the authorized set of roles for a user.
- $currole : Users \rightarrow Rset$, gives the current role of a user.
- $curvm : Users \rightarrow NC$, is a function which gives the VM in which a user is logged on.
- $view : Sub \rightarrow Obj$, is a function that determines the objects that can be viewed by a subject.

2) Secure State: To define the necessary conditions for a secure state, we need to consider the different phases gone through by the system during its operation, we focus on typical network operations :

Login Phase : We require that if the user is logging through a VM, he must have appropriate clearance with respect to the VM. That is, the security class of the user must be above the security class of the VM in which the user is attempting to log on. In addition, the current security class of the user must be below the maximum security class of that user and the role of the user must belong to the authorized role set allocated to that user. So we have the following constraint:

- *Proposition 1 : Login Constraint* :

A state s satisfies the Login Constraint if $\forall x \in Users$:

- $subcls(x) \geq objcls(curvm(x))$
- $subcls(x) \geq curcls(x)$

Connect Phase : Having logged-on to the virtual network, a user may wish to establish a connection with another network component (VM or I/O VM). In determining whether such a connection request is to be granted, both network discretionary and mandatory security policies on connections need to be satisfied. The discretionary access control requirement is specified using the authorization list which should contain an entry involving the requesting subject and the network component. If the network component in question is a VM then the current security class of the subject must at least be equal to the lowest security class of that VM. On the other hand, if the network component is an output device, then the security class of the subject must be below the security class of that component. Hence we have the following constraint:

Proposition 2 : Connect Constraint :

A state s satisfies the Connect Constraint if $\forall (sub, nc) \in connlist$:

- $(sub, nc) \in authlist$
- if $nc \in VMs$, then $curcls(sub) \geq objcls(nc)$
- if $nc \in OD$ then $objcls(nc) \geq curcls(sub)$

Other Conditions We require two additional conditions :
(1) The classification of the information that can be "viewed" through an I/O device must not be greater than the classification of that device.
(2) The role of the users at a state belong to the set of authorized roles. Now we can give the definition of a secure state as follows :

- Definition : A state s is $Secure$ if :

- s satisfies the $Login$ Constraint
- s satisfies the $Connect$ Constraint
- $\forall z \in (IOD_s \cup OD_s), \forall x \in IU_s,$ $x \in view(z) \Rightarrow objcls(z) \geq subcls(x)$.

We assume that the initial system state s_0 is defined in such a way that it satisfies all the conditions of the secure state described above.

VII. Operations and their security requirements

In this section we will present the security constraints that must be satisfied by the different operations performed by the user of the virtual network : this includes vitual machines management operations done by the administrator (create/remove a VM, checkpoint/restore a VM), network operations such as $connect$ and $bind$ operations and finally operations related to the policy management (assign a security class to an object, assign a role to a user, etc).

A. Virtual machines managment operations

Create a new VM : Only the administrator of the virtual network is allowed to create new virtual machines. Once created, a new VM must be labelled by a security class which should be dominated by the security class of the Dom_0. This leads to the following constraints : if a subject sub wants to create a new virtual machine *newVM* then:

- $Admin \in role(sub)$ and $currole(sub) = Admin$
- $objcls(Dom_0) \geq objcls(newVM)$
- $NC'_s = NC_s \cup \{newVM\}$

Remove a VM : Only a user with the role $Admin$ is allowed to remove virtual machines. The only VM that cannot be removed is the administration VM, even by the administrator of the system (this is the normal case, but when we have other sensitive VMs such as the surveillance VM in our architecture, we can add restriction concerning the removal of this VM). This leads us to define the set *sensitiveVMs* which includes the Dom_0 in the case of Xen, the surveillance VM and may include other important VMs that cannot be removed.

We have the following constraints : if a user sub wants to remove a virtual machine *VM* then:

- $currole(sub) = Admin$
- $VM \notin sensitiveVMs$
- $authlist'_s = authlist_s \setminus (x, VM)$, where $x \in Sub$.
- $connlist'_s = connlist_s \setminus (x, VM)$, where $x \in Sub$.

After removing the VM the lists $authlist$ and $connlist$ are updated by removing the pairs where the deleted VM occurs.

