A Network Policy Model for Virtualized Systems

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

***Abstract*—Modern hypervisors offer the ability to build virtual networks between virtual machines. These networks are very useful in both personal and professional activities since they offer the same opportunities as physical networks, but in a much lower cost in terms of hardware and time. On the other hand, these networks are facing many security threats due to the absence of rigourous security policies that protect the sensitive ressources of the network. In this paper, we propose a multilevel security policy model for these networks, this policy covers not only network operations, but also operations related to the management of the virtual architecture.**

Keywords: Virtualized systems, virtual networks, security policies.

I. Introduction

A *virtualized* system such as Xen [1], VirtualBox, VMWare, or QEmu allows one to emulate one or several so-called *guest* operating systems (OS) in one or several *virtual machines* (VM). The different VMs execute as though they were physically distinct machines, and can communicate through ordinary network connections. 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 : building and configuring a virtual network is a very easy task. On the other hand, most of the security threats we face in a non-virtualized environment exist in virtualized environments as well. We propose in this paper a multi-level security policy model 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.

II. Related work

A body of existing work has already examined the issues arised by virtualized architectures [2][3]. However, not enough work was done for securing virtual networks between VMs.

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 [4], 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.

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

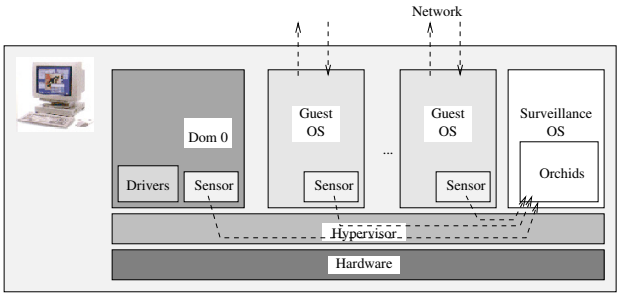

Fig. 1. Decentralized Supervision based on a virtual network

sHype [6] 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 [7], 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. Virtual Networks : advantages and security threats

We call *virtual network* the local network built between virtual machines in an hypervisor-based architecture.

We argue that these networks have several advantages : First, a virtual network reduces the networking hardware investment (fewer cables, hubs) and eliminates dependencies on hardware. Second, one can consolidate hardware by connecting guest systems that run in virtual machines in a single host. Also, consolidating servers in a virtual network allows one to reduce or eliminate the overhead associated with traditional networking components. Besides, by defining a virtual network on a single processor, one does not need to consider network traffic outside the processor. As a result : a high degree of network availability and performance.

In [8] 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 1, which is perhaps more typical of Xen than other hypervisors.

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.

IV. Multilevel Network Security

Multi-level security was formalized by Bell and La- Padula [9] 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.

V. The Security Policy Model

A. Model Representation

In order to be generic, our model takes 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, and sometimes for the processor, 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.

VI. 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 $nc \in subrefobj(sub)$.

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 $sb \in Sub_s$, $iuobj \notin subrefobj(sb)$

We will have $iuobj \in subrefobj'(sub)$. Where $subrefobj'$ refers to the new state s'.

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 $nc2$.

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

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'.

C. Security-related operations

Let us consider some of the security-related operations. We will use the notation x and x' to refer to x at states s and s'.

Assign-cls-nc : The operation *assign-cls-nc(nc,scls)* allows a subject sub to set the security class of a network component object nc, to $scls$. That is, $objcls'(nc) = \{scls\}$. This operation can be performed only when the component is not being used. If this operation is to be performed at state s then the following must be true :
If there exists any $nc \in NC$ such that $objcls(nc) \neq objcls'(nc)$ then :

- for any subject $sb \in Sub_s(sb \neq sub)$, $nc \notin subrefobj(sb)$ and $(sb, nc) \notin connlist$
- $Admin \in role(sub)$ and $currole(sub) = Admin$.

Assign-curcls-user : The operation *assign-curcls-user(usr, scls)* allows a subject sub to set the current security class of a user usr to $scls$. That is, $curcls'(usr) = scls$. If there exists any $usr \in Users$ such that $curcls(usr) \neq curcls'(usr)$ then

- $Admin \in role(sub)$ and $currole(sub) = Admin$ or $usr = sub$.
- $subcls(usr) \geq curcls'(usr)$
- if the user is logged onto a terminal at state s, then $curcls'(usr) \geq objcls(curvm(usr))$.
- if the user is connected to a network component at state s which is not an output device, that is, $(usr, nc) \in connlist$ and $nc \notin OD$, then $curcls'(usr) \geq objcls(nc)$
- if the user is logged in and is connected to an output device, that is, $(usr, nc) \in connlist$ and $nc \in OD$, then $objcls(nc) \geq curcls'(usr)$.

Assign-role-user : The operation *assign-role-user(usr,rlset)* allows a subject sub to assign a role set $rlset$ to a user usr, That is $role'(usr) = \{rlset\}$. For this operation to be secure, we need the following condition to be hold :
If there exists any $usr \in Users$ such that $role(usr) \neq role'(usr)$ then :

- $Admin \in role(sub)$ and $currole(sub) = Admin$
- if the user is logged in at state s, then $currole(usr) \in role'(usr)$.

Setauthlist : The operation $setauthlist(al)$ allows a subject to set the authorization list. The $authlist$ is of the form (sb, nc), where $sb \in Sub$ and $nc \in NC$. Again, this operation can only be performed by a subject who can act as a $Admin$. That is, if $al \notin authlist$ and $al \in authlist'$ then $Admin \in role(sub)$ and $currole(sub) = Admin$ where sub is the subject performing this operation.

Theorem : The model $M =< S, O, s_0 >$ is secure.

VII. 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.

References

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

[2] Adrian Baldwin, Chris Dalton, Simon Shiu, Krzysztof Kostienko, Qasim Rajpoot Providing Secure Services for a Virtual Infrastructure ACM SIGOPS 2009

[3] Trent Jaeger, Reiner Sailer, Yogesh Sreenivasan. Managing the Risk of Covert Information Flows in Virtual Machine Systems. SACMAT 2007

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

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

[6] 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.

[7] 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.

[8] 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.

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