C&ESAR 2012

Computer & **E**lectronics
Security **A**pplications
Rendez-vous

20 - 21 - 22 novembre 2012
Rennes - France

http://www.cesar-conference.fr/

C&ESAR 2012 : *Cloud Computing* menace ou opportunité ?

La 19ème édition de C&ESAR (Computer & Electronics Security Applications), journées SSI de la Défense, est consacrée cette année au thème de l'informatique en nuage, le *Cloud Computing*. L'adoption de l'informatique en nuage a un fort impact sur les infrastructures des technologies de l'information et sur les modes de gestion des risques associés. Si la consommation de services à la demande, le partage de ressources, et la réduction des coûts (apparente du moins) peuvent séduire l'utilisateur, de nombreuses questions apparaissent quant aux conséquences de cette évolution en termes de sécurité. Questions dont les réponses ne sont pas forcément claires ou satisfaisantes pour l'instant. Pour cette édition 2012 de C&ESAR nous avons donc choisi de nous attarder sur les particularités de l'informatique en nuage en matière de sécurité ; depuis le partage d'applications, de services ou d'infrastructures à travers différents domaines d'administration, en passant par les aspects techniques de l'isolation et de l'accès sécurisé, jusqu'à l'impact de ces nouveaux écosystèmes commerciaux et de leurs modèles économiques sur les approches traditionnelles de gouvernance. Quelles seront les conséquences des transformations induites par l'informatique en nuage sur la gestion et la gouvernance de la sécurité dans les systèmes d'information ? Nous espérons que les exposés de ces trois journées vous apporteront quelques éléments de réponse, des avancées techniques et scientifiques aux questions pratiques et opérationnelles, des interrogations en termes de règlementation aux nouvelles problématiques de certification. Nous avons tenté d'évoquer tous ce qui fait de la sécurité une question centrale, et de dimension interdisciplinaire dans la communauté SSI, pour le succès de l'adoption de l'informatique en nuage. Nous tenons à exprimer notre reconnaissance à tous ceux qui ont contribué à cette édition 2012, auteurs de communications, conférenciers invités, membres du comité de programme, ou organisateurs, qui comme chaque année apportent leur pierre à l'édifice pour faire progresser la communauté SSI. Et nous remercions bien sûr aussi nos divers partenaires qui assurent sans faiblir le soutien matériel à cet évènement.

Yves Correc (DGA-MI), Président du comité d'organisation.
Boris Balacheff (HP Labs), Président du comité de programme.
Olivier Heen (Technicolor), Directeur de publicaiton.

Partenaires

ANSSI, DGA, DGSIC, DIRISI, HP, Orange Business Services, Supélec, Technicolor.

Comité d'organisation

José Araujo	ANSSI, SGDSN, French Govt
Florent Chabaud	DGSIC, MoD, French Govt
Yves Correc (Chair)	DGA-MI, MoD, French Govt
Olivier Heen	Technicolor
Ludovic Mé	Supélec
Eric Wiatrowski	Orange Business Services

Comité de programme

Gabriel Antoniu	INRIA Rennes & IRISA
Jean-Francois Audenard	Orange Business Services
Boris Balacheff (Chair)	HP Labs
Emmanuel Bouillon	NC3A, NATO
Yves Correc	DGA-MI, MoD, French Govt
Hervé Debar	Telecom Sud-Paris
Fabrice Derepas	CEA
Paul England	Microsoft
David Grawrock	Intel
Olivier Heen	Technicolor
Isabelle Hirayama	ANSSI, SGDSN, French Govt
Trent Jaeger	PennState University
Derrick Kondo	INRIA / LIG
Volkmar Lotz	SAP Labs
Andrew Martin	Oxford University
Thierry Priol	INRIA Rennes
Hervé Putigny	ANSSI, SGDSN, French Govt
Dominique Rodrigues	NanoCloud & CNRS GRD ASR
Ahmad Sadeghi	T.U. Darmstadt
Simon Shiu	HP Labs
Paul Waller	CESG, UK Govt

Site officiel : `http://www.cesar-conference.fr/`

Table des matières

20 novembre 2012

Towards understanding multiparty trust and security issues in multitenant cloud ecosystems

Adrian Baldwin, Boris Balacheff, Dirk Kuhlmann, Brian Monahan, Simon Shiu

HP Labs
Bristol BS34 8QZ
England, UK

Email: {firstname.lastname}@hp.com

Abstract. *As companies look to moving their IT functions into the cloud the traditional IT security management lifecycle breaks down. The decision is not a simple one to decide to trust a given cloud service; we need to ask about complex service aggregation and supply chains; underlying platform and infrastructure properties; how users interact with those services and the impact of security properties of their client devices. Companies will need new frameworks to think about how they manage the emerging trust and security issues that come with the adoption of cloud based computing. Here we propose an approach that starts by developing a model-based understanding of tasks and information flows in cloud-based infrastructure. This provides us with a structural context to explore the desired platform properties, management processes and assurance characteristics necessary to convey and achieve trust in the use of cloud services. This approach based on the definition of a Trust Domain concept then becomes practical when we can rely on trusted infrastructure that offers the required security properties.*

1 Introduction

Over the next few years we believe that a cloud eco-system will emerge consisting of a variety of cloud service providers (SaaS) [1] who will operate business level services on top of a smaller number of platforms (a merger of IaaS and PaaS). As the range and complexity of services grows, many companies will consider moving their critical business services to the cloud, as well as using cloud to support various partnerships and collaborative projects.

Many companies see cloud as an attractive proposition because it offers flexible services use models and huge cost savings. However, issues such as a

perceived loss of control, whether the appropriate level of security is reached and questions of the resilience and trustworthiness of cloud service providers are preventing any real switch from internal or outsourced IT provision into the use of cloud services. For established companies, these factors represent uncertainty that keeps delaying the likely time to switch IT provision to the cloud [2].

Within this paper we argue that a company that is considering moving a business function to the cloud should start by trying to understand the information flows that underpin this very business function. This will lead them to consider the trustworthiness of those same information flows in a cloud context: from the cloud service providers that process their data, to client systems and access patterns of those involved with the business process. This approach becomes particularly important as business functions support collaborations and data sharing between multiple end user organisations that individually rely on different service providers for their part of the business function.

This approach is designed to allow us to consider the information assurance or information stewardship [3] requirements for that task (business process or project). This in turn allows us to consider what trust and security requirements are required from both the cloud services and the individuals using the services. This brings us to a number of questions:

1. What are the right abstractions to think consistently about properties of cloud services, infrastructure, users and access devices?
2. Are there certain architectural components that simplify such abstractions?
3. How do we help people understand risk characteristics associated with different architectures?

Within this paper we present a conceptual framework for thinking about these different trust and security issues and we discuss how it differs from the conventional view of enterprise IT. Within section 2 we talk about the cloud eco-system and some of the emerging trust and security problems that are different in nature to those facing enterprises currently. This is followed by looking at how a Trust Domain abstraction can be used as a conceptual

framework for considering these issues. The last section talks about the use of modelling to help evaluate different policy options and help chose an appropriate solution for a given task.

2 Cloud Eco-System

Within our work on cloud we assume a simple cloud eco-system model [pym11] as shown in figure 1. Here we have a layer of potential cloud consumers; that is companies who would consider using cloud to support business processes or collaborative projects. In the middle we have a number of service providers who run Software as a Service (SaaS), offering a wide variety of services to support businesses. Note that here we start by ignoring cloud services aimed at individual consumers. Finally we have a layer of platform providers who provide the basic infrastructure (such as Infrastructure as-a-Service, or private infrastructure implementations) on which all the services may be run. Lastly on the edge of the eco-system we have criminals who are looking to attack individual players to make money, and regulators who are seeking to reduce risks.

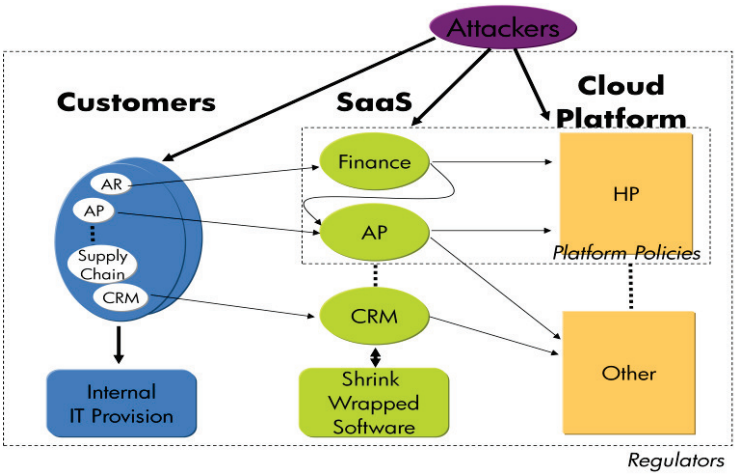


Fig. 1. A three layered description of the cloud eco-system

In previous work we have described ways to model and visualise such an eco-system [3, 4] here we draw out a number of points relating to the security and trustworthiness of how companies use cloud.

Firstly, we envision that as cloud platforms develop there will be lower barriers for software companies and many new start-ups to develop services that in turn will be available to businesses at a very low upfront cost. This will create a very heterogeneous set of software probably built from a large number of different libraries, alongside bespoke service level software code. The cost pressures associated with cloud and the small size of many start ups means that many services will not be operated according to best practice ITIL [5] and COBIT [6] standards expected within the enterprise. In effect this means that it will be hard for those relying on services to gain good assurance over their quality, particularly for hard to observe events such as security incidents. In practice this means that any reliance we can put on platform properties derived from the infrastructure providers are likely to bring huge security and trust management benefits.

Secondly, we should be concerned about a service supply chain where multiple independent organisations become responsible for our security posture. The simplest example is where we use one service provider who runs their service on top of a cloud platform, for example that provided by Amazon, Microsoft or HP. More complex supply chains will occur as a number of different services are combined to perform a customer's task. It may be that they use a number of independent services themselves. For example in a software development project a company may choose a service offering for source control, a separate service offering for bug tracking and other services offering things like code testing and security analysis of source code. Alternatively one service provider may integrate these independent offerings into a single service, which may even be run on a variety of different platforms.

In thinking through their security requirements, software developer would then need to consider how they can trust all of these services. This suggests that there needs to be some exchange of technical information around system configurations, security properties as well as how systems are managed to maintain good configurations. The exchange of such evidence will become a critical issue in establishing trustworthiness for operating in the cloud. Approaches here include looking at how to demonstrate the provenance of a system from hardware to software [7, 8] as well as looking at assurance over management properties [9].

In thinking about the security of a company using a service within the cloud eco-system we also need to consider the security of access devices used to interact with that service. For many attackers client systems have become easy entry points into the enterprise [10]. To put trust into a cloud service's operation we therefore need to think about security and trust related to the management of client systems used to access the services. For a company's internal IT this is an issue because of the increased push for adoption of Bring-Your-Own-Device models where end-user personal devices are used for interacting with business services. For a company that relies on publicly hosted multi-tenant cloud services, the issue also extends to the security management of client system used by end-users of other companies that use the same publicly hosted service, and that may put multi-tenant isolation at risk in the operation of the service. As cloud develops, this will get worse with an increase in collaborative working outside of a company's usual physical circle of trust. For example, e-lancing and business process services have already started developing where service providers are taking on elements of the business task and not just providing IT services.

Finally this movement to more collaborative business situations within the cloud presents the issue of trust in the people you are working with, and trust that they will handle your information according to your information assurance needs. With our software development example we may have a company trying to create a software product but using another organisation to take care of requirements analysis, using a number of individuals hired to write specific pieces of code. Trust in the use of a cloud-based solution now clearly extends to trust in the processes used to ensure that the right individuals have access to the right information only. In our example, each entity could be working for competitors or simply have poor information security practices. The information assurance problem for the project now involves getting trust in each of these entities corresponding people, business processes, information systems and security practices, as well as those of the service supply chain supporting the project.

Today most companies have a security management life-cycle that takes into account all aspects that may impact the company's business, from setting policies, to defining architectures, operational requirements, and assurance

standards, which all should feed back into an ongoing risk assessment process. As we move to cloud and collaborative working we rely on each of the other parties to run their own security management life-cycle in a way that meets our information assurance or stewardship needs [11]. This switch does allow us to think about security from a task perspective rather than as an IT stack, however, we need new frameworks to help us reason and understand the ways risk, system architecture, policy, operations and assurance join together in a multi-party world.

3 Trust domains

Here we propose the notion of a trust domain to provide a way for an organisation to think through the security and trust issues that occur as the enterprise disaggregates as described above. Here we see a trust domain as a group of entities sharing the same expectation of the security that they exchange with each other. Figure 2 represents a trust domain in more detail. Here we can think of a trust domain in three ways. Firstly, the definition of the trust domain and hence the trust properties and policies that it defines and for which it represents an enforcement boundary; secondly, the task being performed and hence the people and information involved and the services and access devices that they use to perform the task; and thirdly, we can think of how we achieve the trust domain implementation through a technology stack.

Within the first area for consideration we have an owner who creates the trust domain for a given task and sets up policies to achieve the required information assurance characteristics. These policies will include statements around how information should be handled by people, how it should be secured at rest and in transit. They would also include connection and authentication constraints specifying how individuals and services can connect into the domain. For example, the type of authorisation given: the individual's access device, the properties it exhibits and its location. In defining the trust domain it becomes very important to have policies and processes that define who is a member of the domain (both individuals as employees of a participating entity and cloud services).

The purpose of the trust domain is to allow a task to be performed whilst maintaining good security. Hence the second concern we have in considering the abstract properties of the trust domain is to think of the way the data flows as a task is performed. That is, how is information created and moved, who moves the information, who stores it and how does it get modified. Here we do not want to consider a detailed workflow (formal or informal) but rather have an idea of how information is spread over the domain and how that varies over time. In understanding the tasks being performed we have an opportunity to consider the information assurance (IA) requirements for the information being shared.

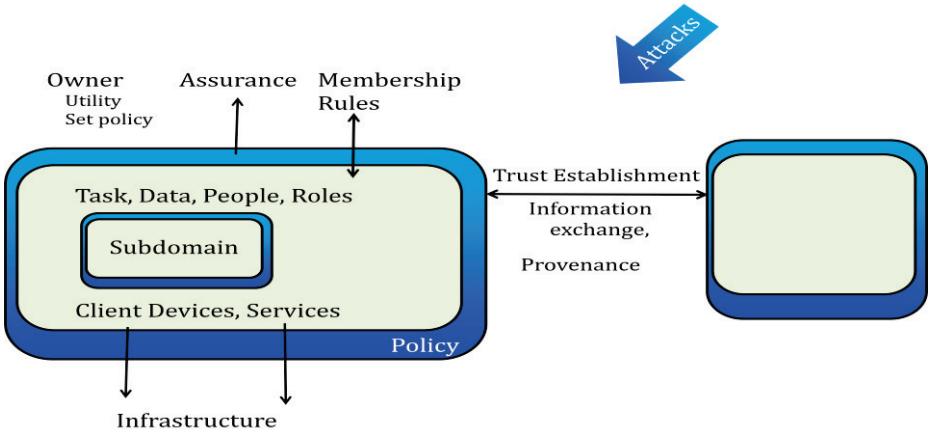


Fig. 2. An abstract description of a trust domain.

Having an understanding of the IA requirements along with an idea of the impacts when things go wrong and the value an attacker would put on the information or the ability to disrupt the process should help in designing the policies for the trust domain. However, in designing policies we need to think about the overall organisational goals and how they trade-off security needs with IT costs and the productivity of the services. Pym *et al* [12, 13] discuss the use of utility as a way of framing these tradeoffs.

Our final point is the need to relate the abstract trust domain description to how it is implemented. Whilst a wide variety of technologies could be used to realise the various pieces of a trust domain we believe trusted virtualisation offers a reliable solution where policies can be enforced within the infrastructure fabric. We see this as being true both on the client side [14] and within the cloud infrastructure [15][16]; avoiding some of the issues with a complex software stack for services.

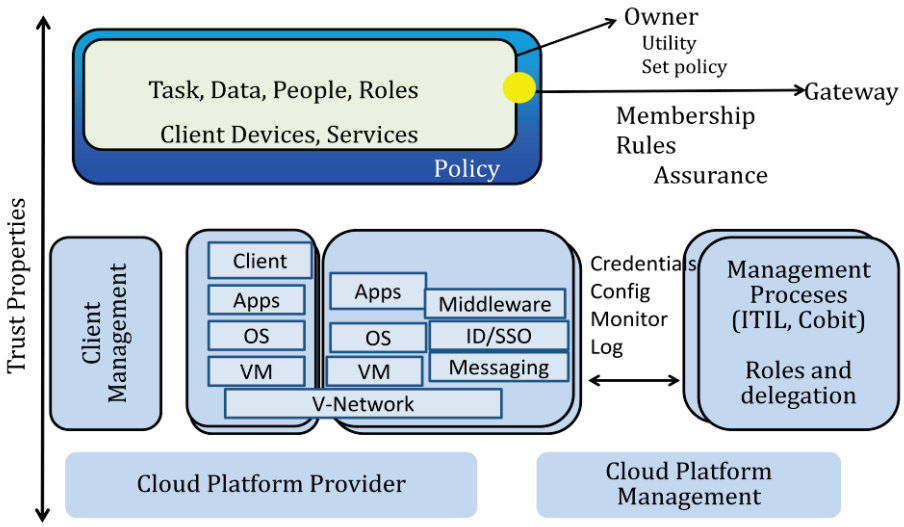


Fig. 3. The technology stack supporting a trust domain,

Figure 3 represents the technology stack supporting the trust domain. There are two immediate parts: the access devices and the cloud services (the application stack). As we look at the technology stack each service and each client device needs to be configured so as to ensure that the trust domain policies are met. This configuration is not just about setting up the software and devices to have the right properties (e.g. the right encryption and connection policies) but the policies may also reflect aspects of the management processes. For example, we may specify that security patches must be applied within 1 week or that the administrators have been vetted to a given standard. Hence here see trust domain policies being enforced by configuring systems, technical and procedural controls.

It is also necessary to provide provenance or trust statements to the trust domain on joining, connecting and dynamically at runtime, to demonstrate that each service or client can be trusted to maintain the trust domain requirements. Such evidence could use mechanisms like trusted computing-based attestation to prove that systems have started in an appropriate configuration, combined with means of demonstrating that management processes will ensure that those systems remain in a trustworthy state. In this way clients and services establish trust remotely from the trust domain but this removed the need for adhoc pair wise relationships to be established.

With the cloud service stack we describe cloud services providing applications, running middleware, as well as maintaining their own OS images. Underneath this the cloud platform providers run a virtualisation fabric (e.g. Xen or KVM) and the physical fabric of servers, network elements and data centres. The need to push configurations down the stack to meet trust domain policies, and the need for assurance information to be provided up the stack for compliance monitoring will impact all layers of the stack. Ideally, cloud platforms would support trusted virtual infrastructure [14, 17, 18] where security policies can be enforced within the VMM and virtual management area (e.g. disk and network encryption, network filtering, device access). Where the technologies offer weaker boundary enforcement more monitoring technologies, management processes and assurance information may be required both at the cloud platform and cloud services.

In our definition of trust domains, we are developing a taxonomy to classify and describe the various aspects of this stack from the virtualisation layer up to the way we conceive and describe the abstract trust domain. Within this taxonomy we stress the importance of the trust and security properties at each layer and how they link together.

4 Modelling

Our trust domain approach allows us to think through the trust requirements for a business project or process being supported through cloud services. As we look at the policy space there are many different choices and an organisation needs to choose ones that meet its utility balancing cost, productivity

and security. The security analytics approach [19] allows us to model a system and potential variations using our modelling languages Gnosis [20] based on a process, resource, location calculus [21]. Within the model we can represent events using stochastic distributions allowing us to run Monte Carlo simulations to understand how the system runs with different policy variations. In previous work we have modelled security processes and looked at different choices.

Our approach with trust domains is to model the movement of information as a resource being moved by processes between locations. Individual's actions can be represented as processes (e.g. those performing a task, administrators and attackers) moving data between locations. This allows us to understand how information spreads as well as modelling human behaviour. Locations here represent storage associated with applications, servers and clients here we can model how these different infrastructure components act.

5 Conclusion

Using cloud in a trustworthy way is not just a simple matter of choosing a good service with the right terms and conditions. As companies start to use cloud in complex ways we need to know that the whole information flows involved in the tasks are secure from the client systems of them and their partners along with a complex service supply chain. We propose that the notion of a trust domain with services and users as members who need to satisfy security constrains as a model for thinking about the security management life-cycle as the enterprise IT stack breaks up. We also suggest that modelling can help in choosing and maintaining an appropriate set of policies for a given task.

6 References

1. Mell P Grance T (2011) The NIST Definition of Cloud Computing (Draft). Technical report, National Institute of Standards and Technology, US Department of Commerce, 2011. Special Publication 800-145 (Draft)

2. Yam, C., Baldwin, A., Shiu, S., and Ioannidis, C. 2011. Migration to Cloud as Real Option: Investment Decision under Uncertainty. In *Proceedings of the 2011IEEE 10th international Conference on Trust, Security and Privacy in Computing and Communications* (November 16 - 18, 2011). IEEE Computer Society, Washington, DC, 940-949. DOI= http://dx.doi.org/10.1109/TrustCom.2011.130

3. Baldwin, A., Pym, D., Sadler, M., and Shiu, S. 2011. Information Stewardship in Cloud Ecosystems: Towards Models, Economics, and Delivery. In *Proceedings of the 2011 IEEE Third international Conference on Cloud Computing Technology and Science* (November 29 - December 01, 2011). IEEE Computer Society, Washington, DC, 784-791. DOI= http://dx.doi.org/10.1109/CloudCom.2011.121

4. Baldwin, A. , Y. Beres, L. Carrotte, T. Koulouris, B. Monahan, D. Pym, S. Shiu, and C.Y. Yam (2012). Exploring Information Stewardship with the cloud ecosystem model. Intl. Conf. on Modeling, Simulation, and Visualization (MSV)

5. Lloyd V (2011) Planning to implement service management (IT Infrastructure Library). The Stationery Office Books. http://www.itil.co.uk/ publications.htm (accessed 01/01/2012)

6. ITGI, Control Objectives for Information and Related Technologies (COBIT), 3rd edition (1998).

7. HASAN, R., SION, R., AND WINSLETT, M. Introducing secure provenance: problems and challenges. In StorageSS '07: Proceedings of the 2007 ACM workshop on Storage security and survivability

8. Namiluko, C. and Martin, A. Provenance-Based Model for Verifying Trust-Properties Lecture Notes in Computer Science, 2012, Volume 7344, Trust and Trustworthy Computing, Pages 255-272

9. Baldwin, A., Casassa Mont, M., Beres, Y., and Shiu, S. 2010. Assurance for federated identity management. *J. Comput. Secur.* 18, 4 (Dec. 2010), 541-572.

10. Mandiant M-Trends™ 2010: The Advanced Persistent Threat (http://www.mandiant.com/resources/m-trends/)

11. Baldwin, Adrian, David Pym and Simon Shiu. Enterprise information risk management: Dealing with cloud computing. To appear: Privacy and Security for Cloud Computing: Selected Topics, Siani Pearson and George Yee (editors), Springer, Computer Communications and Networks series, 2012.

12. Beres Y Pym D Shiu S (2010) Decision Support for Systems Security Investment. Proc. Business-driven IT Management (BDIM) 2010. IEEE Xplore, 2010

13. Beautement A Coles R Griffin J Ioannidis C Monahan B Pym D Sasse A Wonham M (2009) Modelling the Human and Technological Costs and Benefits of USB Memory Stick Security.In *Managing Information Risk and the Economics of Security*, M Eric Johnson (editor), Springer 2009

14. Dalton C Plaquin D Weidner W Kuhlmann D,Balacheff B, Brown R (2009) Trusted virtual platforms: akey enabler for converged client devices. *SIGOPS Oper. Syst. Rev.* 43, 1 (January 2009), 36-43. doi:10.1145/1496909.1496918

15. Bussani, A, n Gri, J L Jansen, B Julisch, K Karjoth, G Maruyama, H Nakamura, M Perez, R Schunter, M Tanner, A Doorn, L V Herreweghen, E A V Waidner, M Yoshihama, (2005) Trusted Virtual Domains: Secure foundations for business and IT services. IBM Technical Report
http://domino.watson.ibm.com/library/cyberdig.nsf/papers/293320BC87A5D430852570BD005D2837/$File/rc23792.pdf

16. Catuogno, L.; Löhr, H.; Manulis, M.; Sadeghi, A.-R.; Stüble, C. & Winandy, M. (2010), 'Trusted virtual domains: Color your network.', Datenschutz und Datensicherheit 34 (5) , 289-294 .

17. John Linwood Griffin, Trent Jaeger, Ronald Perez, Reiner Sailer, Leendert van Doorn, and Ramceres. 2005. Trusted virtual domains: toward secure distributed services. In Proceedings of the First conference on Hot topics in system dependability (HotDep'05). USENIX Association, Berkeley, CA, USA, 4-4.

18. Cabuk, Serdar, Chris I. Dalton, HariGovind Ramasamy, and Matthias Schunter. 2007. Towards automated provisioning of secure virtualized networks. In Proceedings of the 14th ACM conference on Computer and communications security (CCS '07). ACM, New York, NY, USA, 235-245. DOI=10.1145/1315245.1315275
http://doi.acm.org/10.1145/1315245.1315275

19. Pym D Shiu S Coles R van Moorsel A Sasse MA Johnson H (2011) Trust Economics: A systematic approach to information security decision making. Final Report for the UK Technology Strategy Board 'Trust Economics' project.
http://www.hpl.hp.com/news/2011/oct-dec/Final_Report_collated.pdf (accessed 01/01/2012)

20. Core Gnosis (2012) http://www.hpl.hp.com/research/systems_security/gnosis.html (accessed 01/01/2012)

21. Collinson M, Monahan B Pym D (2012) A Discipline of Mathematical Systems Modelling. Forthcoming monograph, College Publications, 2012

Information Asymmetry in Classified Cross Domain System Accreditation

Joe Loughry

Department of Computer Science, University of Oxford
Wolfson Building, Parks Road, Oxford, OX1 3QD, UK

Abstract. The difficulty of cross domain systems security accreditation lies inherent in the fact that, by definition, such systems always span at least one boundary between security domains controlled by different data owners. Consequently, approved solutions regularly encounter security testing criteria that represent the duplicated responsibility for residual risk of multiple security accreditors. Each data owner perceives a site-specific set of risks that would be desirable to mitigate, a technology-dependent set of risks that it is possible to mitigate, and a residual risk it is felt acceptable not to mitigate. Time and cost inefficiency in cross domain system accreditation are shown to originate from asymmetry of knowledge; Spence's criteria for market signalling are shown to hold by analogy for accreditor–accreditor communication in the presence of unequal or non-hierarchical security clearances. By formalising signals, an efficient route to agreement about the true level of residual risk might avoid repeated re-testing and redundant risk mitigations. If successful, the unnecessarily high cost of duplicated security test and evaluation effort could be greatly reduced.

1 Introduction

'Sometimes it is necessary to violate your own security policy' [1]. A concrete example is the existence of cross domain systems, whose reason for existence is to violate security policy in a controlled manner [2]. Cross domain systems are interesting because they nearly always guarantee an adversarial environment during validation testing. Owners of classified systems rarely trust outsiders—among whom they number their users, their own software developers, owners of other classified systems, and vendors or installers of cross domain solutions.

Caught in the crossfire of all this mistrust is the Designated Accrediting Authority (DAA), a government official whose unenviable task it is to try to determine the true level of risk in a system, reduce it to acceptable levels, and then formally accept personal responsibility for the correct operation of the system, on penalty of going to jail for a long time if the cross domain system should fail in use.

In this paper we show that the DAA's predicament is the same thing as the problem of market failure in the presence of asymmetric information familiar to economic theory. Furthermore the criteria for *signalling* established by Spence

and Akerlof are met [3, 4]. This suggests a possible solution to the present high cost of certification and accreditation of cross domain systems that currently manifests in repeated testing and retesting of the same security criteria by DAAs at different security classifications.

1.1 Applicability

This is an important problem for a specific, if not very visible, application area. More generally, though, it is a microcosm of the problem of setting security parameters consistently across a network of security devices in the cloud—but all occurring in one box.

1.2 Organisation of the Paper

The first part of this paper defines cross domain solutions and systems, designated accrediting authority, and the assurance requirements of security certification and accreditation for systems and networks handling classified information. Next, a simple example is used to motivate the development of a model of DAA–DAA interaction that is sufficiently powerful to reason about problems that have been observed to occur in real situations. The concept of residual risk is defined and shown to be the primary driver of DAA decisions. Cross domain system accreditations have a tendency to prompt ineffectual repeated testing because of security clearance rules that limit inter-DAA communication; this leads to high costs. Economic theories of asymmetric knowledge are shown to be applicable to the problem. Finally, a solution is proposed using an artificial market to resolve the asymmetry and reach an improved equilibrium resulting in lower overall cost.

1.3 Purpose of this Paper

The purpose of this paper is to put forth the idea and validate whether or not working accreditors are likely to find the model and the proposed tool useful.

2 Cross Domain Systems and Cross Domain Solutions

Cross domain solutions are needed anywhere that security boundaries exist. As David Bell put it, 'In our real-world environment made up of multiple single-level networks—that is, relatively isolated networks each of which comprises a security enclave or domain at a particular security classification—connected to the network cloud, it is often necessary to move information across security boundaries, and by the Intermediate Value Theorem for Computer Security (CS-IVT), at least one multi-level component must exist in the cloud' [5, §6.2], [1]. A cross domain solution or *controlled interface* is employed to interconnect systems or networks in different security domains, thereby forming a Cross Domain System, or CDS [2]. By definition, CDS installations always span at least one boundary between security domains controlled by different data owners [6].

In the most general case, data owners nearly always mistrust one another, because the relationship between their security classifications may be non-hierarchical, or incommensurable, or simply equipotent, as happens in international installations [7]. Cross domain solution developers and installers regularly encounter situations that have no clear precedent yet need to be resolved; they do this by means of a combination of high-assurance software/hardware and through negotiation with data owners and security accreditors.

Each data owner is represented by a DAA whose job it is to approve connection of the cross domain solution to a classified system or network and to permit operation for a specified period of time [8–11]. DAAs work closely with the cross domain solution developer or installer and other DAAs to ensure adequate protection of the classified information in their security domain. Data owners worry about two potential threats: accidental compromise of the confidentiality of classified information outside the security boundary (called a spill), and negative impacts to the integrity or availability of their information from the introduction of malicious code or denial-of-service attacks. DAAs, being people, in addition operate under the constraints of their government security clearance and security classification rules.

3 A Model of DAA Interactions Constrained by Different Security Clearances

Figure 1 shows a very simple example of a CDS that is nevertheless sufficient to illustrate the problem. There is generally no DAA for unclassified systems, so let us imagine that the low side is classified Confidential and the high side contains Sensitive Compartmented Information (SCI). The low-side[1] DAA represents one of the military services because the information on the low side has a collateral classification, that is, it is classified but not protected by additional code words. But because the high side is SCI, which has a non-hierarchical relationship to collateral security classifications, the high-side DAA must represent one of the members of the intelligence community, for example the National Geospatial-Intelligence Agency (NGA). In reality, cross domain systems commonly are more complicated than this example, with multiple data flows in more than one direction, more than two endpoints, conditional information flows dependent on content, sanitisation and/or transliteration functions, and non-hierarchical security classification relationships. The model presented in this section, however, is sufficient to reason about cross domain systems in collateral, compartmented, and international installations.

In our model, DAAs having responsibility for information at different classification levels have security clearances and accesses that match their responsibilities. In the real world, that might not be true; all DAAs might be cleared

[1] By convention, cross domain solution developers habitually refer to data flows as being 'low to high' or 'high to low' despite the fact that the distinction may be a matter of opinion depending on the data owner's perspective.

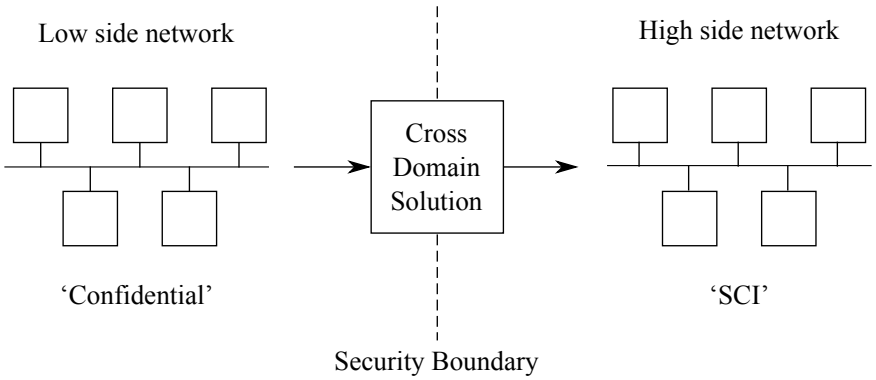


Fig. 1. A simple cross-domain system with asymmetric information

for Top Secret/SCI. Our model presupposes the more limited case for two reasons: firstly, because it better reflects the intent of security policy irrespective of administrative convenience, and secondly because it allows us to analyse the important case of international CDS installations, where DAAs most definitely do not share mutual clearances.

Consider the following situation. DAA 1 holds a Confidential security clearance and has need-to-know, so is therefore privy to classified information about certain threats that are known to exist by the data owner of the low-side system. DAA 1 perceives a site-specific set of risks A_1 that affect the low side system, each risk computed from the *probability* of occurrence of an identified *threat* leading to an *impact* which is derived from the value of the asset [12]. Risks can be avoided, mitigated, transferred, or accepted [13]. DAA 1 assesses a set of risks based on the known threats at his or her clearance, the best available estimate of the probability of occurrence pT_i of each, and the value V of the information on the low side as perceived by the low side data owner; this set of risks that DAA 1 thinks it would be desirable to mitigate is:

$$A_1 = \bigcup_i [pT_i \times V] \tag{1}$$

where $0 \leq pT_i < 1$.

DAA 2 holds a Top Secret/SCI security clearance with accesses similarly determined by DAA 2's need-to-know. It can be understood that DAA 2 knows about some highly classified threats that are not known to DAA 1. In practice, DAA 2 should be aware of all the threats that DAA 1 knows about, but this is not required by the model. DAA 2 perceives a site-specific set of risks A_2 affecting the high side based on probably a larger set of known threats, an estimate pT_j of their probability of occurrence, and the value V' of the information on the high side as perceived by the high side data owner. This is the set of risks that

DAA 2 thinks would be desirable to mitigate:

$$A_2 = \bigcup_j [pT_j \times V'] \tag{2}$$

where $0 \leq pT_j < 1$.

A_2 is not necessarily a proper superset of A_1 or even needs be larger than A_1. DAA 1 values low side information independently of DAA 2, and quite possibly assesses different probabilities for similar risks—although if they are seriously different, it might be better to treat them as distinct threats—simply because it is DAA 1's own asset on the line. Similarly, DAA 2 values high side information independently of DAA 1.

3.1 The Idea of Residual Risk

DAA 1 perceives a technology-dependent set of risks B_1 that it is possible to mitigate, and DAA 2 similarly perceives a set of risks B_2 that is is possible to mitigate. Because both sides are presumed to be aware of what is technically possible, it is likely that $B_1 = B_2$, although there is always the possibility that DAA 2 is aware of some highly classified risk mitigation for a threat that DAA 1 does not even know exists.

The job of a DAA is formally to accept responsibility on behalf of the Principal Accrediting Authority (PAA) for the *residual risk* of connecting their classified information system to the overall CDS. Residual risk for each DAA i is defined as the relative complement,

$$(A_i - B_i) \quad , \tag{3}$$

that being the set of risks which it is felt, by a particular DAA, to be acceptable not to mitigate. The goal of the DAA is always to minimise residual risk. This is achieved through a combination of choosing the right cross domain solution vendor and product based on Certification Test and Evaluation (CT&E) results, correctly configuring the cross domain solution according to its technical capabilities, and rigorous testing of the CDS before and after issuance of approval-to-connect to verify that the CDS adequately protects the security domain of the data owner each DAA represents. In real installations, the PAA responsible for the highest-classification information in the CDS generally is responsible for choosing a cross domain solution vendor. The process of testing a cross domain solution in situ forming a CDS is called Security Test and Evaluation (ST&E) and results, in our model, in the granting of an Approval to Operate (ATO) from each DAA. ATO lasts for a limited amount of time, usually three years, and is periodically reviewed.

3.2 Asymmetric Knowledge

To reiterate, by definition a CDS installation always spans at least two security domains controlled by different data owners. With multiple data owners come

multiple DAAs. With each DAA, under present rules, comes another round of ST&E, oftentimes performed by the same team of Independent Verification and Validation (IV&V) contractors for reasons described in [6].

It is from the asymmetry of knowledge just described that the well-known time and cost inefficiency of the CDS accreditation process arises. If the true level of residual risk could be agreed upon by all DAAs and validated by a single round of ST&E to the satisfaction of all parties, then the cost of CDS accreditation would be greatly reduced. Towards that goal, we now show that the problem is isomorphic to a well-known result from economic theory.

Problems that can be caused by asymmetric information are well understood. In markets characterised by a lack of knowledge on the part of buyers, rational behaviour by all participants can lead to a collapse of the market to the point where no seller will offer a product for sale [3]. Conversely, in markets characterised by a lack of knowledge on the part of sellers, *adverse selection* results in a lopsided distribution of risk, which can lead to a situation called *moral hazard* in which participants who know they are insulated from the consequences of a risk behave differently than if they were fully exposed to it [14]. Game theory offers a handful of compensating strategies for asymmetric knowledge, among them the concept of *signalling* [4, 15]. In signalling, sellers in a market under conditions of asymmetric information can resolve the asymmetry by communicating information to buyers in a convincing way, but in order for the buyer to believe the signal, the cost of asserting the signal must be high [4].

Can we apply these ideas to the problem of improving the situation of a temporary non-optimal equilibrium amongst the individual assessments of n different DAAs about the total residual risk resulting from the installation of a complex CDS? In one sense, the problem is that non-communicating DAAs can end up stuck in isolated local minima because they lack an important piece of information about a risk mitigation already implemented by another DAA in response to a threat the existence of which is above the first DAA's clearance level.[2] In another sense, the problem is analogous to a covert channel, through which we wish to communicate some information in violation of the system security policy [16]. In that case, the security policy we need to violate is not that of the CDS, but of the security clearances of at least some of the DAAs.

3.3 Justification for the Accreditor Model

Is it even meaningful to talk about a single value for the residual risk of a complex CDS interconnecting many different security enclaves, thereby exposing data of widely differing perceived—and maybe even objectively intrinsic—values to the

[2] The related problem of highly classified threats for which there is no known risk mitigation is a very real one, but in the absence of a fix, from the perspective of the higher-cleared DAA it is a worry he or she cannot talk about, and from the perspective of the lower-cleared DAA, ignorance is bliss.

risk of damage, disclosure, or loss? It is attractive to call the overall residual risk

$$R = \sum_{i}^{n} (A_i - B_i) \tag{4}$$

from the residual risks in (3) assessed by each individual DAA—who is, after all, responsible for the safety of data in his or her security enclave—because this metric behaves the right way in the intuitive sense that if one DAA feels that the residual risk to one enclave is unusually high, it properly increases the overall level of risk of the CDS.

We claim that this model is sufficiently powerful to address every situation encountered in the field. To show this, first consider a collateral CDS where each accreditor has a security clearance that is one of confidential, secret, or top secret.[3] The highest security clearance of any accreditor in the system, and consequently the security classification of the CDS, is called 'system-high'. The lowest security clearance of any accreditor in the CDS, in our model, determines the classification of 'system-low'. In a collateral CDS, each accreditor's security clearance is hierarchically related to all of the others such that when any accreditor cleared at system-high is satisfied that the residual risk is acceptable, all the accreditors immediately agree because they know that the system-high accreditor already knows everything *they* know about the threats and vulnerabilities of the CDS.

Now consider the case of a CDS containing SCI. Here there is no strictly hierarchical relationship between the security clearances of accreditors, in practice some of whom might have collateral clearances. System-high floats to SCI (which dominates all collateral classifications) with the union of all applicable compartments; the definition of system-low remains as before. Now if there are any accreditors who are not cleared for SCI, or there are at least two SCI-cleared accreditors who do not share at least one compartment, we are at an impasse—at least one accreditor may know of a threat or risk mitigation that affects the residual risk of the CDS but is prohibited from communicating that information to at least one other accreditor. In this asymmetric information situation, only a limited amount of information can be legally communicated without violating clearance or classification rules: the fact that a particular accreditor believes the residual risk of the CDS to be too high.[4] We have no solution for this prob-

[3] The presence of uncleared accreditors, who can be considered to have a clearance of 'unclassified', such as might be represented by private organisations with information security responsibility such as health care providers, does not invalidate the relation.

[4] Interestingly, this leaks information; recall the example given earlier of an accreditor who is cleared to know about a highly classified threat or risk mitigation. There are only three possibilities: firstly, if the accreditor says only that the residual risk is unacceptably high, that statement reveals two facts: the existence of a classified threat and the fact that no one knows how to mitigate it. The second possibility is if the accreditor says only that the residual risk is acceptable; this leaks the fact of the existence of a classified risk mitigation, although not necessarily the fact of a higher classified threat, as the classified risk mitigation may be for an already known

lem yet; we have merely constructed a model that illustrates the difficulty. The scenario is drawn from personal experience of CDS installations in the field.

Finally, consider the case of international accreditations. Without loss of generality, international accreditors can be treated as SCI-cleared accreditors who have access to only one compartment, that compartment being the name of their country. (The uncleared accreditors mentioned previously could equivalently be modelled as foreign country accreditors.) This is consistent with the extension of collateral classifications with handling caveats such as NOFORN ('not releasable to foreign nationals') or EYES ONLY. The model is therefore complete.

4 Proposed Solution

With that out of the way, let us consider the problems of simulating a market amongst DAAs who are constrained from communicating freely about their individual assessments of residual risk of a CDS because of security classification rules. It is a weird sort of market in which participants offer to buy and sell commodities that they do not know the value of, although someone else does. Since some of the DAAs are prohibited from describing the exact details of a threat or a risk mitigation, or even the lack of any known risk mitigation for a threat with no countermeasure, they must in some way signal (in Spence's use of the word) the actual value of the residual risk as they perceive it.

4.1 Predicting the Behaviour of Accreditors

The traditional view of signals holds that for a signal to be convincing, it must have a high cost to preclude dishonest use of signals to gain unfair advantage [4]. We believe that Spence's cost constraint is satisfied in this adaptation of the model because there is negative incentive for cheating when the result of dishonesty—that is, to communicate a false reading of the residual risk as perceived by DAA k—would either raise the value of R in (4), thereby increasing the amount of risk that DAA k must accept formal responsibility for, possibly even to a level exceeding DAA k's authority; or conversely, to artificially depress the apparent level of risk below what DAA k knows the true value to be, again raising the level of personal risk to DAA k's own self when he or she signs on the dotted line.

The required high cost of signals in this market is made manifest by the very real risk that a cleared DAA takes in choosing to communicate information about the true level of residual risk in violation of his or her oath to protect classified information. It works both ways, as even DAAs with low security clearance understand the need-to-know rule and would hesitate to casually provide classified information to another absent a clearly communicated need-to-know decision from their authorised security officer. The necessary and sufficient criteria for signalling, therefore, are met [3, 4, pp. 499–500 and 367, respectively].

threat with no known risk mitigation. It is an unavoidable leak, however, as the third alternative would be for the accreditor to remain silent, thereby accepting personal responsibility for a risk that is intolerably large. The accreditor must say something.

4.2 Controlling the Schedule of Certification

One final aspect remains to be examined. This is the seldom-acknowledged incidence of 'turf wars' in the certifier and accreditor communities. We have observed in CT&E activities, and have anecdotal reports from practising DAAs in ST&E activities, of prima facie obstructionist behaviour exhibited by accreditors and their supporting casts of penetration testers against cross domain solution developers and in some cases even against other DAAs. The reasons for such activity are not yet clear. The outcome, however, leads almost always in the direction of increased certification and accreditation cost; in the worst case, pathological turf wars could even lead to another well known economics result, the tragedy of the commons [17].

The developer cannot accurately forecast the schedule of certification testing during CT&E because the duration of the penetration testing phase is unknown. Very much like covert channel analysis in high-assurance Common Criteria evaluations, penetration testing tends to be unbounded in the possible effort that could be expended and always is effectively terminated either by funding or schedule, not by the completeness of testing. We found, however, that the cross domain solution developer can indirectly control at least the duration of penetration testing. In a study of the successful U.S. Department of Defence Information Assurance Certification and Accreditation Process (DIACAP) certification of a cross domain solution in 2010–11, a causal correlation was observed between certifier finding reports and the form of the developer's responses. When the developer responded by disagreeing with the findings of the certifier's penetration testers, this invariably prompted a follow-on report containing more findings. When the developer concurred with the findings, no further reports of findings appeared [1]. We believe that this mechanism may be usable by the developer to bound the schedule of certification.

5 Summary and Future Work

The main contribution of this paper is that we have shown that the market for residual risk satisfies Spence's criteria for signalling in the presence of asymmetric information. We have not yet constructed or tested a simulation of the DAA market for information, but intend to do so in future after receiving feedback from the certification and accreditation community about the applicability of the model described for the first time in this paper. The model is sufficiently general to reason about collateral, compartmented, and international accreditations, including unclassified accreditors, thereby covering the gamut of situations encountered by cross domain solution developers and installers in the field.

A new tool being developed at the University of Oxford, called *nihil obstat*, is designed to facilitate the determination of an equilibrium in the market for residual risk amongst DAAs by soliciting a series of bid/ask quotations from accreditors at different security classification levels and using them to set a 'market price' for the residual risk that each DAA is prepared to accept.

Acknowledgements

The author wishes to thank his supervisors, including Andrew Martin, who observed that the author seemed to be trying to build a covert channel machine, and Ivan Fléchais, who gave the clearest formulation of risk yet. Thanks are owed as well for the help of anonymous reviewers who improved the argument.

