PIGA-Virt: An Advanced Distributed MAC Protection of Virtual Systems

J. Briffaut, E. Lefebvre, J. Rouzaud-Cornabas, and C. Toinard

ENSI de Bourges – LIFO, 88 bd Lahitolle, 18020 Bourges cedex, France
{jeremy.briffaut,jonathan.rouzaud-cornabas,
christian.toinard}@ensi-bourges.fr

Abstract. Efficient Mandatory Access Control of Virtual Machines remains an open problem for protecting efficiently Cloud Systems. For example, the MAC protection must allow some information flows between two virtual machines while preventing other information flows between those two machines. For solving these problems, the virtual environment must guarantee an in-depth protection in order to control the information flows that starts in a Virtual Machine (VM) and finishes in another one. In contrast with existing MAC approaches, PIGA-Virt is a MAC protection controlling the different levels of a virtual system. It eases the management of the required security objectives. The PIGA-Virt approach guarantees the required security objectives while controlling efficiently the information flows. PIGA-Virt supports a large range of predefined protection canvas whose efficiency has been demonstrated during the ANR Sec&Si [1] security challenge. The paper shows how the PIGA-Virt approach guarantees advanced confidentiality and integrity properties by controlling complex combinations of transitive information flows passing through intermediate resources. As far as we know, PIGA-Virt is the first operational solution providing in-depth MAC protection, addressing advanced security requirements and controlling efficiently information flows inside and between virtual machines. Moreover, the solution is independent of the underlying hypervisor. Performances and protection scenarios are given for protecting KVM virtual machines.

1 Introduction

A virtualization layer, i.e. an hypervisor, brings isolation between multiple systems, i.e. Virtual Machines, hosted on the same hardware. The hypervisor reduces the interferences between the VMs. But the virtualization is not a security guarantee. It increases the attack surface and adds new attack vectors. As a consequence, the virtualization must not be the sole technology for providing isolation within a Cloud. For example, in [14], the isolation is broken through drivers that allow the access of the underlying hardware from inside a VM. Indeed, these drivers can access the physical memory without passing through the

[1] http://www.agence-nationale-recherche.fr/magazine/actualites/detail/resultats-du-defi-sec-si-systeme-d-exploitation-cloisonne-securise-pour-l-internaute/

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part II, LNCS 7156, pp. 416–425, 2012.

kernel or the hypervisor thus by passing the protection layers. Preventing such attacks requires to guarantee the integrity of 1) the VM, 2) the hypervisor and 3) the underlying Operating System. Moreover, a VM can produce information flows that come to another VM in order to break some of the requested security objectives. Accordingly, a VM can attack the integrity, confidentiality and availability of other VMs that run on the same hardware.

With cloud paradigm, the data and the entire computing infrastructure is outside the scope of the final users. Thus, security is one of the top concerns in clouds [9]. Indeed, with a cloud infrastructure relying on virtualization, the hardware is shared between multiple users (multi-tenancy) and these users can be adversaries. Moreover, as explained in [13,4], various securities and functionalities are needed to enforce the criticality of missions within a cloud. Thus, the major goal of this work is to increase the security assurance by 1) hardening the isolation of the virtualization layer and 2) providing a mission-aware security component for the virtualization layer and 3) balancing the security with the performance.

The first section defines the precise objectives of our solution. Second, the paper describes the different protection modes supported by PIGA-Virt. Third, it describes how PIGA-Virt enforces the protection. Forth, it gives the efficiency for the different modes of the versatile PIGA-Virt solution. Fifth, it describes the related works. Finally, the paper concludes by defining the future works.

2 Motivation

In-depth end-to-end Mandatory Protection Inside a VM
The mandatory control minimizes the privileges that a process (a subject) has regarding the various objects. But existing MAC approaches mainly deal with direct flows. A first set of objectives consider the control of the flows inside a given virtual machine.

The first purpose is thus to control indirect information flows transiting through intermediate resources or processes (covert channels). The second purpose is to ease the definition of a large set of security objectives, such as separation of privileges or indirect accesses to the information through covert channels. The third purpose is to provide a mandatory protection controlling all the levels (in-depth protection) of a virtual machine, such as processes, graphic interface, network, etc. Our fourth purpose is to provide an efficient mandatory protection that guarantees all the supported security properties with satisfying performance in the context of the virtual machines sharing the same host. In contrast with our previous works [5], the fourfth objective is addressed in that paper since performances need to be improved for hosting multiple VMs on the same machine.