Checkpoint and restore a VM : These functionalities are offred by most modern hypervisors. By creating checkpoints for a virtual machine, one can restore the virtual machine to a previous state. A typical use of checkpoints is to create a temporary backup before applying updates to the VM. The $restore$ operation enables to revert the virtual machine to its previous state if the update fails or adversely affects the virtual machine. Any user can checkpoint and restore his own VM, the user with the role $Admin$ can do this with any VM. To make sure that these two operations do not represent security threats, we need the following constraints.

If a user sub wants to checkpoint a virtual machine *vm1* then:

- $curvm(sub) = vm1$ or $currole(sub) = Admin$
- $VM \neq Dom_0$

In addition to these constraints, when restored, a VM must keep the same security class as before the checkpoint. Let s and z be respectively the states of the system bebore and after the checkpoint, we should have :

- $objcls_z(vm1) = objcls_s(vm1)$

B. Network operations

Connect operation : The operation $connect(sub, nc)$ allows a subject sub to connect to a remote network entity nc. From the Connect Constraint given earlier, for this operation to be secure, we require that :

- $(sub, nc) \in authlist$
- if $nc \in VMs$, then $curcls(sub) \geq objcls(nc)$
  or
  if $nc \in OD$ then $objcls(nc) \geq subcls(sub)$

After the operation is performed we should have : $(sub, nc) \in connlist'$ and $nc \in subrefobj(sub)$.

Having connected to a remote VM, a subject can perform operations which allow the manipulation of information objects. We envisage the information manipulation phase to consist of two stages : a binding stage and a manipulation stage. The binding stage involves a subject linking itself to the VM on which the operation is to be performed. At the manipulation stage, typically the operations include those operations defined by the Bell-LaPadula model such as $read$, $append$, $write$ and $execute$. In our model, we will only consider one basic manipulation operation which allows the transfer of an object from one VM to another, as this is perhaps the most important operation from the network point of view. This operation causes information to flow from one entity to another over the network. (In fact, this operation will form part of other operations as well. For instance, consider a read operation, whereby a user reads a file stored in a remote entity. This operation must include the transfer of the file from the remote network component to the local network component in which the user resides.) There are also other operations which modify certain security attributes of objects and subjects. In the usual computer security model, these include operations for assigning and changing security classes to users and information objects and assigning and modifying access sets for information unit objects. Note that in general for any operation to be performed, the subject must have authorized access to the connection with the remote entity. That is, the $Connect\ Constraint$ must be satisfied to begin with.

Bind operation : The operation $bind(iuobj, nc)$ allows a subject sub to link an information object $iuobj$ in a network component nc. The constraints that must be satisfied by this operation are:

- $(sub, iuobj) \in accset(iuobj)$
- $curcls(sub) \geq objcls(iuobj)$
- for any $sb \in Sub_s$, $iuobj \notin subrefobj(sb)$

After the operation is performed, we should have $iuobj \in subrefobj'(sub)$. Where $subrefobj'$ refers to the new state s'.

Note that we have included a simple access control based on $accset$ at the remote network component. In practice, a comprehensive access control mechanism is likely to be provided by a mechanism located in the remote entity. Note that we could have defined the bind operation as part of the connect operation, thereby making the connection to a particular information object at the connect stage rather than to a network component.

Transfer operation :
The operation *transfer(iuobj1,nc1,iuobj2,nc2)* allows a subject sub to append the contents of an information unit object $iuobj1$ in a network component object $nc1$ to the contents of another information unit object $iuobj2$ in a network component object $nc2$. For this operation to be secure, we require that :

- $objcls(iuobj2) \geq objcls(iuobj1)$
- $curcls(sub) \geq objcls(iuobj1)$

Further both $iuobj1$ and $iuobj2$ referenced by the subject sub must not be referenced by any other object. That is, for any $sb \in Sub_s$, $sb \neq sub$, $iuobj1$ and $iuobj2 \notin subrefobj(sb)$. Also $iuobj1$ and $iuobj2 \in subrefobj(sub)$.

After the operation is performed the security classes of the objects $iuobj1$ and $iuobj2$ remain unchanged. That is,

- $objcls'(iuobj1) = objcls(iuobj1)$
- $objcls'(iuobj2) = objcls(iuobj2)$

where $objcls'$ refers to the new state s'.

Unbind : The operation $unbind(sub, iuobj)$ allows a subject sub to release its link to an information object $iuobj$. That is, before this operation $iuobj \in subrefobj(sub)$. After the operation, we have $iuobj \notin subrefobj(sub)$.