References

1. Loughry, J.: Security Test and Evaluation of Cross Domain Systems. PhD thesis, University of Oxford (Trinity Term 2012)
2. Director of Central Intelligence: Protecting sensitive compartmented information within information systems. DCID 6/3 (1 August 2000)
3. Akerlof, G.A.: The market for 'lemons': Quality uncertainty and the market mechanism. Quarterly Journal of Economics **84**(3) (August 1970) 488–500
4. Spence, M.: Job market signaling. The Quarterly Journal of Economics **87**(3) (August 1973) 355–374
5. Bell, D.E.: Looking back at the Bell–LaPadula model. In: 21st Annual Computer Security Applications Conference, Tucson, Arizona, USA (5–9 December 2005) 337–351
6. Loughry, J.: Unsteady ground: Certification to unstable criteria. In: Proceedings of the Second International Conference on Advances in System Testing and Validation Lifecycle, Nice, France (22–27 August 2010)
7. Sun Microsystems, Inc.: Compartmented Mode Workstation Labeling: Encodings Format DDS-2600-6216-93. Trusted Solaris 2.5, 2550 Garcia Avenue, Mountain View, California 94043-1100 USA. (July 1997) Revision A.
8. National Institute of Standards and Technology: Guide for Applying the Risk Management Framework to Federal Information Systems. (February 2010) NIST Special Publication 800-37 Revision 1.
9. Ross, R., Johnson, A., Katzke, S., Toth, P., Stoneburner, G., Rogers, G.: Guide for Assessing the Security Controls in Federal Information Systems. (July 2008) NIST Special Publication 800-53A.
10. United States Department of Defense: DoD information assurance certification and accreditation process (DIACAP). ASD(NII)/DoD CIO (November 28, 2007) DoD Instruction 8510.01.
11. U.S. Department of Commerce, National Institute of Standards and Technology: NIST Special Publication 800-53, Revision 3: Recommended Security Controls for Federal Information Systems and Organizations. (June 2009) Final Public Draft.
12. Flechais, I.: Designing Secure and Usable Systems. PhD thesis, University College, London (2005)
13. Tockey, S.: Return on Software. Addison–Wesley Professional, Reading, Massachusetts (2004)
14. Crosby, E.U.: Fire prevention. Annals of the American Academy of Political and Social Science **26** (1905) 224–238
15. Stigler, G.J.: The economics of information. Journal of Political Economy **69**(3) (June 1961) 213–225
16. National Computer Security Center: A Guide to Understanding Covert Channel Analysis of Trusted Systems. (November 1993) NCSC-TG-030 Version 1.
17. Hardin, G.: The tragedy of the commons. Science **162**(3859) (13 December 1968) 1243–1248

Configuring Cloud Deployments for Integrity

Trent Jaeger, Nirupama Talele, Yuqiong Sun, Divya Muthukumaran,
Hayawardh Vijayakumar, and Joshua Schiffman
{tjaeger,nrt123,yus138,muthukumaran,hvivay,jschiffm}@cse.psu.edu

Systems and Internet Infrastructure Security Lab
Pennsylvania State University

Abstract. Many cloud vendors now provide pre-configured OS distributions and network firewall policies to simplify deployment for customers. However, even with this help, customers have little insight into the possible attack paths that adversaries may use to compromise the integrity of their computations on the cloud. In this paper, we leverage the pre-configured security policies for cloud instances to compute the integrity protection required to protect cloud deployments. In particular, we show that it is possible to compute security configurations for cloud instance deployments that can prevent information flow integrity errors and that these configurations can be measured into attestations using trusted computing hardware. We apply these proposed methods to the OpenStack cloud platform, showing how web server application instance can be configured to protect their integrity in the cloud and how integrity measurement can be used to validate such configurations for approximately 3% overhead.

1 Introduction

Cloud computing is a realization of computing as a utility, where customers submit their computing tasks to a centralized service that provides the resources necessary to execute those tasks. According to NIST [33], "[c]loud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources." Rather than purchasing and maintaining an abundance of hardware resources themselves, customers can "plug in" to the cloud, paying for only the quantity of resources used. This is particularly attractive to those customers whose resource utilization may vary dramatically or where the costs of hardware and its maintenance form a significant fraction of their overall budget.

Despite a promising business model, security is a major concern that may limit the impact of the cloud computing paradigm. Customers need assurance that their personal or proprietary processing can be protected on systems administered by a third party. For example, if a medical billing organization wants to use a cloud system to run their processing, they will have to consider how to protect their processing from remote adversaries and achieve compliance with privacy requirements, such as HIPAA, when such computing is moved to a third party. Key to the safe use of any computing system is the ability to configure security policies that limit adversary access to critical processing. While the cloud vendors aim to make security configuration simpler, through pre-configured OS distributions and default firewall policies, significant responsibility for security decisions still falls upon the customers [18].

At Penn State's Systems and Internet Infrastructure Security (SIIS) Lab, we are studying how cloud vendors and customers can work together to configure cloud computations that protect customer processing. This study spans two critical issues in cloud computing: (1) configuring cloud computations to prevent known attacks on such pre-configured OS distributions and (2) validating the runtime compliance of cloud computations with the expected configurations.

First, in modern systems, much of the security configuration is already done in advance. Now, administrators configure hosts by selecting an OS distribution, selecting the desired application programs, and

configuring network access for the deployed distribution. Selecting a commodity OS distribution now often implies selecting a mandatory access control (MAC) policy to be enforced by the distribution [53, 59, 36, 35, 28], which limits the number of processes accessible to remote adversaries and aims to confine the rest. Due to the complexity of MAC policies, administrators do not modify such policies manually, limiting the custom defenses that they can apply to the selection of software packages and configuration of firewall rules. Further, such policies are designed for *least privilege* [41], meaning that functionality drives which permissions are included, not security concerns. As a result, the MAC policy may not accurately enforce the integrity requirements that the administrators may expect, in such cases permitting operations that would compromise their integrity requirements (i.e., if the administrators understood the MAC policy). With the advent of cloud computing, the limitations of this approach to configuration has been transferred into a problem for cloud customers.

Second, once the customer has settled on a configuration for their cloud instance, they want to know whether the runtime execution of their instance will behave as expected. Trusted computing mechanisms have been developed to collect measurements of system events necessary to produce proofs of system integrity (attestations) that can be verified by remote parties [39, 49, 16, 50, 51, 44, 25, 24, 5]. However, trusted computing has not been widely adopted on traditional hosts, and cloud computing provides additional challenges because the cloud is opaque to customers. For example, the node controllers upon which instances are deployed may not be addressable by remote customers and cloud instances may be migrated dynamically.

In this paper, we describe methods for: (1) identifying and resolving integrity problems in the security policies resulting from the deployment of cloud computations using pre-configured OS distributions and (2) measuring and validating that the runtime integrity of deployed systems complies with the integrity-protecting configuration. This work leverages several research projects underway at the SIIS Lab. First, we demonstrate that we can use available package configurations and MAC policies in OS distributions to locate individual program entrypoints that are accessible to adversaries [56], called the *attack surface* of the program [15]. Second, using this information, we can configure information flow problems whose solutions mediate adversary access, blocking potential attacks paths [29]. Such problems can be constructed to evaluate remote or local threats against individual cloud instances or threats against a computation consisting of several instances. Third, we describe methods for enforcing such mediation in operating systems [55] and programs [47, 21, 31], enabling customers to deploy cloud instances that protect their integrity proactively. Finally, we describe our trusted computing mechanism for measuring cloud instances, based on an *integrity verification proxy* (IVP) service [46, 45], which monitors customer integrity criteria on VMs running in the cloud.

We demonstrate these methods for Ubuntu Linux OS distributions using SELinux deployed on the OpenStack cloud platform. What we find is that: (1) we can produce policies for cloud instances that approximate classical integrity, in the form of Clark-Wilson integrity [10] and (2) we can monitor the enforcement of such policies using trusted computing with low overhead. Using such techniques, cloud vendors can provide customers with insight into how customization may impact the integrity of their deployments and monitor the execution of their deployments.

The remainder of this paper is as follows. Section 2 provides background on Infrastructure as a Service (IaaS) cloud platforms, which are the target of this work. Section 3 defines the information flow integrity problem that we aim to address in this paper. Section 4 describes our approach for configuring cloud computations for integrity. Section 5 outlines how we monitor cloud computations for their adherence to that configuration. Section 6 concludes and discusses future work.

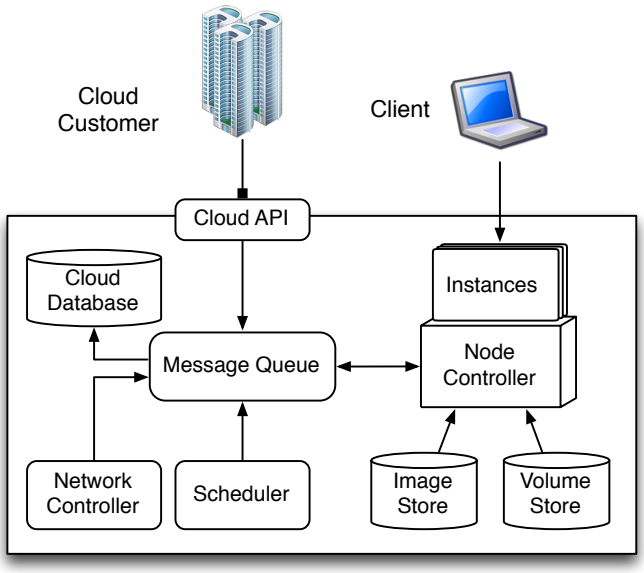


Fig. 1. IaaS Clouds.

2 IaaS Cloud Architecture

Clouds come in a variety of architectures with differing levels of service and features. While the definition of a "cloud" is as nebulous as its name, NIST has begun to categorize clouds based on the degree of administration and services the clouds offer to the customer [33]. Cloud computing provides a platform for customers to run *instances*, computations containing customer-chosen software and data. Cloud platforms offer different granularity of processing, such as Software as a Service [40, 14] (SaaS), Platform as a Service [26, 13] (PaaS), and Infrastructure as a Service [4, 1] (IaaS). In this paper, we consider only IaaS clouds as they are the building blocks of higher level cloud abstractions. Moreover, the IaaS paradigm gives the customer more control over how cloud components manage sensitive data and code. IaaS clouds provide the basic infrastructure launch instances in the form of virtual machines (VMs), such as VM hosting, virtualized networking, and storage for VM images and disk volumes. Customers can use IaaS clouds to replace or supplement a traditional data center by hosting the service in the cloud and scaling up compute and storage requirements on-demand.

As an illustrative example, consider the high-level IaaS cloud architecture in Figure 1. The primary component is the *node controller*, a VM host for customer VM instances. Clouds are composed of thousands of these nodes, which are broken into clusters that provide a level of redundancy and can be spread out geographically based on demand of the region. Within each cluster, a *network controller* is responsible for configuring virtual networking between instances and translating public IPs to private intra-cloud addresses. When a new instance is requested, a *scheduler* chooses a node controller to host the VM based on various scheduling policies like resource fairness. In addition, the cloud's state (e.g., which instances are running) is stored in a *cloud database* that is updated and referenced by the various components.

Instances are created from disk images stored in the *image store*. They are uploaded to the cloud by the customer or a third party vendor and remain static across reboots. Additional mutable storage is provided

through additional services like a key-object stores (e.g., Amazon's S3 [3]) or network attached block storage from a *volume store*. Customers control their instances through an API endpoint, which also exposes options for configuring firewall rules, `ssh` host identity keys, and other policies. Finally, instances open to the internet can interact with clients to provide services for which they were designed.

The cloud instances that we deploy are Ubuntu Linux OS distributions that run a particular application package, such as a web server (Apache), database (MySQL), or web client (Firefox). Each of the distributions we use includes an SELinux mandatory access control (MAC) policy [36] that governs the accesses of the running processes. SELinux policies are composed from individual policies designed for software packages. The SELinux policies are produced using a runtime analysis, which biases them toward *least privilege* [41]. That is, SELinux policies restrict processes based on the functionality required for the associated program to run, not based on protecting the integrity of the process. As a result, adversaries often have access to some of the resources used by processes, but there is no principled approach to identify those cases and protect the process. Other commodity MAC enforcement works similarly [35, 59, 53, 28].

In discussing cloud integrity, we assume the physical security of the cloud is maintained and that attacks on the hardware are prevented. We also trust the cloud at an organizational level to provide services without malicious intent. That is, we assume the cloud's components were honestly configured with the purpose of protecting the integrity and confidentiality of its customers. However, we do not trust the cloud beyond that point and accept that curious or malfeasant administrators may attempt to alter cloud systems in an untrustworthy way. Thus, we consider threats like an administrator logging in to the node controller to directly read instance memory or locally cached disk images. We also consider threats from network attackers both within the cloud intranet and externally that can snoop on, alter, or inject packets. We do not guarantee that satisfaction of an integrity criteria implies that the system will not perform undesirable behavior. We leave it up to the relying party to design integrity criteria that would ensure this property.

3 Information Flow Integrity Problem

To detect all possible integrity threats, we must identify when an adversary can write to data that may be read or executed the victim [6], which can be modeled as an information flow problem. Traditionally, an information flow problem is defined as follows:

Definition 01 *An* information flow problem*, $\mathcal{I} = (\mathcal{G}, \mathcal{L}, \mathcal{M})$, consists of the following concepts:*

1. *A* directed data flow graph $G = (V, E)$ *consisting of a set of nodes V connected by edges E.*
2. *A* lattice $\mathcal{L}$= $\{L, \preceq\}$. *For any two levels $l_i, l_j \in L$, $l_i \preceq l_j$ means that l_i 'can flow to' l_j.*
3. *There is a* level mapping function $M : V \rightarrow \mathcal{P}^L$ *where $\mathcal{P}^L$ is the power set of L (i.e., each node is mapped either to a set of levels in L or to $\emptyset$).*
4. *The lattice imposes security constraints on the information flows enabled by the data flow graph. Each pair $u, v \in V s.t.$ $[u \hookrightarrow_G v \wedge (\exists l_u \in M(u), l_v \in M(v).\ l_u \not\preceq_{\mathcal{L}} l_v)]$, where $\hookrightarrow_G$ means there is a path from u to v in G, represents an* information flow error.

Such an information flow problem can be constructed for mandatory access control (MAC) policies. The data flow graph is determined by the information flow (i.e., read and write operations) authorized by the MAC policy. The integrity lattice identifies the types of security-sensitive entities, such as integrity-critical resources and adversary-accessible resources. The mapping assigns the relevant levels in the lattice to the MAC policy labels for those entities. It has been shown that information flow errors in programs [30] and MAC policies [17, 42, 9] can be automatically found using such a model.

However, resolving such information flow errors has been a complex manual task. In general, information flow integrity errors can be resolved by changing the data flow graph (i.e., the MAC policy) or adding *mediation* to change the integrity of data propagated by information flows. However, changing the data flow graph is difficult in practice because it implies a change in the operations a system may perform, which may prevent one or more programs from functioning correctly. As a result, we explore methods to to resolve information flow errors using mediation.

The challenge in this work is two-fold. First, we must map the configuration of cloud components to a system-wide information flow problem and determine which mediation placements are viable. Prior work examines information flows among VMs in a cloud environment [7], but does not examine problems caused by such flows within the cloud components themselves. Second, we want enable customers to verify that their running instances only receive untrusted inputs via mediators. We leverage trusted computing technologies [54], but we must adapt this work to cloud computing in a manner that enables verification of the guarantee above at runtime.

4 Configuring for Integrity

In this section, we explore a method for configuring cloud instances and computations consisting of multiple cloud instances to protect their integrity. As described above, each cloud instance is a pre-configured OS distribution, but the security policies in each distribution are not designed to protect integrity. However, these distributions do include information sufficient to compute adversary accessibility to program entrypoints (see Section 4.1) from which threats to cloud computation deployments can be identified (see Section 4.2). We then examine defenses for these threats that would provide proactive integrity protection system-wide in Section 4.3.

4.1 Computing Attack Surfaces

Researchers have explored several methods to describe how adversaries may use their access rights to launch attacks. For example, methods have been developed to compute *attack graphs* [48, 37, 34], which generate a sequence of adversary actions that may result in host compromise. However, these methods treat programs as black boxes, where rules describe possible compromises without principled information about the programs. Should a particular vulnerabilities be patched in the program code, then attack graphs may diverge from the actual deployment, leading to false positives.

As an alternative, researchers have argued for defenses at a program's *attack surface* [15], which is defined by the entry points of the program accessible to adversaries. In practice, a program entrypoint is an instruction in the program that invokes the system call library to receive system resources (e.g., files, network, or IPC data). Unfortunately, programs often have a large number of entrypoints, and it is difficult to know which of these are accessible to adversaries using the program alone. Some experiments have estimated attack surfaces using the value of the resources behind entrypoints [23]. However, if the goal is simply to take control of a process, any entrypoint may suffice. While researchers have previously identified that both the program and the system security policy may impact the attack surface definition [15], methods to compute the accessibility of entrypoints had not been developed.

In a recent paper, we develop a method to compute program attack surfaces for OS distributions [56]. Calculating the attack surface has two steps. First, for a particular subject (e.g., the SELinux label `httpd_t`), we need to define its adversaries (e.g., processes with the SELinux subject label `user_t`), and locate OS objects under adversarial control (e.g., files with the SELinux object label `httpd_user_content_t`). We do this using the system's MAC policy. Next, we need to identify the program entry points that access

these adversary-controlled objects. Statically analyzing the program cannot tell which permissions are exercised and which OS objects accessed at each entry point, and thus we use a runtime analysis to locate such entry points.

The adversaries of a subject are identified conservatively. For a particular subject in the MAC policy, the only other subjects that it trusts are those that have permission in the MAC policy to modify its executable program file (directly or indirectly). All other subjects are untrusted. A detailed method is specified [56] to identify all labels accessible to subject that may be modified by adversaries.

Practical runtime analysis is possible for many programs because several Linux software packages now include comprehensive test suites. These test suites test program functionality across multiple configuration options, which often identifies attack surfaces that would not be found through normal manual execution. Despite the conservative definition for adversaries, our study showed that only a small fraction of the program entrypoints are actually accessible to adversaries. For 23 system subjects in the Ubuntu Linux 10.04 Desktop distribution, only 81 out of 2138 entrypoints are accessible to adversaries. For well-known server programs, 14 out of 78 entrypoints in OpenSSH and 5 out of 30 entrypoints for Apache were accessible to adversaries. Using knowledge of these entrypoints, we identified two previously-unknown vulnerabilities, demonstrating the importance of tracking attack surfaces to prevent vulnerabilities. As a result, we found that computing attack surfaces is possible, yields useful information for extending proactive integrity protections, and reduces the effort of defenders greatly in determining where to provide integrity protections.

4.2 Computing System-Wide Mediation

With knowledge of how to compute attack surfaces[1], which identify adversary accessibility, our next goal is to compute the mediation mediators necessary to resolve all information flow errors in a cloud computation. Researchers have found that it is possible to find resolution to an information flow errors by computing a solution to a graph-cut problem [19, 20, 38]. In this case, the graph-cut solution corresponds to the placement of mediation code necessary to resolve all information flow errors. This solution applies for a two-level integrity lattice (e.g., high and low integrity), but in practice, more than two integrity requirements may be necessary resulting in a *multiway cut problem*, which has been shown to be NP-Hard for directed graphs [12]. However, greedy solutions are generally effective for finding possible mediations.

The problem is how to use this knowledge to configure integrity protections for cloud computations, possibly spanning multiple cloud instances. Using available security policies (i.e., MAC and firewall policies), it is possible to produce an information flow problem, as defined in Definition 01 above. However, often over 100 mediators are required per instance. This seems like to much work for customers, programmers, administrators, or OS distributors.

In a paper to appear in December 2012 [29], we use the insight that pre-configured instances face several threats in any deployment of that instance. As a result, these threats should always be mediated proactively, and the focus should be on the new threats that emerge in the specific customer deployment. A customer deployment may impact the cloud instance in two ways: (1) it may add new remote threats by expanding the network connections to any instance and (2) it may add local threats through the introduction of unprivileged code and unverified data to the instance. Thus, we solve information flow problems that identify the program entrypoints necessary to protect the instance integrity by default, from remote threats given the deployment, and from local threats given the deployment.

[1] In particular, how to compute which program entrypoints access which object labels in the MAC policy. This computation does not use the same threat model as the attack surface calculation, but rather adversaries are based on the information flow problem lattice in Definition 01.

For an Apache web server cloud instance, no more than 217 mediators are necessary to protect all the subjects in a default network configuration. These should be defended by default, otherwise all uses of this OS distribution would be flawed. In practice, cloud vendors and OS distributors should work together to ensure proactive integrity protection for these entrypoints.

When a web application is deployed on this server, 24 additional entrypoints must mediate remote threats, 10 of which are specific to the new application code. Thus, to protect the integrity of the customer's instance from new remote threats caused by the deployment, the customer must determine whether the additional program entrypoints accessible to adversaries are protected. In this case, the customer must check their own code for mediation as well as obtain insight into mediation for 14 new attack surface entrypoints in existing OS distribution code. At present, no automated approach exists to resolve these issues, but hopefully, the identification of attack surfaces will motivate such methods.

In addition, we also examine attack surfaces that may result from local threats. In this experiment, we identify local threats as the untrusted object labels relative to the web server process that are not reachable from remote adversary input. If each of these object labels do indeed include malicious data, then 56 additional mediators are necessary to protect the web server and trusted computing base processes. This insight is not altogether surprising, as local exploits are a common vector for launching attacks. In this case, it behooves the customer to ensure that any data assigned to these untrusted object labels is vetted prior to deploying the cloud instance.

Finally, our method also computes the mediation required for a cloud computation consisting of multiple cloud instances. We configured a web application that included a database as well as two distinctly-configured web clients. In addition, we also evaluated the mediation required in the node controller hosts of the cloud instances[2], resulting five VMs total. An advantage is that the same mediation requirement may be present across the system at large, where 2/3 of the attack surface entrypoints appeared in multiple VMs. As a result, 525 mediators total are needed for the combination of VMs by default. Redundant mediation also helped limit the impact of dealing with remote threats, as only three new attack surface entrypoints required mediation once the customer deployment was configured. Local threats seem to be more dependent on the application being hosted on the VMs, however. The new attack surface resulting from local threats on web clients had little overlap with that of the web server, meaning that addressing local threats will be an important challenge in the future.

4.3 Approximating Clark-Wilson Integrity

Once mediation locations have been identified, enforcement code that implements that mediation is necessary. Enforcement code may be added to the operating system or the program to mediate attack surfaces. We describe the two cases and their impact relative to achieving Clark-Wilson integrity, a classical integrity model.

In the first case, the operating system provides access control to limit access to processes, but as we have shown that available access control does not prevent attack surfaces from appearing. The operating system can do two additional things to limit attack surfaces. First, the operating system can limit the entrypoints that processes can use to access resources controlled by adversaries. That is, operating systems can prevent processes from expanding their attack surface beyond what is known. Second, when a process requests a resource from the operating system by name, the operating system can limit the adversaries' abilities to redirect that name resolution. For example, when a process requests a resource by pathname, if an adversary can modify a directory used in name resolution then an adversary may redirect the victim to a resource of the adversary's choosing. In a recent study, we found that many latent vulnerabilities to this threat exist in

[2] In this case, we used a Xen-based system.

programs, even mature ones [58]. To control access to the program entrypoints, we have built a *process firewall* mechanism that can enforce rules to protect entrypoints from adversary access in name resolution and to the returned resource [57].

In the second case, a process entrypoint expects to receive some untrusted input. In this case, it is up to the program to protect itself from such inputs. In recent work, a variety of mechanisms have been explored to enforce type safety [32, 11], enforce execution semantics [2, 8], enforce information flow [31], and use capabilities to control interaction with adversaries [22, 21, 47]. At present, program defenses of attack surfaces are ad hoc in practice, but methods such as these and others may greatly limit the choices that adversaries can make that may impact process integrity.

Protecting program entrypoints from adversaries is an approximation of Clark-Wilson integrity [10]. In this approximation, the Clark-Wilson requirement that all programs that process high integrity data be certified (i.e., formally-assured) is dropped (rule C2), so the Clark-Wilson requirement that all untrusted inputs to such processes must be discarded or upgraded (C5) is strengthened to only allow untrusted inputs to mediating entrypoints. That is, the set of mediating entrypoints must be a superset of the attack surface entrypoints. That is the ultimate goal of this work.

5 Monitoring Computation Integrity

In this section, we describe a method for monitoring the runtime integrity of cloud computations. Because we have carefully evaluated that the security configuration provides the required integrity protection, as described in the previous section, the goal is to monitor that the cloud computation is enforcing that configuration. Monitoring a configuration is much more efficient than monitoring memory in general, as configurations change infrequently, if at all.

5.1 Measuring Integrity

Validating trust that a remote system will behave as expected is a fundamental problem in computer security. The introduction of trusted computing hardware, in particular the Trusted Platform Module [54] (TPM), has been one of the most significant advances to solve this problem in recent years. TPM devices provide protected storage for collecting measurements of host integrity (integrity measurements) and cryptographic mechanisms suitable for constructing proofs from integrity measurements that can be verified by remote parties (attestations). Despite the broad availability of TPMs, in over 200 million hosts, and several trusted computing mechanisms [49, 51, 50, 39, 25] hardware-based trusted computing has not yet led to the widespread adoption of mechanisms to validate trust in commodity systems. We find that such usage has been limited by: (1) the difficulty in predicting what "integrity" means for complex commodity system and (2) the lack of support for runtime integrity measurement. In this research thrust, our goals have been to design new integrity measurement mechanisms based on the TPM hardware to support practical integrity measurement for commodity systems and apply these techniques to different kinds of system deployments.

We have developed a number of integrity measurement designs over the years to capture more events that may impact integrity and ease the verifiers' task [52, 27, 44], and our most recent work unifies these ideas [46]. The key concept in this paper is the *integrity verification proxy* (IVP), an integrity monitor framework that verifies system integrity at the proving system on behalf of the remote clients (e.g., cloud customers). The IVP is a service resident in a virtual machine (VM) host that monitors the integrity of its hosted VMs for the duration of their execution through a combination of load-time and VM introspection mechanisms. Client connections to the monitored VM are proxied through IVP and are maintained so long as the VM satisfies client-supplied integrity criteria. Runtime VM introspection can be costly, but we show that

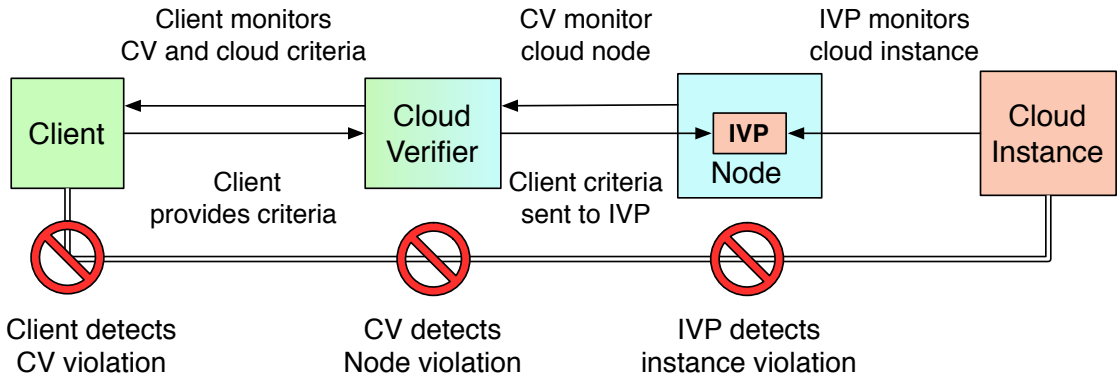


Fig. 2. Integrity Measurement Protocol

focusing runtime monitoring on security policy enforcement can approximate Clark-Wilson integrity for low monitoring overhead (less than 2% on the benchmarks we ran).

To verify the IVP platform's integrity, we use the Root of Trust for Installation (ROTI) approach to attest the filesystem of hosts that are modified infrequently [52]. At install time, a TPM-signed proof is generated that binds the installed filesystem to the installer and system image that produced it. Since host VMs modify few files at runtime, this proof will be useful across boot cycles. The ROTI method has been adapted for network installation, such as is done in the cloud [43]. In network installation, the phase of gathering the installer and system image may be under the control of an adversary, but the network-based ROTI mechanism produces of proof of exactly the inputs used to create the filesystem enabling remote clients to verify VM hosts upon which IVPs run.

5.2 Monitoring Cloud Integrity

Figure 2 shows the architecture of our integrity measurement mechanism for cloud computing. There are three key concepts. First, customers can provide *integrity criteria* dictating what the integrity requirements that must be satisfied in order to create a secure communication channel to the cloud instance. This enables the customer to dictate the terms of integrity to guide measurement, which is done using both traditional load-time and run-time (e.g., based on VM introspection) techniques. Second, the measurement framework use the IVP to track the integrity of the cloud instance and enforces customers' integrity criteria. We show elsewhere that it is possible to deploy IVPs on the cloud nodes to enforce a variety of criteria, including those based on CW-Lite [46]. Third, a *cloud verifier* provides the layer of indirection to enable the customers to use IVPs even though they may not know where their instances are deployed.

Figure 3 illustrates a simplified view of the changes to the basic OpenStack infrastructure to implement this approach. OpenStack is composed of various 'projects' that add additional services, but we will fo-

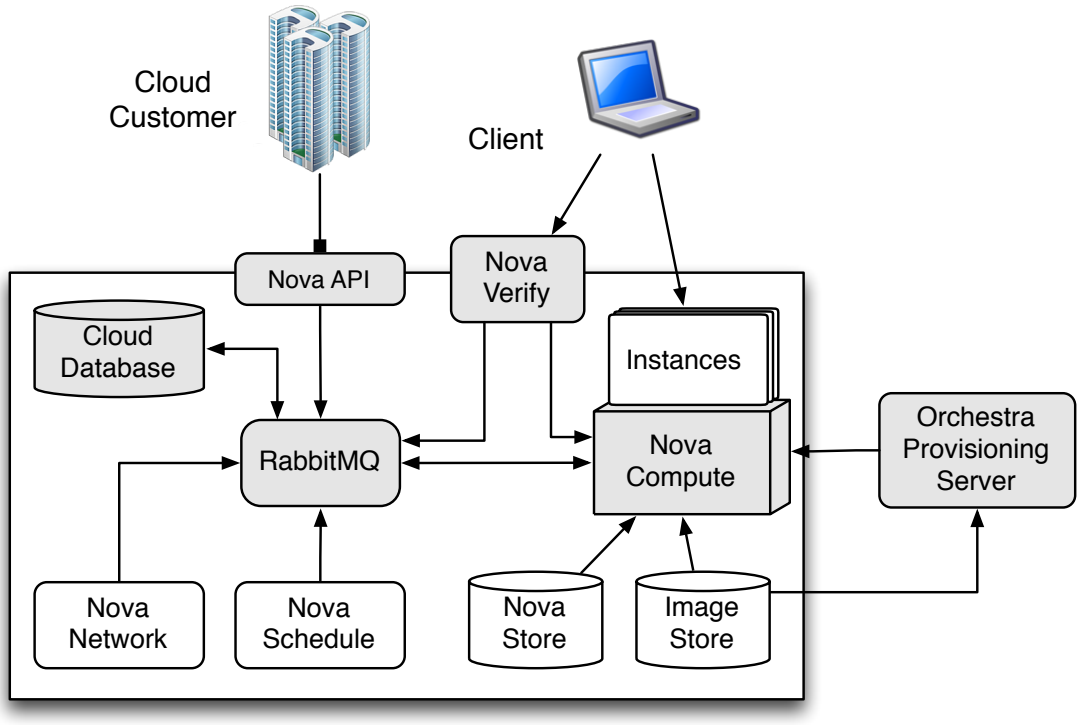


Fig. 3. Changes to OpenStack

cus on the `nova` project for our implementation because it provides the necessary components for hosting instances. The components are largely the same as those presented in the general IaaS architecture in Section 2, but some names have been replaced with their `nova` equivalent. The shaded components represent the changes we have made to implement the CV framework. First, the `nova-api` interface has been extended with a new commands to specify client criteria. Second, the cloud database has been updated to store node identity keys. Third, the `nova-compute` component (compute node) that hosts cloud instances was extended with our IVP implementation. We also modified the RabbitMQ message queue service to deny all messages except over an SSL channel authenticated by a CV signed certificate. This implements our requirement that only verified components can communicate through the RabbitMQ and thus participate in the cloud. Finally, the `nova-verify` service has been added as a standalone component to implement the CV. `nova-verify` is another public facing service in addition to the `nova-api` service. Overall, the additional code added to OpenStack was roughly 2,600 SLOC in Python. Additional code for implementing modules and measurement interfaces was under 1,000 SLOC of Python.

We evaluated this design on an OpenStack version 2011.3-nova-milestone cloud installed on three Dell M620 blades. These machines have two quad core Xeon processors with 64GB of RAM and two 1Gb network cards for the private and public network. The first blade serves as the compute node that hosts virtual machines. The second blade serves as the Cloud Verifier as well as other necessary controlling components of the cloud like scheduler, API server and network controller. The last blade is used to simulate clients of

Operation	Mean
CV Verification	1.09
TPM Quote	0.84
Communication	0.17
Others (read, write file etc.)	0.08
Node Join Protocol	1.68
TPM Quote	0.82
OpenSSL Node Key Generation	0.29
Node Certificate Generation	0.22
Communication	0.23
Others (read, write file etc.)	0.12
Client Criteria Registration	0.56
Openstack Processing	0.30
Instance Certificate Generation	0.22
Certificate Verification	0.04

Table 1. Protocol time delay breakdown. Times are in seconds and are averages of 30 runs.

the cloud and performs the benchmark programs. Each system runs Ubuntu-11.10 (x86_64) with a Linux 3.0 kernel for hosts and Ubuntu-11.10 (x86_64) with kernel 3.0.1 for virtual machines.

Table 1 shows the breakdown of three major protocols in our CV framework. These numbers are the average of 30 runs of the protocol. The first is the time due to verifying the `nova-verify` service, which the client must do before using the cloud. The second is the compute node's cloud join protocol. In both cases, the majority of the overhead comes from the TPM quote operation. Despite this, the delay is less than 2 seconds. For the node join protocol, this is a significantly shorter time than the boot process the node must go through and is only a one-time cost per boot. For the CV verification, we envision techniques like Asynchronous Attestation [27] can be used to to reduce this delay since it will be a major bottleneck for potentially thousands of clients connecting to the cloud. Finally, the client registration operation has a negligible overhead compared to the typical delay incurred by using the API server in general. It is worth noting that the total time will depend on the modules that must be checked for the client's criteria. This represents the minimum time only.

We benchmarked the G-WAN (G-WAN 3.3.28 64-bit) web server using ab (Apache Benchmark). We made 30 runs of the benchmark on the [1-1000] concurrency range. As shown Figure 4, the G-WAN web server running on the IVP framework performs at an average overhead of 3.1% compared to the G-WAN web server normally. We have found that this overhead is primarily due to VM introspection with `gdb` [46]. Since `gdb` uses the `ptrace` interface in the kernel to monitor processes for debug signals, every syscall incurs a small processing overhead by `gdb` to parse the signal and resume process execution. A possible solution for this would be to modify the `ptrace` interface to notify the `gdb` process only when debug signals are raised. This is future work.

6 Conclusions

In this paper, we describe a method and supporting tools for configuring cloud computations to protect their integrity proactively. Integrity protection is defined as an information flow problem, where any adversary access to an authorized operation has the potential to exploit a vulnerability. As security policies are too

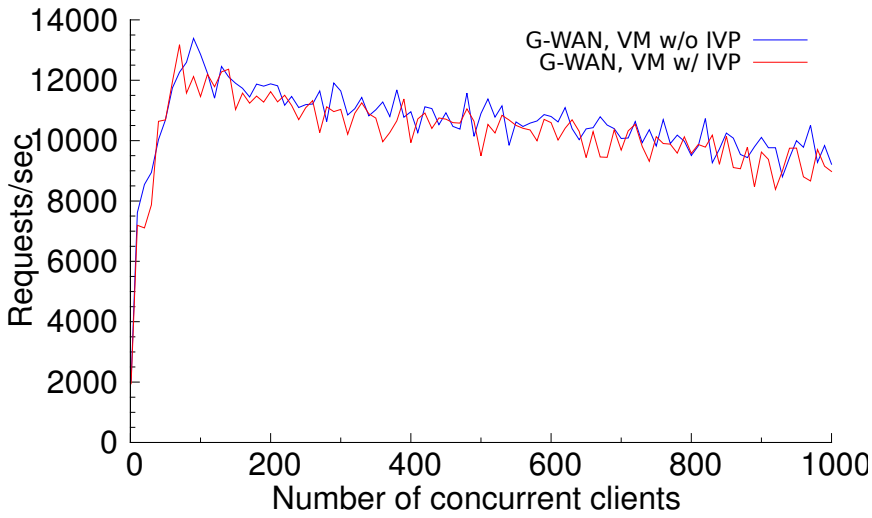


Fig. 4. G-WAN Server Performance

complex to evaluate manually, we describe a runtime analysis method to compute adversary accessibility to individual program entrypoints, a method that can compute the mediation necessary to resolve information flow errors system-wide given such adversary access, and defensive mechanisms to enforce that mediation approximating Clark-Wilson integrity. Using the resultant configuration, we define an integrity monitoring mechanism that uses trusted computing to enable cloud customers to ensure that their compute instances satisfy their integrity requirements. We demonstrate that this monitoring mechanism can be applied to monitor instances running in the OpenStack cloud platform for low overhead. In the future, we plan to extend our work in runtime vulnerability testing [58] to find and fix vulnerabilities in mediation.

References

1. Rackspace Cloud Servers, http://www.rackspace.com/cloud/
2. Abadi, M., Budiu, M., Erlingsson, U., Ligatti, J.: Control-flow integrity: Principles, implementations and applications. In: Proceedings of the 12th ACM Conference on Computer and Communications Security (November 2005)
3. Amazon Simple Storage Service(Amazon S3). https://drive.google.com/
4. Amazon EC2, http://aws.amazon.com/ec2
5. Azab, A.M., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.C.: HyperSentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity. In: Proc. 17th ACM Conference on Computer and Communications Security (2010), http://doi.acm.org/10.1145/1866307.1866313
6. Biba, K.J.: Integrity Considerations for Secure Computer Systems. Tech. Rep. MTR-3153, MITRE (April 1977)
7. Bleikertz, S., Groß, T., Schunter, M., Eriksson, K.: Automated information flow analysis of virtualized infrastructures. In: Proceedings of the 2011 European Symposium on Research in Computer Security. pp. 392–415 (2011)
8. Castro, M., Costa, M., Harris, T.L.: Securing software by enforcing data-flow integrity. In: OSDI '06: Proceedings of the 7th USENIX Symposium on Operating System Design and Implementation. pp. 147–160 (2006)
9. Chen, H., Li, N., Mao, Z.: Analyzing and Comparing the Protection Quality of Security Enhanced Operating Systems. In: NDSS (2009)
10. Clark, D.D., Wilson, D.: A Comparison of Military and Commercial Security Policies. In: IEEE Symposium on Security and Privacy (1987)

11. Dhurjati, D., Kowshik, S., Adve, V.: Safecode: enforcing alias analysis for weakly typed languages. In: PLDI '06: Proceedings of the 2006 ACM SIGPLAN Conference on Programming Language Design and Implementation. pp. 144–157. ACM, New York, NY, USA (2006)
12. Garg, N., Vazirani, V.V., Yannakakais, M.: Multiway cuts in directed and node weighted graphs. In: in Proc. 21st ICALP, Lecture Notes in Computer Science 820. pp. 487–498. Springer-Verlag (1994)
13. Google App Engine. https://developers.google.com/appengine/
14. Google Docs. https://drive.google.com/
15. Howard, M., Pincus, J., Wing, J.: Measuring Relative Attack Surfaces. In: Proceedings of Workshop on Advanced Developments in Software and Systems Security (2003)
16. Jaeger, T., Sailer, R., Shankar, U.: PRIMA: Policy-Reduced Integrity Measurement Architecture. In: Proc. 11th ACM SACMAT (2006)
17. Jaeger, T., Sailer, R., Zhang, X.: Analyzing integrity protection in the SELinux example policy. In: USENIX Security Symposium (Aug 2003)
18. Jaeger, T., Schiffman, J.: Cloudy with a chance of security challenges and improvements. IEEE Security & Privacy (Jan/Feb 2010)
19. King, D., Jha, S., Jaeger, T., Jha, S., Seshia, S.A.: Towards Automated Security Mediation Placement. Tech. Rep. NAS-TR-0100-2008, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA (November 2008)
20. King, D., Jha, S., Muthukumaran, D., Jaeger, T., Jha, S., Seshia, S.A.: Automating security mediation placement. In: ESOP (2010)
21. Krohn, M.N., Yip, A., Brodsky, M., Cliffer, N., Kaashoek, M.F., Kohler, E., Morris, R.: Information flow control for standard OS abstractions. In: Proceedings of the 21st ACM Symposium on Operating Systems Principles. pp. 321–334 (Oct 2007)
22. Levy, H.M.: Capability-Based Computer Systems. Butterworth-Heinemann (1984)
23. Manadhata, P., Tan, K., Maxion, R., Wing, J.M.: An Approach to Measuring A System's Attack Surface. Tech. Rep. CMU-CS-07-146, School of Computer Science, Carnegie Mellon University (2007)
24. Mannan, M., Kim, B.H., Ganjali, A., Lie, D.: Unicorn: Two-factor attestation for data security. In: 18th ACM Conference on Computer and Communications Security (2011)
25. McCune, J.M., Parno, B.J., Perrig, A., Reiter, M.K., Isozaki, H.: Flicker: An Execution Infrastructure for TCB Minimization. In: Proc. 3rd ACM SIGOPS/EuroSys (2008)
26. Microsoft Azure. http://www.windowsazure.com/en-us/
27. Moyer, T., Butler, K., Schiffman, J., McDaniel, P., Jaeger, T.: Scalable asynchronous web content attestation. In: Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC '09). ACSA (2009)
28. MSDN: Mandatory Integrity Control (Windows). http://msdn.microsoft.com/en-us/library/bb648648%28VS.85%29.aspx
29. Muthukumaran, D., Rueda, S., Talele, N., Vijayakumar, H., Teutsch, J., Jaeger, T., Edwards, N.: Transforming commodity security policies to enforce clark-wilson integrity. In: Proceedings of the 2012 Annual Computer Security Applications Conference (Dec 2012)
30. Myers, A.C., Liskov, B.: A decentralized model for information flow control. ACM Operating Systems Review 31(5), 129–142 (Oct 1997), http://www.cs.cornell.edu/andru/papers/iflow-sosp97/paper.html
31. Myers, A.C., Zheng, L., Zdancewic, S., Chong, S., Nystrom, N.: Jif: Java information flow. http://www.cs.cornell.edu/jif, July2001-2003
32. Necula, G.C., McPeak, S., Weimer, W.: CCured: Type-safe retrofitting of legacy code. In: Proceedings of the ACM Conference on the Principles of Programming Languages (January 2002)
33. NIST Definition of Cloud Computing. http://csrc.nist.gov/groups/SNS/cloud-computing/
34. Noel, S., Jajodia, S., O'Berry, B., Jacobs, M.: Efficient minimum-cost network hardening via exploit dependency graphs. In: ACSAC (2003)
35. Novell: AppArmor Linux Application Security. http://www.novell.com/linux/security/apparmor/, http://www.novell.com/linux/security/apparmor/
36. Security-enhanced linux, http://www.nsa.gov/selinux

37. Ou, X., Boyer, W.F., McQueen, M.A.: A scalable approach to attack graph generation. In: CCS (2006)
38. Pike, L.: Post-hoc separation policy analysis with graph algorithms. In: Workshop on Foundations of Computer Security (FCS'09). Affiliated with Logic in Computer Science (LICS) (August 2009)
39. Sailer, R., Zhang, X., Jaeger, T., van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: USENIX Security Symposium (2004)
40. Salesforce. http://www.salesforce.com/
41. Saltzer, J.H., Schroeder, M.D.: The protection of information in computer systems. Proceedings of the IEEE 63(9) (Sep 1975)
42. Sarna-Starosta, B., Stoller, S.D.: Policy analysis for security-enhanced linux. In: WITS (April 2004)
43. Schiffman, J., Moyer, T., Jaeger, T., McDaniel, P.: Network-based root of trust for installation. IEEE Security & Privacy (Jan/Feb 2011)
44. Schiffman, J., Moyer, T., Shal, C., Jaeger, T., McDaniel, P.: Justifying integrity using a Virtual Machine Verifier. In: Proceedings of the 25th Annual Computer Security Applications Conference (ACSAC '09). ACSA (2009)
45. Schiffman, J., Sun, Y., Vijayakumar, H., Jaeger, T.: Cloud verifier: Verifiable auditing service for iaas clouds (Sep 2012)
46. Schiffman, J., Vijayakumar, H., Jaeger, T.: Verifying system integrity by proxy. In: 5th International Conference on Trust and Trustworthy Computing. pp. 179–201 (2012)
47. Shankar, U., Jaeger, T., Sailer, R.: Toward Automated Information-Flow Integrity Verification for Security-Critical Applications. In: Proceedings of the 2006 ISOC Networked and Distributed Systems Security Symposium (February 2006)
48. Sheyner, O., Haines, J., Jha, S., Lippmann, R., Wing, J.M.: Automated generation and analysis of attack graphs. In: Proceedings of the 2002 IEEE Symposium on Security and Privacy. p. 273. IEEE Computer Society, Washington, DC, USA (2002)
49. Shi, E., Perrig, A., van Doorn, L.: BIND: A Fine-Grained Attestation Service for Secure Distributed Systems. In: IEEE SP '05 (2005)
50. Shrivastava, S.: Satem: Trusted Service Code Execution across Transactions. In: SRDS '06 (2006)
51. Smith, S.W.: Outbound authentication for programmable secure coprocessors. In: In 7th European Symposium on Research in Computer Science (2002)
52. St. Clair, L., Schiffman, J., Jaeger, T., McDaniel, P.: Establishing and sustaining system integrity via root of trust installation. In: Proceedings of the 2007 Annual Computer Security Applications Conference. pp. 19–29 (Dec 2007)
53. Sun Microsystems: Trusted solaris operating environment - a technical overview, http://www.sun.com
54. TCG: Trusted Platform Module. https://www.trustedcomputinggroup.org/specs/TPM/ (2005)
55. Vijayakumar, H., Schiffman, J., Jaeger, T.: A Rose by Any Other Name or an Insane Root? Adventures in Name Resolution. In: Proc. of 7th European Conference on Computer Network Defense (EC2ND). Gothenburg, Sweden (Sep 2011)
56. Vijayakumar, H., Schiffman, J., Jaeger, T.: Integrity walls: Finding attack surfaces from mandatory access control policies. In: 7th ACM Symposium on Information, Computer, and Communications Security (ASIACCS) (May 2012)
57. Vijayakumar, H., Schiffman, J., Jaeger, T.: Process Firewalls: Enforcing Safe Resource Access with Attack-Specific Invariants. Tech. Rep. NAS-TR-0153-2012, Network and Security Research Center, Department of Computer Science and Engineering, Pennsylvania State University, University Park, PA, USA (Jan 2012)
58. Vijayakumar, H., Schiffman, J., Jaeger, T.: STING: Finding name resolution vulnerabilities in programs. In: 21st USENIX Security Symposium (2012)
59. Watson, R.N.M.: TrustedBSD: Adding trusted operating system features to FreeBSD. In: Proceedings of the FREENIX Track: 2001 USENIX Annual Technical Conference. pp. 15–28 (2001)

21 novembre 2012

Self-Defending Clouds: Myth and Realities*

Marc Lacoste[1], Aurélien Wailly[1], and Hervé Debar[2]

[1] Orange Labs, Security and Trusted Transactions Dept.
38-40, rue du Général Leclerc, 92794 Issy-Les-Moulineaux, France
{marc.lacoste,aurelien.wailly}@orange.com
[2] Télécom SudParis, CNRS Samovar UMR 5157
9, rue Charles Fourier, 91011 Evry, France
herve.debar@telecom-sudparis.eu

Abstract. Security is a growing concern as it remains the last barrier to widespread adoption of cloud environments. However, is today's cloud security Lucy in the Sky with Diamonds? Expected to be strong, flexible, efficient, and simple? But surprisingly, being neither? A new approach, making clouds *self-defending*, has been heralded as a possible element of answer to the cloud protection challenge. This paper presents an overview of today's state and advances in the field of cloud infrastructure self-defense. Four key self-protection principles are identified for IaaS self-protection to be effective. For each layer, mechanisms actually deployed to deliver security are analyzed to see how well they fulfill those principles. The main remaining research challenges are also discussed to yield truly mature self-defending clouds.