In-depth end-to-end Protection between VMs
With multiple multiple VMs sharing the same host, the flows between the VMs must also be efficiently controlled. For example, the protection must prevent a malicious information flow, coming from (VM1), from going to (VM2) through a NFS share. But, some NFS flows between VM1 and VM2 must be allowed

while others must be denied. A second set of objectives address specifically that control of the flows between the VMs. The corresponding objectives have not been addressed during our previous works.

A fifth purpose is that the in-depth end-to-end protections must control the flows between the different VMs. Such a protection consists in controlling 1) all the indirect flows that are visible to the VMs and 2) all the indirect flows using, intermediate entities of the target host, that are invisible to the VMs. A sixth purpose is to have a protection independent from the target hypervisor. It is an important issue since several kinds of hypervisor technologies can cohabit in the context of a cloud. A seventh purpose is that the proposed protection must be easy to configure. A eigth purpose is to ease the tuning of the protection efficiency. Thus, the administrator can tune the protection to balance the performance with the security.

3 Architecture of PIGA-Virt

PIGA-Virt provides mechanisms to reach our 8 objectives.

PIGA-VIRT is a protection system that controls interactions inside and between VMs. It has two layers:

- Local layer : each VM runs a local PIGA-Decision [5] engine associated with a SELinux/PIGA-Kernel. The combination of PIGA-Kernel and PIGA-Decision provides an efficient reference monitor guaranteeing a large set of security objectives. Thanks to our Security Protection Language (SPL), it eases the definition of security objectives and controls the information flows inside each VM.

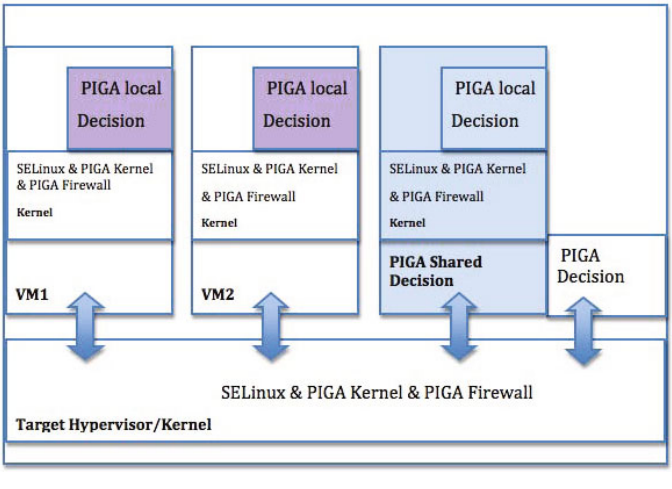

Fig. 1. Architecture of PIGA-Virt

– Shared layer : a dedicated secure VM runs a shared PIGA Decision engine that 1) improves the performance for the intra flows and 2) controls the information flows between the VMs. Thanks to the proposed extension of SPL, the shared layer controls the information flows starting in a VM and finishing in another VM. The extension consists 1) in adding a virtual machine identifier to the security contexts and 2) in processing a flow graph for each virtual machine.

The administrator can combine the local and shared layers according to the required security objectives. So, the administrator can choose between three modes: local, shared/local and shared modes. The shared mode simplifies the task of the security administrator since it controls both the inter and the intra flows through a central management. The sequel describes the unified MAC approach provided by the local and the shared mode and compares the performances between the local and the shared modes.

4 In-Depth End-to-End Mandatory Protection

PIGA-Virt provides a unified MAC approach for the local and the shared mode:

A] SELinux and XSELinux control the direct information flows of the applications and XWindow; Those controls are always performed in a local manner since it consists in reusing the SELinux approach.

B] PIGA includes two major components:

- PIGA-PROTECT (PIGA-KERNEL and PIGA-DECISION) that controls the transitive information flows. PIGA-PROTECT prevents against millions of vulnerabilities inside the SELinux policies. In practice, illegal activities allowed by the SELinux policies are precomputed and PIGA-DECISION compares in real-time the precomputed set of illegal activities with the real activities occuring on the system. When a real activity matches with a precomputed one, PIGA-DECISION denies the corresponding system call.