VIII. Conclusion and future work

The flexibility that makes virtual networks such a useful technology can also undermine security within organizations and individual hosts. Current research on virtual machines has focused largely on the implementation of virtualization and its applications. But less effort was done for securing communication under virtualized systems. We proposed in this paper a security policy model for communication under virtual networks, this model can be implemented easily under most virtualized architectures.

Currently, we are extending our security policy to cover not only local networks, but also wide networks composed of many virtualized systems involving policy agreements and the protection of information flows that leave the control of the local hypervisor. We need to establish trust into the semantics and enforcement of the security policy governing the remote hypervisor system before allowing information flow to and from such a system.

References

[1] Xen, 2005–2011. http://www.xen.org/.

[2] VirtualBox, 2011. http://www.virtualbox.org/.

[3] VMware Workstation, 2011. http://www.vmware.com/.

[4] Qemu, 2011. http://www.qemu.org/.

[5] Adrian Baldwin, Chris Dalton, Simon Shiu, Krzysztof Kostienko, Qasim Rajpoot Providing Secure Services for a Virtual Infrastructure ACM SIGOPS Operating Systems Review archive Volume 43 Issue 1, January 2009 ACM New York, USA.

[6] Trent Jaeger, Reiner Sailer, Yogesh Sreenivasan. Managing the Risk of Covert Information Flows in Virtual Machine Systems. In Proceedings of ACM Symposium on Access Control Models and Technologies (SACMAT), 2007.

[7] Bernhard Jansen, HariGovind V. Ramasamy, Matthias Schunter. Policy Enforcement and Compliance Proofs for Xen Virtual Machines. In proceedings of the 2008 ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments.

[8] S. Smalley, C. Vance, and W. Salamon. Implementing SELinux as a Linux security module. Technical report, NSA, 2001.

[9] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. SIGOPS Operating Systems Review.

[10] Reiner Sailer, Trent Jaeger, Enriquillo Valdez, Ramon Caceres, Ronald Perez, Stefan Berger, John L. Griffin, and Leendert van Doorn. Building a macbased security architecture for the xen opensource hypervisor. In Proceedings of the 21st Annual Computer Security Applications Conference, pages 276-285, December 2005.

[11] Introducing Role-based Access Control to a Secure Virtual Machine Monitor: Security Policy Enforcement Mechanism for Distributed Computers. In : IEEE Asia-Pacific Services Computing Conference 2008.

[12] Till Mossakowski, Michael Drouineaud, Karsten Sohr. A temporal-logic extension of role-based access control covering dynamic separation of duties. In TIME-ICTL 2003. IEEE Computer Society.

[13] J. Olivain and J. Goubault-Larrecq. The Orchids intrusion detection tool. In K. Etessami and S. Rajamani, editors, *17th Intl. Conf. Computer Aided Verification (CAV'05)*, pages 286–290. Springer LNCS 3576, 2005.

[14] J. Goubault-Larrecq and J. Olivain. A smell of Orchids. In M. Leucker, editor, *Proceedings of the 8th Workshop on Runtime Verification (RV'08)*, Lecture Notes in Computer Science, pages 1–20, Budapest, Hungary, Mar. 2008. Springer.

[15] H. Benzina. Logic In Virtualized Systems In ICCANS'11, IEEE Computer Society.

[16] H. Benzina and J. Goubault-Larrecq. Some Ideas on Virtualized Systems Security, and Monitors. In DPM/SETOP'10, LNCS 6514, pages 244-258. Springer, 2010.

[17] Bell, D.E., Padula, L.J.L.: Secure computer system: unified exposition and MULTICS interpretation. Report ESD-TR-75-306, The MITRE Corporation (1976)

[18] Foley, S.: Aggregation and separation as noninterference properties. Journal of Computer Security 1(2)(1992) 159-188

[19] Sandhu, R.: Lattice based access control models. IEEE Computer 26(11) (1993) 9-19

[20] Lee, T.: Using mandatory integrity to enforce 'commerical' security. In: Proceedings of the Symposium on Security and Privacy. (1988) 140-146

[21] Foley, S.: The specification and implementation of commercial security requirements including dynamic segregation of duties. In: ACM Conference on Computer and Communications Security. (1997) 125-134

[22] Sandhu, R.: Role hierarchies and constraints for lattice-based access controls. In: ESORICS. (1996)