Keywords: Cloud Security Supervision; Cloud Security Management; Self-Defending Clouds; Cloud Threats; Autonomic Security; IaaS Infrastructures.

1 What's Wrong with Today's Cloud Security?

Security is undoubtedly the # 1 barrier for cloud technology adoption, as its very nature raises multiple concerns regarding protection of computing, networking, and storage resources. What about it?

Ideally, cloud security should be *strong*, *flexible*, *efficient*, and *simple* (Figure 1):

- *Strong security* is required to face new threats introduced by virtualization, or by resource sharing, as clouds are by essence multi-tenant environments. The crux is to achieve strict isolation between Virtual Machines (VMs), which may fail if the virtualization layer is compromised. To guarantee perimetric security, dissolved organization boundaries also make it more difficult to define the "inside" and the "outside" of an infrastructure.
- *Flexible security* is needed to respond to the multiplicity of threats and to their evolution. This means security resource provisioning should adapt to changes in the cloud environment to provide an optimal level of protection. Flexibility is also a must considering the diversity of stakeholders using cloud services, each with their security objectives, processes, and technical components.

* This work was supported by the OpenCloudWare project, funded by the French Fonds national pour la Société Numérique (FSN).

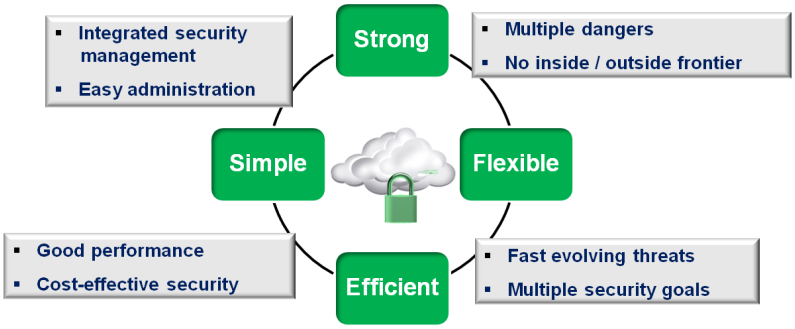


Fig. 1. The Myth – Cloud Security Expectations.

- *Efficient security* is desirable to guarantee security SLAs, e.g., to keep end-to-end incident response times short. Managing vulnerabilities, detecting intrusions, and activating defenses should be performed smoothly, transparently, and in a way which is both highly scalable and non-intrusive regarding performance.
- Above all, *simple security* is essential to keep infrastructure administration within manageable bounds. This requires to abstract away heterogeneity of security building blocks protecting systems, networks, and data to provide an integrated view of cloud defense.

But unfortunately, the reality of today's cloud security is a bit different. Protection appears neither as *so strong*, *so flexible*, *so efficient*, nor *so simple* (Figure 2):

- Cloud security is not really as *strong* as expected: vulnerabilities are hard to detect, and counter-measures hard to place. If in theory, mainstream hypervisors have a low surface of attack, in practice, new classes of threats such as malicious device drivers or rootkits in the virtualization layer call for higher levels of assurance.
- Similarly, security is neither very *flexible* nor *efficient*: many security configurations are still static, policies being often hardcoded within the protection mechanisms. Dynamic, reactive protection strategies are thus hard to set up, as policies must be configured and updated manually – a costly and error-prone process. A generic security supervision architecture is also lacking for the cloud.
- Finally, cloud security is definitely *not simple*: available security components are highly fragmented. How to orchestrate them to guarantee end-to-end security is still undefined, with as main barriers heterogeneity, scalability and interoperability. The problem also turns into a maintenance nightmare for a nebula of different administrators who are not even aware of the existence of one another. Traditional approaches to protection are thus clearly not enough!

Self-protection has recently raised growing interest as possible element of answer to the cloud computing infrastructure protection challenge. Research initiated by IBM on *autonomic security architectures* [7] has the ambition to build infrastructures where security is self-managed: the idea is that security parameters are autonomously negotiated with the environment to match the ambient estimated risks to provide an optimal level of protection. For cloud computing, the result is a *self-defending cloud* infrastructure.

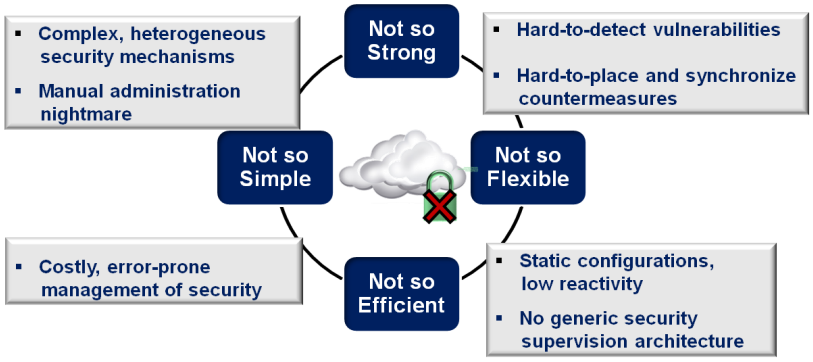


Fig. 2. Reality – Current Cloud Security.

Faced with multiple threats and heterogeneous defense mechanisms, the autonomic approach appears tempting by proposing simpler, stronger, and more efficient cloud security management. Quite a few projects [1, 9, 19, 23] have already investigated the design, implementation, and deployment of self-protection architectures and mechanisms regarding different aspects of infrastructure security. Prospects look rather good, as fully automated security supervision of cloud and inter-cloud infrastructures could be at hand! At the same time, policy-driven security automation still remains at a very early stage for the cloud. First attempts seem to fall at the last hurdle as they overlook several key cloud features.

The objective of this paper is to clarify where we currently stand regarding self-defense of cloud infrastructures. After providing some background on self-protection and its benefits, we identify four *fundamental principles* that a IaaS infrastructure should satisfy for self-protection to be effective: (1) flexible security policies; (2) cross-layered defense; (3) multiple loops; and (4) open security architecture.

We analyze how well current cloud security mechanisms fulfill those principles for each infrastructure layer. We also identify the main *remaining challenges* to solve to yield truly mature self-defending cloud infrastructures.

The rest of this paper is organized as follows. After a survey of threats to sensitive assets in a IaaS infrastructure (Section 2), we recall the approach of cloud self-defense and its benefits (Section 3). We then present the self-protection principles and their coverage by existing cloud security mechanisms (Section 4). Finally, we discuss remaining research challenges and perspectives (Section 5).

2 Threats in a IaaS Infrastructure

A typical IaaS infrastructure is generally organized in three main layers (Figure 3):

– The *physical layer* consists of computing (CPU, memory), networking, and storage resources spread over the cloud infrastructure. Those hardware resources are virtualized and shared among virtual machines.

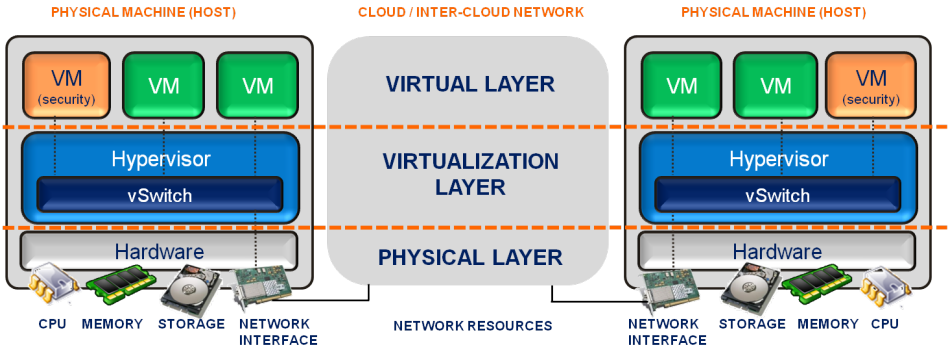


Fig. 3. Elements of a IaaS Infrastructure.

- The *virtualization layer* enables to deploy and run concurrently multiple instances of a software stack (applications, middleware, and OS) on a same physical host in the form of VMs. This task is performed by a specific component, the *hypervisor*, which manages the allocation of physical machine resources among VM instances. It notably guarantees strict isolation between VMs. The hypervisor also enables communications between VMs, either locally on the same host, or through the cloud network through a low-level software bus, often called *virtual switch*.
- The *virtual layer* is composed of the VMs running over the cloud infrastructure. From a security viewpoint, two broad types of VMs may be distinguished: (1) *user VMs* are the customer VMs to be protected. They typically contain customer code and data which may be security-sensitive; (2) *security VMs* are security services (firewalls, intrusion detection systems, anti-malware tools...) running as virtual machines in order to protect the user VMs. This class of VMs is often referred to as *virtual security appliances*.

Some of the main threats against a IaaS infrastructure are described next.

VM-to-VM Threats. A malicious VM may attempt to fool the IaaS VM placement strategy to run on the same physical machine as the attack target (another VM). It may then take advantage of a flaw in hypervisor isolation to launch a side-channel attack to steal or corrupt information from the target VM.

Hypervisor Subversion. More potent attacks attempt to take control of the hypervisor itself from a malicious VM (*hyperjacking*). Such attacks undermine the commonly established idea that commodity hypervisors have a relatively low surface of attack. A VM is able to "escape" from hypervisor isolation enforcement to take full control of the virtualization layer. Possible attack vectors include misconfigurations, or malicious or poorly confined device drivers in the hypervisor. Possible subsequent steps include compromising hypervisor integrity, installing rootkits, or launching an attack against another VM, resulting in breaches in confidentiality, integrity, or availability (e.g., Denial of Service). In the past few years, such isolation breakout attacks have been published for nearly all main hypervisors.

Network Threats. Traditional network security threats such as *traffic snooping* (i.e., intercepting network traffic), *address spoofing* (i.e., forging VM MAC or IP addresses), or *VLAN hopping* (i.e., breaking out of traffic segregation) are also possible, either between physical hosts, or between VMs on the same host.

Availability Threats. This is also a major issue due to resource sharing. Faulty or malicious VM behaviour may lead to resource starvation and interruption of service. For instance, the cloud platform may be brought to a halt, hosts running out of memory due to greedy VM behaviors. Such events may greatly alter the cloud provider image. Crimeware-as-a-Service scenarios may be even worse: the large amount of cloud resources are then deliberately used by hackers to launch massive attacks against juicy targets, for instance using botnets or other malwares.

Information Security Threats. This class of threat such as violations of confidentiality or integrity should also be taken into account for stored data. Few techniques are available to address those threats specifically, apart from traditional cryptographic counter-measures (encryption, signature, authentication).

3 Self-Defending Clouds

3.1 Approach: Automation of Security Management

To address most of the previous threats, it is necessary to detect and prevent intrusions in a way which is as automated as possible. Indeed, cloud infrastructures provide dynamic and flexible services by federating highly heterogeneous resources, hiding underlying complexity. By nature, such infrastructures involve a significant number of physical resources, further growth requiring new management capacities. Unfortunately, current administration solutions, involving manual intervention, do not scale to such complexity. Moreover, the absence of central knowledge may result in unintended conflicts, wasting administrator time. *Automated management of cloud security* is thus decisive for infrastructure stability, protection, as well as for cost-effectiveness.

This is precisely the aim of the approach put forward by IBM regarding *autonomic management of security*, where a system becomes self-protected to automate response to incidents, and react rapidly to detected attacks [7]. Autonomic security applies the idea of flexibility to the security space itself. It goes a step further than simple adaptation by automating the entire process of reconfiguration, thus making the security mechanisms self-responsive, almost running without any user intervention[3]. From a design perspective, this introduces a control structure which oversees the different aspects of the reconfiguration activity – sense, analyze and respond. A feedback loop is set up to select the adequate security policies matching the ambient estimated risk, and achieve an optimal level of protection of system resources.

[3] The system is running "almost" without user intervention, as the user is still part of the loop: users notably specify the security adaptation strategy defining the response to changes in the security context. The user can also be notified, or prompted to take a decision, in case unexpected events occur that the system cannot manage autonomously. Thus, security management is viewed as a closed loop, but in which the user has still an active role to play.

This approach is related to *Intrusion Detection and Prevention Systems (IDPS)*, whose goal is also to monitor a system, analyzing its behavior to detect attacks, and possibly react to them [10]. Self-protection has been viewed as a promising next step of such well-established techniques to fight intrusions [11]. In addition to yielding stronger security management of infrastructures, automated capabilities of detection of and reaction to attacks also bring clear benefits such as lighter administration, lower incident response times, or reduced error-rates.

3.2 The Autonomic Security Loop

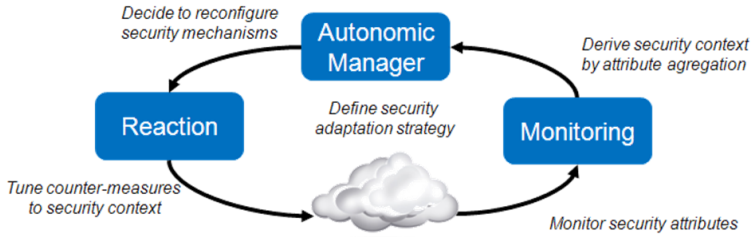


Fig. 4. A Typical Self-Protection Loop.

Operation of a self-protecting system includes the following phases (Figure 4):

- *Detection*: information regarding monitored system activity is collected through sensors, and correlated to derive an overall picture of the risk level. Alerts are raised whenever abnormal activity or known attack patterns are detected.
- *Decision*: alerts are analyzed and checked against the system security policy. In case of policy violation, launching of counter-measures in the infrastructure may be triggered during this phase.
- *Reaction*: counter-measures selected during the decision phase are activated to mitigate the detected attack.

More precisely, an autonomic security loop can be instantiated using a small number of components:

- *Security Context Provider*: this component provides a high-level generic description of the current context. Low-level input data are gathered from different sources (e.g., system/network monitoring components) and aggregated into a higher-level generic representation of the current context – e.g., through a context management infrastructure. The Security Context Provider also provides a description of the ambient security context. A set of security-related attributes (e.g., security levels) are extracted from the generic information provided by the context management infrastructure – e.g., through an inference engine.
- *Decision-Making Component*: based on the security context, this component decides whether or not to reconfigure the security infrastructure, for instance to relax the strength of authentication, or change cryptographic key lengths.

– *Adaptable Security Mechanisms*: The decision is then transmitted to the security mechanism which should be adapted. Reconfiguration is performed by changing the needed component. Security mechanisms should be flexible enough to be reconfigured, by tuning security configuration parameters, or by replacing them with other components offering similar security services. Specific support mechanisms are needed to guarantee that the reconfiguration process is safe and secure.

The autonomic loop is then set up, with monitoring, decision, and action steps. At this point, the system is able to negotiate the security parameters autonomously with its environment, fulfilling the vision of a *self-protecting system*.

3.3 Benefits

The autonomic paradigm applied to manage the security of IaaS infrastructures yields *self-defending clouds*. This approach enables to lighten administrator workload, and offers increased reactivity in an environment where threats are constantly evolving. This results in stronger, more flexible, more efficient, and simpler cloud security.

With such an approach, a cloud infrastructure provider can leverage a self-defense framework to secure virtual machines simply, quickly, dynamically, and automatically while benefiting from increased cost effectiveness, thus making it a key element of a differentiation strategy. Key benefits are lighter administration of the infrastructure, increased reactivity and agility, and lower operating costs. The cloud provider also benefits from an automated solution to set up a "graduated response" against cloud threats while ensuring effective isolation of virtualized resources. This solution should therefore be directly relevant to achieve strong and flexible security in the corresponding products, offering a clear competitive advantage. Finally, an autonomic security framework could be an enabler for providing integrated security supervision of cloud and multi-cloud infrastructures.

Risk-aware, dynamic VM quarantine is a typical application of the self-defense approach. It may be realized as follows. In each monitored VM, a lightweight agent (similar to an anti-virus) gathers and analyzes all files loaded into memory before they are accessed. A VM in charge of system supervision aggregates and orchestrates all decisions concerning the security of the architecture. Decisions are transmitted to different security elements of the infrastructure to confine infected VMs. Once the threat has been eradicated by the agent, the quarantine may be released. The user recovers his VMs free of viruses immediately, and without loss of data.

4 Reality: Mechanisms

4.1 Design Principles

How far are we today from building and operating such systems? Cloud environments come with a number of distinguishing features that should be considered from a security architecture standpoint. The three main dimensions to keep in mind are:

– *Multi-layering*: a cloud infrastructure is built from many independent software layers with their specific security mechanisms, while attacks may target several layers.
– *Multi-laterality*: a cloud involves multiple organizations with their own security objectives, calling for flexible policies and monitoring granularities for security management.
– *Openness*: clouds are increasingly evolving towards interoperability with other clouds or third-party IT systems, making a closed-world vision of security not adequate.

Those dimensions in turn allow deriving a small set of principles to guide the design of new, self-defending cloud architectures. In any case, it seems reasonable to assume that those design principles may be satified by any such existing systems. Those principles are stated and discussed below.

Principle 1 (Policy-Based Self-Protection) *The architecture should be a refinement of a well-defined security adaptation model based on policies.*

This approach has well-known benefits to increase self-management adaptability and extensibility [29]. Therefore, a richer choice of security strategies can be supported in each phase of the control loop [18]. For instance, a wider range of detection and reaction policies can be defined and enforced.

Principle 2 (Cross-Layer Defense) *Detection and reaction should not be performed within a single software layer (VM, VMM, physical), but may also span several layers.*

This means that events detected in one layer may trigger reactions in other layers. The cross-layer approach by coordinating security mechanisms who are not aware of one another, improves security by helping to capture the overall extent of an attack, often not limited to a single layer, and to respond to it with greater accuracy.

Principle 3 (Multiple Loops) *Several control loops of variable level of granularity should be defined and coordinated.*

A single loop has insufficient flexibility for supervision perimeter and does not allow trade-offs for response optimality: either local and fast, but of variable accuracy, or broader and more relevant due to more knowledge available, but slower [20]. Trade-offs with other concerns than security are also possible.

Principle 4 (Open Architecture) *Multiple detection and reaction strategies and mechanisms should be easily integrated in the architecture, to defend the system against both known and unknown threats.*

The architecture should allow to integrate simply heterogeneous off-the-shelf security components, to support the widest possible array possible of defense mechanisms.

To build a self-protecting cloud, three broad classes of solutions are available:

1. Solutions to *protect the VM*;
2. Architectures to *secure the hypervisor*;
3. And *generic tools for detection and reaction*, independent from those two layers.

We give an overview below of each family of techniques, and how well they satisfy design principles 1–4.

4.2 Protecting VMs

Virtual machine introspection [21] sparked a whole stream of research to use the capabilities of the hypervisor to supervise VM behaviors, such as detecting integrity violations. Different alternatives have been proposed to place the monitoring component: *embedded in the VM*, *in the VMM*, or *in an "out-of-VM" appliance*.

"In-VM" placement benefits from proximity with the monitored resources to increase detection accuracy. However, placing an agent in the VM to verify its code may compromise stealth and transparency: programs co-located within the VM may detect the security component, and possibly alter its behavior.

Appliances greatly contribute to improving security: as the protection mechanism does not share the same execution context as the protection target, the former is less likely to be vulnerable if the latter is under attack. Peformance is generally enhanced for similar reasons. For instance, the well-known "anti-virus storm" phenomenon (which may induce VM resource starvation in case of extensive scans for malicious code) may be avoided if the detection mechanism is not located in the VM to protect.

Alternatively, *hypervisor* mechanisms offer transparent and privileged access to resources of the VM to monitor. For instance, a number of malware detection tools intercept system calls, and compare monitoring information gathered from different layers [16, 17]. Unfortunately, such efforts are mostly focused on detection with almost no (or very simple) remediation policies (e.g., restart, kill a VM) [16]. Some systems [13] based on a trusted VMM allow verification of flexible integrity policies. Overall, the corresponding architectures generally prove difficult to be compatible with legacy anti-malware software. A number of reaction mechanisms have also been proposed, mainly in terms of firewalls [25] or self-recovery mechanisms after an intrusion [14]. But few of them have self-protection loops or flexible security policies.

4.3 Hypervisor Defense

One layer below, a variety of techniques were proposed to protect the VMM from subversion, with special attention to buggy or malicious device drivers which enable kernel exploitation due to poor confinement. *Trusted computing architectures* [2, 26] provide strong VMM code integrity and authenticity guarantees using hardware mechanisms. VMM capabilities may be proven to third parties using attestation protocols, therefore allowing to build trust. Those techniques are limited to detection in the form of integrity measurement, without possibilities of remediation in case of violation. For instance, [6] describes a solution controlling integrity of several components of a IaaS infrastructure, based on the TPM (Trusted Platform Module) [5] and on a secure hypervisor [25].

Sandboxing techniques [12] were also heavily explored to confine malicious code by controlling communications between driver and device, kernel, or user space. Many techniques are based on isolating software faults or intercepting system calls. This class of mechanisms is expected to provide strong security if driver code is confined strictly. Unfortunately, the code of the reference monitor enforcing access control remains difficult to protect without any additional hardware mechanism.

Virtualizing some components of the VMM [30] also strengthens isolation, without requiring modifications to existing code. Partial VMM virtualization increases architecture modularity, going in the same direction as new evolutions regarding "disaggregated" VMM security architectures (coming from micro-kernels and component-based architectures) [8, 28]: simpler VMM integrity verification is achieved through a thin core hypervisor providing strong isolation by design, which can be simply formally proven and audited. Unfortunately, those new architectures are not yet compatible with mainstream hypervisors. However, this could change in the near future [22].

A number of *language-based techniques* [34] have also been proposed to protect VMM control flow by detecting safety violations through type systems. Unfortunately, this is usually limited to programming errors, malicious code being hardly considered.

Overall, solutions remain limited either to pure integrity detection [2, 34] or simple containment [12, 30], proposing no actions to sanitize the kernel. Security policies are also often not well separated from the interception mechanisms themselves, making them hard-to-manage. Security mechanisms usually require extensive code rewriting, making them hard to apply to legacy hypervisors. VMM self-protection is thus still a widely unchartered area.

4.4 Detecting Intrusions and Malware

Finally, an extensive number of generic techniques have been explored to detect and prevent intrusions (IDPS) and fight against malware [3, 33], leveraging virtualization to mitigate both known and unknown attacks [24]. Those systems are usually based on a single control loop, with a few attempts at cross-layering to detect elusive malwares. Their architecture is usually more open to allow selection and composition of several detection algorithms to improve accuracy. However, in most cases, they have been little applied to the cloud.

4.5 Towards a Big Picture

Table 1 gives a coverage estimation of principles 1–4 for the classes of mechanisms described previously. One tends to see that, although the use of policy-based design can still be improved, it is already present in tools to protect VMs and to fight against intrusions. Multi-layer defense remains however for the moment limited to the VM protection arena. Existing mechanisms still offer little flexibility in terms of security supervision granularity or openness – even though IDPS systems have made considerable progress in such directions in recent years. Unsurprisingly, hypervisor defense appears as the area where almost everything remains to be invented to realize self-protection in the virtualization layer, despite the wide range of techniques already explored.

Class of Mechanisms	Principle Satisfied?			
	Principle 1	**Principle 2**	**Principle 3**	**Principle 4**
Protection of VMs	A few	Yes	No	No
Hypervisor Defense	No	A few	No	No
IDPS / Anti-Malware	Yes	No	A few	Yes

Table 1. Principle Coverage by Some Classes of Security Mechanisms.

5 Open Issues and Perspectives

We conclude by discussing a number of research issues that still need to be explored (and solutions found!) to reach truly mature self-defending clouds. The remaining challenges are analyzed according to each step of the autonomic security loop: detection, reaction, and decision-making.

5.1 Detection

Here, "the" hard issue is the *placement of security mechanisms* in the infrastructure. This problem may be addressed *horizontally*, i.e., where in the security architecture (network or host) are protection mechanisms most relevant? It may also be considered *vertically*, i.e., in which infrastructure layers are they most effective?

- *Horizontal placement* has been traditionally addressed by detecting intrusions, either from network and host perspectives with: *Network Intrusion Detection Systems (NIDS)* to analyze network traffic; and *Host-Based Intrusion Detection Systems (HIDS)* to collect, correlate, and analyze system information on each host.
  NIDS sensors are spread throughout the network to get a distributed picture of the attack perimeter. This approach is fine for network threats, but could overlook malicious traffic between VMs on the same physical host.
  An HIDS provides precise audit data, which facilitates knowledge of the attack context. But, monitoring and monitored system often reside on the same host, which has performance and security impacts.
  Hybrid monitoring architectures orchestrating detection elements in the network and on hosts should bring many benefits, but are still lacking today. The situation is similar for emerging multi-cloud environments for which there is currently no integrated detection architecture.
- *Vertical placement* is an issue that was magnified with virtualization, a choice becoming possible between *virtual* (e.g., self-defending VMs), *virtualization* (e.g., VM introspection-based architectures), and *physical layers* to set the detection components. This choice depends on the stakeholders operating the layers (e.g., Cloud provider or customer). It may also vary according to: (1) their objectives regarding cloud infrastructure security management; (2) the level of openness provided to other parties for layer resource management; (3) the Cloud model. As for the horizontal case, hybrid architectures federating detection components in the different layers are promising, but have up-to-now little been explored.

To reconcile both dimensions, a *multi-layered and multi-lateral detection architecture* is clearly needed. A first idea could be to build the architecture around the well-known abstraction of *security domain* for defining monitoring scopes, that can then be flexibly coordinated in the overall infrastructure. We are currently working on such an architecture for the inter-cloud setting to enable 360°, both vertical and horizontal, self-protection [32].

Another tough question is *whether the monitoring system should be co-located (in the same layer, machine, network, ...) with the protection target?* A number of "out-of-VM" monitoring appliances have already been proposed to introduce a horizontal separation [27]. Similarly, nested virtualization [4, 36] is increasingly being explored to add one or more privileged layer of protection below the hypervisor for strong sandboxing of vulnerable upper IaaS layers. More generally, this approach can be used to realize "islands" in a IaaS infrastructure. The virtualization layer is then composed of two separate sub-layers:

- The lower layer is under the control of the cloud provider, and contains a mainstream VMM;
- The upper layer is under the control of groups of users, which may install their own VMM, without changing cloud provider.

This type of design allows to introduce more modularity in the IaaS infrastructure of a provider, often deemed too monolithic [35].

5.2 Reaction

The *placement issue* of mechanisms in the infrastructure is totally symmetric to the detection case. One may also wonder *whether reaction mechanisms are truly adaptable to a changing context*, as for the most part, configurations of defense equipments remain largely static. The emerging area of *cloud networking* aims to explore a few of such adaptability issues, e.g., real-time reconfiguration of virtual networks, or on-demand chaining of network security services. Particularly important is the influence of VM migration (across data centers, across clouds). How to migrate the security state (e.g., the IP address space) consistently, securely, and efficiently is still an unsolved issue.

Overall, *hypervisor protection* is still little addressed, notably protection of device drivers which are generally their weakest point, allowing VM isolation bypass. A rich litterature in the system community is available but, unfortunately, not yet applied. Hypervisor self-defense could be a big step towards automated hardening of a layer which remains highly vulnerable. A first design in that direction may be found in [31].

Along other lines, the evolution of regulations in the cloud area induce new compliance and auditability requirements. Thus, all techniques tending to *improve the overall level of assurance* of the cloud infrastructure (such as control flow integrity mechanisms, attestation protocols, proofs of compliance, and more generally all techniques inspired from trusted computing) should be mushrooming in the near future.

5.3 Decision-Making

The main challenge for decision-making in the cloud and inter-cloud setting is how to peform multi-lateral security policy management – starting first by *how to formalize security policies*. Despite many advances, current security policies are not sufficiently flexible for such environments, which require policies spanning organizational boundaries, geographical borders, and applicable in multiple contexts, among multiple actors. Frameworks are needed for flexible policy aggregation and deployment within clouds (for detection and reaction), as well as for policy negotiation between clouds. Despite some first frameworks being available [15], interoperability between different security domains, multiple conflicting stakeholder responsibilities, and multiple jurisdictions remain major barriers. Semantic Web technologies may facilitate such interoperability by progressing towards commonly-agreed formats for security negotiation, but much work remains to be done in that area.

Among a few other hard (but essential) unsolved problems: How to authenticate communications between detection and decision, or between decision and reaction? When coordinating multiple self-protection loops, how to guarantee the stability of the result? And, how to learn from past attacks to improve security and build defenses against future threats?

5.4 Perspectives

Overall, despite many elementary mechanisms being there and great foreseen benefits, there are still quite a few challenges remaining to be solved to reach a fully automated security supervision architecture – still lacking today both for the cloud and inter-cloud settings. Thus the road may still be long towards fully-fledged cloud self-defense. Nonetheless, not all roadblocks have the same maturity level: while self-defending VMs are already running, self-defending hypervisors are still far away down the road. In the meantime, a few self-defending mechanisms may be already running in your cloud infrastructure today...

References

1. Y. Al-Nashif, A. Kumar, S. Hariri, G. Qu, Y. Luo, and F. Szidarovsky. Multi-Level Intrusion Detection System. In *International Conference on Autonomic Computing (ICAC)*, 2008.
2. A. M. Azab, P. Ning, Z. Wang, X. Jiang, X. Zhang, and N. C. Skalsky. HyperSentry: Enabling Stealthy In-Context Measurement of Hypervisor Integrity. In *ACM Conference on Computer and Communications Security (CCS)*, 2010.
3. A. Baliga, L. Iftode, and X. Chen. Automated Containment of Rootkits Attacks. *Computers & Security*, 27:323–334, 2008.
4. M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B.-A. Yassour. The Turtles Project: Design and Implementation of Nested Virtualization. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.
5. S. Berger, R. Cáceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn. vTPM: Virtualizing the Trusted Platform Module. In *USENIX Security Symposium*, 2006.
6. S. Berger, R. Cáceres, D. Pendarakis, R. Sailer, E. Valdez, R. Perez, W. Schildhauer, and D. Srinivasan. TVDc: Managing Security in the Trusted Virtual Datacenter. *SIGOPS Oper. Syst. Rev.*, 42:40–47, January 2008.

7. D. Chess, C. Palmer, and S. White. Security in an Autonomic Computing Environment. *IBM Systems Journal*, 42(1):107–118, 2003.
8. P. Colp, M. Nanavati, J. Zhu, W. Aiello, G. Coker, T. Deegan, P. Loscocco, and A. Warfield. Breaking Up is Hard to Do: Security and Functionality in a Commodity Hypervisor. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.
9. N. De Palma, D. Hagimont, F. Boyer, and L. Broto. Self-Protection in a Clustered Distributed System. *Parallel and Distributed Systems, IEEE Transactions on*, 23(2):330 –336, 2012.
10. H. Debar. *Intrusion Detection Systems – Introduction to Intrusion Detection and Analysis*. Security And Privacy In Advanced Networking Technologies (Nato Science Series / Computer and Systems Sciences). IOS Press, 2004.
11. D. Frincke, A. Wespi, and D. Zamboni. From Intrusion Detection to Self-Protection. *Comput. Netw.*, 51:1233–1238, April 2007.
12. V. Ganapathy, M. J. Renzelmann, A. Balakrishnan, M. M. Swift, and S. Jha. The Design and Implementation of Microdrivers. In *International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2008.
13. T. Garfinkel, B. Pfaff, J. Chow, M. Rosenblum, and D. Boneh. Terra: A Virtual Machine-Based Platform for Trusted Computing. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2003.
14. A. Goel, K. Po, K. Farhadi, Z. Li, and E. de Lara. The Taser Intrusion Recovery System. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2005.
15. D. A. Haidar, N. Cuppens, F. Cuppens, and H. Debar. XeNA: An Access Negotiation Framework Using XACML. *Annales des Télécommunications*, 64(1–2):155–169, 2009.
16. A. Ibrahim, J. Hamlyn-Harris, J. Grundy, and M. Almorsy. CloudSec: A Security Monitoring Appliance for Virtual Machines in the IaaS Cloud Model. In *International Conference on Network and Systems (NSS)*, 2011.
17. X. Jiang, X. Wang, and D. Xu. Stealthy Malware Detection through VMM-Based "Out-of-the-Box" Semantic View Reconstruction. *ACM Trans. Inf. Syst. Secur.*, 13:1–28, 2010.
18. J. Kephart and W. Walsh. An Artificial Intelligence Perspective on Autonomic Computing Policies. In *Fifth IEEE International Workshop on Policies for Distributed Systems and Networks (POLICY)*, 2004.
19. R. Koller, R. Rangaswami, J. Marrero, I. Hernandez, G. Smith, M. Barsilai, S. Necula, S. M. Sadjadi, T. Li, and K. Merrill. Anatomy of a Real-Time Intrusion Prevention System. In *International Conference on Autonomic Computing (ICAC)*, 2008.
20. O. Mola and M. Bauer. Towards Cloud Management by Autonomic Manager Collaboration. *Journal of Communications, Network and System Sciences*, 4(12):790–802, 2011.
21. K. Nance, M. Bishop, and B. Hay. Virtual Machine Introspection: Observation or Interference? *IEEE Security and Privacy*, 6:32–37, September 2008.
22. A. Nguyen, H. Raj, S. Rayanchu, S. Saroiu, and A. Wolman. Delusional Boot: Securing Hypervisors without Massive Re-Engineering. In *7th ACM European Conference on Computer Systems (EUROSYS)*, 2012.
23. ObjectSecurity. OpenPMF White Paper, 2011. `http://www.openpmf.org/`.
24. A. Patel, Q. Qassim, and C. Wills. A Survey of Intrusion Detection and Prevention Systems. *Information Management & Computer Security*, 18(4):277–290, 2010.
25. R. Sailer, T. Jaeger, E. Valdez, R. Caceres, R. Perez, S. Berger, J. L. Griffin, and L. v. Doorn. Building a MAC-Based Security Architecture for the Xen Open-Source Hypervisor. In *Annual Computer Security Applications Conference (ACSAC)*, 2005.
26. R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and Implementation of a TCG-Based Integrity Measurement Architecture. In *USENIX Security Symposium*, 2004.
27. D. Srinivasan, Z. Wang, X. Jiang, and D. Xu. Process Out-Grafting: An Efficient "Out-of-VM" Approach for Fine-Grained Process Execution Monitoring. In *ACM Conference on Computer and Communications Security (CCS)*, 2011.

28. U. Steinberg and B. Kauer. NOVA: A Microhypervisor-Based Secure Virtualization Architecture. In *European conference on Computer systems (EuroSys)*, 2010.
29. J. Strassner. *Policy-Based Network Management: Solutions for the Next Generation*. Morgan Kaufman, 2003.
30. L. Tan, E. Chan, R. Farivar, N. Mallick, J. Carlyle, F. David, and R. Campbell. iKernel: Isolating Buggy and Malicious Device Drivers Using Hardware Virtualization Support. In *International Symposium on Dependable, Autonomic and Secure Computing (DASC)*, 2007.
31. A. Wailly, M. Lacoste, and H. Debar. KungFuVisor: Enabling Hypervisor Self-Defense. In *EUROSYS Doctoral Workshop (EURODW)*, 2012.
32. A. Wailly, M. Lacoste, and H. Debar. VESPA: Multi-Layered Self-Protection for Cloud Resources. In *International Conference on Autonomic Computing (ICAC)*, 2012.
33. Y.-M. Wang, D. Beck, B. Vo, and C. Verbowski. Detecting Stealth Software with Strider GhostBuster. In *Conference on Dependable Systems and Networks (DSN)*, 2005.
34. Z. Wang and X. Jiang. HyperSafe: A Lightweight Approach to Provide Lifetime Hypervisor Control-Flow Integrity. In *IEEE Symposium on Security and Privacy*, 2010.
35. D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize Once, Run Everywhere. In *7th ACM European Conference on Computer Systems (EUROSYS)*, 2012.
36. F. Zhang, J. Chen, H. Chen, and B. Zang. CloudVisor: Retrofitting Protection of Virtual Machines in Multi-Tenant Cloud with Nested Virtualization. In *ACM Symposium on Operating Systems Principles (SOSP)*, 2011.

Survey of Security Problems in Cloud Computing Virtual Machines

Ivan Studnia[1,2], Eric Alata[1,3], Yves Deswarte[1,2], Mohamed Kaâniche[1,2], and Vincent Nicomette[1,3]

[1] CNRS, LAAS, 7 Avenue du colonel Roche, F-31400 Toulouse, France
[2] Univ. Toulouse, LAAS, F-31400 Toulouse, France
[3] Univ. Toulouse, INSA, LAAS, F-31400 Toulouse, France
{studnia,ealata,deswarte,kaaniche,nicomett}@laas.fr

Abstract. Virtualization techniques are at the heart of Cloud Computing, and these techniques add their own vulnerabilities to those traditional in any connected computer system. This paper presents an overview of such vulnerabilities, as well as possible counter-measures to cope with them.

Keywords: virtual machine, hypervisor, vulnerabilities, virtualization-based monitoring, isolation enforcement.

1 Introduction

Virtualization has become an attractive and widely used technology in today's computing. Indeed, the ability to share the resources of a single physical machine between several isolated virtual machines (VM) enabling a more optimized hardware utilization, as well as the easier management and migration of a virtual system compared to its physical counterpart, have given rise to new architectures and computing paradigms. In particular, virtualization is a key element in cloud computing.

However, adding another abstraction layer between hardware and software raises new security challenges. This paper gives an overview of some security problems related to the use of virtualization and shows that the widely used virtual machine managers cannot be considered fully secure. This observation, added to the ever growing popularity of virtualized architectures has led to an important number of studies aiming at enforcing security in virtual systems.

This paper presents first the virtualization principles, then discusses some vulnerabilities related to virtual systems before describing various attempts to address the security challenges raised by virtualization.

2 Context

2.1 Definition

In [21], Popek and Goldberg define a virtual machine as *"an efficient, isolated duplicate of a real machine"*. They also give three necessary conditions to reach this goal.

– Efficiency: Virtualization shall not induce a significant decrease in performance. Therefore, the greatest amount of instructions must not require an intervention from the virtual machine manager (VMM).
– Resource control: The VMM must have a complete control over the virtualized resources.
– Equivalence: A program must behave the same way on a virtual machine as it would do on its physical counterpart.

However, this definition, given in 1974, does no longer cover the whole range of possibilities offered by virtualization. For example, it is now possible to emulate (parts of) systems which do not have a physical equivalent on the host machine. Therefore, new definitions have been established. We will use a definition given in [22]: *"Virtualization is defined as a framework dividing the resources of the device from the execution environment, allowing environment plurality by using one or more techniques such as time-sharing, emulation, partitioning."*

This definition allows us to include emulation or paravirtualization (cf. 2.3) into virtualization and therefore covers a broader range of currently used techniques.

2.2 Classification

Virtualization offers many possibilities, and according to the needs, various kinds of virtualization can be used:

– Process virtualization: virtualizing this layer consists in providing an interface between an application and the underlying system. This allows to create an application without concerning about the specificities of the OSes it will run on, as long as they possess the required virtualization layer. The Java Virtual Machine is an example of process virtualization.
– Server virtualization: here, the virtualization is applied to the hardware. This will allow many OSes to run simultaneously on a physical machine. In this paper, we focus on server virtualization techniques.
– Network virtualization: VPNs (Virtual Private Networks, which enable the interconnection of distinct private networks through a public infrastructure such as the Internet) and VLANs (Virtual LANs, distinct local networks sharing the same physical infrastructure) are examples of network virtualization.
– Storage virtualization: SANs (Storage Area Networks) fall into this category.

2.3 Server virtualization

There are several ways of implementing server virtualization, either by interacting with the hardware, (nPar[4] for example), the OS (Linux-VServer[5]) or through a hypervisor, that is to say a program dedicated to the management of virtual machines. Our study will focus on the latter[6]. These hypervisors are often categorized within two groups:

– Type 1: Type 1 managers are installed directly above the hardware and run with the highest level of privileges. Xen [4] and VMWare ESX [1] are type 1 hypervisors.
– Type 2: Type 2 managers are installed above an operating system, like any other program. QEMU and VirtualBox are type 2 hypervisors.

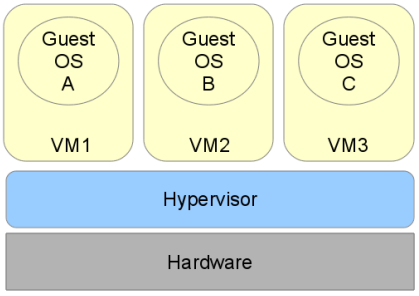


Fig. 1. Type 1 hypervisor

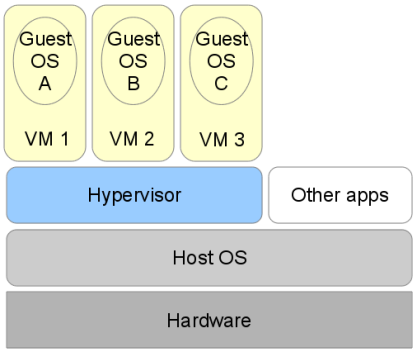


Fig. 2. Type 2 hypervisor

Furthermore, we can distinguish three major techniques of hypervisor based virtualization: full virtualization, paravirtualization and emulation.

Full virtualization In this case, one or more unmodified operating systems (called "guests") are sharing hardware resources from the host system. The presence of the hypervisor is transparent from the guests viewpoint.

Paravirtualization The kernels of the guest systems are modified to make them able to communicate with the hypervisor below them via *hypercalls*. Hypercalls can be seen as equivalents to system calls of a non virtualized OS. Paravirtualization is historically the standard method of virtualization used by Xen.

Emulation Like full virtualization, emulation allows unmodified operating systems to run. However, in that case, the resources seen by the guest OS are completely simulated by software. This allows to execute an operating system compiled on an architecture different from the architecture of the host. This results in lower throughput than with the previously mentioned techniques. QEMU is an example of an emulator.

2.4 Hardware assisted virtualization

Leveraging hardware capabilities to support virtualization has been done for a long time (IBM System/370, 1972). However, hardware assisted virtualization was not implemented on x86 CPUs. This prevented Popek and Goldberg's conditions from being met because some privileged instructions were not correctly trapped. Therefore, the hypervisor had to perform extra tasks in order to address those lacks. This increased the overall complexity and impacted the performance. However, since 2006, Intel VT[7]and AMD-V[8] technologies implement such techniques. A new CPU state was

[4] `http://en.wikipedia.org/wiki/HP_nPar_%28Hard_Partitioning%29`
[5] `http://linux-vserver.org/`
[6] We will use indifferently the terms "hypervisor" and "virtual machine manager" in this paper.
[7] `http://www.intel.com/technology/itj/2006/v10i3/1-hardware/6-vt-x-vt-i-solutions.htm`
[8] `http://sites.amd.com/us/business/it-solutions/virtualization/Pages/virtualization.aspx`

introduced, orthogonal to privilege rings 0-3, called `root mode` on Intel chips and `guest mode` on AMD chips. This state is accessed whenever the hypervisor needs to take (resp. give back) the control over (to) a virtual machine. The guest OS can actually run in ring 0 but not in root mode. Furthermore, the handling of I/O memory virtualization allows to prevent DMA (Direct Memory Access) requests issued from a virtual machine to tamper with unauthorized zones of the host memory.

Although these various features were introduced to facilitate the work of the hypervisor, they can also be used to carry out new kinds of attacks. An example of such an attack is presented in 3.2.

3 Vulnerabilities and attacks

Virtualization technologies offer new economical and technical possibilities. However, the addition of a new layer of software introduces new security concerns. Garfinkel and Rosenblum give in [9] a list of challenges raised by virtualisation that are discussed hereafter.

Scaling. Virtualization enables quick and easy creation of new virtual machines. Therefore, security policies of a network (setup, updates...) have to be flexible enough to handle a fast increase in the number of machines.

Transience. With virtualization, machines are often added to or removed from a network. This can hinder the attempts to stabilize it. For example, if a network gets infected by a worm, it will be harder to find precisely which machines were infected and clean them up when these machines exist only during brief periods of time on the network. Similarly, infected machines or still vulnerable ones can reappear after the infection was thought to be wiped out.

Software lifecycle. The ability to restore a virtual machine into a previous state raises many security concerns. Indeed, previously patched vulnerabilities (programs flaws, deactivated services, older passwords...) may reappear. Moreover, restoring a virtual machine into a previous state can allow an attacker to replay some sequences, which renders obsolete any security protocol based on the state of the machine at a given time.

Diversity. In an organization where security policies are based on the homogeneity of the machines, virtualization increases the risk of having many versions of the same system at the same time on the network.

Mobility. A virtual machine is considered like any other file on a hard drive. It can be copied and moved to another disk or another host. This feature, cited as a benefit of virtualization, also adds security constraints because guaranteeing the security of a virtual machine becomes equivalent to guaranteeing the security of every host it has been on.

Identity. Usual methods used to identify machines (like MAC addresses) are not necessarily efficient with virtual machines. Moreover, mobility increases even more the difficulties to authenticate the owner of a virtual machine (as it can be copied or moved).

Data lifetime. A hypervisor able to save the state of its VMs can counter the efforts made by a guest to delete sensitive data from its memory. Indeed, there may always be a backup version of the VM containing the data.

If many of these challenges can be addressed with good use policies (for cloud computing, examples can be found in [2]), attackers may still exploit flaws in the system to perform their attacks. The rest of this section will focus on these malicious attempts to break into a virtualized system.

3.1 Vulnerabilities

Technically, virtualization has enabled the emergence of new attack scenarios. First, a virtual machine being the equivalent of a physical machine, an attacker willing to take control over it can use the same tools in both cases. On the other hand, the addition of a virtualization layer can give an attacker new ways of subverting its target. Indeed, as shown in figure 3, for a type 2 hypervisor, vulnerabilities in the virtualization layer may allow attacks to be performed from a virtual machine against another VM, the VMM or the host operating system. The picture would be similar for a type 1 hypervisor, except that there is no host OS underneath the hypervisor, so the corresponding attack surface is also removed. If there are any system management tools, these are also considered as a potential attack vector.