- PIGA-FIREWALL targets mandatory network access hardening. It guarantees an end-to-end control of network flows. For example, a firefox process associated with the security context $firefox\_taxes\_t$ has the guaranty to transmit taxes' data only to the network site providing the e-taxes.

The `piga` policies are the same in the local and shared mode. In contrast with the local mode, the shared mode processes the controls within the dedicated virtual machine and is able to control the inter flows. Since `piga-protect` is the heart of our advanced MAC protection, let us give few examples of canvas written using our SPL.

Canvas of Mandatory Protection Supporting Security Objectives
The following property aims at guaranteeing the confidentiality of a set $sc2$ of objects regarding a set $sc1$ of subjects. That property prevents against reading flows that can be direct $>$ or indirect $>>$.

```
1  define confidentiality( sc1 in SCS, sc2 in SCO ) [ ¬(sc2 > sc1) AND ¬(sc2 >> sc1) ];
```

The following property prevents a process from creating a file, executing an interpreter (e.g. bash) that then attempts to execute the created file.

```
1  define dutiesseparationbash( sc1 IN SC ) [ Foreach sc2 IN SCO, Foreach sc3 IN SC,
2  ¬ ( ( sc1 >write sc2) −−then→ (sc1 >execute sc3 ) −−then→ (sc3 >read sc2)) ];
```

Usage of the Proposed Canvas for Intra and Inter VM Protections
The administrator defines a small number of protection rules with the relevant parameters (SELinux contexts) for the canvas. The following rule protects a VM against the attacks relying on a shell interpretation of downloaded scripts. It prevents all the user processes, associated with the regular expression $user\_u : user\_r : user.* \_t$, from downloading a script in order to read and execute that script. That rule can be set-up within each virtual machine (local mode) or within the dedicated virtual machine (shared mode).

```
1  dutiesseparationbash( "user_u:user_r:user.*_t" );
```

In contrast with our previous works [5], the major improvement is the way the dedicated virtual machine computes the controls in the shared mode. The PIGA shared decision engine computes independant data for each virtual machine. The PIGA shared decision engine communicates with the different PIGA-KERNELs available into the different virtual machines. When PIGA shared decision finds a real activity of a virtual machine matching with the precomputed set of illegal activities associated with that virtual machine, it sends a deny to the corresponding PIGA-KERNEL in order to cancel the corresponding system call.

The following rule guarantees the confidentiality of the $/etc$ files in VM1 regarding the users of VM2. In contrast with previous works, a virtual machine identifier is added to the SELinux context. Thus, the administrator easily express the control of the flows between two different virtual machines. The corresponding control is only available in the shared mode. The PIGA shared decision engine breaks an illegal activity into different subactivities for the corresponding virtual machines. When all the subactivities are detected, the PIGA shared decision engines sends a deny for the lattest system call.

```
1  confidentiality(user_u:user_r:user.*_t:vm2, system_u:object_r:etc_t:vm1);
```

5 Experimentations

PIGA-Virt is integrated into a Scientific Linux 6 host with KVM as hypervisor. The PIGA-Kernel consists in a small patch that captures the SELinux hooks. PIGA-Decision is available as a Java process. Experimentations run on an AMD Phenom(tm) II X4 965 Processor with 8 Giga bytes of RAM.

Performances

Figure 2 presents several types of PIGA-Virt instances for protection two Linux virtual machines $VM1$ and $VM2$. PIGA-Virt runs a) without SELinux, b) with the targeted SELinux policy, c) with the strict SELinux policy, d) in local mode to detect the violations of the requested properties and e) in shared mode to detect the violations, f) in local mode to prevent the intra violations and g) in shared mode to prevent the inter violations. The local mode controls the flows inside a VM whereas the shared mode controls the flows between VMs. In contrast with the PIGA-Virt detection, the PIGA-Virt protection enables to evaluate the overhead introduced to prevent the violations of the required security properties.

Several benchmarks (open/close of files, executing the `ls -lR` command to parse the whole file system, fork and file access latency) show the performances of PIGA-Virt. As shown in the figure 2, the overhead due to the in-depth end-to-end protection of PIGA-Virt is a very low. The performance of the environment without any MAC protection corresponds to the a) column i.e. SELinux OFF. The performance of the controls inside the VM is given by the f) column i.e. local protection. The performance of the controls between the VMs, is given by the g) column i.e. shared protection. Sometimes the MAC protections improve the performances e.g. the `ls` command takes more time without any MAC protection since the MAC protections minimize the accesses to the file system. In contrast with SELinux, the PIGA-Virt protection either reduces or equals the overhead. The only exception is the fork result but this benchmark is very stressful since it corresponds to unusual millions of simultaneous fork operations.

Globally, the PIGA-Virt protection brings a very low overhead. In contrast with no MAC protection, PIGA-Virt improves the performances. In contrast with the local mode associated with our previous works, the shared mode factories the PIGA-Decision within a single instance. So, our new shared approach minimizes the overhead due to the security mechanisms. Moreover, the shared mode uses TCP connections between the VMs and PIGA-Decision. So, PIGA-Decision can be run on a dedicated machine with high performance capabilities, improving thus more the CPU consumption.

Protection Efficiency

In contrast with the local mode i.e. our previous work, the shared mode is of major importance in term of security assurance. Indeed, it is the only way to control the flows between the VMs sharing the same host.

Let us give a small example of the protection carried out by the $confidentiality$ property. For example, the following global illegal activity, with a subactivity on $vm1$ ($user\_t$ reading $/etc$ before writing into nfs_t) and a subactivity on $vm2$ ($user\_t$ reading nfs_t), is a violation of $confidentiality(\$sc1 := "user\_u : user\_r : user.* \_t : vm2", \$sc2 := system\_u : object\_r : etc\_t : vm1)$. In such a case, the shared decision engine cancels the reading of nfs_t on $vm2$ since it is the lattest system call of the global activity. Such an activity corresponds, for example, to a malware, executed from the user environment of $vm1$, and transmitting the $/etc/shadow$ password to a distant virtual machine $vm2$. Thus, the shared mode eases the protection against generic malicious activities such as

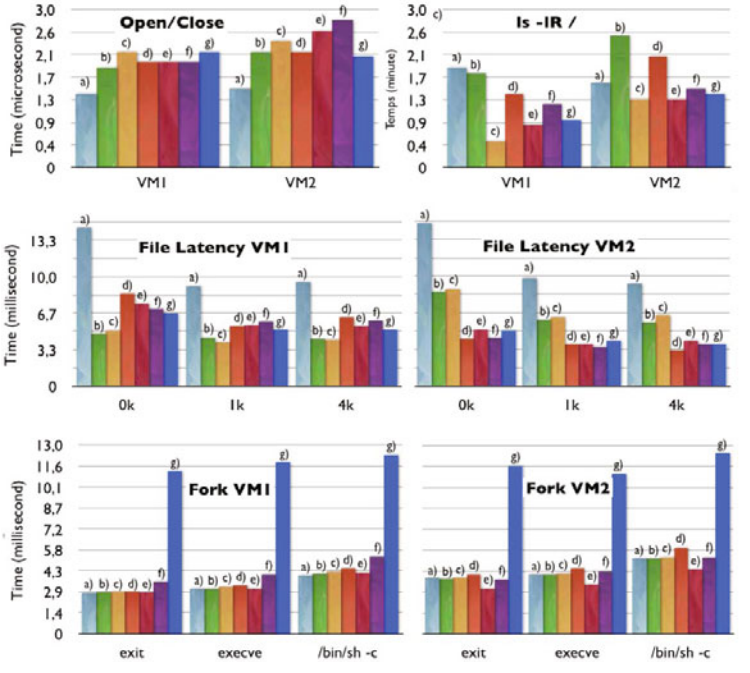

Fig. 2. Performances of PIGA-Virt

NFS threats. It enables to prevent illegal flows while authorizing safe flows since it allows, for example, *sysadm_t* to transmit data through NFS to a distant user.

PIGA-Virt is very efficient since defining a safe SELinux policy is a tricky task. So, a couple of SPL rules is simpler than writing a SELinux policy 1) including millions of rules and 2) that does not control the transitive flows.

Mission Efficiency

PIGA-Virt is a mission-aware environment. First, it takes into account the security objectives i.e. the requested security properties. Second, it provides the efficiency of each security objective.