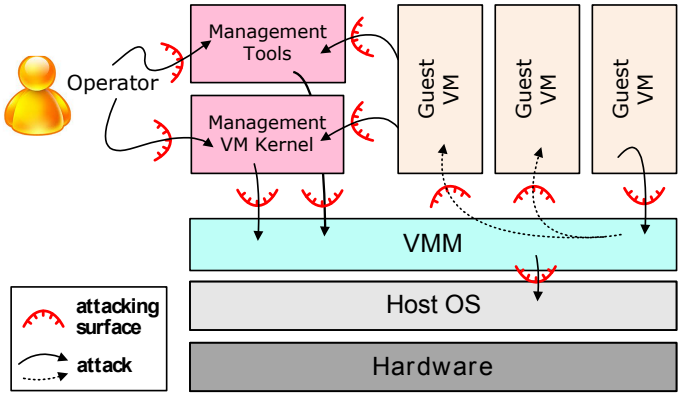


Fig. 3. (after figure 1 from [30]) Attack surfaces on virtual machines

3.2 Attack examples

In this part, we will describe some typical examples of attacks targeting virtualized systems or using virtualization properties to corrupt a machine.

Detecting a virtualized environment. Virtualization technologies (except paravirtualization) are supposed to provide the guest OS with an identical duplicate of a real system. Actually, this

is not completely true and it is therefore possible to detect whether there is a hypervisor running underneath the OS.

Such techniques are useful for both attackers and defenders. On the one hand, an attacker can check if the targeted system is virtualized and acts accordingly. On the other hand, a user willing to check the integrity of his machine can check for the presence of a hypervisor installed against his will under his OS (cf 3.2). [8] gives some generic examples of detection. First, a hypervisor can be detected by checking the execution time of some instructions because the processing of such instructions by the VMM requires more time than when the system is not virtualized. However, such comparisons are possible only if the measurements were previously done on an identical, known to be safe system. [8] also describes a method based on the measurements of the time needed to access the *Translation Lookaside Buffers* (TLB): once those buffers have been filled with some known data, calling the CPUID instruction triggers the cleaning of at least one portion of the TLB if a hypervisor trapped it. Then, by comparing the access times to the TLB measured before and after calling CPUID, one can establish the presence of a hypervisor. One advantage of this approach is that it does not require preliminary baseline measurements.

Identifying the hypervisor. Ferrie describes in [8] the processes he used to identify with precision which of six hypervisors (VMWare, VirtualPC, Parallels, Bochs, Hydra and QEMU) was used. To do so, he used specific instructions that are not handled by some hypervisors the same way as they are on a real system (this can lead to hypervisor-specific exceptions or, on the contrary, to the absence of exception whereas some would have been raised on a real system). Among the known applied examples of such methods, we can cite RedPill[9] and ScoopyDoo[10].

Once a VMM is correctly identified, an attacker can adapt his attack scenario to the vulnerabilities known as characteristic of this VMM. Such techniques also allow the creation of viruses targeting only the systems where a specific kind of hypervisor is running. It should be noted that some of these methods allow to detect and identify various popular hypervisors. However, in most cases they are legitimately installed on a machine, so it is understandable that they do not try to hide themselves.

Breach in the isolation. One of the main goals of a hypervisor is to ensure that the hosted virtual machines are isolated, which means that one guest system is not able to reach more resources than it has been granted, especially the memory used by the other guests or by the host system. However, bad configuration or design flaws within the VMM can allow an attacker to break out of the isolation. This can lead to:

- Denial of service: A virtual machine uses all the computing capacities of the real host, preventing the other VMs from running correctly.
- System halt: A specifically crafted instruction causes the VM or the hypervisor to crash
- *VM escape*: This category covers the situations where an attacker gains access to memory located outside the region allocated to the corrupted VM. The attacker can access the memory of other systems (guest or even host) and can read, write or execute its content. This can lead to a complete takeover of another VM, of the hypervisor or of the entire host system (for a type 2 hypervisor).

[9] `http://invisiblethings.org/papers/redpill.html`
[10] `http://www.trapkit.de/research/vmm/scoopydoo/index.html`

VM escape documented attacks targeting Xen, VMWare and Linux KVM are respectively described in [28], [15] and [7]. In these three situations, an attacker was able to run some custom code with the highest privileges and therefore took over the whole system. This kind of attacks exploits some vulnerabilities in the source code or the design of the hypervisor. In [19], random sequences of code are sent (by fuzzing techniques) to several VMMs. For every tested hypervisor, some sequences have caused a system crash. This is often the first step to discover new exploitable vulenrabilities. Moreover, the amount of code shipped with every new version of a hypervisor seems to be increasing (cf. Table 1), which raises the probability of new vulnerabilities.

Accurate targeting of VMs in a cloud. In this paragraph, we will focus on the works done by Ristenpart et al. [23]. Their research work consists in identifying techniques to ensure a co-residence in a public cloud (Amazon EC2 in their experiments). This means that the authors are able to obtain a VM that is hosted on the same real host as another VM they want to attack. Thanks to this co-residence, they are able to monitor some activities of the targeted VM through a covert channel.

Let us note that, in these experiments, only "legitimate" tools were used and no modification of the targeted systems has been done.

The authors started by establishing a map of the cloud: they examined how the IPs were distributed according to the characteristics of the corresponding VMs (number of cores, storage capacity, ...). Then, they identified several techniques for checking the co-residence between an attacker's VM and his target, first through a network analysis (if the first node is the same, or if the IPs are within a certain range, they are co-resident), then by measuring the host activity. To do so, they compared the time required for reading a given portion of the memory before and during an intense activity imposed to the target (by sending multiple requests to the targeted server). Results of this experiment are shown in figure 4. The target is a web server to which multiple `HTTP` get requests were periodically sent. In the first two graphs, there is a clear distinction between the activity recorded during the `HTTP` gets and the "standard" activity ("`No HTTP gets`"): the two machines are co-resident. In the third graph, no distinction can be made: two machines are not resident on the same real host.

Once that co-residence is confirmed, an attacker can obtain more information about the target activities with a similar method (by monitoring the host activity without interacting with the target) or try to take over it by other means.

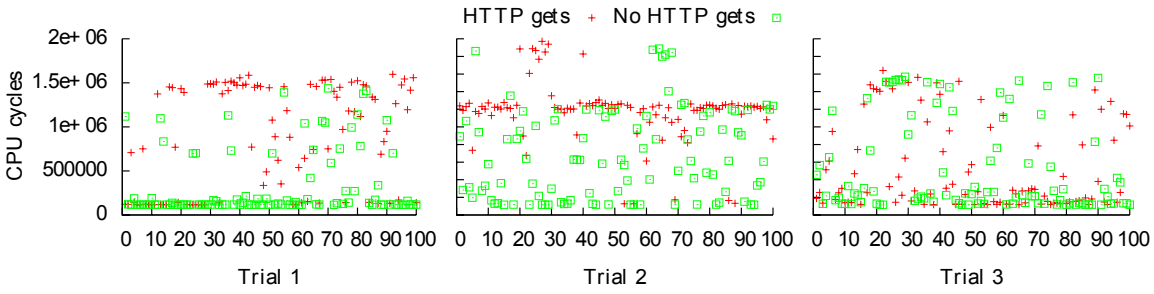


Fig. 4. (Figure 5 from [23]) Establishing co-residence in a cloud through load measurements

Virtual Machine Based Rootkits. The previously described attacks were based on some characteristic features or design flaws of the VMMs. *Virtual Machine Based Rootkits* (VMBRs) are different. Here, the attacker uses some features of hardware-assisted virtualization of x86 processors (cf 2.4) in order to install a hypervisor underneath the targeted OS, putting it into a virtual machine. As the attacker totally controls the hypervisor, he can monitor the whole virtualized OS. Some VMBR implementations can virtualize the OS on the fly (BluePill [24]), others can survive a reboot (SubVirt [13]). Finally, J. Rutkowska [25] and her team were able to install BluePill under Xen after subverting it from one of its VM, therefore putting the hypervisor itself into a VM (*nested virtualization*).

4 Virtualization and security

As seen previously, virtualized systems have many advantages over their physical counterparts. However, they also have their own vulnerabilities (in addition to the "traditional" vulnerabilities inherited from physical systems). Although an important portion of the threats can be prevented by following some rules (for the administrator as well as the users), an attacker may still be able to reach his goal by exploiting some flaws. Many works are therefore published in order to improve the security of virtual environments. We will describe some of them hereafter, but this list is not comprehensive. Like [6], we split up these works in two categories: VM monitoring and isolation enforcement.

4.1 VM monitoring

The hypervisor having higher privileges than the virtualized operating systems, it can be used to monitor their activities "from the outside". This section describes applied examples of this idea.

Virtual machine introspection. Virtual machine introspection (VMI) techniques are based on the ability of the hypervisor to access VM-allocated memory space in order to analyze its content from outside. Data retrieval and analyses can be done directly within the hypervisor or from a dedicated VM (which is recommended because this will require less modifications in the hypervisor code). However, the VMM has basically no knowledge of the abstractions used by the guest OS and is therefore unable to understand the meaning of the retrieved memory zones. This is the *semantic gap* that VMI programs have to fill. There are many VMI implementations, from frameworks (like LibVMI[11] for Xen and KVM) to entire hypervisors, according to one's needs. [18] classifies them according to three characteristics:

- Monitoring or interfering: is the system only watching and reporting any anomaly or can it counter attacks by itself ?
- Semantic awareness: the amount of knowledge the system possesses about the guest OS. For example, Lares[20], using hooks located into the guest system, needs a strong knowledge of this system. On the other hand, Cloudsec [11] knows nothing about the guest OS and tries to fill the semantic gap according to the gathered information.
- Event replay: the ability of the system to record the state of the monitored VM for further analyses.

[11] http://code.google.com/p/vmitools/

Kernel protection. Virtualizing an OS puts it in a less privileged state, the hypervisor getting the highest privileges. The latter can then enforce security in the OS from outside.

SecVisor [26] is a hypervisor designed to ensure the integrity of an OS. It is a small VMM (less than 5000 lines of code) aimed at allowing only the execution of user approved code. The design of SecVisor had to fulfill three criteria:

1. A small amount of code in order to allow a formal checking or a manual audit.
2. A minimal exterior interface in order to reduce the attack surface.
3. The lowest possible amount of changes made into the monitored kernel in order to ease portability.

SecVisor does not prevent code from being added to the kernel, although it forbids its execution unless the code has been approved by the user. To do so, SecVisor relies on hardware features (cf. 2.4) such as hardware memory protection.

The `Hytux` prototype[16], developed at LAAS, is also a lightweight hypervisor that implements protection mechanisms in a more privileged mode than the Linux kernel. It is especially designed to counter kernel rootkits, malware that aims at corrupting the kernel of operating systems.

Rootkit detection. Similarly, Patagonix [17] is a hypervisor designed to detect rootkits hiding into the monitored OS, without relying on the OS capacities. To do so, the code of the virtualized OS is tagged as non executable, which triggers an action from the hypervisor the first time this code is executed of when it has been modified from the previous execution. Then, the code is identified by comparing it with a database of known binaries. Therefore, it becomes possible to detect modified binaries or programs trying to hide their presence to the OS. By using a database to deal with the semantic gap, Patagonix can theoretically monitor any OS as long as it possesses a sufficient database of its correct binaries.

4.2 Isolation enforcement

Some other studies focus on means to prevent a breach in the isolation property. Corrupting the hypervisor (by exploiting design flaws or other vulnerabilities) is a way to break this isolation. Again, various approaches are possible to deal with this problem.

Security modules. This category contains the modules developed to provide security features into a hypervisor. For example, Xen Security Modules [5] is a framework implementing new security rules into Xen that an administrator can include while compiling it.

Another example would be sVirt[12], which uses Mandatory Access Control (MAC) to set a stronger isolation between the VMs in Linux-based virtualization solutions (such as KVM). sVirt assigns a distinct MCS (Multi Category Security) label to each VM but gives the same label to a VM process and its corresponding VM image. Therefore, a VM process will be allowed to read and modify only its corresponding image but any access from one VM process to another or to another image will be prevented. Thus, if one VM becomes compromised, it will not be able to access other VMs (images or processes) hosted on the same computer.

[12] http://selinuxproject.org/page/SVirt

Protecting the VMM. As seen in section 4.1, a hypervisor can be used to monitor the virtualized systems it is hosting. However, as seen in 3.2, the hypervisor can in turn be targeted and modified by an attack. As the hypervisor possesses every privilege on its guest systems, it is crucial to preserve its integrity. However, while it is possible to ensure the integrity of a system during boot (we can for example cite the Trusted Computing Group[13] which works on this topic), it is much harder to ensure runtime integrity. So, to ensure runtime integrity, one could think of installing a second hypervisor under the initial hypervisor dedicated to monitoring it, similarly to 4.1. However, one would have to guarantee that the most privileged hypervisor cannot in turn be corrupted. Several studies have therefore focused on using other means to ensure the integrity of the most privileged element.

First, HyperSentry [3] aims at providing a secure environment to tools willing to enforce the integrity of the most privileged element. For that purpose, it leverages both the SMM (*System Management Mode*) specificities to protect its code from the hypervisor and the Trusted Boot as defined by the TCG [14] to ensure the integrity of its code at startup. Moreover, through the use of communication channels uncontrolled by the hypervisor (like IPMI[15] in [3]), an analysis can be executed without the hypervisor noticing it. Therefore, HyperSentry can access the state of a hypervisor at a given time without making it aware of the process, theoretically preventing it from tampering with the data before it is sent to the analysis tools.

Other works, like HyperSafe [27], try to protect the hypervisor from modifications through the strict enforcement of certain principles. HyperSafe patches an existing hypervisor to enforce two features (*memory lockdown* and *restricted pointer indexing*) aiming at guaranteeing control flow integrity. The objective is to make the hypervisor protect itself against unwanted modifications or executions of its code. HyperSafe has already been ported on Bitvisor and (not yet completely, according to [27]) on Xen.

However, if such tools allow to enhance the ability of a hypervisor to protect itself, using them implies increasing the amount of code running with high privileges (since it is part of the hypervisor). Thus, any design mistake present into these modules can become an entry point for new attacks. Such an example is detailled in [29], where an attack is made possible only if the FLASK module for XSM is installed, which is not the default configuration for Xen.

Protecting the VMs against their VMM. The purpose of CloudVisor [30] is to ensure data confidentiality and integrity for the VM, even if some elements of the virtualization system (hypervisor, management VM, another guest VM) are compromised[16]. The idea is that data belonging to a VM but accessed by something else than this VM appears encrypted. To reach its goal, CloudVisor virtualizes the monitored hypervisor (realizing nested virtualization), therefore removing the latter from the most privileged zone while still giving it the illusion of the opposite. This means that the monitored VMM is now running in *guest mode* while CloudVisor is the only one in *root mode*. Any access, requested by the VMM, to some memory belonging to a VM is then trapped by CloudVisor. If the access is not requested by the owner of the requested page, CloudVisor encrypts its content. Such a concept seems particularly interesting within a cloud context, where multi-tenancy

[13] `http://www.trustedcomputinggroup.org`

[14] `http://www.trustedcomputinggroup.org/resources/trusted_boot`

[15] *Intelligent Platform Management Interface* : `http://download.intel.com/design/servers/ipmi/IPMIv2_0rev1_0.pdf`

[16] Let us note that CloudVisor does not try to protect a VM from any other kind of attacks.

(unrelated users sharing the same physical resources) can be the norm and where privacy therefore represents one of the main concerns of cloud customers.

Microkernels. Microkernels result from the will to design an operating system kernel in another way: a feature shall be allowed in kernel space only if its removal (and execution in user space) prevents the system from working correctly. Everything else is designed as user space modules, communicating between themselves through IPC (*Inter Process Communication*). Figure 5 (source Wikipedia[17]) shows that with such a principle, there is much less code running in kernel space in a microkernel than in a so-called monolithic kernel (such as the Linux kernel). This makes a microkernel critical code easier to audit.

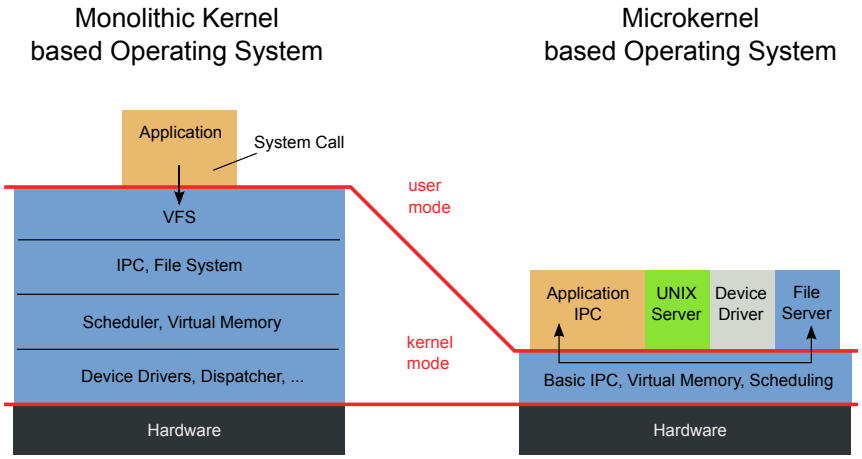


Fig. 5. Overview of the differences between a monolithic kernel and a microkernel

Conceptually, since a privileged module deals with isolation and communication between the processes running above it, microkernels are close to hypervisors. However, only the formers have a claim on minimal size. Those similarities [10] allow some microkernels to work as virtual machine managers. Among these is L4 [18] for which one implementation, called seL4, was formally proven to be secure [14]. This means that seL4 is theoretically flawless (unless a flaw is found in the proof system itself, or in the underlaying layers: memory protection, context switching, hardware management, etc.). Running such a microkernel with the highest level of privileges theoretically prevents their compromission. The use of microkernels, with their smaller amount of critical code, as a basis for virtualization can also help to counter the trend of the increasing code size into some hypervisors (example with Xen in Table 1).

Removing the VMM. NoHype [12] represents yet another case. The hypervisor being a potential attack surface between two virtual machines, one solution to remove this surface is to get rid of

[17] `http://en.wikipedia.org/wiki/Microkernel`
[18] `http://os.inf.tu-dresden.de/L4/overview.html`

Table 1. Amount of lines of critical code into Xen, after [30]

	VMM	Dom0 Kernel	Tools	Total
Xen 2.0	45K	4,136K	26K	4,207K
Xen 3.0	121K	4,807K	143K	5,071K
Xen 4.0	270K	7,560K	647K	8,477K

the hypervisor. This is made possible by using hardware assisted virtualization and putting several constraints on the conditions of virtualization: each virtual machine is linked to one unique processor core, the portion of memory allowed to it is fixed and managed by the hardware (see 2.4) and each (virtual) device is dedicated to only one VM. Figure 6 gives an overview of NoHype implementation. MMC stands for *Multi-core Memory Controller*. It is the device dedicated to managing the allocation of memory to each core. The system manager (on the left) also runs on one dedicated core, which is the only one allowed to send IPIs (*Inter Processor Interrupts*) to other cores, enabling the startup, shutdown or migration of a VM on them. For example, to start a VM, the system manager prepares the required resources and sends an IPI to the core on which the VM is supposed to run. This core runs a program responsible for configuring it (by mapping the allocated resources) then starts the VM image. The effective virtualization of the guest OS is realized by a tiny hypervisor, called *core manager* (the little grey squares on the schema). The system manager is then able to communicate exclusively with the core managers and only to trigger the migration or termination of a VM. However, due to its design, NoHype loses some features offered by more traditional virtual machine managers, such as the ability to share resources (devices, memory buffers) between several VMs.

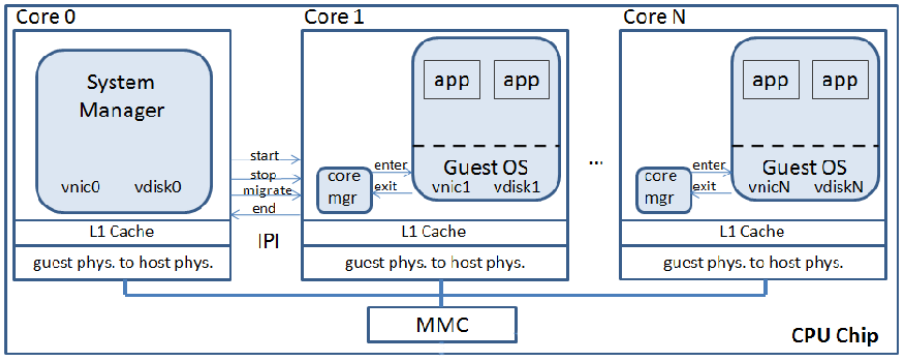


Fig. 6. NoHype architecture (excerpt from [12])

5 Conclusion

Virtualization technologies can bring many interesting new features (optimized use of hardware resources, eased restauration, machine migration, ...), but they also introduce new means to perform attacks. These attacks may either be directed against virtualized systems or leverage some features

related to virtualization in order to take over a system. Therefore, especially if a system is shared between many users, as is the case in cloud computing, strong security of virtual machines is crucial to protect their data and gain the customer's trust. We have presented various implementations dealing with those concerns on two main topics:

- Leveraging virtualization to secure a system located on top of a hypervisor.
- Securing the VMM. This also raised issues concerning the security of the most privileged element of a system.

The research on these topics is very active today, following a great diversity of possibilities, going from devising ways to secure popular systems like Xen, KVM or VMWare solutions, to creating new models such as the microkernel-based virtualization. However, even if these solutions are satisfying security-wise, one should still consider the possible issues of their large-scale deployment. Indeed, additional control procedures will cause a decrease in efficiency. One must therefore find the right balance between performance and security, according to their needs.

Acknowledgements

This research has been partially supported by the French ANR SOP project (ANR-11-INFR-0001), and by the French project Investissements d'Avenir SVC (Secure Virtual Cloud).

References

1. VMWare ESX homepage. `http://www.vmware.com/products/vsphere/esxi-and-esx/index.html`
2. Alliance, C.S.: Security guidance for critical areas of focus in cloud computing v3.0. Cloud Security Alliance (2011)
3. Azab, A., Ning, P., Wang, Z., Jiang, X., Zhang, X., Skalsky, N.: HyperSentry: enabling stealthy in-context measurement of hypervisor integrity. In: Proceedings of the 17th ACM conference on Computer and communications security. pp. 38–49. ACM (2010)
4. Barham, P., Dragovic, B., Fraser, K., Hand, S., Harris, T., Ho, A., Neugebauer, R., Pratt, I., Warfield, A.: Xen and the art of virtualization. ACM SIGOPS Operating Systems Review 37(5), 164–177 (2003)
5. Coker, G.: Xen Security Modules (XSM). Xen Summit (2006)
6. Douglas, H., Gehrmann, C.: Secure virtualization and multicore platforms state-of-the-art report. Tech. rep., SICS, Swedish Institute of Computer Science (2009)
7. Elhage, N.: Virtunoid: Breaking out of KVM. Black Hat USA (2011)
8. Ferrie, P.: Attacks on more virtual machine emulators. Symantec Technology Exchange (2007)
9. Garfinkel, T., Rosenblum, M.: When virtual is harder than real: Security challenges in virtual machine based computing environments. In: Proceedings of the 10th conference on Hot Topics in Operating Systems-Volume 10. p. 20. USENIX Association (2005)
10. Heiser, G., Uhlig, V., LeVasseur, J.: Are virtual-machine monitors microkernels done right? ACM SIGOPS Operating Systems Review 40(1), 95–99 (2006)
11. Ibrahim, A., Hamlyn-Harris, J., Grundy, J., Almorsy, M.: Cloudsec: A security monitoring appliance for virtual machines in the IaaS cloud model. In: Network and System Security (NSS), 2011 5th International Conference on. pp. 113–120. IEEE (2011)
12. Keller, E., Szefer, J., Rexford, J., Lee, R.: Nohype: virtualized cloud infrastructure without the virtualization. In: ACM SIGARCH Computer Architecture News. vol. 38, pp. 350–361. ACM (2010)
13. King, S., Chen, P.: Subvirt: Implementing malware with virtual machines. In: Security and Privacy, 2006 IEEE Symposium on. pp. 14–pp. IEEE (2006)

14. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., et al.: sel4: Formal verification of an os kernel. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles. pp. 207–220. ACM (2009)
15. Kortchinsky, K.: Cloudburst—a vmware guest to host escape story. Black Hat USA (2009)
16. Lacombe, E., Nicomette, V., Deswarte, Y.: Enforcing kernel constraints by hardware-assisted virtualization. Journal in computer virology 7(1), 1–21 (2011)
17. Litty, L., Lagar-Cavilla, H., Lie, D.: Hypervisor support for identifying covertly executing binaries. In: Proceedings of the 17th conference on Security symposium. pp. 243–258. USENIX Association (2008)
18. Nance, K., Bishop, M., Hay, B.: Virtual machine introspection: Observation or interference? Security & Privacy, IEEE 6(5), 32–37 (2008)
19. Ormandy, T.: An empirical study into the security exposure to hosts of hostile virtualized environments. In: Proceedings of CanSecWest Applied Security Conference (2007)
20. Payne, B., Carbone, M., Sharif, M., Lee, W.: Lares: An architecture for secure active monitoring using virtualization. In: Security and Privacy, 2008. SP 2008. IEEE Symposium on. pp. 233–247. IEEE (2008)
21. Popek, G., Goldberg, R.: Formal requirements for virtualizable third generation architectures. Communications of the ACM 17(7), 412–421 (1974)
22. Ramos, J.: Security challenges with virtualization (2009)
23. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM conference on Computer and communications security. pp. 199–212. ACM (2009)
24. Rutkowska, J.: Subverting vista™ kernel for fun and profit. BlackHat Briefings USA (2006)
25. Rutkowska, J., Tereshkin, A.: Bluepilling the xen hypervisor. Black Hat USA (2008)
26. Seshadri, A., Luk, M., Qu, N., Perrig, A.: Secvisor: A tiny hypervisor to provide lifetime kernel code integrity for commodity oses. In: ACM SIGOPS Operating Systems Review. vol. 41, pp. 335–350. ACM (2007)
27. Wang, Z., Jiang, X.: Hypersafe: A lightweight approach to provide lifetime hypervisor control-flow integrity. In: Security and Privacy (SP), 2010 IEEE Symposium on. pp. 380–395. IEEE (2010)
28. Wojtczuk, R.: Subverting the Xen hypervisor. Black Hat USA 2008 (2008)
29. Wojtczuk, R., Rutkowska, J., Tereshkin, A.: Xen 0wning trilogy. In: Black Hat conference (2008)
30. Zhang, F., Chen, J., Chen, H., Zang, B.: Cloudvisor: Retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In: Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles. pp. 203–216. ACM (2011)

Reconsidering Isolation in IaaS Clouds: a Security Perspective

Kahina Lazri[1], Sylvie Laniepce[1] and Jalel Ben-Othman[2]

[1] Orange Labs, Security and Trusted Transactions Dept.
42 rue des Coutures. Caen, France.
[2] L2TI Laboratory, University of Paris Nord.
99 avenue Jean Baptiste Clement. 93430 Villetaneuse. France.
{kahina.lazri,s.laniepce}@orange.com
jalel.ben-othman@univ-paris13.fr

Abstract. Multi-tenancy is the core feature that enables efficiency and cost effectiveness of cloud computing. However, it brings several new security concerns. Ensuring 'strong isolation' between co-localized tenants remains the most critical issue. This work aims at presenting technical vulnerabilities brought by the resource sharing paradigm in multi-tenant elastic IaaS Cloud. The first part of this paper surveys the literature related to accepted vulnerabilities. Several Proofs of Concepts are described and classified according to the results of the exploitation of these vulnerabilities. In the second part, we argue the existence of new attack strategies able to take advantage of the mechanisms which enable autonomic elasticity. These mechanisms are by nature sensitive to VMs resource consumption which can be easily manipulated by attacks. Finally, we give a representation of the presented vulnerabilities to engage a discussion on the limitations of pure user-centric security approaches for guarantying VM security.

Keywords: Cloud Computing, Security, Multi-tenancy, Vulnerabilities, Isolation, Resource Sharing, Elasticity.

1 Introduction

The definition of cloud computing paradigm has remained for a long time an ambiguous concept. This may be justified by the fact that cloud computing does not only represent a technology evolution but brings several new concepts in the economical, architectural and indisputably technological areas. The emergence of all these concepts has led to a deep revisiting of IT ecosystems in a global manner. Even if several definitions have been established in the last decade [1,2,3], they all agree that Cloud is a new technology which enables delivering Software, Platform and Infrastructure as a Service to end users over the Internet. A key feature that differentiates the Cloud from legacy outsourcing solutions is elasticity [4]. Thus, the Cloud is usually associated for cloud customers to the resource *on demand* and *pay-as-you-use* model. However, we deem that even if the Cloud enables tailoring resource consumption to computation footprint, current deployed models are still far from the dream of the Computer as a Utility model [5].

Hardware virtualization constitutes the heart technology of Cloud, it enables running multiple instances of software stack (application, middleware and OS) in the form

of Virtual Machines (VMs), within a single hardware machine in the aim of improving efficiency and cost effectiveness both for cloud customers and providers. The sharing of resources between VMs possibly belonging to different tenants – called the multi-tenancy – is the key feature that enables providers to cope with unforeseeable workload demand, either by providing more (less) resources to existing VMs or by instantiating new (removing) VMs.

Security is certainly the main roadblock for a massive adoption of Cloud model. Cloud security has been the subject of a great deal of research in both academic and industrial research communities. While some works focus on evaluating whether or not to adopt the Cloud, we believe that Cloud *is* becoming the fifth generation of IT architecture after the Services Oriented Architecture (SOA) model, and the real issue is now how to make this model reliable.

A major security challenge faced by the Cloud comes from running different VMs in the same pool of resources. From the security point of view, this resource sharing requires what we call *'Strong Isolation'* of the resources that are allocated by the Cloud infrastructure to the co-localized VMs (VMs running in the same physical machine). Strong isolation means that the co-localized VMs should have the same isolation as if they were running in separate physical machines. Therefore, we consider that ensuring isolation is more than simply preventing one VM from accessing or modifying data and code of co-localized VMs. The isolation should guarantee that one VM behavior cannot *'disturb'* another co-localized VM running.

In this paper, we discuss the vulnerabilities induced by the resource sharing in the IaaS model. Beyond highlighting some isolation-related vulnerabilities specific to Cloud platforms, the goal of this work is to consider the existence of new attack strategies able to take advantage of the mechanisms enabling the multi-tenancy associated with autonomic elasticity, to disturb the running of VMs.

The rest of this paper is organized as follows. Section 2 provides some background on virtualization architectures and resource management mechanisms. Section 3 presents a survey of exploited Cloud vulnerabilities focusing on breaking isolation. Section 4 identifies new malicious profiles that we suspect to be able to exploit the elasticity mechanisms to disturb co-localized VMs. A last section provides a global view of the previously presented vulnerabilities to engage a discussion on the limitations of pure user-centric security approaches for guarantying VM security.

2 Background on resource management in virtualization platforms

We first provide an overall view of the technologies involved in resource management in Cloud virtualization platforms, as an introduction to the study of the vulnerabilities induced by multi-tenancy and autonomic elasticity.

2.1 Hardware Virtualization

Hardware virtualization is the core technology of the Cloud. It enables running multiple VMs concurrently in the same hardware machine with the help of the hypervisor – also

referred to as Virtual Machine Monitor – which is the entity responsible for running, scheduling and containing the VMs.

Virtualization has appeared for the first time during the beginning of 70s [6], its success was short because of the drop of hardware costs and the emergence of multi-tasking operating systems. However, in the 90s, virtualization came back since computing resources appeared to be often underused. Thus, virtualization has been seen as the solution which may maximize hardware utilization.

The hypervisor is a thin layer of software intermediating between the VMs and the hardware resources (CPU, Memory and I/O devices). Its primary role is to manage resources and to ensure isolation between the various running tenants. To guarantee the control of the VMs execution, the hypervisor provides a virtual form of hardware resources to the VMs (virtual CPU, Shadow Page Tables, virtual NIC)[7]. According to the way in which they handle guest operating systems, virtualization softwares may be broken down into two categories.

The former is Paravirtualization. This technique relies on modifying guest operating systems to cooperate with the hypervisor via *hypercalls*. Even if this means presents good performance, portability remains its main drawback. The popular Xen hypervisor relies in this technique [8].

The latter, which is called Full virtualization, enables running VMs without any OS modification. This method relies on the Binary Translation technique which enables the hypervisor to intercept sensitive instructions during guest OSs executions, and converts them into hypervisor instructions [9]. The most well-known hypervisor using Binary Translation technique is VMware ESX Server [10].

Interposition at the virtualization layer level enables many interesting functions [11] such as network traffic inspection, out-of-VM security management, and flexibility by enabling VM migration. Unfortunately, interposition leads to significant performance deterioration. It impacts throughput, latency and CPU load [12,13]. To reduce the hypervisor complexity and improve overhead, hardware builders have introduced many functions at the hardware level to assist software virtualization [14,15]. Despite the many benefits of *hardware assisted virtualization*, these new features bring several security challenges highlighted in the underlying section.

2.2 Elastic resource provisioning

The subscription-based *pay-as-you-use* model associated with rapid elasticity is arguably the most attractive service of the Cloud. Autonomic elasticity consists in dynamically resizing allocated resources according to VM workload fluctuation, allowing customers to pay only for the used resources. In order to ensure the fair sharing of resource between the co-localized VMs, the hypervisor sets a resource cap for each VM. The main challenge faced by the autonomic resource provisioning systems is the proper tuning of the resource caps. Over-estimation is wasteful since the Cloud provider provisions unused resources, while under-estimation is ineffective since it tends to violate the Service Level Objectives (SLO). Thus, it is crucial for Cloud providers to find the right trade-off between maximizing cost effectiveness and meeting SLO requirements.

Current virtualization platform implementations maximize cost effectiveness and hardware resources utilization through *overcommitment*. Resource overcommitment con-

sists in configuring to VMs more resources than the total available capacity of the host, expecting that the VMs do not use all the resources that have been configured to each of them. If resource contention occurs, the hypervisor migrates one VM to another host having enough available resources [16].

Some emerging approaches aim at automatically readjusting the resource cap. The re-calculation of this cap may be achieved by two ways. The first one consists in configuring some pre-established rules, which when verified, trigger the cap readjustment. Such a system is implemented in [17,18]. The second one is based on adaptive resource prediction models in order to estimate the coming VM behavior [19]. This latter way still suffers from over and under-estimation errors.

Resizing the amount of VMs assigned resources is still challenging. Two main methods have been proposed to deal with workload fluctuation. The *scaling-up* method – vertical scalability – consists for the underlying infrastructure to provide (or release) resources to the VM *within* the hosting physical machine the VM is running on, while the operating system is running. Unfortunately, this method is not yet usable since current operating systems do not support on-the-fly modification (without rebooting) of the amount of resource allocated to them. The method referred to as *scaling-out* – horizontal scalability – consists in aggregating resources of several hosts in a same virtual pool by duplicating (or removing) instances of VM while load balancers distribute the load among the different VM instances hosted on the different servers [20].

We now first survey the literature related to some vulnerabilities of virtualization platforms aiming at hurting strong isolation between VMs, while some elasticity vulnerabilities are considered for their impacts on strong isolation in Section 4.

3 Survey of accepted isolation-related vulnerabilities

In this section, we classify accepted attacks among four categories according to the results of some hypervisor-related code and design vulnerabilities.

3.1 Hypervisor subversion

As stated above, the hypervisor is the entity responsible for running and containing the VMs. Therefore, the security level of the VMs is strongly bound to the robustness of the hypervisor. It has been said that the hypervisor may be trusted since it contains few lines of code and is consequently easy to audit [21,22]. However, current hypervisor implementations are always aiming at incorporating more features such as network monitoring or virtual appliance integration [23], which increases the attack surface of the hypervisor [24]. After all, even if hypervisors are smaller than legacy operating systems, hypervisors remain software and they certainly suffer from bugs and flaws, as regularly reported in the Common Vulnerabilities and Exposures Database [25].

The main sources of hypervisor vulnerability are the software flaws.

In [26], Ormandy has audited the most popular virtualization platforms, including *VMware Workstation* [27], *VMware ESX*, *XEN* and *Qemu* [28], using fuzzing tools.

These tools generate random byte sequences and I/O port activity within the VMs until an error or a crash occurs. Results show that none of the audited hypervisor versions does resist to the test. Multiple flaws have been exploited, notably buffer overflows or flaws at the VM driver level which enable a malicious VM to escape from the VM state and to take the full control of the hypervisor.

VM Escape is very critical because it enables the worst cases of isolation breakout once acquired the control of the hypervisor such as, i) VM hopping in which the attacker accesses and controls co-localized VMs or, ii) resource starvation in which the attacker prevents VMs from using their allocated resources.

Another vector of vulnerability comes from the hardware platform. The growing complexity of the hardware increases significantly the hypervisor attack surface.

One of the first attacks exploiting *Hardware Assisted Virtualization* is the Blue Pill Rootkit [29]. In 2006, Blue Pill have exploited the AMD pacifica hardware extensions by manipulating control structures in order to install a malicious hypervisor in a running OS, which turns the OS to a guest state and lets the attacker taking the full control of the system. Latter, in 2008, Blue Pill has been performed against the Xen hypervisor running on *nested based hardware virtualization* allowing an attacker to get the full control of the hypervisor. Once the attacking VM has such privilege, it enables it to switch the running hypervisor from host state to VM state.

In [30], L. Duflot et al. survey and discuss about the feasibility of many relevant hardware exploited flaws such as RAM-based backdoor, human interaction device corruption and several other flaws at the CPU and Chipset level.

Hypervisor subversion exploits have been demonstrated in most of the virtualization softwares [31,32,33,34]. However, it is prominent to note that most of these attacks require to gain Dom0 privilege. This may be obtained by exploiting misconfiguration or flaws in the device drivers.

3.2 Theft of Resource

The goal of 'theft of resource' attacks – often referred to as 'theft of service' – is to unduly obtain resources at the expense of the provider or at the one of co-localized VMs possibly preventing them from properly benefiting from their own service subscription. This type of isolation breakout may result in billing issue, cross-VM performance degradation, resource starvation and denial of service.

In [35], the authors exploit the Xen scheduler design vulnerabilities to demonstrate the effectiveness of this type of attack. The key idea of the implemented attack consists in exploiting the periodicity and the predictability of the Xen scheduling mechanism. To ensure fair sharing, the Xen scheduler assigns and debits the vCPU credits allocated to each VM at a fixed frequency [36]. Hence, a malicious VM can bypass the foreseeable debiting stage by deliberately interrupting itself – entering in the sleep mode – just before the fixed-frequency checking occurs. This allows the VM to keep its allocated credit not debited and consequently to gain greater priority for future vCPU use. Moreover, such a malicious VM can preempt the currently running VMs since the Xen scheduler enables interrupted VMs to have higher execution priority on the currently running VMs sharing the same CPU core (known as the Boost priority mode [37]).

The authors have demonstrated in labs that a malicious VM can, this way, consume up to 98% of the CPU core shared with co-localized VMs (configured with the same fair share and priority levels). They have also performed the attack on Amazons Elastic Compute Cloud (EC2) platform which runs on the Xen virtualization platform. In this latter case, a malicious VM can consume up to 85% of the idle CPU core (which was 40% exceeding the amount of vCPU allocated to the VM), although VMs run in a *non-conservative-work* mode which in principle does not allow any use of unallocated idle vCPU. On EC2 platform, the attack was unable to steal the VM-allocated vCPU; this lets the authors believe that the EC2 Xen has been patched against cross-VM theft of service attacks.

3.3 Information leakage through covert channel attacks

The sharing of resources between the different VMs makes the Cloud a vulnerable target to *covert channel attacks*. With these attacks, a malicious VM can totally bypass the hypervisor security policy to steal information without subverting the hypervisor. According to [38], *covert channels involve two or more processes collaborating to communicate via a shared resource that they can both affect and measure*. There are several types of covert channel attacks. The most prevalent ones are the 'side channel attacks' and the 'timing covert channel attacks'. In side channel attacks, a process probes the use of a shared physical resource made by processes that are not aware of this probing, in order to deduce some information. With timing covert channel attacks, two or more cooperative processes intentionally communicate information by manipulating and probing the timing and the sequencing of the use of a shared resource, in such a way that the involved processes have an agreed understanding of the performed manipulation.

Regarding the side channels, an attack is demonstrated in [39]. The authors use the energy consumption footprint to determine the exact combination of VMs running in the same host. This attack may be useful for verifying the presence of a given target service on a host, in the aim of subverting it.

For timing covert channels, the most easily exploitable shared resource is the CPU. This is because modifying the load does not require any administrator privilege and CPU cores are often shared between multiple tenants. In [40], the authors achieve data leak using CPU load variation on the Xen hypervisor. The attack requires to take the control of two VMs running in the same host. The first VM belongs to the victim tenant, in which the attacker has to install a spyware responsible for reading confidential data such as credit card numbers. The second VM, called the attacking VM, remains under the control of the attacker and collaborates with the remote spyware which is in charge of performing meaningful manipulation of the shared physical CPU to let the attacking VM deduce the stolen data. This attack is not viewable from firewalls or Intrusion Detection Systems on the virtual network because the spyware does not send the stolen data on the network (but through the covert channel). Results show that the two processes can communicate at 0.49 bps with 91% to 100% accuracy in advantageous environments where co-localized VMs consume little CPU time.

The most noteworthy attack exploiting covert channels is presented in [41]. In this work, T. Ristenpart et al. demonstrate the effectiveness of a covert channel attack on the

Amazon EC2 platform, to discover VMs co-residency. The attack includes two steps. The first step consists in the placement of the malicious VM in the same host as the target VM. This is achieved by combining external and internal network probing to detect evidences of VMs co-residence. The second step consists in verifying the co-residency via *hard-disk-based* timing covert channel. The authors have also demonstrated CPU caches-based covert channels for information leak purpose, between VMs sharing the same CPU core on the Xen platform.

In [42], the authors quantify the limits of CPU cache-based cross-VM covert channel attacks similar to the one conducted in [41], both in laboratory and on EC2 platform, varying the hardware characteristics, the running VM workload load and the hypervisor configuration. They observed that even if they succeed in achieving a better bit rate than the one reported in [41], the information that may be stolen remains limited to small data.

3.4 Performance Degradation

'Performance isolation' is the property ensuring that one VM will not suffer from intensive resource consumption made by a co-localized VM. While the use of the allocated CPU and memory seems to be perfectly partitioned on a per VM basis, current hypervisor implementations potentially fail in ensuring this performance isolation property when applied to the processing of the I/O traffic. From a security perspective, this lack of I/O peformance isolation raises two main problems. Firstly, a bystander VM may suffer from an external attack targeting a co-localized VM. Secondly, a malicious tenant can exploit this flaw by running intensive I/O tasks at its own VM level (e.g. auto network flooding), in order to lead co-localized VMs to experience SLO violation.

For security reasons, some virtualization platforms such as Xen, contain device drivers in a privileged VM, called the *Driver Domain*. With this type of models, all virtual machines share the same physical device driver for I/O traffic sorting [43]. Therefore, this model increases the risk of interference between tenants, since the hypervisor is unable to ensure fair sharing of the driver domain CPU consumption required to process I/O traffic, among the hosted VMs.

In order to quantify the performance degradation observed by one VM when exposed to co-localized intensive VMs, Jeanna et al. [44] have performed a suite of intensive stress tests on various hypervisors, with several profiles of intensive VM. They have considered memory, CPU, disks and network resources. Even if the results reported in this work are not alarming in terms of response time, the tests reveal that, in the Xen hypervisor, innocent VMs experiment deficient isolation of network transmission and receiving. This might be more critical for latency-sensitive processes.

In [45], Barker et al. evaluate impacts of intensive VM loads on the response time of latency-sensitive applications running in co-localized VMs. They demonstrate that the performance of latency-sensitive tasks is impacted by the activity spikes of co-localized tenants. Results show that network and disk resources are the ones suffering the most from the lack of isolation (*degradation of 75% for disk-bound latency-sensitive tasks*[45]).

Following this presentation of accepted attacks, the next section considers the existence of new generation of vulnerabilities exploiting elasticity mechanisms – presented in Section 2 – when implemented in multi-tenant environments.

4 Security considerations for elasticity

With the growing maturity of the Cloud, conventional attacks such as hypervisor subversion will be more and more challenging. The attackers in the near future may try to look for new attack strategies which could enable them to disturb Cloud platforms more easily.

Tailoring dynamically the assigned resources to the needs of the hosted VMs is the major attractive feature of Cloud. However, we consider that elasticity may be a source of vulnerabilities threatening strong isolation if this new paradigm is not well controlled. Mechanisms in charge of automatically adjusting resource provisioning to elastic resource demands are by nature sensitive to VMs resource consumption which can be easily manipulated by attacks.

4.1 Unintended VM migration

Live migration enables virtualization platforms to ensure high flexibility and availability by migrating VMs from an over-loaded host to another host having enough resources. Migration may also be triggered for other reasons such as cross server load balancing, energy management and maintenance.

The migration mechanism is however not costless. Migrating a VM consists in transferring its memory pages and CPU state from a source to a destination host. This operation is intensive and leads the migrated VM to observe a down time and some performance deterioration. According to [46], the migrated VM may experience a down time ranging from 60 ms to 3 seconds depending on the hosted applications behavior, and a slow down by up to 20% during the whole migration time for an ordinary web server. Several parameters may worsen these results. Migration cost depends on network link bandwidth, page dirty rate, pre- and post-migration overheads [47] and available amount of resources on the source and destination hosts [48].

Two observations worth attention. Firstly, when resource contention occurs, the migrated VM is not necessarily the one that requested additional resources. Hence, a bystander VM may be migrated and consequently unfairly observe a down time. Secondly, the consumption of the migration process may impact the performances of the other co-localized VMs (not migrating) during the whole migration period, in case of severe resource contention.

Thus, an attacker deliberately fluctuating the resource consumption of its own VM may trigger abusively unintended migrations in the aim of, i) threatening the availability of the migrated VM, ii) causing performance deterioration for the co-localized VMs, iii) wasting resources because of the intensive activity performed by the source and destination hypervisors during the whole migration time.

Depending on the number of migrations triggered by the attacker, this vulnerability may have more or less consequences on the Cloud infrastructure stability.

4.2 Attack Propagation

Scaling-out mechanisms tend to increase the number of distributed applications running within the Cloud. With these systems, when a running VM claims additional resources, the Cloud resource manager will replicate this VM and instantiate the VM replica in another host. These two VMs belong to the same tenant and continue to run the same kind of applications.

From a security perspective, if the origin VM is under the control of an attacker, this replication may result in the propagation to the VM replica, of the attack already running in the origin VM.