Table 1 provides the efficiency of the different properties used during our experimentation. For example, the efficiency of the *confidentiality* property is 108.045. That value shows 108.045 illegal activities enabling to violate the confidentiality property within the considered SELinux policy. Such a value has two meanings: 1) it gives the security enforcement of a property i.e. the higher the value is, the stronger the property enforces the security and 2) it evaluates the cost of the property i.e. the higher the value is, the higher the processing time is needed by PIGA-Virt.

Mission Tuning

As demonstrated, more a security property is strong the higher the overhead is. It is a well known relationship between security and performance. However, the important point here is that the administrator has a precise evaluation

of each security property. Thus, he can tune the security objectives to fit the performance needings. For example, the *dutiesseparationbash* property is a large overestimation of the separation of duties, that protects against malicious scripts, since it prevents millions of potential vulnerabilities. In contrast with the *dutieseparationbash*, the *dutiesseparation* property is less large since it protects only against binary executions preventing thus only 208.240 illegal activities. However, *dutiesseparationbash* and *dutiesseparation* do not tackle the same security objective. In order to tune a property such as *dutiesseparationbash*, several facilities are available.

PIGA-Virt eases the tuning of the security missions in different ways. Thus, the administrator can adjust a security objective by 1) providing different security contexts for the security canvas, 2) modifying the definition of the canvas and 3) modifying the SELinux policy. That latter solution is usually the trickiest. However, PIGA-Virt facilitates this task. Let us consider the *confidentiality* property preventing illegal activities including:

```
1  user_u:user_r:user_t−(dbus{send_msg})−>user_u:user_r:user_dbusd_t; user_u:user_r:user_dbusd_t−(
       file{write})−>user_u:object_r:user_home_t; user_u:user_r:gpg_agent_t−(file{read})−>user_u:
       object_r:user_home_t; user_u:user_r:gpg_agent_t−(file{write} )−>system_u:object_r:nfs_t
```

Thus, the administrator sees that the *dbus* and *gpg* are involved into that threat. PIGA-Virt shows that the problem can be corrected with a separation of duties for *dbus* or *gpg*. Thus, the tuning consists in a new SELinux policy including, for example, separation of duties for *dbus* (e.g. removing the permission of writing into *user_home_t* for *dbus_t*).

Table 1. Efficiency of the requested security mission

	Property	Efficiency
Security	transitionsequence	101 533
mission	notrereadconfigfile	2
	ourreadconfigfile	4
	dutiesseparation	208 240
	dutiesseparationbash	194 629 680
	confidentiality	108045
	integrity	30
	trustedpathexecution	8 715
	trustedpathexecutionuser	204
	trustedpathexecutionuser	26
	consistentaccess	50 470

6 Related Works

A frequent approach is to use integrity verification technologies. [1] uses a dedicated hypervisor to encrypt the data and the network transmission. GuardHype [2]

and [10] verifies the integrity of the hypervisor itself or the integrity of the kernel and critical applications. But these approaches are limited to statically verify the integrity of an image, a binary or a part of the memory. However, those solutions do not control the access to the ressources. The followed approach is to put Madatory Access Control outside of the VMs. Thus, the multiple virtual machines can be controlled consistently and safely using a single security monitor. MAC [6] is the only way to guarantee security objectives. In [3], that approach is limited to the control inside an untrusted virtual machine and cannot guaranty the isolation between the virtual machines. For example, sHype [12] brings Type Enforcement to control the inter-VM communications. But, sHype only controls overt channels thus missing implicit covert channels. Moreover, they do not propose a way to express security properties. The MAC enforcement of the hypervisor can be extended to the MAC enforcement inside the virtual machine. Thus, [8] divides the overall policy into specialized policies (one per VM and one for the interaction between VM)s. For example, Shamon [7] is a prototype based on Xen/XSM (Inter-VM MAC) and SELinux (OS Level MAC) to control applications running on different VMs. As explained in [11], the common way to analyze MAC policies is to search for illegal information flows inside them. In order to reduce the complexity, [11] analyses each layer (hypervisor then OS). The analysis is too complex and the illegal flows cannot be blocked in real time. So existing solutions cannot control in real-time advanced security properties associated with multiple information flows between the different virtual machines.

7 Conclusion

That paper presents the first mission-aware security approach for VMs that supports a large range of security objectives and provides a precise evaluation of the security efficiency. In contrast with existing approaches, it provides a real time protection of advanced security objectives with a very low overhead. Moreover, PIGA-Virt eases the work of the administrator since around ten security rules are generally sufficient to control efficiently the flows between the different VMs sharing the same host. Finally, PIGA-Virt is an extensible approach. Indeed, it requires only security contexts associated with the different system resources. For example, a Windows 7 module is available providing consistent security labels that can be processed through PIGA-Virt. It is an excellent way to improve the security of heterogeneous VMs such as required in Cloud infrastructures. Future works deal with distributed scheduling of VMs as a security mission-aware service providing Security as a Service ([Sec]aaS) in the context of anything as Service approaches (XaaS Clouds).

References

1. BitVisor 1.1 Reference Manual (2010), `http://www.bitvisor.org/`
2. Carbone, M., Zamboni, D., Lee, W.: Taming virtualization. IEEE Security and Privacy 6(1), 65–67 (2008)

3. Chen, X., Garfinkel, T., Christopher Lewis, E., Subrahmanyam, P., Waldspurger, C.A., Boneh, D., Dwoskin, J., Ports, D.R.K.: Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. SIGOPS Oper. Syst. Rev. 42, 2–13 (2008)
4. Jaeger, T., Schiffman, J.: Outlook: Cloudy with a chance of security challenges and improvements. IEEE Security and Privacy 8, 77–80 (2010)
5. Briffaut, C.T.J., Peres, M.: A dynamic end-to-end security for coordinating multiple protections within a linux desktop. In: Proceedings of the 2010 IEEE Workshop on Collaboration and Security (COLSEC 2010), pp. 509–515. IEEE Computer Society, Chicago (2010)
6. Loscocco, P.A., Smalley, S.D., Muckelbauer, P.A., Taylor, R.C., Turner, S.J., Farrell, J.F.: The Inevitability of Failure: The Flawed Assumption of Security in Modern Computing Environments. In: Proceedings of the 21st National Information Systems Security Conference, Arlington, Virginia, USA, pp. 303–314 (October 1998)
7. McCune, J.M., Jaeger, T., Berger, S., Caceres, R., Sailer, R.: Shamon: A system for distributed mandatory access control. In: Proceedings of the 22nd Annual Computer Security Applications Conference, pp. 23–32. IEEE Computer Society, Washington, DC (2006)
8. Payne, B.D., Sailer, R., Cáceres, R., Perez, R., Lee, W.: A layered approach to simplified access control in virtualized systems. SIGOPS Oper. Syst. Rev. 41, 12–19 (2007)
9. Pearson, S., Benameur, A.: Privacy, security and trust issues arising from cloud computing. In: Proceedings of the 2010 IEEE Second International Conference on Cloud Computing Technology and Science, CLOUDCOM 2010, pp. 693–702. IEEE Computer Society, Washington, DC (2010)
10. Quynh, N.A., Takefuji, Y.: A real-time integrity monitor for xen virtual machine. In: ICNS 2006: Proceedings of the International Conference on Networking and Services, p. 90. IEEE Computer Society, Washington, DC (2006)
11. Rueda, S., Vijayakumar, H., Jaeger, T.: Analysis of virtual machine system policies. In: Proceedings of the 14th ACM Symposium on Access Control Models and Technologies, SACMAT 2009, pp. 227–236. ACM, New York (2009)
12. Sailer, R., Jaeger, T., Valdez, E., Caceres, R., Perez, R., Berger, S., Griffin, J.L., Van Doorn, L., Center, I.B.M.T.J.W.R., Hawthorne, N.Y.: Building a MAC-based security architecture for the Xen open-source hypervisor. In: 21st Annual Computer Security Applications Conference, p. 10 (2005)
13. Sandhu, R., Boppana, R., Krishnan, R., Reich, J., Wolff, T., Zachry, J.: Towards a discipline of mission-aware cloud computing. In: Proceedings of the 2010 ACM Workshop on Cloud Computing Security Workshop, CCSW 2010, pp. 13–18. ACM, New York (2010)
14. Wojtczuk, R.: Subverting the Xen hypervisor. BlackHat USA (2008)