As presented for some of the above vulnerabilities, one malicious VM behavior can affect the performances of co-localized VMs, threatening the Cloud platforms stability. If the VM to be replicated presents such a malicious behavior, the consequence of this attack may be the propagation of the performance degradation across several servers. Thus, we believe that this type of scaling systems may enable attackers to increase the effects of their attack since an attacker deliberately claiming more resources will gain the control of several VMs across the Cloud Infrastructure.

In [49], authors have demonstrated an effective propagation of an anomaly (bug, attack, SLO violation) across physical machines running distributed applications in the Cloud. For the best of our knowledge, there is no work demonstrating propagation of attack effects to the VMs co-localized with the VM replica, in case of attack-driven VM replication.

4.3 Economic Denial of Service

Predictable systems are ones of the most promising approaches enabling autonomic resource cap readjustment. These methods rely on the learning of VMs behavior in order to build their corresponding workload pattern. To be congruent to the Cloud model, these systems must take into account the dynamic variations in workload requests, by regularly updating predicted values.

These systems may be particularly vulnerable to resource starvation attacks, such as flooding attacks or malicious processes deliberately claiming more resources. This is because the Cloud resource manager will readjust the cap and provide more resources in order to cope with the workload need, while being unaware that the source of resource demand is under the control of an attacker.

The first issue raised by this attack relates to the billing. A customer may object to pay for resources consumed by an attack. Other possible side effects are migration of innocent VMs or performance deterioration and SLO violation of co-localized VMs.

The following section engages a discussion in order to highlight our understanding of the Cloud security distinctiveness regarding the vulnerabilities presented so far.

5 Discussion

In this paper, we have described some hypervisor vulnerabilities – implementation or design vulnerabilities – of the resource sharing mechanisms, that might be exploited in the context of multi-tenancy.

In table 1, we propose a representation in two dimensions of the previously presented attacks which exploit these vulnerabilities. The rows represent the results of the attacks, such as categorized in the paper among the various sub-sections. The columns represent the entity which the attacker manipulates to perform the attack. That may be the attacker's own VM ('My VM'), an attacked VM or the hypervisor. Labels indicate the bibliography references of the attacks. No label indicates that no proof of concepts has as yet been reported; such plots refer to attacks that could exploit the vulnerabilities regarding elasticity that we discussed in Section 4.

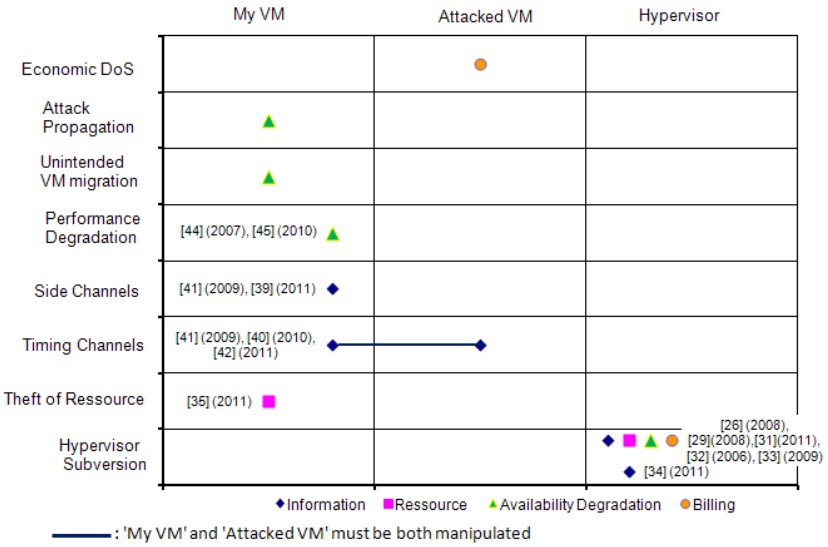


Table 1. Grouping of the main attacks according to the manipulated entity

The attacks are plotted using different symbols to let appear the kind of 'disturb' observed by the victim VM. That may be information theft, resource theft, availability degradation or attack-driven billing.

This table highlights two main observations specific to Cloud on virtualized platforms:

- The third column of this table confirms that the hypervisor is the cornerstone of the security of entire cloud stack, since attacks running against the hypervisor – exploiting implementation hypervisor vulnerabilities – let the victim VM suffer from the four previously defined types of 'disturb',

- The first column lets appear attacks which manipulate only the attacker's VM resources to disturb co-localized VMs – these latter ones being the victim VMs – by exploiting hypervisor design vulnerabilities. This kind of attacks runs *within* the

attacker's VM space, where the attacker has obviously no VM-attached security means to bypass, and *out* of the victim VM space, where the VM-attached security means is unable to protect the victim VM because its resources are not directly manipulated.

These observations let us consider that VMs may be not properly protected with pure user-centric security models, due to the fact that such models fail to take into account the out-of-VM execution context. Moreover, we assume that there is a need for the hypervisor to ensure some of the required VM security, enlarging hypervisor self-protection to hypervisor-delivered protection of the VMs (against the exploitation of the hypervisor design vulnerabilities).

In conclusion, the analysis provided in this paper lets us consider that conventional security approaches may not be sufficient to ensure an efficient protection of Cloud-hosted VMs, since these approaches do not well take into account the shared nature of virtualized Cloud platforms. We believe that Cloud infrastructures require security mechanisms which take into consideration an enlarged set of security attributes covering i) VMs local execution context, for example with location attributes, and ii) Cloud-level global execution context, for example with numbers of observed migrations within the Cloud. Such means could improve VMs security and Cloud platforms stability in the aim of increasing Cloud dependability.

To argue the relevance of the questions that we raise in this paper regarding resource management security in Cloud, we are currently developing practical proofs of concepts on possible hypervisor design vulnerabilities derived from the elasticity feature.

References

1. Timothy Grance Peter Mell. The nist definition of cloud computing, September 2011.
2. CSA. Security guidance for critical areas of focus in cloud computing. 2009.
3. Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, December 2008.
4. Dustin Owens. Securing elasticity in the cloud. *Commun. ACM*, 53(6):46–51, June 2010.
5. Vyas Sekar and Petros Maniatis. Verifiable resource accounting for cloud computing services. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 21–26, New York, NY, USA, 2011. ACM.
6. R. P. Goldberg. Architecture of virtual machines. In *Proceedings of the workshop on virtual computer systems*, pages 74–112, New York, NY, USA, 1973. ACM.
7. Mendel Rosenblum and Tal Garfinkel. Virtual machine monitors: Current technology and future trends. *Computer*, 38(5):39–47, May 2005.
8. Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the nineteenth ACM symposium on Operating systems principles*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.
9. Ole Agesen, Alex Garthwaite, Jeffrey Sheldon, and Pratap Subrahmanyam. The evolution of an x86 virtual machine monitor. *SIGOPS Oper. Syst. Rev.*, 44(4):3–18, December 2010.

10. Carl A. Waldspurger. Memory resource management in vmware esx server. *SIGOPS Oper. Syst. Rev.*, 36(SI):181–194, December 2002.
11. Tal Garfinkel and Mendel Rosenblum. When virtual is harder than real: Security challenges in virtual machine based computing environments. In *In 10th Workshop on Hot Topics in Operating Systems*, 2005.
12. Carl Waldspurger and Mendel Rosenblum. I/o virtualization. *Commun. ACM*, 55(1):66–73, January 2012.
13. James E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, May 2005.
14. Intel. Pci-sig single root i/o virtualization (sr-iov) support in intel® virtualization technology for connectivity. Technical report.
15. Virtual machine device queues. an integral part of intel® virtualizationtechnology for connectivity that delivers enhanced network performance. Technical report.
16. Timothy Wood, Prashant Shenoy, Arun Venkataramani, and Mazin Yousif. Black-box and gray-box strategies for virtual machine migration. In *Proceedings of the 4th USENIX conference on Networked systems design & implementation*, NSDI'07, pages 17–17, Berkeley, CA, USA, 2007. USENIX Association.
17. Benny Rochwerger, David Breitgand, Amir Epstein, David Hadas, Irit Loy, Kenneth Nagin, Johan Tordsson, Carmelo Ragusa, Massimo Villari, Stuart Clayman, Eliezer Levy, Alessandro Maraschini, Philippe Massonet, Henar Munoz, and Giovanni Toffetti. Reservoir - when one cloud is not enough. *Computer*, 44:44–51, 2011.
18. http://aws.amazon.com/fr/autoscaling/.
19. Zhiming Shen, Sethuraman Subbiah, Xiaohui Gu, and John Wilkes. Cloudscale: elastic resource scaling for multi-tenant cloud systems. In *Proceedings of the 2nd ACM Symposium on Cloud Computing*, SOCC '11, pages 5:1–5:14, New York, NY, USA, 2011. ACM.
20. Luis M. Vaquero, Luis Rodero-Merino, and Rajkumar Buyya. Dynamically scaling applications in the cloud. *SIGCOMM Comput. Commun. Rev.*, 41(1):45–52.
21. Peter M. Chen and Brian D. Noble. When virtual is better than real. In *Proceedings of the Eighth Workshop on Hot Topics in Operating Systems*, HOTOS '01, pages 133–, Washington, DC, USA, 2001. IEEE Computer Society.
22. Mendel Rosenblum. The reincarnation of virtual machines. *Queue*, 2(5):34–40, July 2004.
23. Shudong Zhou. Virtual networking. *SIGOPS Oper. Syst. Rev.*, 44(4):80–85, December 2010.
24. Yanpei Chen, Vern Paxson, and Randy H. Katz. Whats new about cloud computing security? Technical Report UCB/EECS-2010-5, EECS Department, University of California, Berkeley, Jan 2010.
25. http://web.nvd.nist.gov/view/vuln/search.
26. Tavis Ormandy. An empirical study into the security exposure to hosts of hostile virtualized environments. 2008.
27. Jeremy Sugerman, Ganesh Venkitachalam, and Beng-Hong Lim. Virtualizing I/O devices on Vmware workstation's hosted virtual machine monitor. In *USENIX Ann. Technical Conf. (ATC)*, volume 15, pages 1–14, New York, NY, USA, November 2001. ACM.
28. http://wiki.qemu.org/main-page.
29. Joanna Rutkowska and Alexander Tereshkin. Bluepilling the xen hypervisor. *Black Hat USA*, 2008.
30. Loic Duflot, Olivier Grumelard, Olivier Levillain, and Benjamin Morin. On the limits of hypervisor- and virtual machine monitor-based isolation. In Ahmad-Reza Sadeghi and David Naccache, editors, *Towards Hardware-Intrinsic Security*, Information Security and Cryptography, pages 349–366. Springer Berlin Heidelberg, 2010. 10.1007/978-3-642-14452-316.
31. Nelson Elhage. Virtunoid: A kvm guest ! host privilege escalation exploit. In *Black Hat USA 2011*, 2011.

32. Samuel T. King, Peter M. Chen, Yi min Wang, Chad Verbowski, Helen J. Wang, and Jacob R. Lorch. Subvirt: Implementing malware with virtual machines. In *In IEEE Symposium on Security and Privacy*, pages 314–327, 2006.
33. Kostya Kortchinsky. Cloudburst: A vmware guest to host escape story. *Black Hat USA*, 2009.
34. Martin Mulazzani, Sebastian Schrittwieser, Manuel Leithner, Markus Huber, and Edgar Weippl. Dark clouds on the horizon: using cloud storage as attack vector and online slack space. In *Proceedings of the 20th USENIX conference on Security*, SEC'11, pages 5–5, Berkeley, CA, USA, 2011. USENIX Association.
35. Fangfei Zhou, M. Goel, P. Desnoyers, and R. Sundaram. Scheduler vulnerabilities and coordinated attacks in cloud computing. In *Network Computing and Applications (NCA), 2011 10th IEEE International Symposium on*, pages 123 –130, aug. 2011.
36. Ludmila Cherkasova, Diwaker Gupta, and Amin Vahdat. Comparison of the three cpu schedulers in xen. *SIGMETRICS Perform. Eval. Rev.*, 35(2):42–51, September 2007.
37. See hwan Yoo, Kuen-Hwan Kwak, Jae-Hyun Jo, and Chuck Yoo. Toward under-millisecond i/o latency in xen-arm. In Haibo Chen, Zheng Zhang, Sue Moon, and Yuanyuan Zhou, editors, *APSys*, page 14. ACM, 2011.
38. Nate Lawson. Side-channel attacks on cryptographic software. *IEEE Security and Privacy*, 7:65–68, 2009.
39. H. Hlavacs, T. Treutner, J. Gelas, L. Lefevre, and A. Orgerie. Energy consumption side-channel attack at virtual machines in a cloud. In *Dependable, Autonomic and Secure Computing (DASC), 2011 IEEE Ninth International Conference on*, pages 605 –612, dec. 2011.
40. Keisuke Okamura and Yoshihiro Oyama. Load-based covert channels between xen virtual machines. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 173–180, New York, NY, USA, 2010. ACM.
41. Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*, CCS '09, pages 199–212, New York, NY, USA, 2009. ACM.
42. Yunjing Xu, Michael Bailey, Farnam Jahanian, Kaustubh Joshi, Matti Hiltunen, and Richard Schlichting. An exploration of l2 cache covert channels in virtualized environments. In *Proceedings of the 3rd ACM workshop on Cloud computing security workshop*, CCSW '11, pages 29–40, New York, NY, USA, 2011. ACM.
43. Diwaker Gupta, Ludmila Cherkasova, Rob Gardner, and Amin Vahdat. Enforcing performance isolation across virtual machines in xen. In *Proceedings of the ACM/IFIP/USENIX 2006 International Conference on Middleware*, Middleware '06, pages 342–362, New York, NY, USA, 2006. Springer-Verlag New York, Inc.
44. Jeanna Neefe Matthews, Wenjin Hu, Madhujith Hapuarachchi, Todd Deshane, and Demetrios Dimatos Gary Hamilton. Quantifying the performance isolation properties of virtualization systems. In *In Proc. of the Workshop on Experimental Computer Science (ExpCS*. USENIX Association, 2007.
45. Sean K. Barker and Prashant Shenoy. Empirical evaluation of latency-sensitive application performance in the cloud. In *Proceedings of the first annual ACM SIGMM conference on Multimedia systems*, MMSys '10, pages 35–46, New York, NY, USA, February 2010. ACM Press.
46. William Voorsluys, James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Cost of virtual machine live migration in clouds: A performance evaluation. In *Proceedings of the 1st International Conference on Cloud Computing*, CloudCom '09, pages 254–265, Berlin, Heidelberg, 2009. Springer-Verlag.

47. Sherif Akoush, Ripduman Sohan, Andrew Rice, Andrew W. Moore, and Andy Hopper. Predicting the performance of virtual machine migration. *Modeling, Analysis, and Simulation of Computer Systems, International Symposium on*, 0:37–46, 2010.
48. Yangyang Wu and Ming Zhao. Performance modeling of virtual machine live migration. In *Proceedings of the 2011 IEEE 4th International Conference on Cloud Computing*, CLOUD '11, pages 492–499, Washington, DC, USA, 2011. IEEE Computer Society.
49. Hiep Nguyen, Yongmin Tan, and Xiaohui Gu. Pal: Propagation-aware anomaly localization for cloud hosted distributed applications. In *Managing Large-scale Systems via the Analysis of System Logs and the Application of Machine Learning Techniques*, SLAML '11, pages 1:1–1:8, New York, NY, USA, 2011. ACM.

Verified Secure Kernels and Hypervisors for the Cloud

Matthieu Lemerre[1], Nikolai Kosmatov[2], and Céline Alec[2]

[1] CEA, LIST, Embedded Real-Time System Laboratory, PC 172, 91191 Gif-sur-Yvette, France.

[2] CEA, LIST, Software Reliability Laboratory, PC 174, 91191 Gif-sur-Yvette, France.

firstname.lastname@cea.fr

Abstract. Cloud-computing systems share hardware resources (CPU time and memory) between mutually untrusted applications. As such, they must provide isolation barriers between these applications, but must also secure resource sharing such that applications cannot stop, slow down, or provoke incorrect resource accounting of other applications. These isolation barriers are implemented by a trusted operating system kernel, which must be built according to security principles.
Confidence in the system can be further increased with the help of tools for formal verification and proof of programs. Using these tools is eased because thorough application of security principles leads to a small system, but still poses many challenges for formal verification, like concurrency and the need to model the behavior of the hardware.
This paper shows how modern tools for proof of programs can be applied to verification of secure kernels and hypervisors for the Cloud. We illustrate this approach on a critical module of a prototype Cloud hypervisor, called Anaxagoros, using the Frama-C software verification platform.

1 Introduction

Secure systems are traditionally built according to design principles that decompose the system into *isolated* components with minimal *rights*, and with communication between components tightly controlled.

This isolation is very important for cloud computing, that executes applications from mutually untrusted users: a failure or attack from an application of a user must not interfere with applications of other users (or other applications of the same user).

But cloud systems have another requirement, which is to mutualize the resources of the system on which they are built, so as to maximize efficiency. Resources of the system must be shared *securely* between the tasks, for instance, in order to make denials of service impossible (i.e. to prevent an application to slow down or stop another one, for instance by requesting all resources), or to correctly account for the memory and CPU time spent executing the applications, and thus, to correctly bill the clients. Secure resource sharing requires to

complement the traditional behavioral security principles [1] with new *resource security* principles.

The Anaxagoros [2,3] system has been designed to maximize both traditional and resource security. It is composed of:

- A *microkernel*, that implements the isolation between tasks, controls access to the services; it is trusted by all tasks in the system;
- Some *services*, each sharing one resource (memory, network...) securely between the tasks, and being trusted only by the tasks that need the resource;
- *Libraries*, helping to use the resources in the system (e.g. the C library), and being trusted only by the task that uses them.

The amount of system code that must work properly to correctly execute a program is called its *trusted computing base (TCB)*. The implementation of Anaxagoros tries to put the code first into libraries then into shared services, and then into the kernel, leading to minimizing the TCB of each task of the system. Critical tasks can minimize the amount of shared services used and avoid using the provided libraries, thereby having a minimal TCB.

To minimize TCB, the kernel and services present a low-level interface (like exokernels [4]): they consist in a thin layer that only refuses or authorizes operations requested by user applications. This low-level interface also allows efficient virtualization of operating systems [5], such as Linux, to provide security and isolation between existing systems.

In particular, the Anaxagoros kernel is globally trusted, even by critical tasks, which makes it the most critical component of the system. The fact that it is minimized (approximately 3500 lines of code) decreases the likeliness of bugs and helps in code reviews. But its minimality and criticality make it a good target to further increase confidence in the system, by providing a formal proof that the kernel offers the required security functions.

Verification of such system software represents an interesting and challenging target for software verification because of its critical and complex functions, such as concurrency, the use of assembly code, and the need to model interactions with the hardware.

This paper is organized as follows. Sec. 2 presents a short tutorial on proof of programs with the JESSIE tool. Sec. 3 introduces basic system security rules, describes Anaxagoros microkernel and in particular its virtual memory system. Sec. 4 shows how this system it can be formally verified. Sec. 5 and 6 provide related work and conclusion.

2 A short tutorial on proof of programs

In this section we show how a program can be formally verified using automatic tools for proof of programs.

We use FRAMA-C [6,7], an open-source platform dedicated to analysis of C programs, developed at CEA LIST. FRAMA-C has a plugin-oriented architecture that allows the user to run analyzers already available in the platform

```
1 /*@ ensures \result >= a && \result >= b &&
2         ( \result == a || \result == b );
3     assigns \nothing;
4 */
5 int max(int a, int b){
6   if( a > b )
7     return a;
8   else
9     return b;
10 }
```

Fig. 1. File `max.c` with a function returning the maximum of its inputs `a` and `b`

as well as to develop new plugins. FRAMA-C offers various analyzers, many of them are open-source. They implement a wide range of modern software analysis techniques, such as abstract interpretation, impact analysis, dependency analysis, program slicing, pointer analysis, weakest precondition, etc. The structural test generation tool PATHCRAWLER [8,9] is also available as a FRAMA-C plugin. FRAMA-C contains two plugins for proof of programs, JESSIE and WP.

Proof of programs, also known as *theorem proving,* is a powerful technique of program verification that provides a formal mathematical proof that the program meets its specification. While the theoretical foundations [10,11] of this approach were developed in the 1970s, its automation became possible only during the last decade thanks to the spectacular progress made by the developers of modern program verification tools.

Among them, automatic theorem provers, like Simplify [12], ALT-ERGO [13,14] and Z3 [15], are capable to perform simple proofs automatically. On the other hand, interactive provers, such as COQ [16,17,18], ISABELLE [19,20], and HOL [21,22], allow the engineer to indicate proof steps that are then checked by the tool. Interactive provers may be suitable for more complex properties but require human interventions.

In this approach, to prove a program p, one proceeds in the following way.

- First, each function f in the program source code must be *annotated,* or *specified,* by inserting into the code special clauses, or *annotations*, indicating the hypotheses h and conclusions c. They describe the state of program variables at different execution points and program actions. Typically, the hypotheses describe the state before the function is called, while the conclusions may specify how this state is modified by the function, and the return values. In this manner, each function receives a *specification,* or a *contract.* Basically, the verification task is then reduced to the proof of the implication $h \Rightarrow c$.
- Next, these annotations are used to compute *proof obligations*, i.e. propositions that must be proved to ensure that if the hypotheses h are verified, then the conclusions c hold. The proof obligations can be very complex since they should take into account all program instructions including variable assignments, conditionals, loops, other function calls, etc.
- Finally, a theorem prover is called to prove the proof obligations.

```
1  /*@ requires l >= 0;
2      requires \valid(a + (0..(l-1)));
3      requires \forall integer i, j; (0 <= i <= j < l ==> a[i] <= a[j]);
4
5      assigns \nothing;
6
7      behavior present:
8        assumes \exists integer i; (0 <= i < l && a[i] == x);
9        ensures 0 <= \result < l;
10       ensures a[\result] == x;
11
12     behavior absent:
13       assumes \forall integer i; (0 <= i < l ==> a[i] != x);
14       ensures \result == -1;
15 */
16 int searchInArray(int* a, int l, int x){
17   int k;
18
19   /*@ loop invariant 0 <= k <= l &&
20         \forall integer i; 0 <= i < k ==> a[i] < x;
21       loop assigns \nothing;
22       loop variant l-k;
23   */
24   for(k = 0; k < l; k++){
25     if(a[k] == x)
26       return k;
27     else if(x < a[k])
28       return -1;
29   }
30   return -1;
31 }
```

Fig. 2. Function `searchInArray` looks for element `x` in sorted array `a` of length `l` and returns the index of this element in `a` if it is found, or -1 otherwise

Frama-C analyzers share a common specification language called acsl (ANSI/ISO C Specification Language) [23]. Let us show how a C program can be specified in acsl and proved in the jessie plugin of Frama-C.

Example 1. Fig. 1 shows a simple example of function `max` returning the maximum of its two inputs, specified in acsl. The `ensures` clause provides a postcondition. Here it states that the return value `\result` must be not less than both inputs, and equal to at least one of them. The `assigns` clause specifies the (non local) variables that the function is allowed to modify. All other variables accessible outside the current function call cannot be affected by the function. Thus this clause also expresses a postcondition. In Fig. 1, this clause (line 3) states that the function should not modify any non local variable. Running the proof of this program in jessie with the command `frama-c -jessie max.c` proves the program. In other words, it finds a formal mathematical proof that the program meets its specification.

Example 2. Fig. 2 shows the function `searchInArray` specified in acsl using behaviors. *Behaviors* provide a very convenient notation when it is necessary to specify the function separately in several cases. Common precondition and postcondition can be provided and must be true for all cases, and a default behavior can be used to specify the default case (see [23] for more detail). A behavior

applies to the situation when its `assumes` clause is true, and in this case the postcondition (`ensures` and `assigns` clauses) must be verified. Notice that a behavior's postcondition should be true in addition to the common postcondition.

In Fig. 2, the common precondition for all behaviors includes lines 1–3. Line 1 states that the array size `l` is not negative. Line 2 specifies that the memory locations `a[0]`, ..., `a[l-1]` are valid, that is, the program can access them. Line 3 specifies that the array `a` is sorted. The common `assigns` clause at line 5 states that the function should not modify any non local variable.

Two behaviors describe separately the case when the element `x` is present in `a` (lines 7–10) and the case it is absent (lines 12–14). Behavior `present` describes the presence case defined by line 8. In this case the postconditions of lines 9–10 must be true. Similarly, the second behavior `absent` describes the absence case defined by line 13. In this case the postcondition of line 14 must be satisfied.

In order to help JESSIE to prove programs in presence of loops, specific clauses for loops may be used, such as `loop invariant`, `loop variant` and `loops assigns`. Here, lines 19–20 indicate a *loop invariant* that must be true before the loop and after any loop iteration. Line 21 indicates variables that can be modified after a number of iterations. Line 22 provides a *loop variant* V, that is, a positive, integer expression strictly decreasing after any loop iteration. It allows the tool to prove loop termination (since the positive integer V cannot decrease infinitely).

3 The Anaxagoros microkernel and hypervisor

3.1 Introduction to systems security

The basic design principles to build a secure operating system have been put forth by Saltzer and Schroeder in the famous paper "The Protection of Information in Computer Systems" [1]. The idea is to build the system as a set of small components called *domains*, isolated from one another, and with tightly-controlled communication between the components. This design brings several benefits:

- inspecting the security in one domain is easier (because it is smaller),
- it is easy to inspect the impact of the loss of one domain on the security of the system,
- the loss of one domain does not weaken the security of other domains.

A good analogy would be the design of a medieval castle: if you only have one level of fence around your large castle, then a single breach looses the entire castle to the opponent. A well-designed castle has several level of fences, with isolated small regions in the castle, and tightly-controlled pathways between the regions, to ease resisting to an attacker.

The eight design principles for designing secure systems are:

Separation of privilege: The idea is to build the system as a set of separate domains with different *privileges*, or *rights*. The domain that may have the

right to read from the network, while a second will have the right to write to a secret file; both domains must be involved to compromise the file with a remote attack.

Least common mechanisms: We call the *trusted computing base* (TCB) of a domain the set of code on which the domain depends to operate properly. The goal of the *least common mechanism* principle is to minimize the TCB of each task in the system. Indeed, a shared domain is an opportunity to breach isolation between several domains; furthermore, an error in shared code may affect several, or even all domains.

There are two main ways to interpret this principle:

- *Microkernel* systems split base *services*, such as network or hard drive access, into isolated domains; so that failure or compromission of a base service affect only the domains that need it. This decrease the TCB for each task.
- *Exokernel* systems and hypervisors decrease the amount of code in base services to put it into the upper layer code. This globally decrease the size of shared system code.

These two ways are not incompatible; for instance, Anaxagoros follows both ways.

Complete mediation: Every access to every resource or object is a privileged operation, and this privilege must be checked, i.e. there should be access control for all objects. For instance, writing to a file, reading a network packet, receiving keyboard inputs, displaying data on the screen, are all privileged operations, and the system must check that the domain that tries to perform these operations has sufficient privileges. A hidden benefit of this principle is to indentify all privileged operations, and the domain that may perform them.

Principle of least privilege states that domains should be given only the privilege of the actions they need to accomplish. The program that displays PDFs on the screen need not access the network, and so is not given this right; thus its eventual compromission does not lead to divulge secret files on the Internet. This principle limits the impact of a compromission of a domain.

Fail-safe defaults: It basically states that "whatever is not explicitly allowed, is forbidden". Forgetting to give enough privilege to a program will only prevent it to work, a problem that will be quickly found; forgetting to remove a privilege leads to a potential security breach that can remain unnoticed.

Economy of mechanism: It states that the design of the system should be as simple as possible. This simplifies reviews. Especially the mechanism for access control should be simple, as the security of the whole system relies on it. As a result, there are two main systems of access control:

- *Access control lists*, where to each resource is associated a list of the domains that can access this resource;
- *Capabilities*, where to each domain is associated the list of accessible resources.

Open design opposes "security through obscurity". The security of the system should not rely on the fact that some features are known only by the team that designed it.

Psychological acceptability states that if the security rules in a system are too complex to use or understand, they are likely to be not applied.

3.2 Anaxagoros

Anaxagoros [2,3] is a secure microkernel that is also capable of virtualizing pre-existing operating systems, for example Linux virtual machines. It is capable of executing hard real-time tasks or operating systems, for instance the PharOS real-time system [24], securely with non real-time tasks, on a single chip.

This goal has required to put a strong emphasis on security in the design of the system, and not only on traditional "behavioral" security (isolation and access control to protect confidentiality and integrity, as presented in the previous section) but also on availability (being able to slow down or steal resources from another task is considered a breach in security).

As it is a microkernel, Anaxagoros is the only piece of code that requires to run in the privileged mode of the CPU in an Anaxagoros-based system. Every piece of code that can be moved out of the kernel is placed in a separated user-level *service*, with limited rights.

This approach contributes to TCB minimization in two ways:

- first, the kernel, which is the only globally trusted piece of code, is minimized,
- second, as services are isolated, their faults do not affect applications that do not require them. For instance, a bug in a network stack would not affect a task that does not use the network.

For safety and concurrency reasons [2], the interface of the kernel and the main user services is low-level, close to the hardware (this is contrary to other microkernel approaches which attempt to provide a generic interface that abstracts the hardware). This approach also allows to classify Anaxagoros as an exokernel [4] or as an hypervisor.

This approach also contributes to TCB minimization: as the interface provides no abstraction, the code of the kernel and services becomes much simpler, as it only has to check that the required hardware operations are permitted.

The kernel generally strictly enforces Saltzer and Schroeder behavioral security principles [1]. In addition to minimizing TCB, the kernel provides protection domains using the machine's virtual memory mechanisms and controls access to shared services using capabilities.

The kernel and services are designed to prevent availability attacks, which are a problem often ignored in conventional system design. In particular the *denial of resources* attack can be made when a task can issue requests that make the kernel or a service allocate a resource (e.g. memory): by issuing a sufficient number of requests, the system can run out of memory. New resource security mechanisms and principles have been built in Anaxagoros to avoid this kind of attack (for instance the kernel does not allocate any memory, while still allowing dynamic creation of new tasks and virtual machines).

3.3 The virtual memory system of Anaxagoros

A critical component to ensure security in Anaxagoros is its *virtual memory system* [3]. The x86 processor (and many other high-end hardware architecture) provide a mechanism for *virtual memory translation*, that translates the address manipulated by a program to real address. One of the goals of this mechanism is to help organizing the program address space, for instance to allow a program to access big contiguous memory regions.

The other goal is to control the memory that a program can access; we will be focusing on that part. The physical memory is split into same-sized region, called *pages* (pages are of size 4kb on standard x86 configurations).

Anaxagoros does not decide what is written to pages; rather, it allows tasks to perform any operations on pages, provided that this does not affect the security of the kernel itself, and of the other tasks in the system.

To do that, it ensures only two simple properties. The first is that a program can only change the page that it "owns". We will not explain here how ownership is represented or checked, and rather concentrate on the second property stating that pages are used according to their types.

Indeed, the hardware mechanism works as follows: a page p is accessible if a special register b points to a page p_d, that points to a page p_t, that points to p (pointing to x means containing a pointer to x in a special format, that we call a *mapping* to p). If a page p_d is pointed by b, we say that p_d is used as a *page directory;* if p_t is pointed by a page directory, we say that it is used as a *page table*. If p is accessible, we say that it is used as a *data page*. The program can write directly to any accessible pages. To write to the other pages, it sends a request to the virtual memory algorithm in the kernel, that checks the request and performs the writing operation if it is allowed.

The key point here is that the hardware does not prevent a page table or page directory to be also used as a data frame. Thus if nothing is done, a task can change the mappings in any page table or page directory it owns. By doing the right modifications, it can access (and write to) any page, including those that it does not own.

3.4 The memory system algorithm: an overview

The goal of the algorithm we are presenting (and verifying) is to prevent these unauthorized modifications. It works by recording:

- The type of the page (the basic types are `zero`, `data`, `pagetable` and `pagedirectory`).
- The number of times the page is being used as a data page, page table, or page directory.

The types are used to ensure the following rules:

Rule 1 Only pages of type `pagedirectory` can be used as page directories;
Rule 2 Only pages of type `pagetable` can be used as page tables;
Rule 3 Only pages of type `data` can be used as data pages.

The kernel ensures these rules by checking requests for changes of page table and page directory entries, so that page directories can only point to pages of type `pagetable`, and pagetables can only point to pages of type `data`. Other requests are denied.

In other words, the algorithm ensures that pages can be used only according to their role. With these rules, we know that page directories cannot be used as data pages, and thus cannot be changed directly by the program, preventing unauthorized modifications.

Now, we allow dynamic reuse of memory, meaning that a page once used as a data page can later be used as page directory. To allow that, the type of the page has to change. But the kernel cannot allow arbitrary change of type, otherwise several types of attacks are possible, and that would lead to the first three rules becoming false.

For instance, a program can access any page p (including those it does not own) by changing a data page p' to contain a mapping to p, change the type of p' to `pagetable`, then use p' as a page table. To prevent this attack, the page p' must be cleaned by the kernel before changing the type. This is the role of the type `zero`: when the kernel receive a request to change the type of a page to `zero`, it first cleans up the contents of that page.

Rule 4 Pages can change their types only from, and to the type `zero`
Rule 5 Pages of type `zero` are filled with zeros, and thus do not point to other pages.

These rules are not sufficient, and other kinds of attacks to access p are possible, by having a page p' used simultaneously as a data page and as a page table:

- Either p' is used as a data page (of type `data`), then cleaned and changed to type `zero`, to type `pagetable`, and used as pagetable;
- Or it is used as a page table (of type `pagetable`), then cleaned and changed to type `zero`, to type `data`, and used as data page.

After both situations, even if p' has been cleaned, nothing prevents the attacker to directly add mappings to p in p'. To prevent these attacks we use a "number of mappings" counter for each page:

Rule 6 The "number of mappings" counter of a page of type `data` is equal to the number of mappings to this page in pages of type `pagetable`
Rule 7 The "number of mappings" counter of a page of type `pagetable` is equal to the number of mappings to this page in pages of type `pagedirectory`
Rule 8 The "number of mappings" counter of a page of type `pagedirectory` is equal to the number of mappings to this page in the b register (at most one for monoprocessor systems).
Rule 9 A page of type `zero` is not used as data page, page table, or page directory, i.e. its number of mappings is 0.

The kernel enforces these rules by denying requests to clean pages to type zero when their "number of mappings" counter is not zero, and by adjusting the "number of mappings" counter every time a pointer to a page is added or removed.

When these checks are present, all the above rules hold whatever the requests from the tasks. Formal proof that these rules are fulfilled by the algorithm is illustrated in Section 4.

3.5 The actual algorithm

The algorithm presented above is simplified: the actual algorithm present additional cases. In particular:

1. There are two kinds of mappings: read-only and writable. Only writable mappings need to be accounted for in this algorithm, but we also have a "number of readable mappings" counter that has other uses. For instance ensuring that there are no more mappings to a page when it changes its owner ensures absence of communication between the previous and the new owner.
2. Pages of type `pagedirectory` and `pagetable` can be used as data pages, but only in read-only mappings. Also, pages of type `pagedirectory` can be used as page directories and as page tables (i.e. they can be pointed by other pages of type `pagedirectory`, or by themselves). The "number of mappings" meaning for pages of type `pagedirectory` is changed: it is equal of the number of times it is pointed by a b register plus the number of times it is pointed by pages of type `pagedirectory`.
3. Cleaning a page is a long operation that can be interrupted. If a page cleanup is interrupted, the page's type is set to a special "partially cleaned" type, and the number of entries cleaned up so far is recorded. Types of page `data` must have a "number of mappings" counter equal to zero at the beginning of the cleanup, otherwise the page contents could be changed during the cleanup. But pages of type `pagedirectory` and `pagetable` can be cleaned up while they are still being used, because the kernel can refuse any concurrent modification. If the "number of mappings" of a page table or a page directory is non zero at the end of a cleanup, the page stays with a "partially cleaned" type, so that there are no active mappings to pages of type `zero`.

4 Anaxagoros verification

The purpose of our ongoing work is the formal verification of the Anaxagoros hypervisor. Our approach is based on the specification of the code in ACSL [23] and proving it using the Frama-C plugins jessie and wp for proof of programs. We are currently working on the proof of the virtual memory management module, one of the most critical modules.

In this section we show how critical system C code can be specified and formally verified using proof of programs. We illustrate it on a partial simplified version of Anaxogoros virtual memory module given in the Appendix. This extract focuses on cleaning `data` pages with interruptions and resuming at the right place. It takes into account the "number of mappings" counter for `data` pages (cf Sec. 3.3, 3.4).

The types `DToZero`, `PTToZero` and `PDToZero` (line 4) are assigned respectively to pages of types `Data`, `Pagetable` and `Pagedirectory` in cleanup, i.e. for which cleaning has started but has not yet terminated (it can be in progress or interrupted, cf Sec. 3.5.3). In this simplified version, the size and maximal number of pages are limited (lines 1–2), and their contents and attributes are represented by arrays (lines 5–10). `Cleaning[p]` specifies if the page `p` is being cleaned now, and `Cleaned[p]` indicates how many elements from the beginning of the page have been already cleaned if the cleaning has been interrupted.

Lines 13–37 define some predicates that will be used in the specification. For instance, `R9` (lines 26–27) states that there are no mappings to a page of type `Zero` (cf Rule 9), while `R5` (lines 28–29) states that a page of type `Zero` is filled with zeros (cf Rule 5). The predicate `ToZeroPagesStartWithZeros` (lines 30–36) specifies that the first `Cleaned[p]` elements of a page `p` in cleanup are filled with zeros, with no valid mappings. Line 37 defines the global invariant.

Page cleaning can be started by `CleanData` (lines 97–107), or resumed by `CleanPartiallyCleanedData` (lines 129–137). After necessary checks, they call `putOnePageToZero` (lines 57–76) that cleans the page from a given position. At each iteration, it checks for interruptions (line 67) modeled in this simplified version by a global variable (line 11).

This example shows that formal program specification takes more than 80% in the resulting specified C code, and writing specification represents very significant effort. Function contracts are the most important part of it, but proving the program with JESSIE may require writing additional clauses, for instance, for loop iterations (cf lines 59–64), or intermediate assertions that help the prover (about 15 lines not presented here).

100% of the 489 proof obligations generated for the code of the Appendix are proved by JESSIE (using JESSIE version 2.29, FRAMA-C Carbon version, and the provers Alt-Ergo version 0.93 and/or Simplify version 1.5.4).

5 Related work and discussion

A recent work [25] presented formal verification for the OS microkernel seL4, allowing devices running seL4 to achieve the EAL7 evaluation level of the Common Criteria [26]. Another formal verification of a microkernel was presented in [27]. In both cases, the verification used interactive, machine-assisted and machine-checked proof with the theorem prover Isabelle/HOL. Although interactive theorem proving requires human intervention to construct and guide the proof, it has the benefit to serve a general range of properties and is not limited

to specific properties treatable by more automated methods of verification as static analysis or model checking.

The formal verification of a simple hypervisor [28] used VCC [29], an automatic first-order logic based verifier for C. The underlying system architecture was precisely modeled and represented in VCC, where the mixed-language system software was then proved correct. Unlike [25] and [27], this technique was based on automated methods.

[30] reports on verification of TLB (translation lookaside buffer) virtualization, a core component of modern hypervisors. Because devices run in parallel with software, they necessitate concurrent program reasoning even for single-threaded software. The authors give a general methodology for verifying virtual device implementations, and demonstrate the verification of TLB virtualization code in VCC.

Formal verification nowadays remains rather costly. According to [31], the cost of the verification of the seL4 microkernel was around 25 person-years, and required highly qualified experts. seL4 contains only about 10,000 lines of C code, and verification cost is about $700 per line of code.

6 Conclusion and future work

Recent advances in formal verification and security engineering have shown that it is now possible to perform a formal machine-checked proof of a complete microkernel. In this paper, we presented a short tutorial on proof of programs with Frama-C, described the Anaxagoros hypervisor with its security principles, and illustrated on the Anaxagoros virtual memory system how critical system code can be specified and formally verified using modern verification tools.

There are still many interesting challenges to be addressed. An important future work perspective is the verification of a concurrent hypervisor. For instance, the verification in [25,28] was carried out for a sequential version. This research direction is extremely important for an OS or a hypervisor since concurrency naturally appears both for parallel execution on a multi-core architecture and for non-deterministic interleaving via threads on a unique processor. We expect that such verification may require the development of new algorithms and specifications, adapted for the proof of a concurrent version, in particular for the execution on multi-core processors.

Future work also includes an extension of the verification to complex mixed software and hardware designs in order to avoid that a hardware failure alters the expected behavior of a verified hypervisor.

Whatever particular verification approach is used, formal verification of a microkernel or a hypervisor represents a great effort and remains valid only for a particular version being verified. Therefore, any evolution of the software requires new verification. To allow industrial usage of formally verified system software in a real-life environment, the verification of a new version should require only a limited effort, without performing a new specification and proof of the whole system. Another important future work direction is developing formal verification

methodologies for modular proof such that any evolution has a limited impact on the verification.

References

1. Saltzer, Schroeder: The protection of information in computer systems. Communication of the ACM **7** (1974)
2. Lemerre, M., David, V., Vidal-Naquet, G.: A communication mechanism for resource isolation. In: Proceedings of the Second Workshop on Isolation and Integration in Embedded Systems. IIES'09, New York, NY, USA, ACM (2009) 1–6
3. Lemerre, M., David, V., Vidal-Naquet, G.: A dependable kernel design for resource isolation and protection. In: IIDS '10: Proceedings of the First Workshop on Isolation and Integration in Dependable Systems, ACM (2010) 1–6
4. Engler, D.R., Kaashoek, M.F., J. O'Toole, J.: Exokernel: an operating system architecture for application-level resource management. In: Proceedings of SOSP '95. (1995) 251–266
5. Legout, V., Lemerre, M.: Paravirtualizing linux in a real-time hypervisor. In: Embed with Linux (EWili) 2012. (2012)
6. Cuoq, P., Kirchner, F., Kosmatov, N., Prevosto, V., Signoles, J., Yakobowski, B.: Frama-C: A program analysis perspective. In: Proc. of the 10th International Conference on Software Engineering and Formal Methods (SEFM 2012), Springer (2012) 233–247
7. Correnson, L., Cuoq, P., Kirchner, F., Prevosto, V., Puccetti, A., Signoles, J., Yakobowski, B.: Frama-C User Manual. (October 2011) `http://frama-c.com`.
8. Kosmatov, N.: PathCrawler online (2010–2012) http://pathcrawler-online.com/.
9. Kosmatov, N., Williams, N., Botella, B., Roger, M., Chebaro, O.: A lesson on structural testing with pathcrawler-online.com. In: TAP'2012. Volume 7305 of LNCS., Prague, Czech Republic, Springer (May 2012) 169–175
10. Hoare, C.A.R.: An axiomatic basis for computer programmation. Communication of the ACM **12**(10) (1969) 576–580
11. Dijkstra, E.W.: Guarded Commands, Nondeterminacy and Formal Derivation of Programs. CACM **18**(8) (1975) 453–457
12. Detlefs, D., Nelson, G., Saxe, J.B.: Simplify: a theorem prover for program checking. J. ACM **52** (May 2005) 365–473
13. The Alt-Ergo theorem prover. `http://ergo.lri.fr/`
14. Conchon, S., Contejean, E., Kannig, J., Lescuyer, S.: Lightweight Integration of the Ergo Theorem Prover inside a Proof Assistant. In Rushby, J., Shankar, N., eds.: AFM, New York, NY, USA, ACM (2007) 55–59
15. de Moura, L.M., Bjørner, N.: Z3: An efficient SMT solver. In: TACAS. Volume 4963 of LNCS., Springer (2008) 337–340
16. The Coq proof assistant. `http://coq.inria.fr/`
17. Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development. SpringerVerlag (2004)
18. Bertot, Y.: A short presentation of coq. In: TPHOLs. (2008) 12–16
19. The Isabelle proof assistant. `http://www.cl.cam.ac.uk/research/hvg/Isabelle/`
20. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL — A Proof Assistant for Higher-Order Logic. Volume 2283 of LNCS. Springer (2002)
21. The HOL4 proof assistant. `http://hol.sourceforge.net/`

22. Gordon, M.J.C., Melham, T.F., eds.: Introduction to HOL: A theorem proving environment for higher order logic. Cambridge University Press (1993)
23. Baudin, P., Filliâtre, J.C., Hubert, T., Marché, C., Monate, B., Moy, Y., Prevosto, V.: ACSL: ANSI/ISO C Specification Language. (February 2011) http://frama-c.cea.fr/acsl.html.
24. Lemerre, M., Ohayon, E., Chabrol, D., Jan, M., Jacques, M.B.: Method and Tools for Mixed-Criticality Real-Time Applications within PharOS. In: Proceedings of AMICS 2011: 1st International Workshop on Architectures and Applications for Mixed-Criticality Systems. (2011)
25. Klein, G., Elphinstone, K., Heiser, G., Andronick, J., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an OS kernel. In: Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles (SOSP'09), ACM (2009) 207–220
26. The Common Criteria Recognition Arrangement. http://www.commoncriteriaportal.org
27. Alkassar, E., Paul, W., Starostin, A., Tsyban, A.: Pervasive Verification of an OS Microkernel. In Leavens, G., O'Hearn, P., Rajamani, S., eds.: Verified Software: Theories, Tools, Experiments. Volume 6217 of LNCS. Springer Berlin / Heidelberg (2010) 71–85
28. Alkassar, E., Hillebrand, M.A., Paul, W.J., Petrova, E.: Automated verification of a small hypervisor. In: Verified software: theories, tools, experiments (VSTTE), Springer (2010) 40–54
29. Cohen, E., Dahlweid, M., Hillebrand, M., Leinenbach, D., Moskal, M., Santen, T., Schulte, W., , Tobies, S.: VCC: A practical system for verifying concurrent C. In: Theorem Proving in Higher Order Logics (TPHOLs). Volume 5674 of LNCS. (2009) 23–42
30. Alkassar, E., Cohen, E., Kovalev, M., Paul, W.J.: Verification of TLB virtualization implemented in C. In: the 4th International Conference on Verified Software: Theories, Tools, Experiments (VSTTE 2012), Philadelphia, PA, USA, Springer (January 2012) 209–224
31. Klein, G.: From a verified kernel towards verified systems. In: the 8th Asian Symposium on Programming Languages and Systems (APLAS 2010), Shanghai, China, Springer (November 2010) 21–33

Appendix. A simplified version of data page cleaning

```
#define MaxNumPages 100
#define PageSize 10
int NumPages; // number of pages
enum pageType {Zero,Data,Pagetable,Pagedirectory,DToZero,PTToZero,PDToZero};
unsigned int Contents[MaxNumPages * PageSize];
unsigned char Valid[MaxNumPages * PageSize];
enum pageType Type[MaxNumPages];
unsigned int Mappings[MaxNumPages];
unsigned char Cleaning[MaxNumPages];
unsigned int Cleaned[MaxNumPages];
int pending_preemption; // interruption iff non zero value
```

```
/*@
  predicate zero_Contents{L}(integer pIndex) =
    \forall integer k; 0<=k<PageSize ==> Contents[pIndex*PageSize+k]==0;
  predicate structures_validity = 0<=NumPages<=MaxNumPages &&
    \valid( Contents + (0.. (NumPages*PageSize -1) ) ) &&
    \valid( Valid + (0.. (NumPages*PageSize -1) ) ) &&
    \valid(Type + (0..NumPages-1)) && \valid(Mappings + (0..NumPages-1)) &&
    \valid(Cleaning + (0..NumPages-1)) && \valid(Cleaned + (0..NumPages-1)) &&
    ( \forall integer j; 0<=j<NumPages ==> (Type[j]==Zero || Type[j]==Data ||
      Type[j]==Pagetable || Type[j]==Pagedirectory || Type[j]==DToZero ||
      Type[j]==PTToZero || Type[j]==PDToZero) ) &&
    ( \forall integer j; 0<=j<NumPages ==> 0<=Cleaned[j]<=PageSize ) &&
    ( \forall integer j,l; 0<=j<NumPages && 0<=l<PageSize ==>
      0<=Valid[j*PageSize+l]<=1 );
  predicate R9 =
    \forall integer j;  ( 0<=j<NumPages && Type[j]==Zero ) ==> Mappings[j]==0 ;
  predicate R5 =
    \forall integer j;  ( 0<=j<NumPages && Type[j]==Zero ) ==> zero_Contents(j) ;
  predicate ToZeroPagesStartWithZeros =
    ( \forall integer j;  ( 0<=j<NumPages && (Type[j]==DToZero ||
      Type[j]==PTToZero || Type[j]==PDToZero) ) ==>
      (\forall integer i; 0<=i<Cleaned[j] ==> Contents[j*PageSize+i]==0) ) &&
    ( \forall integer j;  ( 0<=j<NumPages && (Type[j]==DToZero ||
      Type[j]==PTToZero || Type[j]==PDToZero) ) ==>
      (\forall integer i; 0<=i<Cleaned[j] ==> Valid[j*PageSize+i]==0) );
  predicate Inv = structures_validity && R9 && ToZeroPagesStartWithZeros && R5;
*/

/*@
  requires Inv && 0<=pIndex<NumPages && 0<=startIndex<=PageSize &&
    ( \forall integer k; 0<=k<startIndex ==> Contents[pIndex*PageSize+k]==0 ) &&
    Cleaning[pIndex]==1 && Mappings[pIndex]==0 && Type[pIndex] == DToZero;
  behavior finished:
    assumes pending_preemption == 0;
    ensures Inv && Type[pIndex]==Zero && Mappings[pIndex]==0 &&
      Cleaning[pIndex]==0 && Cleaned[pIndex]==0;
    assigns Contents[(pIndex*PageSize+startIndex) .. (pIndex*PageSize +
      PageSize-1)], Type[pIndex], Cleaning[pIndex], Cleaned[pIndex];
  behavior interrupted:
    assumes pending_preemption != 0;
    ensures Inv && Type[pIndex]==DToZero && Mappings[pIndex]==0 &&
       Cleaning[pIndex]==0 && startIndex < Cleaned[pIndex] <= PageSize;
    assigns Contents[(pIndex*PageSize+startIndex) .. (pIndex*PageSize +
      PageSize-1)], Type[pIndex], Cleaning[pIndex], Cleaned[pIndex];
*/
void putOnePageToZero(int pIndex, int startIndex){
  int l;
  /*@ loop invariant startIndex <= l <= PageSize && Inv &&
        ( \forall integer k; 0<=k<l ==> Contents[pIndex*PageSize+k]==0 );
      loop assigns Contents[(pIndex*PageSize+startIndex) .. (pIndex*PageSize +
        PageSize-1)];
      loop variant PageSize - l;
  */
  for(l=startIndex; l<PageSize; l++){
    Contents[pIndex*PageSize + l] = 0;
    if(pending_preemption){
      Cleaned[pIndex]=l+1;
      Cleaning[pIndex]=0;
      return;
    }
  }
  Type[pIndex] = Zero;
  Cleaning[pIndex]=0;
  Cleaned[pIndex]=0;
}
```

```
77
78 /*@
79   requires Inv && 0<=pIndex<NumPages;
80   behavior finished:
81     assumes pending_preemption == 0 && Type[pIndex]==Data && Mappings[pIndex]==0;
82     ensures Inv && \result==0 && Type[pIndex]==Zero &&
83       Cleaning[pIndex]==0 && Cleaned[pIndex]==0;
84     assigns Contents[(pIndex*PageSize) .. (pIndex*PageSize + PageSize-1)],
85       Type[pIndex], Cleaning[pIndex], Cleaned[pIndex];
86   behavior interrupted:
87     assumes pending_preemption != 0 && Type[pIndex]==Data && Mappings[pIndex]==0;
88     ensures Inv && \result==0 && Type[pIndex]==DToZero &&
89        Cleaning[pIndex]==0 && 0 < Cleaned[pIndex] <= PageSize;
90     assigns Contents[(pIndex*PageSize) .. (pIndex*PageSize + PageSize-1)],
91       Type[pIndex], Cleaning[pIndex], Cleaned[pIndex];
92   behavior failure:
93     assumes Type[pIndex]!=Data || Mappings[pIndex]!=0;
94     ensures Inv && \result==1;
95     assigns \nothing;
96 */
97 int cleanData (int pIndex){
98   if(Type[pIndex] != Data)
99     return 1;
100  if(Mappings[pIndex] != 0)
101    return 1;
102  Cleaning[pIndex]=1;
103  Cleaned[pIndex]=0;
104  Type[pIndex] = DToZero;
105  putOnePageToZero(pIndex,0);
106  return 0;
107 }
108
109 /*@
110   requires Inv && 0<=pIndex<NumPages;
111   behavior finished:
112     assumes pending_preemption == 0 && Type[pIndex]==DToZero && Mappings[pIndex]==0;
113     ensures Inv && \result==0 && Type[pIndex]==Zero &&
114       Cleaning[pIndex]==0 && Cleaned[pIndex]==0;
115     assigns Contents[(pIndex*PageSize+Cleaned[pIndex]) .. (pIndex*PageSize +
116       PageSize -1)],Type[pIndex],Cleaning[pIndex],Cleaned[pIndex];
117   behavior interrupted:
118     assumes pending_preemption != 0 && Type[pIndex]==DToZero && Mappings[pIndex]==0 &&
119       Cleaning[pIndex]==0;
120     ensures Inv && \result==0 && Type[pIndex]==DToZero &&
121       Cleaning[pIndex]==0 && \old(Cleaned[pIndex]) < Cleaned[pIndex] <= PageSize;
122     assigns Contents[(pIndex*PageSize+Cleaned[pIndex]) .. (pIndex*PageSize +
123       PageSize -1)],Type[pIndex],Cleaning[pIndex],Cleaned[pIndex];
124   behavior failure:
125     assumes Type[pIndex]!=DToZero || Mappings[pIndex]!=0;
126     ensures Inv && \result==1;
127     assigns \nothing;
128 */
129 int cleanPartiallyCleanedData (int pIndex){
130   if(Type[pIndex] != DToZero)
131     return 1;
132   if(Mappings[pIndex] != 0)
133     return 1;
134   Cleaning[pIndex] = 1;
135   putOnePageToZero(pIndex,Cleaned[pIndex]);
136   return 0;
137 }
```

Improving security of virtual servers and the storage in the Cloud by cutting SSH access and using EncFS encryption

Dominique Rodrigues[1] and Henri Pidault[2]

[1] CTO, nanoClouD
dominique.rodrigues@nanocloud.com
[2] EVP and Managing Director IT, Technology & Products, Kyriba
henri.pidault@kyriba.com

Abstract. In order to improve security of data which are stored in the cloud, including files for virtualization of servers or desktops, we propose to combine several existing technologies and to develop new tools. For instance, for a server having public visibility, we have developed a methodology to remove any remote access when this server is launched in production. The upgrades are then performed by the replacement of a new server delivered by a template. On the other hand, we propose to make an extensive use of encryption for any file and folder which is used in the cloud. KVM virtualization coupled with QEMU offers a native way to encrypt virtual servers or virtual desktops. EncFS is also a mature technology to transparently encrypt data and directories, and is available for any operating system. At last, we propose to develop tools to combine all these technologies to improve data security and we also encourage companies and individuals to require the implementation of these tools to societies offering services in the cloud.

1 Cloud and fears

One of the main fear of the enterprises which would like to use technologies in the cloud, is the feeling that anybody could access to their data, in particular some competitor. When servers are not anymore hosted within the walls of the company, it is difficult to know who could succeed to have any kind of access to the files which are stored on these machines.

For instance, if you use a virtual server, which is actually a file hosted on a physical server or within a SAN or a NAS system, it may seem easier to steal or download this entire file, instead of robbing a complete server. On the other hand, you can fear that a sysadmin in charge of your infrastructure in the cloud will one day be corrupted by one of your competitor and tries to copy some of your precious data for him. These fears hinder the adoption of this technology and all the advantages that companies could benefit with it. We propose here to present several existing technologies, or which are in development, that could be combined to increase security in the cloud. Our focus will be on SSH access and encryption of data in the cloud, in particular when using virtualization.

2 Limiting access to virtual servers

Any technology that could limit access to a virtual server which is hosted in the cloud is then welcome. So the minimum is to implement an encrypted access to log into its server, usually by the SSH protocol. Use of VPN, often based on proprietary technologies, is also one the favorite means to limit access to remote servers.

But the final solution could be to cut any access. To achieve that goal, we propose here a methodology which combines the use of virtualization with the disabling of any remote access based on SSH.

3 Cutting SSH access

So a simple idea is to avoid any possibility to login into a virtual server.Therefore, we propose to remove any SSH service on a virtual server when the latter is launched into production.

Though easy to understand, this idea needs a clean implementation. Because when the virtual server will be online, there will be not even a mean to check what is happening inside. On the other hand, the template which will be used to provide this secure virtual server still needs to keep access for its configuration.

Therefore, the following policy could be implemented:

1. Configure a template for any service that will run in a virtual machine.
2. Test a virtual server provided by the previous template in a sandbox.
3. Launch a new production server based on the validated template.
4. Stop and destroy the virtual server if needed.
5. Return to step 1 (for an update) or 3 (in the other cases).

By following this policy, an enterprise is ensured to have a virtual server where it is at least quite impossible to log in.

The drawback is that it will run as a black box. But since it is based on a virtual server provided by a template, it is easy and cost effective to often replace this black box by another. In case of any doubt, an incriminated virtual server could even be recovered by the company to deal a forensic analysis, while the service is still delivered by a new virtual machine.

However, if the physical host of this kind of virtual server is stolen, there is no real protection against an access to the hosted files. Whatever is the virtualization technology used, it is not difficult to mount a virtual image in order to have access to the entire virtual machine. The only way to avoid this is to use encryption.

4 Native encryption of a virtual image with QEMU-IMG

Among technologies of virtualization, KVM (Kernel-based Virtual Machine) [1] module, which is included in Linux kernel since 2.6.20 version, is increasingly

popular. To build a virtual machine with KVM, it is convenient to use QEMU [2], an open source emulator, which has now an optimised version for KVM called qemu-kvm [3], with the tool qemu-img. The basic way to do it is as follows

```
$ qemu-img create -f qcow2 virtfile.qcow2 10G
```

This command will create a 10 GB file, based on qcow2 format (a qemu format). Then, to encrypt this file, you can do a convert operation by typing the following

```
$ qemu-img convert -O qcow2 -o encryption virtfile.qcow2 enc_virtfile.qcow2
```

Or to directly encrypt the file at its creation

```
$ qemu-img create -f qcow2 -o encryption enc_virtfile.qcow2 10G
```

You will be prompted to enter a password (128 bit key stored in AES format) in order to encrypt the file. This password will be necessary when the server will be launched. It should be given using the qemu monitor console, after typing the command cont (otherwise nothing happens).

The enterprise, which owns the virtual server based on this encrypted file, may be the only one to know its password if it has access to an admin tool to launch its virtual server. This is the key to be sure that no cloud sysadmin may have an unauthorized access to the enterprise data. Therefore, we recommend that companies require this kind of service to their hosters in the cloud.

One way for a hoster to provide this service could be to use a virtual server template, which is not encrypted, and to let the client enterprise clone it for its needs and encrypt the copy with its own password.

5 EncFS encryption

Encryption can also be implemented for an entire folder, instead of for a single file. EncFS [4] has been developed for this purpose. It allows to encrypt an entire directory, on an existing partition without reformatting it. The idea is to create two folders: one where files are clearly readable, and the other where everything is encrypted. The encrypted folder provides permanent access, but the clear one is mounted only by its owner, using a password.

Development of EncFS is stopped since 2010. However, this technology is stable and usable in production. Portages have been made to deliver EncFS on several operating systems. For instance, EncFS can be used for Windows with the programs provided at [5], [6] and [7]. Port for MacOSX is also now available (see for instance [8]). Under Linux, EncFS makes use of FUSE [9] (file system in user space). To create a new encrypted folder, the following commands must be entered:

```
$ mdkir /path/clear_folder
$ mkdir /path/.encrypted_folder
$ encfs /path/.encrypted_folder /path/clear_folder
Volume key not found, creating new encrypted volume.
Password: [password entered here]
Verify: [password entered here]
```

Files can be stored and edited in the directory */path/clear_folder/*. They will be transparently encrypted in the directory */path/.encrypted_folder/*. To unmount the clear directory, just type

```
$ fusermount -u /path/clear_folder
```

and this folder will then appear to be empty. The encrypted folder is accessible but name of files and their contents are unreadable.

The main drawback of EncFS is that date of modification, size and rights of files are not hidden. Anyway, it greatly improves security of data and offers interesting features, such as the possibility to change the encryption password as often as desired.

6 EncFS for Storage in the cloud and virtual desktops folders

Therefore, EncFS is a good candidate to improve the security of data hosted in the cloud.

Indeed, many paying services are offered today to save its files in the cloud. However, companies proposing this backup have often declared to be owners of data stored on their servers. They are therefore free to do whatever they like with that data.This conflicts with respect for privacy or trade secret, and, once more, hinder adoption of the cloud usage.

With EncFS, it is possible to transparently encrypt its files and to store those, instead of readable ones. To decrypt its files, only a hidden single file, called *.encfs6.xml* and stored in the encrypted folder, and a password are necessary. So files stored in the cloud may be accessed from anywhere, just by keeping this file with you and remembering your password. On the other hand, it is a good idea to not store *.encfs6.xml* in the cloud, to improve security of its data. EncFS may also be used in the virtual desktop domain. When an enterprise offers to its employees to work on virtual desktops, virtual files of the latter could be in the cloud. Many interesting data concerning this enterprise may then be accessible outside its walls. By using EncFS, every employee could work in a folder which is encrypted when he or she logs out from its virtual computer. We think that this option, or an equivalent one, should be available in every virtual desktop commercial offer, which is not the case today.

7 EncFS for virtualization

Since any virtual server or virtual desktop is in fact a file stored on a physical machine (computer or storage array), virtualization could also benefit from EncFS to improve security and privacy of data stored in virtual services. An admin tool should be developed, like in the case of the qemu virtual file, in order to let the possibility for an enterprise to encrypt the folder where its virtual servers are stored. With that encryption, combined with the qemu file encryption, no sysadmin could be able to read informations stored into the virtual server. Furthermore, even in the worst case of a stolen computer containing virtual files, it will be very difficult, even quite impossible, to recover anything from encrypted files.

Conclusion

Security must never rely only on a single technology. Indeed, it is much more safer to combine several means to improve security. By doing so, we increase the number of barriers against a malicious person or a competitor that will try to gain access to your private data or your trade secrets.

This potential access is one of the biggest fear for enterprises thinking to use the cloud for their professional needs, or those who have already switched to this new mode of computing use.

We have then proposed to mix several existing technologies, by developing new tools which will allow this combination. For instance, we could remove any remote access for autonomous servers delivering one service. The encryption of raw data, virtual servers files and of folders which host these files are also the way to improve security of an infrastructure hosted in the cloud.

We encourage enterprises to ask for these features when they address commercial offers in the cloud. Lack of security can be discussed in Service Level Agreements with financial compensations or penalties, but the data of an enterprise are often the core of its competitiveness and are then invaluable.

References

1. http://www.linux-kvm.org/page/Main_Page, main website of Kernel-based Virtual Machine (KVM).
2. http://wiki.qemu.org/Main_Page, main website of QEMU, the open source emulator for several architectures.
3. http://sourceforge.net/projects/kvm/files, sourceforge repository for qemu-kvm, the QEMU version optimized for KVM.
4. http://www.arg0.net/encfs, website of the EncFS project.
5. http://members.ferrara.linux.it/freddy77/encfs.html, main website of encfs4win, the project porting encfs for windows.
6. http://gitorious.org/encfs4win, git repository for encfs4win.

7. https://groups.google.com/group/dokan, Google group of Dokan related to encfs4win. Dokan (http://dokan-dev.net/en/) is a project with the aim to port SSHFS to windows.
8. http://code.google.com/p/encfsvault/, website of the encfsvault, a project porting encfs for MacOSX.
9. http://fuse.sourceforge.net/, sourceforge repository of FUSE, the Linux file system in userspace.

Privacy-supporting cloud computing by in-browser key translation*

Myrto Arapinis Sergiu Bursuc Mark Ryan
School of Computer Science, University of Birmingham
{m.d.arapinis,s.bursuc,m.d.ryan}@cs.bham.ac.uk

Abstract

Cloud computing means entrusting data to information systems that are managed by external parties on remote servers, in the "cloud", raising new privacy and confidentiality concerns. We propose a general technique for designing cloud services that allows the cloud to see only encrypted data, while still facilitating some data-dependent computations. The technique is based on key translations and mixes in web browsers.

We focus on a particular kind of software-as-a-service, namely, services that support applications, evaluations, and decisions. Such services include job application management, public tender management (e.g., for civil construction), and conference management. We identify the specific security and privacy risks that existing systems pose. We propose a protocol that addresses them, and forms the basis of a system that offers strong security and privacy guarantees.

We express the protocol and its properties in the language of ProVerif, and prove that it does provide the intended properties. We describe an implementation of a particular instance of the protocol called ConfiChair, which is geared to the evaluation of papers submitted to conferences.

1 Introduction

Cloud computing means entrusting data to information systems that are managed by external parties on remote servers, "in the cloud." Cloud-based storage (such as Dropbox), on-line documents (such as Google docs), and customer-relationship management systems (such as salesforce.com) are familiar examples. Cloud-based services are being adopted widely throughout business. For example, Google Apps currently supports 40 million individual employees from 4 million businesses that pay US$50 per user per year. Those businesses have outsourced their email, calendar, and documents to Google. Currently, 5000 business sign up to Google Apps each day.

Unfortunately, cloud computing raises privacy and confidentiality concerns because the service provider has access to all the data, and could accidentally or deliberately disclose it. For this reason, many organisations are not willing to sign up to Google Apps. They typically include lawyers, doctors, insurance companies, banks, engineers, and government. Solutions are sought that would offer such companies some of the benefits of cloud computing, without the confidentiality risk.

In this paper, we provide a solution for a particular kind of software-as-a-service (SaaS), namely, services that support applications, evaluations, and decisions. Such services include job application management, public tender management (e.g., for civil construction), and conference management. We identify a set of confidentiality requirements for such SaaS applications, and we propose a way

*This paper is not for formal publication. A substantial part of its content has been published at *Principles of Security and Trust*, LNCS 7215, which was held as part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2012. This extended draft, which generalises the range of application of the techniques and includes more detail, has been invited for publication in the *Journal of Computer Security*, and is currently under review for that journal.

to construct applications which achieves the requirements. The confidentiality guarantees ensure that no-one has access to the application data, beyond the access that is explicitly granted to them by their participation in the service. In particular, this is true about the cloud provider and managers. We describe a protocol in which applicants, evaluators and decision makers interact through their web browsers with the cloud-based management system, to perform the usual tasks of uploading and downloading applications and evaluations. The cloud is responsible for fine-grained routing of information, in order to ensure that the right agents are equipped with the right data to perform their task. It is also responsible for enforcing access control, for example concerning conflicts of interest and to ensure that an evaluator doesn't see other evaluations of an application before writing her own. However, all the sensitive data is seen by the cloud only in encrypted form, and the cloud is not able to tell which applicants are being reviewed by which evaluators.

For brevity, we use the term "cloud" to include all roles that are not an explicit part of the service management; that includes the service management system administrator, the cloud service provider, the network administrator, etc. The security properties that our system provides may be summarised as follows:

- **Secrecy of applications, evaluations and scores.** The cloud does not have access to the content of applications or evaluations, or the numerical scores given by evaluators to applications.
- **Unlinkability of applicant-evaluator.** The cloud *does* have access to the names of applicants and the names of evaluators. This access is required in order to route information correctly, to enforce access control, and to allow a logged-in user to see all his data in a unified way. However, the cloud does not have the ability to tell if a particular applicant was reviewed by a particular evaluator.

Applicability of the ideas The protocol we propose is limited to a particular class of SaaS systems, namely those that support applications, evaluations, and decisions. In future work, we intend to explore wider classes of SaaS systems that might also benefit from the technique of in-browser key translation and mixing, such as the following:

- Customer relationship management systems (such as salesforce.com);
- Cloud-based finance and accounting services;
- Social networks, in which users share posts and status updates without wishing that data to be mined by the cloud provider for profiling purposes.

Our contributions.

1. We identify a class of cloud-based systems for competition management and propose a general framework to reason about the functionality and privacy of such systems (section 2)
2. We propose a general protocol that provides secrecy and unlinkability properties for cloud-based competition management, relying on a trusted competition manager and in-browser key translation (section 3)
3. We show how the desired privacy properties can be formally expressed and automatically verified with ProVerif (section 4)
4. We present the implementation of a privacy-supporting system for conference management (ConfiChair), which is based on our general protocol (section 5).

Contributions 1-3 are a generalization of a specific protocol for conference management and of its verification, that we have proposed in [9].

2 Description of the problem and related work

As mentioned, our target is any SaaS system that supports applications, evaluations, and decisions. There are many systems of this kind in the real world, and increasingly they are cloud-based. We give some examples:

Conference management systems are used to support the reviewing and discussion procedures needed to decide which papers are accepted for presentation at conferences. Authors submit papers online, and programme committee members download the papers to evaluate them, and upload their reviews. The conference management system typically also supports online discussion of the reviews and the acceptance decision. EasyChair and the Editor's Assistant (EDAS) are well-known examples of cloud-based conference management systems. For example, EasyChair currently hosts more than 3000 conferences per year, and has about 600,000 users [39].

Public tender management is the process in which governments and their agencies collect and evaluate bids (or tenders) to supply services. Bidders submit tenders, and evaluators review and compare the bids in order to accept one or more of them. There may be a pre-qualification stage, in which potential tenderers have to prove their eligibility, and there may be a tender-refinement stage, in which bidders can answer questions and supply more information about their bid [3]. A plethora of tender management systems exists [1, 4, 2].

Recruitment for employment is the process of attracting, screening, and selecting a qualified person for a job. Applicants respond to advertisements by making an application; referees and other evaluators comment on the suitability of the applicants, and a panel makes a decision about whom to offer the position. WCN is a UK-based e-recruitment provider.

These systems involve a similar workflow, although they use different terminology and the workflow may differ in small details. We illustrate the terminological differences in the table below:

Generic	*Conference*	*Procurement*	*Employment*
Invitation	Call for papers	Invitation to tender	Job description
Panel	Programme committee	Evaluation team	Panel
Manager	Programme chair	Manager	Recruiter
Panel member	PC member	Evaluator	Referee
Application	Paper	Tender	Application
Applicant	Author	Bidder	Applicant
Evaluation	Review	Evaluation	Evaluation
Outcome	Decision	Announcement	Outcome

In this paper, we use the generic terms in the left-hand column. A panel member will sometimes be called reviewer and the competition manager will sometimes be called chair.

Competition management workflow. In general, a competition comprises an initialisation phase, followed by a certain number of rounds. The initialisation consists in the opening of the competition by the chair, the registration of applicants, and the declaration of conflicts between applicants and panel members. A round consists in 5 consecutive phases: first the applicants submit the documents necessary to the current round (Submission phase), then each submission gets assigned to panel members (Assignment phase) for evaluation (Evaluation phase) and discussion (Discussion phase). Finally the applicants get notified the outcome of the current round and may also get instructions for the following round (Notification phase).

Depending on the considered scenario, there may be several rounds to the competition. Each round consists of five phases (Submission, Assignment, Evaluation, Discussion, and Notification)

described below. For example, in the context of conference management, in the first round authors submit papers which get reviewed and discussed by the programme committes. There can also be a second round, usually called rebuttal, where authors are asked to submit their responses to the first reviews, which in turn get reviewed and discussed. Similarily, tender bidding processes usually comprise three rounds, namely the pre-qualification, the bidding and the refinement rounds.

Security concerns. The different kinds of competition management systems share similar security concerns. In the case of cloud-based conference management systems, these have already been well-documented [36]. Cloud-based systems such as EasyChair contain a vast quantity of sensitive data about the authoring and reviewing performance of tens of thousands of researchers world-wide. This data is in the possession of the system administrators, and could be accidentally or deliberately disclosed. In the case of tender management systems and employment recruitment systems, the risks are similar. The centralised existence of sensitive data about individuals and organisations can create conflicts of interest for the custodians, and also can attract the attention of hackers and crackers.

Note that the data confidentiality issue concerns the cloud system administrator (who administrates the system for all competitions), not the individual competition manager (who is concerned with a single competition). With current systems, the cloud system administrator has access to all the data on the system, across all of the competitions and all of the applicants and evaluators An individual competition manager, on the other hand, has access to the data only for the particular competition of which she is manager. Moreover, an applicant or evaluator that chooses to participate in the competition can be assumed to be willing to trust the manager (for if he didn't, he would not participate); but there is no reason to assume that he trusts or even knows the cloud management system provider.

This paper proposes a competition management protocol which does not allow the cloud infrastructure provider to access any of the data. But solving the problem of data confidentiality has to be done in a way that allows the service provider to offer useful functionality, and in a way that results in a system with good usability features. Thus, our problem is determined by three conflicting sets of requirements, namely functionality, privacy and usability. As we show below, there is much existing work related to our paper, but it can not be used to solve our problem either because of its complexity, or because of its different perspectives on privacy, or because it does not achieve the required balance between privacy and functionality.

2.1 Desired properties and threat model

Functional requirements. As previously mentioned, we use the term "cloud" to refer to the cloud service provider, competition management system and its administrators, and the network. The responsibilities of the cloud are:

- To collect and store data relevant to the competition, including names of evaluators and applicants, as well as applications, evaluations and scores.
- To enforce access control in respect of conflicts of interest and ensuring that evaluators see other evaluations of an application only after they have submitted their own.
- To manage the information flow of the competition: from applicants, to competition manager, to evaluators and back.
- To notify the applicants of the outcome of their applications.

Privacy requirements. We require that the cloud does not know

- the content of submitted applications,
- the content of submitted evaluations,
- the scores attributed to submitted applications.

Further, when data is necessarily known to the cloud in order that it can fulfil the functional requirements, we require what we call the *unlinkability* property: the cloud is unable to determine if a particular applicant is being evaluated by a particular evaluator or not; that is, the cloud cannot infer

- the link between applicants and evaluators

Threat model. It is reasonable to trust the cloud to execute the specified functional requirements. Indeed, an incorrect functionality would be detected in the long run and the users would simply move into another cloud. On the other hand, the cloud may try to violate privacy without affecting functionality, in a way that cannot readily be detected. Our protocol is designed to remove this possibility. Obviously, there are inherent limitations on any protocol's ability to achieve this. For instance, if the cloud provider was invited to participate as an evaluator or a manager, then he necessarily would have access to privileged information. Consequently, the privacy requirements are expected to hold in our threat model only for competitions in which the cloud provider does not participate, except as provider of the cloud service or as applicant corresponding to an application.

We assume that users are running uncorrupted browsers on malware-free machines. The HTML, Java, and Javascript code that they download is also assumed to be obtained from a trustworthy source and properly authenticated (*e.g.* by digital signatures).

Usability requirements. The system should be as easy to use as present-day web-based competition management systems, such as EasyChair, iChair, eTenderer, VHTender, or WCN. The cost of security should not be unreasonable waiting time (e.g. for encryption, data download), or software installation on the client-side (e.g. a browser should be sufficient), or complex key management (e.g. public key infrastructure), etc. We discuss more about usability in section 5 which describes a prototype implementation of an instance of our protocol.

2.2 Related work

Generic solutions. Much work has been done that highlights the confidentiality and security risks that are inherent in cloud computing (e.g., [18] includes an overview), and there is now a conference series devoted to that topic [23]. Although the issue is well-known, the solutions described are mostly based on legislative and procedural approaches. Some generic technological solutions have appeared in the literature. The first one uses trusted hardware tokens [37], in which some trusted hardware performs the computations (such as key translations) that involve sensitive data. Solutions based on trusted hardware tokens may work, but appear to have significant scalability issues, and require much more research. Other papers advise designing cloud services to avoid having to store private data, and include measures to limit privacy loss [32].

Fully-homomorphic encryption (FHE) has been suggested as another generic solution to cloud-computing security. FHE is the idea that data processing can be done through the encryption, and has recently been shown to be possible in theory [25]. However, the range of functionality that can be provided through the encryption is not completely general. For example, one cannot extract from a list the items satisfying a given predicate, but one can return a list of encrypted truth values that indicate the items that satisfy the predicate, which is less useful. It is not clear to what extent FHE could alleviate the requirement to perform the browser-side computations of ConfiChair. Moreover, FHE is currently woefully inefficient in practice, and can only be considered usable in very specialised circumstances.

Data confidentiality and access control. Many works consider the problem of restricting the access of data in the cloud to authorised users only. For example, attribute-based encryption [12, 10] allows fine-grained control over what groups of users are allowed to decrypt a piece of data. A different example is work that aims to identify functionally encryptable data, i.e. data that can be encrypted while preserving the functionality of a system [34]. Such systems, and others, aim to guarantee that the cloud, or unauthorised third parties, do not access sensitive data. Our problem requires a different perspective: how to design systems that allow the cloud, i.e. the intruder, to

handle sensitive data, but at the same time ensure that sensitive data value links between them are not revealed.

Unlinkability. In many applications it is important that links between participants, data, or transactions are kept hidden. In RFID-based systems [20] or in privacy enhancing identity management systems [22] for example, an important requirement is that two transactions of a same agent should not be linkable in order to prevent users from being tracked or profiled. Another exemplar application that requires unlinkability is electronic voting: a voter must not be linked to the vote that he has cast [24]. Moreover, like user identities in our case study, a vote is at the same time functional (to be counted) and sensitive (to be private). Voting systems achieve unlinkability by relying either on mix nets [29, 28], or on restricted versions of homomorphic encryption that allow the addition of plaintexts [7, 11]. Our proposed protocol also relies on mixing, showing how that idea can be adapted to new application areas.

Other systems identify applications where the cloud can be provided with "fake" data without affecting functionality [26]. In that case, privacy of "real" data may be preserved, without the cloud being able to detect the substitution. That is a stronger property than what we aim for, and at the same time the solution proposed in [26] is restricted to very specific applications. In particular, a conference management system can not function correctly with "fake" data provided to the cloud.

Conference management. There has been work exposing particular issues with conference management systems, related to data secrecy, integrity and access control [30, 35]. These are also important concerns, but that are quite orthogonal to ours, where we are interested in system design for ensuring unlinkability properties. More importantly, none of these works considers our threat model, where the attacker is the cloud.

3 The protocol

3.1 Description

The protocol is informally described in Figures 1-3 on the following pages. Some details, such as different tags for messages in each phase of the competition, are left out, but the detailed formal definition is given in appendix. The main idea of the protocol is to use a symmetric key K_{Comp} that is shared among the members of the panel. This key will be used to encrypt sensitive data before uploading it to the cloud. However, the cloud needs access to some sensitive data, like the identity of the panel member in charge of evaluating a particular application, in order to implement the functional requirements of the protocol. To reveal that data to the cloud, without compromising privacy, our protocol makes use of the fact that different types of data are needed by the cloud at different phases of the competition. Thus, in transitioning from one phase to another, the manager can hide the links between applicants and panel members. He does so by performing a random mix on the data he needs to send to the cloud before moving to the next phase. Each competition has a public key, that applicants use to encrypt symmetric keys, that in turn serve to encrypt applications.

Notation. In Figures 1-3 we use the following notations:

- $\{m\}_k$ to represent the encryption of a term m with a key k
- $L \leftarrow_r S$ to denote that L is a random permutation of elements in S
- $n \in_r M$ to denote that n is randomly selected among elements in the set M
- $L \leftarrow \{M \mid \phi\}$ to denote that the elements of the L are constructed as specified by an expression M, whose sub-expressions have been defined in ϕ. For example, $\{\{s\}_k \mid s \in M, k \in_r \mathbb{N}\}$ represents the set of encryptions of elements of M, with a key that is randomly chosen for each element of M.

- the diagrams specify the workflow for a particular panel member P and a particular applicant A. There is therefore an implicit universal quantification over all panel members and all applicants for a conference. We sometimes use the expression "for ϕ" to restrict the diagram from that point on to executions that satisfy ϕ.

3.1.1 Initialisation (Figure 1).

Initialisation consists in the three phases (Opening, Registration, and Conflicts declaration) described in this section.

Opening. The manager generates the symmetric key K_{Comp}, a public key $\mathsf{pub}(\text{Comp})$ and a corresponding private key $\mathsf{priv}(\text{Comp})$. The symmetric key is then shared among the members of the panel in a way that does not reveal it to the cloud (see section 3.2.1). Then the manager requests from the cloud the creation of the competition named Comp, sending along the names of the chosen members $\mathsf{P}_1, \ldots, \mathsf{P}_\ell$ for the panel.

Registration. An applicant A registrates his intention to participate to the competition. He creates a symmetric key k. He uploads to the cloud k encrypted with $\mathsf{pub}(\text{Comp})$. An identifier λ is used to refer to this encrypted message. The first role of the key k will be to provide a symmetric key for the encryption of the applicant's submissions at each round of the process. The second role of k will be to encrypt the outcome of each round, for the notification that will be sent through the cloud back to the applicant.

Conflicts declaration. The panel members declare the possible conflicts they might have with the registered applicants. The manager downloads the database with encrypted keys, decrypts them using the private key $\mathsf{priv}(\text{Comp})$ of the competition and encrypts them back with the shared symmetric key K_{Comp}. A new identifier μ is introduced for each registration key. The manager also associates to each μ the set of conflicts. Finally, he mixes the elements in $\text{DB}^r_{\text{Keys}}$ before sending it to the cloud. The cloud filters the keys according to the conflicts and sends them to the members of the panel.

3.1.2 A round (Figures 2 and 3).

Depending on the considered scenario, there may be several rounds to the competition. Each round consists of five phases (Submission, Assignment, Evaluation, Discussion, and Notification) described below.

Submission. An applicant A creates the document s^i to submit at round i. He uploads to the cloud s^i encrypted with the key k generated at the registration phase. The identifier λ is used to link the submission s^i and the key k, so that the pannel members can access the applicant's submissions (if they are not in conflict) in the following rounds.

Assignment. After applicants have submitted their documents for round i, the panel members can, if the manager allows it, bid for the submitted applications they want to evaluate. To do so they encrypt the corresponding identifiers μ with the shared key of the competition. This information is sent to the manager through the cloud, who can take it into account in his assignment of submissions to panel members at the end of this phase.

Evaluation. The members of the panel download the database with the submissions of round i, and can decrypt those for which they have received the key at the declaration of conflicts phase. For the applications they have been assigned to evaluate, they upload evaluations and scores in encrypted form back to the cloud. Note that the cloud is told to what identifier μ this encrypted

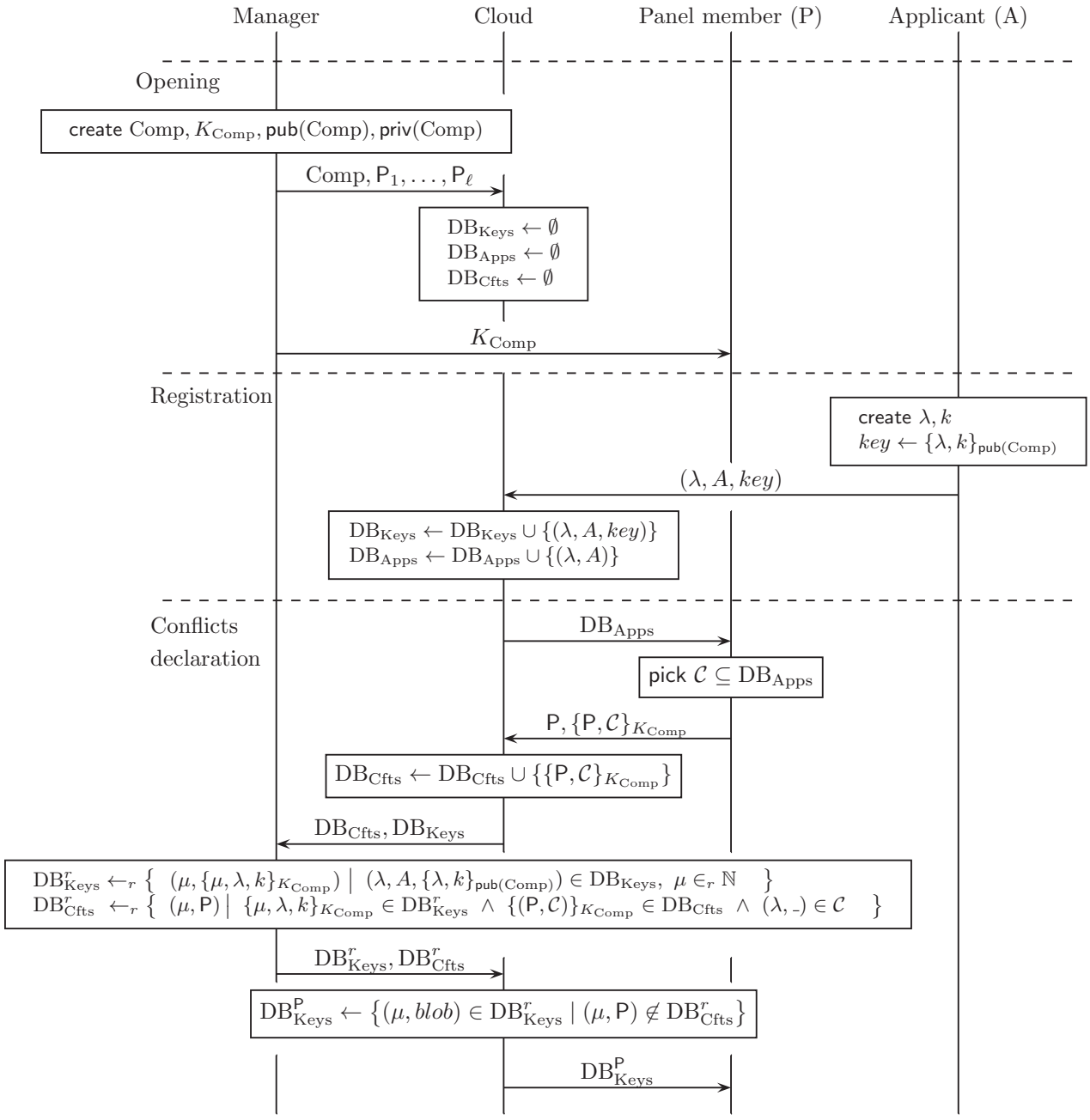


Figure 1: **Initialisation: opening, registration, and conflicts declaration phases**

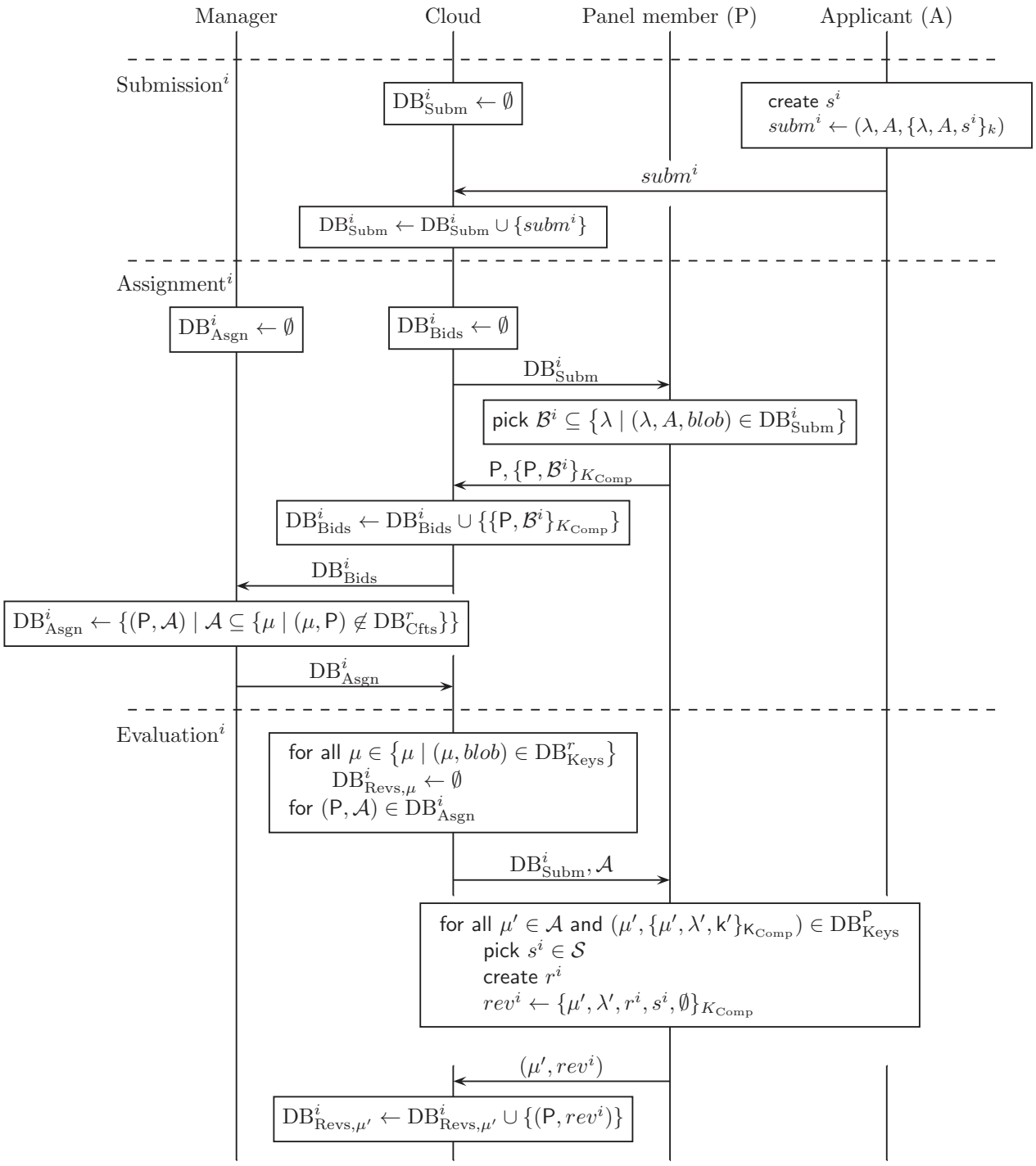


Figure 2: **Round i: submission, assignment, and evaluation phases**

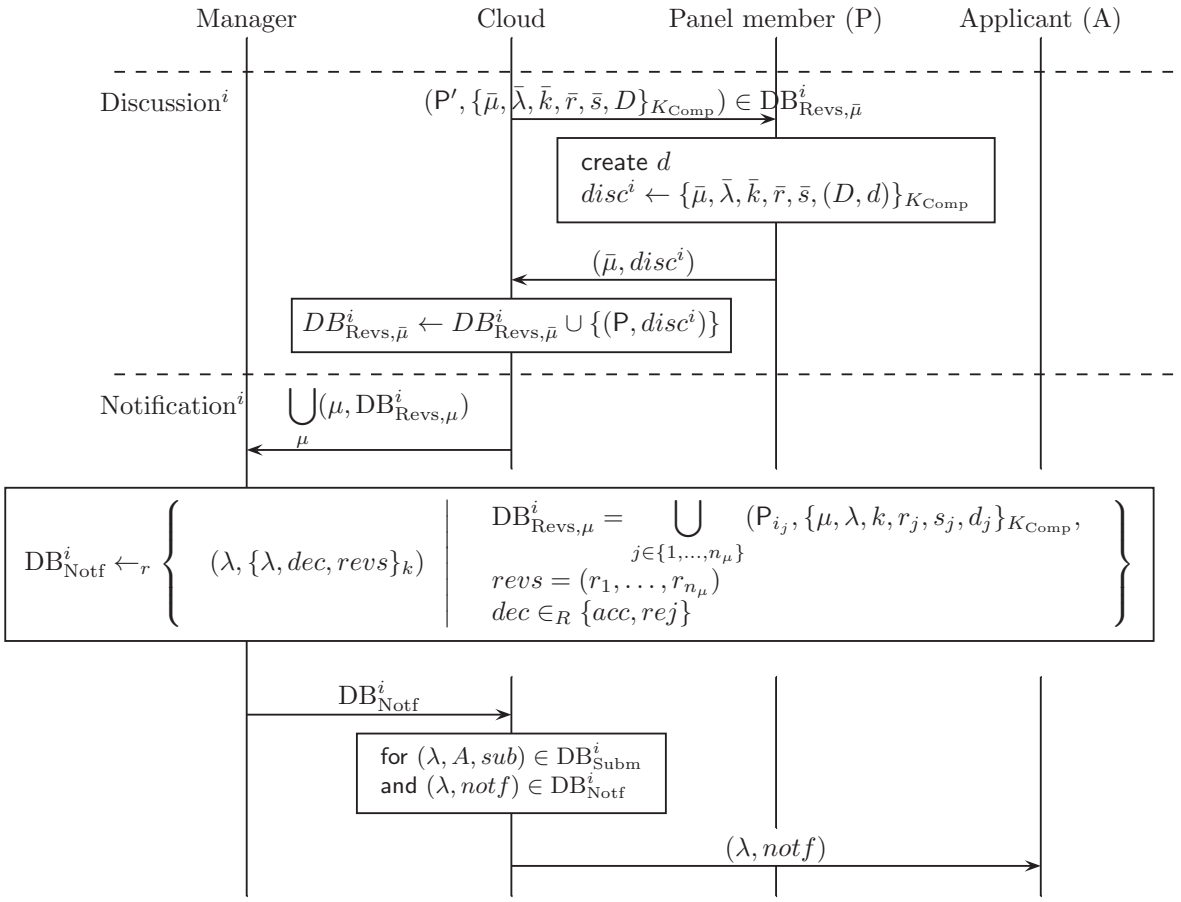


Figure 3: **Round i: discussion and notification phases**

evaluation refers to. This allows the cloud to manage the data flow, without being able to link μ with λ, and hence the panel member with the applicant.

Discussion. The evaluations of each paper are submitted to the panel members (except for the conflicting evaluators) for discussion. Each member of the panel can read a submitted evaluation and the ongoing discussion D and add a comment d to it.

Notification. For each application, the manager of the panel creates a notification including the outcome of the current round i and the evaluations written by the panel members. It can also include instructions for the next round. The outcome is encrypted with the applicant's symmetric key k (chosen at registration). The encrypted notification along with the submission identifier λ is uploaded to the cloud, allowing it to manage the information flow without compromising the privacy of the applicants.

3.2 Discussion

3.2.1 Distribution of the competition symmetric key.

The privacy properties of our protocol rely on the sharing of a symmetric key K_{Comp} among the members of the panel in such a way that the cloud does not get hold of K_{Comp}. Here we suggest a few possible solutions in the context of our applications, reflecting different trade-offs between security and usability. Our protocol is independent of which of the three solutions is adopted:

(1) Public keys. Each member of the panel may be expected to have a public key. Then, the symmetric key can be encrypted with each of the chosen panel member's public key and uploaded to the cloud. The distribution can be made more flexible and efficient by relying on key distribution protocols like [16]. An important issue in this setting is the authentication of public keys of members invited to participate in the competition panel. This may be done either relying on a hierarchical certification model such as *PKI* or, what is more probable in the case of competition management, on a distributed web of trust, such as that of *PGP*.

(2) Token. In this solution, each member of the panel generates a symmetric key k_{P} and uploads $\{k_{\mathsf{P}}\}_{\mathsf{pub}(\text{Comp})}$ to the cloud. Then, the manager sends $\{K_{\text{Comp}}\}_{k_{\mathsf{P}}}$ to the panel member using a channel that is outside the control of the cloud. He does this by checking the evaluator's authenticated email address and sends $\{K_{\text{Comp}}\}_{k_{\mathsf{P}}}$ to that address. The panel member then decrypts this token to obtain K_{Comp}. In this case, even if the cloud has access later to a panel member's email, it cannot compromise the privacy properties that our protocol ensures.

(3) Email. If we assume that email infrastructure is not in the control of the cloud service provider that hosts the competition management system (as is most probably the case), the key K_{Comp} could be sent to panel members directly by email. In that case, if the email of a panel member is compromised later on, its privacy for the competition Comp is also compromised. Note that it is only the key K_{Comp} that must be sent by email, all the rest of the protocol being executed in the cloud.

3.2.2 Computation in the cloud.

We stress that non-trivial computation takes place in the cloud, namely routing of messages, and optionally collection of statistics. It is essential for usability and take-up of the proposed system that these computations are done by the cloud. The difficulties in designing the protocol thus lie in releasing the necessary information for the cloud to perform these operations without compromising the user privacy. In particular, the link between the sender of a message (*e.g.* the applicant behind a submission) and the end receiver of this message (*e.g.* the panel member evaluating this submission) should remain private and this although it is the cloud that is responsible for routing messages.

Optionally, the protocol can be extended to allow the cloud to collect statisticts or other anonymised data about the competition, its applicants, submissions, and panel members. This can

be achieved by adding code which extracts this information during the manipulations performed by the manager's browser. For example, along with the computation of $\mathrm{DB}^r_{\mathrm{Notf}}$ in Figure 3, the manager could also compute the average score $as_\mu = (s_1 + \cdots + s_{n_\mu})/n_\mu$ for each submisstion and return the vector $(as_\mu)_\mu$ to the cloud. (Such optional features must be carefully designed to avoid weakening the security properties, and are not considered in our formal model in Secrion 4.)

3.2.3 Efficiency and usability

It may seem that there is a considerable amount of work to be done by the manager, especially in the transition between phases. As we discuss in section 5, this does not have to be evident to the manager. Our experiments with our prototype implementation of a competition management system following this protocol show that the browser can transparently execute the protocol.

4 Formal model and verification

It is difficult to ascertain whether or not a cryptographic protocol satisfies its security requirements. Numerous deployed protocols have subsequently been found to be flawed, *e.g.* the Needham-Schroeder public-key protocol [31], the public-key Kerberos protocol [19], the PKCS#11 API [17], or the BAC protocol in e-Passports [21]. In this section, we formally show that our protocol satisfies the claimed security properties. The formal verification of the protocol has been done automatically using the ProVerif tool [13, 15], and the corresponding ProVerif scripts are available online [33]. The verification requires a rigorous description of the protocol in a process calculus as well as formal definitions of the desired properties, each discussed in detail in the following sections.

4.1 The process calculus

The ProVerif calculus [13, 15] is a language for modelling distributed systems and their interactions. It is a dialect of the applied pi calculus [6]. In this section, we briefly review the basic ideas and concepts of the ProVerif calculus.

4.1.1 Terms.

The calculus assumes an infinite set of *names*, $a, b, c, \ldots$, an infinite set of *variables*, $x, y, z, \ldots$ and a finite *signature* Σ, that is, a finite set of *function symbols* each with an associated arity. Function symbols are divided in two categories, namely *constructors* and *destructors*. Constructors are used for building messages from other messages, while destructors are used for analysing messages and obtaining parts of the messages they are applied to. Names and variables are messages. A new message M may be built by applying a constructor $f \in \Sigma$, to names, variables and other messages, $M_1, \ldots, M_n$, and denoted as usual $f(M_1, \ldots, M_n)$. A term evaluation D is built by applying any function symbol $g \in \Sigma$ (constructor or destructor) to variables, messages or term evaluations, $D_1, \ldots, D_n$, denoted $g(D_1, \ldots, D_n)$. The semantics of a destructor g of arity n is given by a finite set of rewrite rules of the form $g(M_1, \ldots, M_n) \to M_0$, where $M_0, M_1, \ldots, M_n$ are messages that only contain constructors and variables. Constructors and destructors are used to model cryptographic primitives such as *shared-key* or *public-key encryption*. The ProVerif calculus uses tuples of messages $(M_1, \ldots, M_n)$, keeping the obvious projection rules implicit.

In the following, for the purpose of modelling the encryption primitives in our protocol, we will consider the signature

$$\Sigma_{\mathsf{ciph}} = \{\mathsf{senc}_{/3}, \mathsf{sdec}_{/2}, \mathsf{pub}_{/1}, \mathsf{aenc}_{/3}, \mathsf{adec}_{/2}\}$$

where senc (*resp.* aenc) is a constructor of arity 3 that models the randomised shared-key (*resp.* randomised public-key) encryption primitive, sdec (*resp.* adec) is the corresponding destructor of arity 2, and pub is a constructor of arity 1 that models the public key associated to the private key given in argument.

The semantics of destructors in Σ_{ciph} is given by the following rules

$$\begin{array}{rcl} \mathsf{sdec}(x, \mathsf{senc}(x, y, z)) & \to & z \\ \mathsf{adec}(x, \mathsf{aenc}(\mathsf{pub}(x), y, z)) & \to & z \end{array}$$

We model the probabilistic shared-key encryption of the message m with the key k by $\mathsf{senc}(k, r, m)$, where the r is fresh for every encryption; and the probabilistic public-key encryption of the message m with the pubic key corresponding to the secret key k by $\mathsf{aenc}(\mathsf{pub}(k), r, m)$.

We will write $D \Downarrow M$ if the term evaluation D can be reduced to the message M by applying some destructor rules. For example, if we consider the following term evaluation E and message N

$$\begin{array}{rcl} E & = & \mathsf{senc}(k, r, \mathsf{sdec}(k', \mathsf{senc}(k', r', s))) \\ N & = & \mathsf{senc}(k, r, s), \end{array}$$

by application of the first rewrite rule given above, we have $E \Downarrow N$.

4.1.2 Processes.

Processes are built according to the grammar given below, where M, N are terms, D is a term evaluation and n is a name.

$P, Q, R ::=$	processes
0	null process
$P \mid Q$	parallel composition
$!P$	replication
$\mathrm{new}\, n; P$	name restriction
$\mathrm{let}\ M = D\ \mathrm{in}\ P\ \mathrm{else}\ Q$	term evaluation
$\mathrm{in}(N, M); P$	message input
$\mathrm{out}(N, M); P$	message output

Replication handles the creation of an unbounded number of instances of a process. The process $\mathrm{let}\ M = D\ \mathrm{in}\ P\ \mathrm{else}\ Q$ tries to evaluate D and matches the result with M; if this succeeds, then the variables in M are instantiated accordingly and P is executed; otherwise Q is executed. We will omit the else branch of a let when the process Q is 0. Names that are introduced by a new construct are *bound* in the subsequent process, and they represent the creation of fresh data. Variables that are introduced in the term M of an input or of a let construct are *bound* in the subsequent process, and they represent the reception or computation of fresh data. Names and variables that are not bound are called *free*. We denote by $\mathrm{fn}(P)$, respectively $\mathrm{fv}(P)$, the free names, respectively free variables, that occur in P.

Notation. A process definition P will sometimes be denoted by $P(\vec{v})$, where $\vec{v}$ is a vector of free variables that occur in P and that can be seen as parameters for the process P. Then we will abbreviate the process $\mathrm{let}\ \vec{v} = \vec{w}\ \mathrm{in}\ P$ simply by $P(\vec{w})$, and we will say that $P(\vec{w})$ is an instance of $P(\vec{v})$.

Semantics. The possible actions of the environment are captured by *evaluation contexts*. A *context* is a process with a hole. For a context $C[\_]$, we denote by $C[A]$ the process obtained by filling its hole $\_$ with the process A. An *evaluation context* is a context whose hole is not under a replication, a conditional, an input, or an output.

We define the operational semantics of the process calculus by the means of two relations: structural equivalence and internal reductions. *Structural equivalence* ($\equiv$) is the smallest equivalence relation on processes closed under α-conversion of bound names and bound variables, application of evaluation contexts and of standard rules (see [15] for full definition) that capture the associativity, the commutativity and the interplay of $\mid$ and ν.

Internal reduction ($\to$) is the smallest relation closed under structural equivalence, application of evaluation contexts and such that

$\mathrm{out}(N, M).P \mid \mathrm{in}(N, x).Q \to P \mid Q\{^{M}/_{x}\}$
let $M = D$ in P else $Q \to P\sigma$, if $D \Downarrow N$ & $\sigma = \mu(M, N)$
let $M = D$ in P else $Q \to Q$, otherwise

where we let $\mu(M, N)$ denote the substitution that matches M with N, if such a substitution exists. We write $\to^*$ for an arbitrary (possibly zero) number of internal reductions.

4.2 The protocol in the process calculus

We introduce the set of constants $\Sigma_{\mathsf{tags}} \cup \Sigma_{\mathsf{nums}}$, where

$$\begin{array}{rcl} \Sigma_{\mathsf{tags}} & = & \{\mathsf{reg}, \mathsf{initround}, \mathsf{subm}, \mathsf{initbid}, \mathsf{bid}, \mathsf{revw}, \mathsf{dsc}, \mathsf{ntf}\} \\ \Sigma_{\mathsf{nums}} & = & \{\mathsf{zero}, \mathsf{one}, \mathsf{two}\} \end{array}$$

The constants in Σ_{tags} correspond to tags used to label messages originating from different phases of the protocol. The constants in Σ_{nums} represent numbers, used for instance to assign scores to papers. The initial round of the protocol is represented by the constant zero. Then, if a round is represented by the term t, the next round is represented by the term $\mathsf{succ}(t)$, where succ is an unary function symbol.

Puting all the function symbols together, we obtain the signature

$$\Sigma_{CC} = \Sigma_{\mathsf{ciph}} \cup \Sigma_{\mathsf{tags}} \cup \Sigma_{\mathsf{nums}} \cup \{\mathsf{succ}\}$$

To model our protocol, we first consider the following names:

- c_{shk} to represent a private channel that is used to distribute the symmetric key shared by the panel members.
- c_{pbk} to represent an authenticated channel where the applicants can obtain the public key of the competition.
- c_{round} to represent a channel used by the manager to announce the current round.

Then, the protocol is represented by the process CC (confidentiality in the cloud) defined as follows:

$$CC \stackrel{\text{def}}{=} \mathrm{new}\, c_{\mathsf{shk}};\ \mathrm{new}\, c_{\mathsf{pbk}};\ \mathrm{new}\, c_{\mathsf{round}}; (!M \mid !R \mid !A)$$

where M, R and A respectively model the behaviour of the competition manager, of a panel member (i.e. evaluator) and of an applicant respectively. Note that we do not fix the number of managers, panel members or applicants: our results hold for any number of instances of the protocol. We declare c_{shk} to be new, because the corresponding channel is private. The construct $\mathrm{new}\, c_{\mathsf{pbk}}$ ensures that the data in c_{pbk} is authentic (the intruder can not write data on c_{pbk}), which allows applicants to obtain the correct public key of the competition from M. In addition to sending the public key on c_{pbk}, the manager uploads the public key to the cloud (on a free channel c), thus making it public.

We present A and some parts of R in the following. The description of all processes, including M, is in appendix.

$$\begin{array}{lcl} A & \stackrel{\text{def}}{=} & \mathrm{new}\, ida; !A_{\mathsf{reg}} \\ A_{\mathsf{reg}} & \stackrel{\text{def}}{=} & \mathrm{new}\, \lambda; \mathrm{new}\, k; \mathrm{new}\, r; \mathrm{in}(c_{\mathsf{pbk}}, x_{\mathsf{pbk}}); \\ & & \text{let } key = \mathsf{aenc}(x_{\mathsf{pbk}}, r, k) \text{ in } \mathrm{out}(c, (\lambda, ida, key)); !A_{\mathsf{subm}} \\ A_{\mathsf{subm}} & \stackrel{\text{def}}{=} & \mathrm{in}(c_{\mathsf{round}}, x_{\mathsf{round}});\ \mathrm{new}\, s;\ \mathrm{new}\, r; \\ & & \text{let } x_{\mathsf{subm}} = (\lambda, ida, \mathsf{senc}(k, r, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, ida, s))) \text{ in} \\ & & \mathrm{out}(c, x_{\mathsf{subm}}); \mathrm{in}(c, x_{\mathsf{ntf}}) \end{array}$$

An applicant can register to many different competitions (the process $!A_{\mathsf{reg}}$). During registration, he generates an identifier λ and a submission key k, and obtains from c_{pbk} the corresponding

public key x_{pbk}. The ciphertext $\mathsf{aenc}(x_{\mathsf{pbk}}, r, k)$ is then uploaded to the cloud, using a public channel c. After registration, the applicant can make any number of submissions (the process $!A_{\mathsf{subm}}$). For each submission, the applicant first determines the current round of the competition (stored in x_{round}), creates a new submission s, and then uploads to the cloud a corresponding ciphertext for this submission, that is $\mathsf{senc}(k, r, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, ida, s))$. Note that we attach a tag $(\mathsf{subm}, x_{\mathsf{round}})$ to the plaintext in order to specify that it corresponds to a submission made in round x_{round}.

$$
\begin{array}{lcl}
R & \stackrel{\text{def}}{=} & \text{new}\, idr;\ \text{out}(c, idr);\ !\text{in}(c_{\mathsf{shk}}, x_{\mathsf{shk}});\ !R_{\mathsf{conflicts}} \mid !R_{\mathsf{round}} \\
R_{\mathsf{round}} & \stackrel{\text{def}}{=} & \text{in}(c_{\mathsf{round}}, x_{\mathsf{round}});\ \text{in}(c, x_{db});\ (!R_{\mathsf{bids}} \mid !R_{\mathsf{review}}) \\
R_{\mathsf{review}} & \stackrel{\text{def}}{=} & \text{in}(c, x_{\mathsf{subm}}); \text{let}\ (x_\mu, idr, x_{\mathsf{ciph}}) = x_{\mathsf{subm}}\ \text{in} \\
 & & \text{let}\ ((\mathsf{initbid}, x_{\mathsf{round}}), x'_\mu, x_\lambda, x_{ida}, x_k) = \mathsf{sdec}(x_{\mathsf{shk}}, x_{\mathsf{ciph}})\ \text{in} \\
 & & \text{new}\, \mathsf{review}; \text{in}(c_{\mathsf{nums}}, x_{\mathsf{score}}); \text{new}\, r; \\
 & & \text{let}\ rev = (x_\mu, idr, \mathsf{senc}(x_{\mathsf{shk}}, r, ((\mathsf{revw}, x_{\mathsf{round}}), x'_\mu, x_\lambda, x_k, \mathsf{review}, x_{\mathsf{score}}, \mathsf{zero})))\ \text{in} \\
 & & \text{out}(c, rev); R_{\mathsf{dsc}}
\end{array}
$$

where $R_{\mathsf{conflicts}}, R_{\mathsf{bids}}, R_{\mathsf{dsc}}$ are defined in appendix.

An evaluator can participate in any number of competitions, after he chooses an identity and submits it to the cloud (on the channel c). For each competition, the panel member receives the shared-key of the panel members on the channel c_{shk}, declares conflicts with any number of submissions (the process $!R_{\mathsf{conflicts}}$) and participates in any number of competition rounds (the process $!R_{\mathsf{round}}$). In each round, R can bid for applications and evaluate applications. The input $\text{in}(c, x_{db})$ models the download of all submissions of the current round from the cloud. However, a panel member can access a submission only if he gets the submission key, which is modeled in R_{review}. The tuple $(x_\mu, idr, x_{\mathsf{ciph}})$ has been prepared by key translation and uploaded to the cloud by the manager M. M ensures that x_{ciph} is indeed an encryption of $((\mathsf{initbid}, x_{\mathsf{round}}), x'_\mu, x_\lambda, x_{ida}, x_k)$ with x_{shk}, which allows R to obtain the submission key x_k for a given paper. Then, R uploads a similar tuple to the cloud, where the ciphertext contains a different tag, an evaluation and a score for the paper.

4.3 Properties and analysis

In this section we explain how the desired secrecy and unlinkability properties are formally defined in the process calculus that we consider, and how they can be automatically verified with ProVerif. We define both secrecy and unlinkability as equivalences between processes, adapting the classical approach of [38, 5, 24] to our context.

4.3.1 Observational equivalence.

For a process A, we write $A \Downarrow c$ when A can evolve into a process that can send a message on c, that is, when $A \to^* C[\overline{c}\langle M\rangle.P]$ for some term M, some evaluation context $C[\_]$ that does not bind c (i.e. $c \notin \text{bn}(C[\_])$), and some process P.

Definition 1 *Observational equivalence ($\approx$) is the largest symmetric relation $\mathcal{R}$ between processes such that $A\ \mathcal{R}\ B$ implies:*

1. *if $A \Downarrow c$, then $B \Downarrow c$.*
2. *if $A \to^* A'$ then, for some B', we have $B \to^* B'$ and $A'\ \mathcal{R}\ B'$;*
3. *$C[A]\ \mathcal{R}\ C[B]$ for all closing evaluation contexts $C[\_]$.*

We will express secrecy and unlinkability as the observational equivalence of two processes, that share the same operational structure and differ only on data that they handle.

4.3.2 Formal definition of security properties.

To express security properties we will need to consider particular applicants and evaluators in interaction with the rest of the system. For this we consider a hole in the process CC, where we can plug any process, *i.e.* we let:

$$CC[\_] \stackrel{\text{def}}{=} \text{new}\, c_{\mathsf{shk}};\ \text{new}\, c_{\mathsf{pbk}};\ \text{new}\, c_{\mathsf{round}};\ (!M \mid !R \mid !A \mid \_\ )$$

To express applicants and evaluators who submit some specific data (of which the privacy will be tested), we consider the following *witness* processes:

- $A_{\mathsf{subm}}(ida, s, \lambda, k)$ - for an applicant whose identity is ida, whose random identifier is λ and whose submission key is k. Moreover, among the submissions from ida, there is a particular one whose content is s.
- $R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, sc)$ - for an evaluator whose identity is idr and that behaves like a regular evaluator, with the single difference that amongst other evaluations, he also evaluates a submission whose identifiers are (λ, ida, k), to which it assigns the evaluation rev and a score sc.
- $R_{\mathsf{mu}}(idr, \mu)$ - is an evaluator that has been assigned to evaluate a submission from an applicant whose encrypted submission key is associated to μ in the database $\mathrm{DB}^{r}_{\mathrm{Keys}}$ after key translation (i.e. the conflicts declaration phase in Figure 1).
- $M_{\mathsf{ar}}(ida_1, \mu_1, idr_1, ida_2, \mu_2, idr_2)$ - is a manager that additionally ensures that, for all $i \in \{1, 2\}$, to the encrypted submission key of ida_i is assigned the identifier μ_i after key translation (i.e. the conflicts declaration phase in Figure 1). Moreover, idr_i is among the panel members that evaluate the submission from ida_i corresponding to label μ_i.

The formal definition of these processes is detailed in Appendix B.

4.3.3 Secrecy properties.

To formalise the considered secrecy properties, we rely on the notion of strong secrecy defined in [14].

Application secrecy. We say that a competition management protocol satisfies strong secrecy of applications if, even if the cloud initially knows s_1 and s_2, the cloud cannot make a distinction between an execution of the protocol where an applicant submitted an application containing s_1 and an execution where the same applicant has submitted s_2.

To formally capture this, we construct from $CC[\_]$ two processes: in the first one the hole is filled with an applicant that submits the publicly known (*i.e.* free) value s_1, and in the second one the hole is filled with that applicant submitting the publicly known (*i.e.* free) value s_2. We verify using ProVerif that these two processes are observationally equivalent:

$$CC[\text{new}\,\lambda;\ \text{new}\,k;\ A_{\mathsf{subm}}(ida, s_1, \lambda, k)] \ \approx\ CC[\text{new}\,\lambda;\ \text{new}\,k;\ A_{\mathsf{subm}}(ida, s_2, \lambda, k)]$$

Score secrecy. In order to model the strong secrecy of scores, we build from $CC[\_]$ one process in which the hole is filled with a panel member that attributes one to some submission, and one process in which the hole is filled with the same panel member attributing two to it. We use the witness process R_{eval} to construct the different instances of CC:

$$\begin{array}{l} CC[\text{new}\,s;\ \text{new}\,\lambda;\ \text{new}\,k;\ A_{\mathsf{subm}}(ida, s, \lambda, k) \mid \text{new}\,rev;\ R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, \mathsf{one})] \ \approx \\ CC[\text{new}\,s;\ \text{new}\,\lambda;\ \text{new}\,k;\ A_{\mathsf{subm}}(ida, s, \lambda, k) \mid \text{new}\,rev;\ R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, \mathsf{two})] \end{array}$$

Evaluation secrecy. The secrecy of evaluations is defined similarly:

$$CC[\text{new } s;\ \text{new } \lambda;\ \text{new } k;\ A_{\mathsf{subm}}(ida, s, \lambda, k) \mid \text{in}(c_{\mathsf{nums}}, sc);\ R_{\mathsf{eval}}(idr, \lambda, ida, k, \mathsf{review}_1, sc)] \approx$$
$$CC[\text{new } s;\ \text{new } \lambda;\ \text{new } k;\ A_{\mathsf{subm}}(ida, s, \lambda, k) \mid \text{in}(c_{\mathsf{nums}}, sc);\ R_{\mathsf{eval}}(idr, \lambda, ida, k, \mathsf{review}_2, sc)]$$

where review_1 and review_2 are two free names and c_{nums} is a channel used to model the non-deterministic choice of scores.

4.3.4 Applicant-evaluator unlinkability.

This property aims to guarantee that the cloud can not link a given applicant to an evaluator of his application. Of course, to this end, it must be the case that there are at least two applicants, say ida_1 and ida_2, and two evaluators, say idr_1 and idr_2, participating in the competition. Then, we require that the two executions where

- ida_1 submits an application that is evaluated by idr_1 and ida_2 submits an application that is evaluated by idr_2
- ida_1 submits an application that is evaluated by idr_2 and ida_2 submits an application that is evaluated by idr_1

are indistinguishable from the point of view of the cloud. Formally, this amounts to the following property of observational equivalence:

$$CC\left[\begin{array}{l}\text{new } s_1; \text{new } s_2;\\ A_{\mathsf{subm}}(ida_1, s_1) \mid A_{\mathsf{subm}}(ida_2, s_2) \mid\\ M_{\mathsf{ar}}(ida_1, \mu_1, idr_1, ida_2, \mu_2, idr_2) \mid\\ R_{\mathsf{mu}}(idr_1, \mu_1) \mid R_{\mathsf{mu}}(idr_2, \mu_2)\end{array}\right] \approx CC\left[\begin{array}{l}\text{new } s_1; \text{new } s_2;\\ A_{\mathsf{subm}}(ida_1, s_1) \mid A_{\mathsf{subm}}(ida_2, s_2) \mid\\ M_{\mathsf{ar}}(ida_1, \mu_2, idr_2, ida_2, \mu_1, idr_1) \mid\\ R_{\mathsf{mu}}(idr_1, \mu_1) \mid R_{\mathsf{mu}}(idr_2, \mu_2)\end{array}\right]$$

Automated analysis. We use the ProVerif tool to prove that all four eqivalence properties defined above are true in our model. The complete specification of these properties in ProVerif is available online [33].

5 Implementation of ConfiChair

In order to evaluate the performance and usability of the protocol, we have implemented it for conference management systems in a system we call ConfiChair. The ideal implementation of our protocol would look and feel very similar to existing cloud-based conference management systems such as OpenConf, EDAS and EasyChair, and should require no additional software beyond a web browser.

We constructed a prototype implementation [33], in order to discover any potential problems with a practical implementation and to find how much time and memory such a system may require, both on server-side and on client-side.

5.1 Overview.

We implemented the ConfiChair prototype so that only a browser is necessary for participating as an author, a reviewer, or a chair. Overall, our prototype of ConfiChair feels very similar to current web-based management systems. A user of the system can perform his usual tasks by simply clicking a few buttons.

For example, to submit a paper an author logs to his ConfiChair account, selects the link for the conference to which he wants to submit, clicks the *new submission* button, selects the PDF file of his paper and clicks the *submit* button to complete his submission. All the key generation and the secure storing, as well as the encryption dictated by our protocol is transparently performed

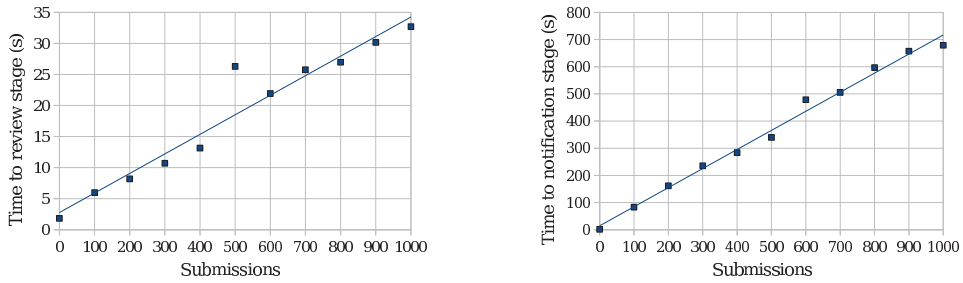


Figure 4: Performance evaluation. The time taken for transitioning to the review stage is about 25s for 500 papers. The time for transitioning to the notification phase is about 350s for 500 papers.

by the browser. The only aspect not currently performed by the browser is the retrieval of the conference public-key $\mathsf{pub}(\mathrm{Comp})$; this key must be manually input by the author (by copy-paste from the call for papers for example).

Similarily, the chair of a conference wanting to create a ConfiChair page for his conference Comp, logs into his ConfiChair account and clicks the *create new conference* button. His browser will transparently generate and securely store the keys K_{Comp}, $\mathsf{pub}(\mathrm{Comp})$, and $\mathsf{priv}(\mathrm{Comp})$.

5.2 Performance.

The system is expected to handle hundreds of papers without overhead on the chair. In particular, browser-side re-encryption and mixing while transitioning between phases should not take more than a few minutes. From that perspective, the results of our experiments with the prototype implementation are promising. They are presented in figure 4. Also, experiments with any number of random files can be easily re-run on [33]. The tables show the waiting time for a corresponding number of papers when transiting between phases: always within a few minutes. The submissions in our experiments are PDF files of scientific papers, thus reflecting a real-case situation.

5.3 Transparency of key management.

To hide the complexity of the encryption keys from the user, these are managed and retrieved by the browser transparently when logging to ConfiChair. The login procedure implemented relies on each user having an identity id and a secret password psw_{id} from which the browser derives two keys: the ConfiChair account key $\mathsf{Kdf}_1(psw_{id})$ to authenticate the user to the the cloud provider, and a second key $\mathsf{Kdf}_2(psw_{id})$ used to encrypt the key purse of the user. This key purse contains the set of keys generated by the browser in previous accesses to the ConfiChair system, for example submission keys if the user has used ConfiChair as an author in the past, or conference keys if he has used it as a programme committee member.

When submitting a paper, the author's browser generates a symmetric key k which it uses to automatically encrypt the paper before sending it to the cloud. This key k is in turn added to the key purse of the user, which is uploaded encrypted with $\mathsf{Kdf}_2(psw_{authorid})$ to the cloud. To the submitter, this does not look like anything other than a normal file upload. Similarly, when the chair moves the conference to the review stage, it appears to be just like clicking on a normal link, since the chair's browser has already retrieved from the cloud the chair's key purse, and decrypted it with $\mathsf{Kdf}_2(psw_{chairid})$, and can then transparently decrypt and reencrypt the submissions according to the protocol.

In this way, the only key that needs to be securely backed up by a user id is his ConfiChair password psw_{id}. All the other keys are stored in encrypted form in the cloud, and retrieved when

needed by his browser.

Currently, the authors need to copy and paste from the call for papers the public key of the conference pub(Comp) to which they want to submit, and the reviewers need to copy and paste from their email the shared-key of the conference K_{Comp} for which they are reviewers.

5.4 Future improvements

In contrast to the ideal system that we envisage, our prototype requires the use of a Java plug-in for the users' browsers, since a Java applet is used to provide cryptographic routines. These routines are called from the JavaScript code using LiveConnect. An alternative to Java would be to use HTML5 which, unlike previous versions of HTML, provides the necessary features to implement ConfiChair, such as the possibility to program on-the-fly stream encryption and decryption. However, as the experiences of [7, 8, 27] suggest, Java applets are not necessarily an impediment in the usability and the take-up of the system. Moreover, while the use of the Java plug-in may look unattractive to some, it presents the following two advantages over HTML5:

- HTML5 is not yet widely adopted. Only the Chrome browser currently supports all the necessary features of HTML5 that an implementation of ConfiChair would require.
- In order for the user to trust the code that their web browser runs, the code should be reviewed, certified and signed by one or more trusted parties. The user's web browser would then verify the certificates without the user's intervention. Currently Java applets are the only way to achieve this.

All these implementation platform related issues will be further investigated in the future, for a real implementation and deployment of the ConfiChair protocols.

6 Conclusion

The accumulation of sensitive data on servers around the world is a major problem for society, and will be considerably exacerbated by the anticipated take-up of cloud-computing technology. Confidential data about the applications and evaluations of different types of bids and tenders is currently stored on a relatively small set of cloud-based competition management systems. For example, in the case of conference management, the authoring and reviewing performance of tens of thousands of researchers across thousands of conferences is stored by one or two well-known cloud-based systems [36].

We have introduced a general technique that can be used to address this problem in a wide variety of circumstances, namely, the technique of translating between keys and mixing data in a trustworthy browser. We have proposed a protocol underlying a competition management system that uses this technique to obtain strong privacy properties while having all the advantages of cloud computing. In this system, the cloud sees sensitive data only in encrypted form, with no single person holding all the encryption keys (our protocol uses a different key for each competition). The competition manager's browser decrypts data with one key and encrypts it with possibly another one, while mixing and re-randomising to ensure unlinkability properties.

We are able to state and prove strong secrecy and unlinkability properties for the protocol. It still enables the cloud provider to route information to the necessary managers, evaluators and applicants, to enforce access control, and optionally to perform statistics collection. In our implementation of ConfiChair, we have demonstrated that the cryptography and key management can be handled by a regular web browser [33] (specifically, we used LiveConnect). We plan to continue developing our prototype into a complete system.

An important design decision in our protocol and in ConfiChair is the fact that a single key K_{Comp} is used to encrypt all the information for the competition. Stronger secrecy properties could be obtained if a different key were used for different subsets of evaluators and applicants, but this would be at the cost of simplicity. Using a single key per competition seems to strike a

good balance between usability and security. Finer-grained access control is implemented (as on current systems) by the cloud, e.g. for managing the conflicts of interest.

In further work, we intend to apply the ideas to work with other cloud-computing applications (such as those mentioned in the introduction), and to provide a framework for expressing secrecy and unlinkability properties in a more systematic way.

Acknowledgments. Thanks to Joshua Phillips for much help with the implementation and typesetting. We also gratefully acknowledge financial support from EPSRC via the projects *Trust Domains* (TS/I002529/1) and *Trustworthy Voting Systems* (EP/G02684X/1).

References

[1] eTenderer. `http://www.etenderer.com/public.aspx`.

[2] INDECO tender management system. `http://www.indeco.co.uk/index.php?id=tendermanagementsystem`.

[3] Tendering for public contracts. A guide for small businesses. `http://www.bis.gov.uk/files/file39469.pdf`.

[4] VHTender - tender management system. `http://www.vhsoft.com/products/vhtender.htm`.

[5] Martín Abadi. Security protocols and their properties. In *Foundations of Secure Computation, NATO Science Series*, pages 39–60. IOS Press, 2000.

[6] Martín Abadi and Cédric Fournet. Mobile values, new names, and secure communication. In *Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01)*, pages 104–115, January 2001.

[7] Ben Adida. Helios: Web-based open-audit voting. In Paul C. van Oorschot, editor, *USENIX Security Symposium*, pages 335–348. USENIX Association, 2008.

[8] Ben Adida, Olivier Pereira, Olivier De Marneffe, and Jean-Jacques Quisquater. Electing a university president using open-audit voting: Analysis of real-world use of helios. In *In Electronic Voting Technology/Workshop on Trustworthy Elections (EVT/WOTE*, 2009.

[9] Myrto Arapinis, Sergiu Bursuc, and Mark Ryan. Privacy supporting cloud computing: Confichair, a case study. In Pierpaolo Degano and Joshua D. Guttman, editors, *POST*, volume 7215 of *Lecture Notes in Computer Science*, pages 89–108. Springer, 2012.

[10] Randolph Baden, Adam Bender, Neil Spring, Bobby Bhattacharjee, and Daniel Starin. Persona: an online social network with user-defined privacy. In Pablo Rodriguez, Ernst W. Biersack, Konstantina Papagiannaki, and Luigi Rizzo, editors, *SIGCOMM*, pages 135–146. ACM, 2009.

[11] Olivier Baudron, Pierre-Alain Fouque, David Pointcheval, Jacques Stern, and Guillaume Poupard. Practical multi-candidate election system. In *PODC*, pages 274–283, 2001.

[12] John Bethencourt, Amit Sahai, and Brent Waters. Ciphertext-policy attribute-based encryption. In *IEEE Symposium on Security and Privacy*, pages 321–334. IEEE Computer Society, 2007.

[13] Bruno Blanchet. An efficient cryptographic protocol verifier based on Prolog rules. In *Computer Security Foundations Workshop (CSFW'01)*, 2001.

[14] Bruno Blanchet. Automatic proof of strong secrecy for security protocols. In *IEEE Symposium on Security and Privacy*, pages 86–, 2004.

[15] Bruno Blanchet, Martín Abadi, and Cédric Fournet. Automated verification of selected equivalences for security protocols. *Journal of Logic and Algebraic Programming*, 2007.

[16] Dan Boneh, Craig Gentry, and Brent Waters. Collusion resistant broadcast encryption with short ciphertexts and private keys. In Victor Shoup, editor, *CRYPTO*, volume 3621 of *Lecture Notes in Computer Science*, pages 258–275. Springer, 2005.

[17] Matteo Bortolozzo, Matteo Centenaro, Riccardo Focardi, and Graham Steel. Attacking and fixing PKCS#11 security tokens. In *ACM Conference on Computer and Communications Security*, pages 260–269, 2010.

[18] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[19] Iliano Cervesato, Aaron D. Jaggard, Andre Scedrov, Joe-Kai Tsay, and Christopher Walstad. Breaking and fixing public-key kerberos. *Inf. Comput.*, 206:402–424, February 2008.

[20] C. Chatmon, T. van Le, and T. Burmester. Secure anonymous RFID authentication protocols. Technical Report TR-060112, Florida Stat University, Department of Computer Science, 2006.

[21] Tom Chothia and Vitaly Smirnov. A traceability attack against e-passports. In *Financial Cryptography*, 2010.

[22] Sebastian Clauß, Dogan Kesdogan, Tobias Kölsch, Lexi Pimenidis, Stefan Schiffner, and Sandra Steinbrecher. Privacy enhancing identity management: Protection against reidentification and profiling. In *Proceedings of the 2005 ACM Workshop on Digital Identity Management*, 2005.

[23] Cloud Security Alliance. *Secure Cloud*. `www.cloudsecurityalliance.org/sc2010.html`, 2010.

[24] Stéphanie Delaune, Steve Kremer, and Mark D. Ryan. Verifying privacy-type properties of electronic voting protocols. *Journal of Computer Security*, 17(4):435–487, July 2009.

[25] C. Gentry. Fully homomorphic encryption using ideal lattices. In *41st ACM Symposium on Theory of Computing (STOC)*, 2009.

[26] Saikat Guha, Kevin Tang, and Paul Francis. NOYB: Privacy in online social networks. In *In Proceedings of the First ACM SIGCOMM Workshop on Online Social Networks*, 2008.

[27] Yan Huang and David Evans. Private editing using untrusted cloud services. In *Second International Workshop on Security and Privacy in Cloud Computing, Minneapolis, Minnesota. 24 June 2011.*

[28] Markus Jakobsson, Ari Juels, and Ronald L. Rivest. Making mix nets robust for electronic voting by randomized partial checking. In Dan Boneh, editor, *USENIX Security Symposium*, pages 339–353. USENIX, 2002.

[29] Ari Juels, Dario Catalano, and Markus Jakobsson. Coercion-resistant electronic elections. In Vijay Atluri, Sabrina De Capitani di Vimercati, and Roger Dingledine, editors, *WPES*, pages 61–70. ACM, 2005.

[30] Swee-Won Lo, Raphael C.-W. Phan, and Bok-Min Goi. On the security of a popular web submission and review software (WSaR) for cryptology conferences. In *WISA'07: Proceedings of the 8th international conference on Information security applications*, pages 245–265, Berlin, Heidelberg, 2007. Springer-Verlag.

[31] Gavin Lowe. An attack on the Needham-Schroeder public-key authentication protocol. *Information Processing Letters*, 56(3):131–133, 1996.

[32] Siani Pearson, Yun Shen, and Miranda Mowbray. A privacy manager for cloud computing. In Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors, *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, pages 90–106, 2009.

[33] Joshua Phillips and Matt Roberts. ConfiChair - prototype privacy-supporting conference management system. `http://www.confichair.org`.

[34] Krishna P. N. Puttaswamy, Christopher Kruegel, and Ben Y. Zhao. Silverline: Toward data confidentiality in third-party clouds. Technical Report 08, University of California Santa Barbara, 2010.

[35] Hasan Qunoo and Mark Ryan. Modelling dynamic access control policies for web-based collaborative systems. In *Data and Applications Security and Privacy XXIV*, volume 6166 of *LNCS*, pages 295–302, 2010.

[36] Mark D. Ryan. Cloud computing privacy concerns on our doorstep. *Communications of the ACM*, 54(1):36–38, 2011.

[37] Ahmad-Reza Sadeghi, Thomas Schneider, and Marcel Winandy. Token-based cloud computing. In Alessandro Acquisti, Sean W. Smith, and Ahmad-Reza Sadeghi, editors, *Trust and Trustworthy Computing, Third International Conference, TRUST 2010, Berlin, Germany, June 21-23, 2010. Proceedings*, pages 417–429, 2010.

[38] Steve Schneider and Abraham Sidiropoulos. CSP and anonymity. In *ESORICS*, pages 198–218, 1996.

[39] Andrei Voronkov et al. EasyChair Conference System. `http://www.easychair.org`.

A Formal definition of the protocol

We will make use of the following syntactic sugar of ProVerif: in the term M from the expressions "let $M = D$ in P else Q" and "in(c, M)" one is allowed to have subterms of the form "$=N$", for some term N. This simply means that the term N has been previously defined in the process, and therefore the instantiation of M has to be consistent with the previous instantiation of N.

```
(**************************************************************************
SIGNATURE & EQUATIONAL THEORY
**************************************************************************)

free c.

fun pub/1. fun aenc/3.
reduc adec(x, aenc(pub(x), y, z)) = z.

fun senc/3.
reduc sdec(x, senc(x, y, z)) = z.

fun succ/1. fun zero/0. fun one/0. fun two/0.

fun empty/0.
fun reg/0. fun initround/0. fun subm/0. fun initbid/0.
fun bid/0. fun revw/0. fun dsc/0. fun ntf/0.

(**************************************************************************
PANEL CHAIR
**************************************************************************)
```

```
let Chair =
    new shkCompetition; new pvkCompetition;
    (Chair_Init | (!Chair_Conflicts_And_Key_Translation) | (!Chair_Round)).

let Chair_Init =
    (!out(cshkCompetition, shkCompetition)) |
    (!out(cpbkCompetition, pub(pvkCompetition))) | (!out(c, pub(pvkCompetition))).

let Chair_Conflicts_And_Key_Translation =
    new mu; new r;
    in(c, (xidReviewer, xblob));
    in(c, (xlambda, xidApplicant, xreg));
    let (xcft, =xidReviewer, =xlambda, =xidApplicant) = sdec(shkCompetition, xblob) in
    ( (*KEY TRANSLATION*)
    let (=reg, =xlambda, =xidApplicant, xk) = adec(pvkCompetition, xreg) in (
    let rSubm = (mu, senc(shkCompetition, r, (initround, mu, xlambda, xidApplicant, xk))) in (
    out(c, rSubm);
    out(c,(mu,xcft,xidReviewer))
    ))).

let Chair_Round = out(dround,zero) | (*FIRST ROUND*)
    in(dround, xround);
    out(cround, xround); out(cround, xround); out(dround, succ(xround));
    ((!Chair_Initbids) | (!Chair_Assign) | (!Chair_Notify)).

let Chair_Initbids =
    in(c, (xmu, xblob));
    let (=initround, xmu', xlambda, xidApplicant, xk) = sdec(shkCompetition, xblob) in (
    new r;
    let initbids =  senc(shkCompetition, r, ((initbid, xround), xmu', xlambda, xidApplicant, xk)) in (
    out(c, (xmu, initbids)))).


let Chair_Assign =
    in(c, (xmu, xblob));
    in(c, xreviewer); in(c, xbids); in(c, xcfts);
    out(c, (xmu, xreviewer,xblob)).

let Chair_Notify =
    in(c, (ymu, yidReviewer, yblob));
    let ((=dsc, =xround), ymu', ylambda, yk, yreview, yscore, ydiscussion) = sdec(shkCompetition, yblob)
    in ( in(cnums, yntf); new r;
    let notification = (ylambda, senc(yk, r, ((ntf, xround), ylambda, yntf, yreview))) in (
    out(c, notification))).

(*****************************************************************************
PANEL MEMBER
*****************************************************************************)

let Reviewer = new idReviewer; out(c, idReviewer); (!Reviewer_Init).

let Reviewer_Init =
    in(cshkCompetition, xshk);
    ((!Reviewer_Conflicts) | (!Reviewer_Round)).

let Reviewer_Conflicts =
    in(c, (xlambda, xidApplicant));
    in(cnums, xcft); new r;
```

```
    out(c, (idReviewer, senc(xshk, r, (xcft, idReviewer, xlambda, xidApplicant)))).

let Reviewer_Round =
    in(cround, xround);in(c, xsubmissions);
    ((!Reviewer_Bids) | (!Reviewer_Review)).

let Reviewer_Bids =
    in(c, (xmu, xblob));
    let ((=initbid, =xround), xmu', xlambda, xidApplicant, xk) = sdec(xshk, xblob) in (
    in(cnums, xbid); new r;
    let bids = (idReviewer, senc(xshk, r, ((bid, xround), idReviewer, xmu'))) in (
    out(c, bids))).

let Reviewer_Review =
    in(c, (xmu, =idReviewer, xblob));
    let ((=initbid, =xround), xmu', xlambda, xidApplicant, xk) = sdec(xshk, xblob) in (
    new review; new r;in(cnums, xscore);
    let rev = (xmu, idReviewer,
               senc(xshk, r, ((revw, xround), xmu', xlambda, xk, review, xscore, empty))) in (
    out(c, rev);
    Reviewer_Discussion)).

let Reviewer_Discussion =
    in(c, (ymu, yblob));
    let  ((=revw, =xround), ymu', ylambda, yk, yreview, yscore, yd) = sdec(xshk, yblob) in (
    new discussion; new r;
    let disc = (ymu, idReviewer,
                senc(xshk, r, ((dsc, xround), ymu', ylambda, yk, yreview, yscore, (yd, discussion)))) in
    out(c, disc)).

(*****************************************************************************
APPLICANT
*****************************************************************************)

let Applicant = new idApplicant; (!Applicant_Registration).

let Applicant_Registration =
    new lambda; new k; new r;
    in(cpbkCompetition, xpbk);
    out(c, (lambda, idApplicant, aenc(xpbk, r, (reg, lambda, idApplicant, k))));
    (!Applicant_Round).

let Applicant_Round =
    in(cround, xround);
    Applicant_Submission.

let Applicant_Submission =
    new s; new r;
    let submission = (lambda, idApplicant, senc(k, r, ((subm, xround), lambda, idApplicant, s))) in
    out(c, submission).


(*****************************************************************************
NUMBERS
*****************************************************************************)

let Nums = (!(out(cnums, one))) | (!(out(cnums, two))).
```

```
(*****************************************************************************
THE PROTOCOL
*****************************************************************************)

process new cpbkCompetition; new cshkCompetition;
new cnums; new dround; new cround;
( Nums | (!Chair) | (!Reviewer) | (!Applicant) )
```

B Formal definition of witnessing processes

As explained in section 4.3, to formalise our secrecy and unlinkability properties we need to consider particular instances of applicants, of reviewers and of the competition manager. These processes are called witnesses (of the considered property). The witnessing applicants are instances of the process A_{subm}. According to the considered property, the witnessing reviewers are either instances of R_{eval} or of R_{mu}. For the property of applicant-evaluator unlinkability, we have to consider the witnessing manager M_{ar}.

B.1 Definition of $A_{\mathsf{subm}}(ida, s, \lambda, k)$

$A_{\mathsf{subm}}(ida, s, \lambda, k)$ models an applicant whose identity is ida, whose random identifier is λ and whose submission key is k. Moreover, among the submissions from ida, there is a particular one whose content is s. The main idea is to consider the definition of A and ommit the pieces of code where ida, s, λ, k are instantiaded, because now they are given as parameters to the process. We have:

$$
\begin{array}{lcl}
A_{\mathsf{subm}}(ida, s, \lambda, k) & \stackrel{\text{def}}{=} & !A_{\mathsf{reg}} \\
A_{\mathsf{reg}} & \stackrel{\text{def}}{=} & \text{new}\, r; \text{in}(c_{\mathsf{pbk}}, x_{\mathsf{pbk}}); \\
 & & \text{let}\ key = \mathsf{aenc}(x_{\mathsf{pbk}}, r, k)\ \text{in}\ \text{out}(c, (\lambda, ida, key)); !A_{\mathsf{subm}} \\
A_{\mathsf{subm}} & \stackrel{\text{def}}{=} & \text{in}(c_{\mathsf{round}}, x_{\mathsf{round}});\ \text{new}\, r; \\
 & & \text{let}\ x_{\mathsf{subm}} = (\lambda, ida, \mathsf{senc}(k, r, ((\mathsf{subm}, x_{\mathsf{round}}), \lambda, ida, s)))\ \text{in} \\
 & & \text{out}(c, x_{\mathsf{subm}}); \text{in}(c, x_{\mathsf{ntf}})
\end{array}
$$

B.2 Definition of $R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, sc)$

$R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, sc)$ models a reviewer whose identity is idr and that behaves like a regular reviewer, with the single difference that amongst other evaluations, he also evaluates a submission whose identifiers are (λ, ida, k), to which it assigns a review rev and a score sc. Again, the idea is to ommit the creation of $idr, \lambda, ida, k, rev, sc$ in the definition of R and to use the given parameters instead. We have:

```
let R_eval(idr, lambda,ida,k,rev,sc) =
    let idReviewer = idr in out(c, idReviewer); (!Reviewer_Init_Wtn).

let Reviewer_Init_Wtn =
    in(cshkCompetition, xshk); ((!Reviewer_Conflicts) | (!Reviewer_Round_Wtn)).

let Reviewer_Round_Wtn =
    in(cround, xround); in(c, xsubmissions);
    ((!Reviewer_Bids) | (!Reviewer_Review) | Reviewer_Review_Wtn).

let Reviewer_Review_Wtn =
    let (review,score) = (rev,sc) in
    new r; in(cnums, score);
    in(c, (xmu, =idReviewer, xblob));
    let ((=initbid, =xround), xmu', =lambda, =ida, =k) = sdec(xshk, xblob) in (
    let rev = (xmu, idReviewer,
```

```
            senc(xshk, r, ((revw, xround), xmu', lambda, k, review, score, zero))) in (
    out(c, rev); Reviewer_Discussion)).
```

where Reviewer_Conflicts,Reviewer_Bids,Reviewer_Review, Reviewer_Discussion are defined in appendix A.

B.3 Definition of $R_{\mathsf{mu}}(idr, \mu)$

$R_{\mathsf{mu}}(idr, \mu)$ is very similar to $R_{\mathsf{eval}}(idr, \lambda, ida, k, rev, sc)$. We simply fix the parameter μ instead of other parameters:

```
let R_mu(idr,mu) =
    let idReviewer = idr in out(c, idReviewer); (!Reviewer_Init_Wtn).

let Reviewer_Init_Wtn =
    in(cshkCompetition, xshk); ((!Reviewer_Conflicts) | (!Reviewer_Round_Wtn)).

let Reviewer_Round_Wtn =
    in(cround, xround); in(c, xsubmissions);
    ((!Reviewer_Bids) | (!Reviewer_Review) | Reviewer_Review_Wtn).

let Reviewer_Review_Wtn =
    in(c, (=mu, =idReviewer, xblob));
    let ((=initbid, =xround), =mu, xlambda, xidApplicant, xkey) = sdec(xshk, xblob) in (
    new review; new rw; in(ccoin,score);
    let rev = (mu, idReviewer,
              senc(xshk, rw, ((revw, xround), mu, xlambda, xkey, review, score, empty))) in (
    out(c, rev); Reviewer_Discussion)).
```

B.4 Definition of $M_{\mathsf{ar}}(ida_1, \mu_1, idr_1, ida_2, \mu_2, idr_2)$

$M_{\mathsf{ar}}(ida_1, \mu_1, idr_1, ida_2, \mu_2, idr_2)$ models a manager that additionally ensures that, for all $i \in \{1, 2\}$, to the encrypted submission key of ida_i is assigned the identifier μ_i after key translation (i.e. the conflicts declaration phase in Figure 1). Moreover, idr_i is among the reviewers that evaluate a submission from ida_i. Assuming that ida_i has submission identifier $lambda_i$ and submission key key_i, we have the following code (the processes not defined below are as in appendix A):

```
 let M_ar(ida_1,mu_1,idr_1,ida_2,mu_2,idr_2) =
    new shkCompetition; new pvkCompetition;
    (Chair_Init | (!Chair_Conflicts) | Chair_Conflicts_Wtn | (!Chair_Round)).

let Chair_Conflicts_Wtn =
    in(c, (xidReviewer1, xblob1));
    in(c, (xidReviewer2, xblob2));
    in(c, (=lambda_1, =ida_1, xreg1));
    in(c, (=lambda_2, =ida_2, xreg2));

    let (xcft1, =xidReviewer1, =lambda_1, =ida_1) = sdec(shkCompetition, xblob1) in
    let (xcft2, =xidReviewer2, =lambda_2, =ida_2) = sdec(shkCompetition, xblob2) in

    let (=reg, =lambda_1, =ida_1, =key_1) = adec(pvkCompetition, xreg1) in
    let (=reg, =lambda_2, =ida_2, =key_2) = adec(pvkCompetition, xreg2) in

    new rw1; new rw2;
```

```
    let rSubm1 = (mu1, senc(shkCompetition, rw1, (initround, mu1,lambda_1,ida_1,key_1))) in
    let rSubm2 = (mu2, senc(shkCompetition, rw2, (initround, mu2,lambda_2,ida_2,key_2))) in
    (
    out(c, rSubm1); out(c, rSubm2);
    out(c,(mu1, xcft1, xidReviewer1));
    out(c,(mu2, xcft2, xidReviewer2))
    ).

let Chair_Round =
    in(dround, xround); out(cround, xround); out(cround, xround); out(dround, succ(xround));
    ( !Chair_Initbids | !Chair_Assign | Chair_Assign_Wtn | !Chair_Notify ).

let Chair_Assign_Wtn =
    in(c, (=mu1,xblob1));
    in(c, (=mu2,xblob2));
    in(c, xbids);in(c, xcfts);
    let assign1 = (mu1,idr_1,xblob1) in
    let assign2 = (mu2,idr_2,xblob2) in
    out(c, assign1);out(c, assign2).
```

User Defined Information Flow Policy for Web Service Orchestration

Thomas Demongeot[1,2], Eric Totel[3], and Valérie Viet Triem Tong[3]

[1] DGA - Maîtrise de l'information, La Roche Marguerite, BP 57419, 35171 Bruz Cedex
[2] TELECOM Bretagne, 2 rue de la Chataigneraie, CS 17607, 35576 Cesson-Sévigné Cedex, France
[3] Supélec, Avenue de la Boulaie, BP 81127, F-35511 Cesson-Sévigné Cedex, France

Abstract. Web Services are currently the base of a lot a e-commerce applications. Nevertheless, the clients often use these services without knowing anything about their internals. Moreover, they have no clue about the use of their personal data inside the global applications. Some programming languages allow the orchestration of Web Services within Service-Oriented Architecture (SOA). As one feature of SOAs is the dynamic discovery of services actually used during execution, an orchestration user does not know prior to the execution how, and by who, the data he provides will be used. In this paper, we offer the opportunity to the user to specify constraints on the use of its personal data and to the provider of an orchestration service to propose a model of information flow policy. To ensure the privacy of data at runtime, we propose an information flow policy model. The policies are configured at runtime by the user of the orchestration. The policies ensure that no information flow can be produced from the user data to unauthorized services. However, the dynamic aspects of the web services lead to situations where the policy prohibits the nominal operation of the orchestration (e.g., when using a service that is unknown by the user). To solve this problem, we propose to the user to dynamically permit exceptional unauthorized flows. In order to make its decision, the user is provided with all information necessary for decision-making.

1 Introduction

Web services [1] were originally designed as a set of reusable services freely available to everyone. Service-orientation eventually offers an elegant way to build new services composed of existing ones using the notion of orchestration.

On one hand, since services are based on encapsulation, the client does not need to understand how a service works. On the other hand, this lack of information also means that the client does not know how his data are used and by who. Currently, most of the efforts in web service security focus on the confidentiality of the information at the communication protocol level, but do not solve the problem of how to make a specific service orchestration trustable for the clients.

Even if the service orchestration provider is trustable, it has no technical solution to guarantee for a specific client that it satisfies his expectations in terms of data protection. User data protection in a service orchestration is thus crucial.

The expectations of the client must be expressed, which implies some security policy language to be available. In this paper, we propose such an elementary data protection policy configurable by the user of the service and we propose a model to define different type of information flow policy. We propose a model to check this information flow policy at runtime and to modify it when necessary.

The solution proposed here is based on dynamic information flow tracking, thus Section 2 presents a state of the art on this topic. We define a model of Service Oriented Architecture using orchestration (Section 3). We define a model of a generic information flow policy that specifies legal information flows. We define what properties it can provide (Section 4) and how to verify the policy (Section 5). Section 6 presents how to dynamically update the security policy. Finally, we conclude and expose future work in Section 7.

2 Related work

The area of information flow tracking has been well-studied during the last decade. The basic idea of information flow tracking is that sensible data are marked with an identifier sometimes called a taint, a label, a tag or a mark. The marks are propagated along the flow to taint objects in the system. The propagation can be either dynamically observed or statically discovered. Several researches have helped to strengthen the control of data privacy in service orchestrations, particularly by statically controlling data flows. In [3], BPEL[4] is considered as the description of a distributed collaborative system with a multi-level security policy. This policy ensures that data from Web Services are used properly, but it lacks flexibility and does not manage dynamic adaptation. [4] and [5] proposed type systems in order to guarantee non-interference property in dynamic service composition. But the method proposed by [4] needs to analyse each service involved in the orchestration and does not support complex orchestration. In [5] each service involved in the orchestration need to produce a contract describing its internal behaviour and the authors propose a framework to analyse service orchestration. In [6], the authors propose an XML schema for specifying an employment policy of available Web-Services statically verified in BPEL programs. In both cases, security policies are defined by the host of BPEL and do not specify a security policy for each user. Moreover, the verification of information flows is done statically: it is impossible to address the problem of dynamic discovery of services. In [7] and [8] Myers and Liskov propose more expressive marks (which are called labels). A label attached to a value denote both owners and readers of this value. An owner decides which principals can access his data, these principals are *the readers*. In [9] Myers presents Jif, where labels are used to annotate data items in a Java program. Jif checks at compile

[4] BPEL (Business Process Execution Language) is a language used to define orchestrations

time, in a manner similar to type checking, if all the executions of annotated programs verify the information flow policy. In their approach, the information flow policy consists of the definition of the readers by the owner. This policy is defined before the analysis and can be updated by relabelling data. Their model authorises only two relabelling rules: restriction and declassification. Data can only be relabelled from label L_1 to label L_2 if L_2 is more restrictive than L_1: intuitively if it removes readers, adds owners, or both. A data item is declassified when it is relabelled to a label containing more readers for an owner o or when an owner o is removed. A declassification process is allowed only when the process acts for o. In [10, 11] the authors explicitly distinguish information from containers and thus propose to mark containers of information with two tags reflecting both the origins of the value and the security policy attached to the container. More precisely sensible data are associated to a numerical identifier and an information flow policy specifies how combinations of these identified data can flow in information containers. The model of marks presented in [11] can be either implemented at system level or at program level. In [10] information flows are tracked at run-time allowing us to check if the current execution is correct with regard to the definition of the policy. The policy is completely defined at the initialisation and can be either deduced from an interpretation of access control rights or manually defined. The policy can be updated at runtime simply by changing the tags. In [11] the authors explain how to perform a modification of the policy by changing tag value but do not define how, why or when to perform such a modification. We propose to adapt these previous models in the particular context of web services. We aim to observe information flows inside an orchestration of web services in order to ensure the user's data protection. We adopt a dynamic observation of these flows since in a context of web services we will dynamically discover the environment. As in [10, 11] we explicitly identify user's data with numerical identifier. As Myers and Liskov in [7] and [8] the security policy will specify owners of the identified information items. A user defines which services can access his information items. The description of an information flow policy could be difficult for uninformed users. To solve this problem we propose to dynamically update the control flow policy when services are discovered. Our model interacts with the users to adapt or complete the control flow policy when required.

3 System model

In this part we consider a Service Oriented Architecture (SOA) based on orchestrations. The various components of the considered architecture are shown in Figure 1.

In our architecture software resources are organized in the form of **services**. Services provide a set of functions that receive **messages** and return them after treatment. Messages consist of one or more **information data**. Services are called by **clients**. Clients are either other services, software, or human.

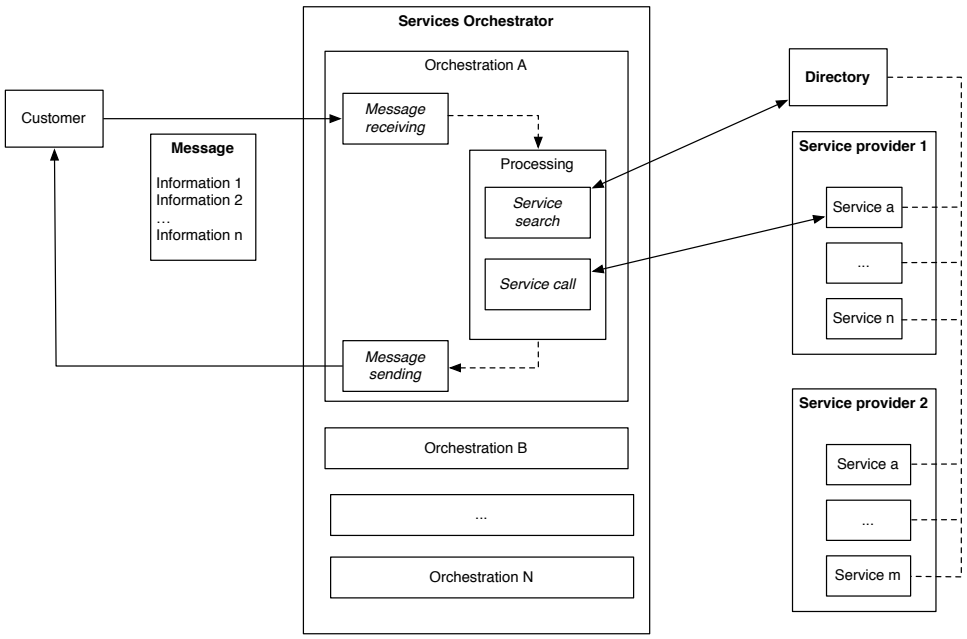


Fig. 1. Service Oriented Architecture

The different services are provided by one or more **service providers**. Each service provider offers one or more services.

Clients access services through their **interfaces**. These define the set of signatures of public operations, especially the syntax of exchanged messages. The interface is a contract between the service provider and the client. It is separated from the implementation and platform independent. This description provides a basis for the implementation of the service by the supplier and the client implementation.

The same service could be provided by several different implementations. These implementations have the same interface. They can be supplied by different suppliers.

All the services of different providers are referenced in a **directory**. A directory allows in particular to search all services providing the same functionality and having the same interface.

The central concept of our architecture is the composition of services into business processes. This business process is implemented within an **orchestration**. An orchestration is primarily a service that receives messages and returns the result of its computation. A number of operations can be performed within the orchestration. However, for each business functionality, orchestration uses an external service that can be searched beforehand in a directory. An **orchestrator of services** provides one or more orchestrations.

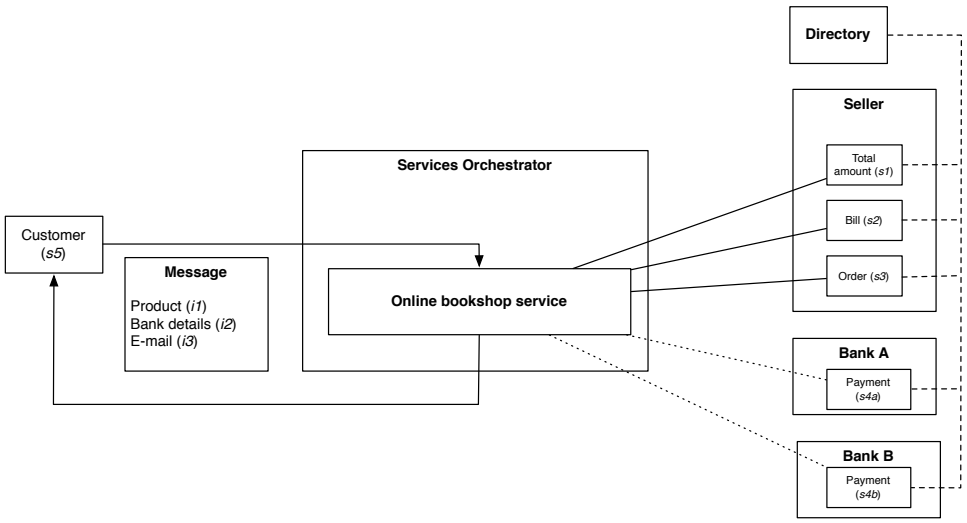


Fig. 2. Online Bookshop Service

A piece of information is a data item, a value such as a string, or an integer. A piece of information is provided to a web service orchestration through a call to this service. This piece of information is manipulated by the orchestration and the services it invoked and mixed with other pieces of information. In this work, we consider that sensitive information and in particular user private data have to be followed in order to monitor where these information data items flow. For that purpose we reuse the notion of *atomic information* first introduced in [11] to identify sensitive or private information. Any piece of information handled in the system is either atomic or obtained after treatments (like calculus) on one or more atomic information. Here any non-atomic information is the compound of one or more atomic information. For example, if x, y are atomic information $2 \times x, x + y, \ldots$ are compound information, the first non-atomic information results from the use of x, the second results from the use of x and y.

In a web service orchestration, informations are located in logical containers of information like the variables manipulated by services. Operations performed by programs or services will generate information flows between variables and consequently information will be mixed and/or will move from one variable to another. In this work we want to prevent private or sensitive information to be accessed by a non-authorized service, i.e., we want to ensure that sensible information flows only occur into variables readable by authorized services.

Before detailing our approach, let us have an example: Figure 2 details an online bookshop service which uses different services, such as a bookstore and a bank providing a payment service. The accesses to these services are provided in an orchestration of services.

The various actors in this transaction are:

– the seller who provides three services: it computes the total amount of the transaction to allow bank payment (s_1), emits the bill (s_2) and finally prepares the order that will be delivered to the client address (s_3);
– bank A and B provides a payment service (s_{4a}) and (s_{4b});
– the client who could be seen as a service (s_5).

The message sent by the customer to the online bookshop service consists of three atomic information:

– the chosen product (i_1);
– bank details (i_2);
– client e-mail address (i_3).

These atomic data are used to compute all information items handled by the complete system, such as the total amount of the transaction, the confirmation of payment, final product delivery notification,...

4 Information Flow Policy

4.1 Scope of the Information Flow Policy

The information flow policy determines for each atomic informations (and by composition for all composite informations) to which services this information may be sent. To do this, we first determine an owner for each atomic information (usually the service that injected it into the system). The owner is responsible for determining the information flow policy for all his atomic information. The following policy is determined by composition. Owners of a composed information are the union of owners of each atomic information composing this information. This information is then accessed by the services that could access all the initial atomic informations.

More formally, we use the following notations:

– **Information:** $I = \{i_1, ..., i_n\}$ is the set of atomic information of the system. Information derived from several atomic information $i_j, \ldots, i_k$ is denoted by $i_j \oplus \ldots \oplus i_k$
– **Services:** $S = \{s_1, ..s_m\}$ is the set of services of the system.
– **Owners** of information i are services that we denote $owner(i) \subseteq S$. They are defined as follows:
  - If i is an atomic information then its owner is the service that injected it into the system.
  - If i is a compound information, i.e., $i = i_j \oplus \ldots \oplus i_k$ then

$$owner(i) = owner(i_j) \cup \ldots \cup owner(i_k) \quad (1)$$

4.2 Generic Information Flow Policy

In [12] Denning formally defines an information flow policy by the following triple $(CS \rightarrow, \oplus)$, where CS is a set of security classes, $\rightarrow$ the relationship allowing the flow of information between security classes and $\oplus$ the operator to combine security classes together. The junction operator specifies the security class of any information obtained by combining informations from different classes.

In our case the information flow policy should define to what services an atomic information can be sent. We propose to define two types of security labels, a security label type for informations, another type of security label for services.

The relation $\rightarrow$ is the relationship allowing a flow from an information to a service. This relationship is used to allow the flow of an information i whose security label is L_i to a service s whose security label is L_s. We use the same notation $\rightarrow$ to express that the flow of an information i to a service s is authorized.

In an orchestration it is possible to produce information composed from several atomic information. It is therefore necessary to define the relation $\oplus$ to combine information security labels and to calculate the security label of composed information.

The security label associated with an atomic information is specified by the owner of the atomic information. Composed information security label is derived from security labels of atomic information who served to produce it.

Security labels associated with services are specified by a certification authority.

4.3 Example of Information Flow Policy

Information Flow Lattice A very simple form of information flow policy can be defined using two security classes, eg *secret* (S) and *public* (P). All flows between these classes of security are allowed except *secret* to *public*. This policy can be formulated as follows:

Information Flow Relationship $\rightarrow = \{(S,S),(P,P),(P,S)\}$, (that is to say $S \rightarrow S$, $P \rightarrow P$ and $P \rightarrow S$)
Classes Combination Operator $\oplus$ is defined as follows :
$S \oplus S = S$, $P \oplus S = S$, $S \oplus P = S$ et $P \oplus P = P$.

In this case we use the same security classes for information and services. And public information can be sent to any type of service while the secret information can be sent only to authorized secret services.

In the example shown in Figure 2, the seller wants to ensure that his bank details (i_2) is well protected. In this case the banks are the only services authorized to access secret information. In this case the security label associated with i_2 is positioned to secret by the user. i_2 can then be sent by the service orchestrator to banking services. Protecting against illegal banking service will be conducted in the same way. Only legitimate banking services would be labeled secret. And banking information would only be sent to authorized banking services.

This security policy is simple and easy to implement. However it requires a certification authority that labelized all services available. But it does not specify to what service each information can be sent. It is by example impossible to specify that only the seller knows which product the client has chosen and that the bank details are provided only to the bank. This is why we offer another example of information flow policy that is more flexible but still more difficult to implement.

Decentralized model In the model proposed by Myers and Liskov ([7]), each variable of the system is associated with a security label. This label indicates which are the owners of the information. Each owner of an information specifies the set of users authorized to access this information. An user authorized to access a variable by all owners is called a reader.

We propose to adapt this information flow policy for our model of service-oriented architecture. We propose that, for each atomic information, the owner of the information specifies the set of services to which this information may be sent.

Security labels associated with each service corresponds to a unique id that identifies this service.

Security labels associated with atomic information defines the set of readers allowed to access this information. Readers are identified by the unique id (the security label of the service). The security class associated with an information i corresponds to the definition of the function readers(i).

Readers of an information i are services defined by the owners of i which we denote $readers(i) \subseteq S$. Readers are defined as follows:

- if i is an atomic information, readers of i are the readers allowed by the service which injected it into the system;
- if $i = i_j \oplus \ldots \oplus i_k$ then

$$readers(i) = readers(i_j) \cap \ldots \cap readers(i_k) \tag{2}$$

The information flow policy defines allowed readers for each atomic information, rules of composition (2) define, by composition, readers of every compound information. The policy is defined by the owners of information, since an owner determines the readers that are allowed to read its atomic information items. A call to a service that brings an information is legal only if the service called is a *reader* for this information. In the same way, a response from a service is only authorized if the caller is a legal reader for the information received.

We can then deduce the definition of the function $\rightarrow$. $i \rightarrow s$ if and only if $s \in readers(i)$.

Let us consider again the example detailed in Section 3 in Figure 2. In this example six services are present:

- s_1, s_2 and s_3 are three services provided by the seller;
- s_{4a} and s_{4b} are payment services provided by the banks;
- s_5 is the user service that calls the orchestration to place an order.

Atomic information items in the system are provided by the user service, i.e., service s_5. i_1 corresponds to the chosen product. The client imposes that i_1 is accessible only to the seller and thus $readers(i_1) = \{s_1, s_2, s_3\}$, the services provided by the seller. i_2 corresponds to the bank details. The client imposes that they are accessible only to the bank A, we thus have $readers(i_2) = \{s_{4a}\}$, the payment service provided by the bank A. Similarly the client wants i_3 (the client email address) to be accessible to the seller only. We have $readers(i_3) = \{s_1, s_2, s_3\}$. In all cases the owner of such atomic information is the service calling the command (called directly by the client), i.e., s_5.

More formally in this example **Readers** of I are defined as follows,

- $readers(i_1) = \{s_1, s_2, s_3\}$,
- $readers(i_2) = \{s_{4a}\}$,
- $readers(i_3) = \{s_1, s_2, s_3\}$.

In other words, the policy is entirely defined Figure 3.

Atomic information name	**Owners**	**Readers**
i_1	s_5	$\{s_1, s_2, s_3\}$
i_2	s_5	$\{s_4\}$
i_3	s_5	$\{s_1, s_2, s_3\}$

Fig. 3. An information flow policy for the example detailed in section 3

5 Dynamic Information Flow Checking

In order to follow the origin of information flow, we add to each variable the list of initial information used to produce the content of this variable that we call *history*. The value of an *history* is initialized as empty and is firstly modified when a new information item is injected in the web service through a call to this service. At this moment, the injected information item is considered as atomic, its owner is the caller. The caller also defines the security policy associated with the initial atomic information. The *history* is further modified at each operation on the variable that modify the content of the variable. *Histories* are modified to reflect atomic informations used to produce informations contained in the variable. When a service calls another service or makes a response to another service, a verifier checks if the flow engendered is legal with respect to the current security policy. More precisely the verifier checks if the resulting security policy, calculated from the security policy associated with the initial informations, allow the flow to the service. In the following, we formally define how *histories* are defined and modified.

5.1 Definition of Information Flow *Histories*

As stated before, a *history* is a meta-data attached to each container and describes atomic informations used to produce the information currently located in the container. If c is a container its information flow *history* is of the form

$$H_c = \{\mathbf{i_1}; ...; \mathbf{i_j}\}$$

Such a *history* means that information i contained in c is based on information $\mathbf{i_1}, ..., \mathbf{i_j}$.

5.2 Initialization and Modification of Information Flow *Histories*

Let us consider a service s_1 injecting an item of information i in another service s_2 by calling s_2 using a variable v. The service s_1 is considered to be the owner of the atomic information i now located in the variable v of s_2. The variable v is the container of i and its information flow *history* is on the form $\{i\}$ where i identified the atomic information it is depending on. Informations about owners and security policy are localized in a table. In practical terms if the service s_1 is executed by a user, this user will be asked to define the security policy associated with its own information.

When a service is called, it makes some internal computation before sending a response. These internal computations induce some information flows and modify the content of containers of information. Since a *history* attached to a container describes the informations used to produce its current content, it has to be updated at each observation of an information flow towards the container.

From a general point of view, we consider a set of containers $c_j, ..., c_k$ labeled by $L_j, ..., L_k$. If we observe an information flow from the containers $c_j, ..., c_k$ to another container c, then we update the *history* of c which is now the union of labels attached to $c_j, ..., c_k$.

Let us have an example with three containers c_1, c_2 and c_3 associated with the following *histories* :

- $H_{c_1} : \{\mathbf{i_1}; \mathbf{i_2}\}$
- $H_{c_2} : \{\mathbf{i_1}; \mathbf{i_3}\}$
- $H_{c_3} : \{\mathbf{i_4}\}$

and the information flow policy associated with this atomic data items :

- $\mathbf{i_1} : \mathbf{s_1} \triangleright L_1$
- $\mathbf{i_2} : \mathbf{s_1} \triangleright L_2$
- $\mathbf{i_3} : \mathbf{s_2} \triangleright L_3$
- $\mathbf{i_4} : \mathbf{s_3} \triangleright L_4$

We denote L_1 the information flow policy associated by $\mathbf{s_1}$ to the atomic information i_1.

We consider an information flow from c_1 and c_2 to c_3. This flow modifies the content of c_3 which is now a value derived from those located in c_1 and c_2. The *history* H_{c_3} is updated to $H_{c_1} \sqcup H_{c_2}$, i.e.

$$\{\mathbf{i_1}; \mathbf{i_2}; \mathbf{i_3}\}$$

Owners of c_3 are now :

$$\begin{aligned} owners(c_3) &= owners(i_1) \cup owners(i_2) \cup owners(i_3) \\ &= \{s_1\} \cup \{s_1\} \cup \{s_2\} = \{s_1, s_2\} \end{aligned}$$

and the security policy associated with c_3 is now :

$$information\_flow\_policy(c_3) = L_1 \oplus L_2 \oplus L_3$$

The history of information flows is carried out via the propagation of the *histories* attached to the containers of information. When a service performs a response using a variable c this response will be authorized according to the security policy.

From a practical point of view, in our work the history of information flows is propagated through the *histories* at runtime in a modified orchestrator[5]. The legality of a call to a service or a response from a service is checked just before the call / response.

6 Dynamic Update of the Information Flow Policy

Let us consider an orchestration performing a call of a service s (or similarly a response to a service s) using data d having a *history* on the form

$$H_d = \{\mathbf{i_1}; ...; \mathbf{i_j}\}$$

and the information flow policy associated with this information :

- $\mathbf{i_1} : \mathbf{s}_\alpha \triangleright L_1$
- ...
- $\mathbf{i_j} : \mathbf{s}_\beta \triangleright L_j$

We have to verify if this call is legal with regard to the information flow policy before performing the call. By definition of the security policy this call is legal if and only if all information items contained in d could be sent to the service s associated with the security label L_s, i.e. $L_1 \oplus ... \oplus L_j \rightarrow L_s$. If the flow is authorized, then the orchestrator performs the call. Otherwise we ask owners of d to confirm if that the call must although be authorized. Indeed, since services can be dynamically discovered we cannot decide if the call is really forbidden or if the owners have not completely defined the security policy.

[5] by example a BPEL interpretor

We use a dedicated service to ask all owners $(\mathbf{s}_\alpha, \ldots, \mathbf{s}_\beta)$ if they authorize or not to send a compound information d computed using their atomic information resp. $\mathbf{i_1}, \ldots, \mathbf{i_j}$.

More precisely the orchestrator calls a dedicated service to contact information owners. This service is an exception to the security policy, we consider that this particular service is a reader for any atomic information. In future work we plan to protect this dedicated service: for instance we plan to encrypt the data send to/by this service. This service is used to ask every owner $\mathbf{s_k}$ of atomic information $\mathbf{i_k}$ if they accept to modify the policy of $\mathbf{i_k}$. The service thus uses a request composed of four parts:

- the initial information item i_k that was used to compute the value d ;
- the value d if the owner is authorized to access the value of d, this part is empty otherwise;
- the service s ;
- if the information actually sent to the service depends explicitly or implicitly on the initial information.

For each owner, this call may have one of these three possible responses:

- (**refusal**) the owner refuses to modify the security policy;
- (**temporary exception**) the owner accepts the update of the security policy only for this call/response of service;
- (**agreement**) the owner accepts the update of the security policy until the end of the execution of the orchestration. In this case the security policy of the atomic information is modified.

If at least one owner refuses the modification, the service call (or the response) is not performed. If all the owners accept the modification but at least one of them authorises only a temporary exception then the call (or the response) is performed and the security policy attached to i_k remains the same. Finally when the owner accepts the modification, the security policy of i_k is modified.

7 Conclusion

The goal of our work is twofold : first to give the service orchestrator a way to define a model of information flow control policy, second to give the user of a web service the ability to restrain the use of his data by services he never heard of. At the time of a service call, he is able to define which user data can be accessed by which web services. This property is guaranteed by a distributed security policy that defines which data can be accessed by which service. Using the security model defined by Myers et al. as a basis, our contribution consists in applying this type of security policy to Web Services and to dynamically define what are the variables in an orchestration of Web Services that are influenced by the user inputs. For this purpose, we follow the information flows that are produced by the various operations available in the orchestrator. When flows are

produced between variables, we update the labels attached to these variables to reflect the information used to produce this data.

However such an approach is difficult to configure for a user of Web-Services. That is why we proposed a mechanism to dynamically update or build the security policy and principles for integrating this mechanism in an orchestrator. In particular, we defined a communication protocol between the orchestration of services and the owner of the information in order to perform declassification.

Future work will focus on detecting implicit or explicit data leakage and ensure the privacy of the user data. We are currently working on the implementation of the mechanisms inside a modified BPEL interpreter.

References

1. Cerami, E.: Web Services Essentials. O'Reilly and Associates, Inc., Sebastopol, CA, USA (2002)
2. OASIS: Web services business process execution language version 2.0. OASIS Standard (2007)
3. Nakajima, S.: Model-checking of safety and security aspects in web service flows. In Heidelberg, S.B.., ed.: Web Engineering. Volume 3140/2004 of Lecture Notes in Computer Science. (2004) 767
4. Hutter, D., Volkamer, M.: Information flow control to secure dynamic web service composition. In: Security in Pervasive Computing. Volume 3934 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg (2006)
5. Rossi, S., Macedonio, D.: Information flow security for service compositions. In: ICUMT, IEEE (2009) 1–8
6. Fischer, K.P., Bleimann, U., Fuhrmann, W., Furnell, S.M.: Security policy enforcement in bpel-defined collaborative business processes. In: ICDEW '07: Proceedings of the 2007 IEEE 23rd International Conference on Data Engineering Workshop, Washington, DC, USA, IEEE Computer Society (2007) 685–694
7. Myers, A.C., Liskov, B.: A decentralized model for information flow control. Proc. ACM Symp. on Operating System Principles (1997) 129 – 142
8. Myers, A., Liskov, B.: Complete, safe information flow with decentralized labels. In: IEEE Symposium on Security and Privacy. (1998)
9. Myers, A.C.: Jflow: Pratical mostly-static information flow control. Proceedings of the 26th ACM Symposium on Principles of Programming Langages (1999) 228 – 241
10. Hiet, G., Viet Triem Tong, V., Me, L., Morin, B.: Policy-based intrusion detection in web applications by monitoring java information flows. Int. J. Inf. Comput. Secur. **3** (2009) 265–279
11. Viet Triem Tong, V., Clark, A., Mé, L.: Specifying and enforcing a fine-grained information flow policy: Model and experiments. In: Journal of Wireless Mobile Networks, Ubiquitous Computing and Dependable Applications. (2010)
12. Denning, D.E.: A lattice model of secure information flow. Commun. ACM **19** (1976) 236–243

22 novembre 2012

"Integrated Governance in the Cloud"
- Need for a New Paradigm -

C&ESAR Conference – November 20th- 22nd, 2012

Daniel Pradelles
EMEA Privacy Office – Hewlett-Packard CCF
Les Ulis, France

daniel.pradelles@hp.com

Abstract. Global data processing, and cloud computing in particular, present governance issues which are proving difficult to address with current tools and approaches. This paper describes the current global challenges and proposes an analysis of the need for change from key stakeholders along three main axes: Organisations that use and provide cloud computing solutions need to develop integrated governance across business processes and technological developments; International regulatory frameworks must become more flexible and be designed to support global interoperability; Finally, technology innovation is critically needed to support these transformational changes, in order to allow for safe and compliant implementation of global cloud solutions.

1 Problem statement

As recent surveys, front page scandals, citizen's outcries and regulators statements have shown, lack of trust is one of the key inhibitors to the adoption of current and future Information society products, services and technologies. People are more and more concerned about where and how their data is collected and what may happen to it once it is collected by Government or Companies. This concern is even worse when it goes into the cloud due to its intangible and ethereal nature. They are worried about where their data will be transferred to, who can access it, and how it will be copied, shared and used.

In summary there is a general sense of losing control and a concern that data integrity and confidentiality may be jeopardized intentionally or accidentally. It has to be noticed upfront that these concerns which originated from Personal data processing and specific Privacy regulations is now also expanding to corporation and government bodies data at large. Then organizations that change from carrying out their computing in-house to using the public cloud are not so much concerned any more about the physical state of servers, but instead the confidentiality and security of their data.

In the current trend to adopt Cloud computing and take advantage of its full benefits in terms of cost effectiveness, flexibility and scalability, there is quite often an underestimation of its potential drawbacks in terms of complexity.
Already today in a large corporation, and even more when dealing with multiple actors, the processing flows are dynamic (change depending on purpose, time of the day), global (not anymore limited to a predefined geographic zone) and fragmented (chunks of data can be processed in different places). This will become truer in a distributed cloud computing context therefore the complexity impacts may be seen along three main axes:

- ***Technical****:* Increased difficulty to monitor where the data goes back and forth, maintain the virtual flows infrastructure and fix issues in an efficient and timely manner;
- ***Regulatory****:* It becomes more and more awkward to define which law is applicable, what jurisdiction is the valid one, what is the relevant regulatory authority and who is the entity ultimately liable for breaches and incidents;
- ***Organizational****:* How to define the individual in charge, the organization managing the decision process and the access governance to define who has rights to access and use the data.

The approach we suggest to address these difficulties focuses on increasing trust for consumers, clients and regulators; such trust will support appropriate and fruitful collaboration between involved parties, ensure adequate information exchange & processing while preserving the capacity to growth and innovate within a constantly evolving environment.
Trust cannot be decided upfront and unilaterally, it comes from sound information stewardship by service providers for which they need to be held accountable. The approach should also decrease regulatory complexity and increase certainty in global business environments, which are particularly critical in a cloud context where services can be aggregated from multiple, sometimes international providers. We need to provide a clear and consistent framework of data protection rules; and avoid a complex matrix of national laws and reduce unnecessary layers of complexity for cloud providers. The business context resulting from this should support innovation in technologies and business practices.

Finally there should be appropriate technology tools supporting this accountable comprehensive & global dynamic.

2 Key points to consider

So, in order to really take advantage of the cloud, we need to find ways to address these issues in a holistic and multidisciplinary manner that takes into account the complexities of the full ecosystem; we attempt to summarize key elements of how this could be achieved:

2.1 No prescriptive solution with complex systems

First we need to acknowledge that we are in a domain which is an example of what is called in engineering a "complex system", with a multitude of variables, actors and constraints. In such a framework we cannot predict results using pre-defined rules based on algorithms or mathematical models. Only experience, observation and simulation may help to gain some understanding of the system behaviour. The environment in which we operate is marked by influences coming from social, cultural, legal, traditional dimensions but also technology, business models, economy… all within an increasingly fast & dynamic environment.

This is why we believe that issues cannot be addressed with predefined, highly prescriptive solutions often based on the understanding we have today. We need fluid, flexible tools; we may say "organic" ones, based on commitment to overarching concepts with a comprehensive approach. These tools should focus on results and deliverables to those commitments, rather than on how they are achieved. By definition in a complex environment, the tools and their implementation have to be defined and their efficiency monitored with a multidisciplinary approach including all parties involved along their lifecycle. We are very pleased to see that this concept of "multi stakeholder's involvement" and "close and transparent dialog" is already mentioned in the regulatory work done on both sides of the Atlantic.

2.2 Keep it Global

The second key element is the increasing need to act locally or regionally while thinking globally. While we fully understand the willingness to listen and protect own citizens, to strengthen and support own economy in own market; the information society services and relevant regulations can no longer be shaped locally in abstraction of the global dimension, which is the scale and reality of today's data flows.
As a result, we strongly support the efforts to create a harmonized and global privacy framework which will ensure the "Global interoperability" required to establish a comprehensive, consistent and flexible workable regulatory framework. We also support the willingness to ensure better and effective enforcement indispensable to provide legal predictability to companies, to incentivize compliance and provide encouragement to those organizations investing in robust compliance procedures. But these rules should be pragmatic, adequately sized, clear and consistent within each "regulatory" region with comparable strength across the Globe.

2.3 Accountability as a "Third way"

The next element lies in the discussions around self-regulatory and formal regulatory approaches which have been going on for ever… each approach has its own benefits and drawbacks. At HP we believe that there is a "third way" which will provide the benefits of both, avoid most of the drawbacks, while providing more effective compliance and Data protection.
Accountability goes beyond simple compliance or mere industry self-regulations. It requires companies to comply with regulations, to make responsible, ethical and disciplined decisions, and additionally demonstrate willingness and commitment to ensure this high level of personal data protection. HP was very pleased to see some elements of this concept being introduced in the recent EU regulation proposal for data protection issued last January 2012.
A component of this concept, which is also introduced in this proposal, is the so-called "Privacy by design". Each organization, independently of its size, business sector, the processing complexity and data confidentiality, should be expected to commit to considering upfront the impact and risks they may create with new technologies, product and business process. It should be pointed out at this point that "Privacy by

design" is a process and a development methodology which has to be deeply embedded in the genes of a responsible and accountable company. This is fundamentally different from "Privacy by default" which is a product or service "initial state" within a well-defined context and for a specific user.
Even further along this line, companies should implement into their organization a comprehensive and well defined Privacy accountability framework. The backbone of it should be a strong commitment to a set of policies, and to full transparency, complemented by robust implementation mechanisms and appropriate validation and audit process.

The other component of this concept is integrated governance with a feedback mechanism to help identify new risks & opportunities. The last but not the least part of this framework is a companies'/ organizations' readiness to demonstrate their capacity to fulfil their engagement to major stakeholders such as Regulators, Data subjects and internal stakeholders.

In a "true cloud" context with a large number of Processors providing elements of the service fulfilled by a Controller for the benefit of a Data subject or a company this Accountability approach may realize its full potential. The vision is to build a flexible and "across the cloud" Regulatory and Company Governance approach that provides a "Chain of accountability" right along the service provision chain, and to underpin this with technological mechanisms.

3 The new paradigm and the three development dimensions

As new innovative business models become increasingly complex, stakeholders understanding of impacts and potential harm become less and less easy to evaluate. The complexity and the intrinsic difference in pace between technology innovation and the regulatory framework also add to this difficulty. To address this issue we would recommend an active "on-going" collaboration between major stakeholders like regulators, industry associations, companies and consumers advocates. We may say that we all have the same customers: "the data subjects" and that such collaboration is the most effective way to serve them in a timely and effective manner.

In order to address the current and future challenges we are facing, or will face, in an efficient way, we believe that active and coordinated work should be done along three main axes:

- ***Innovative regulatory frameworks:*** The current legal landscape across the globe is a patchwork of different laws, approaches and implementations which, even based on pretty similar concepts, has been shaped by a long history of legal, political, cultural and business decisions that are often taken in pretty isolated " self-consistent" regional blocks. In our globalized, borderless context it makes compliance and business extremely complex and cumbersome.

  It is therefore critical that new models such as regulatory standards and efforts to ensure global interoperability are enacted in order to facilitate both the operation of global business and provision of redress within cloud environments. On the practical side we may foresee already some International Standards and Tools (like BCR and CBPR) which would help bridging regulatory gaps and culture and then provide "Interoperability " between regional regulatory blocks;

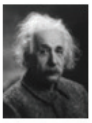
- ***Responsible company governance:*** In current complex environments it becomes obvious that predefined, prescriptive and static solutions and requirements quickly become outdated, inefficient and often counter effective. We then need comprehensive company governance models whereby organizations act as responsible stewards of the data which is entrusted to them within the cloud.

  Such an approach, ensuring responsible behaviour via accountability mechanisms and balancing innovation with individuals' expectations, includes innovative practices such as Privacy by Design, on-going privacy impact assessments and other tools as a way of achieving timely and proactive consideration of the risks and harm which may be generated within business operations. Accountability and related practices allow companies to act as responsible steward of data and ensure "Organization adequacy";

- ***Supporting technologies:*** As stated before in this article, the cloud exacerbates current issues and creates new unanticipated ones, especially due to the pervasive, ubiquitous, global and dynamic nature of how and where processing takes place. The emerging concepts of "Big Data" and the "Internet of things" will make it even more complex, fuzzy and multitenant.

  For addressing this and maintaining a chain of accountability across the cloud there is a need to improve existing technologies and standards and develop new ones aimed at supporting and monitoring data use and obligations linked to data stewardship along its lifecycle through the cloud. These include privacy enhancing technologies, security mechanisms, encryption, anonymisation, etc… allowing Access governance, Rights and obligations management and Data minimization across the cloud and data lifecycle.

By using a combination of these means, users, citizens and companies can be provided with reassurance that their data, personal or not, will be protected, and cloud deployments can be made in compliance with regulations, even within countries where such regulation is relatively strict.



"We can't solve problems by using the same kind of thinking we used when we created them."

Albert Einstein

Reference URL's and related papers:

1. **EU Data Protection framework proposal:** *http://ec.europa.eu/justice/newsroom/data-protection/news/120125_en.htm*
2. **Overview of Binding Corporate rules:** *http://ec.europa.eu/justice/data-protection/document/international-transfers/binding-corporate-rules/index_en.htm*
3. **HP & Accountability:** *http://www.hp.com/hpinfo/globalcitizenship/society/privacy.html*
4. **"Accountability-based Privacy Governance" – CIPL Project:** *http://www.informationpolicycentre.com/accountability-based_privacy_governance/*
5. **"An Interdisciplinary Approach to Accountability for Future Internet Service Provision";** Siani Pearson and Nick Wainwright,. *International Journal of Trust Management in Computing and Communications (IJTMCC)*, volume 1, issue 1, 2012.

6. **"Developing Accountability-based Solutions for Data Privacy in the Cloud";** Andrew Charlesworth and Siani Pearson,. To appear in: Volume 26, Issue 1, Innovation, Special Issue: Privacy and Technology, *European Journal for Social Science Research*, Taylor & Francis, UK, 2012.
7. **"Toward Accountability in the Cloud"**; Siani Pearson, View from the Cloud, *IEEE Internet Computing,* IEEE Computer Society, July/August issue, vol. 15, no. 4, pp. 64-69, 2011.
8. **"Accountability as a Way Forward for Privacy Protection in the Cloud";** Siani Pearson and Andrew Charlesworth, *Proc. 1st CloudCom 2009*, ed. M.G. Jaatun, G. Zhao, C. Rong, Beijing, Springer LNCS 5931, pp. 131-144, December 2009.
9. **"Accountability for Cloud and Other Future Internet Services"**; Siani Pearson et al, to appear in Proc. CloudCom 2012, IEEE, December 2012