A Methodology for the Development and Verification of Access Control Systems in Cloud Computing

Antonios Gouglidis, Ioannis Mavridis

A Methodology for the Development and Verification of Access Control Systems in Cloud Computing

Antonios Gouglidis and Ioannis Mavridis

Department of Applied Informatics, University of Macedonia
156 Egnatia Str., 54006, Thessaloniki, Greece
{agougl, mavridis}@uom.gr

Abstract. Cloud computing is an emergent technology that has generated significant interest in the marketplace and is forecasted for high growth. Moreover, Cloud computing has a great impact on different type of users from individual consumers and businesses to small and medium size (SMBs) and enterprise businesses. Although there are many benefits to adopting Cloud computing, there are significant barriers to adoption, viz. security and privacy. In this paper, we focus on carefully planning security aspects regarding access control of Cloud computing solutions before implementing them and, furthermore, on ensuring they satisfy particular organizational security requirements. Specifically, we propose a methodology for the development of access control systems. The methodology is capable of utilizing existing security requirements engineering approaches for the definition and evaluation of access control models, and verification of access control systems against organizational security requirements using techniques that are based on formal methods. A proof of concept example is provided that demonstrates the application of the proposed methodology on Cloud computing systems.

Keywords. Security, Inter-organizational systems, Cloud business, Verification

1 Introduction

Cloud computing is an emergent technology that has generated significant interest in the marketplace and is forecasted for high growth. Specifically, Cloud computing is expected to be a significant growth driver in global IT spending. In fact, by 2013, about 82% of all net-new software firms coming to market will be operationalized on service based software versus a packaged product. By 2016, approximately 25% of all new business software purchases will be of service-enabled software. Software-as-a-Service (SaaS) delivery models will constitute about 14.9% of worldwide software spending across all primary markets and 18% of applications spending, according to IDC [1]. Furthermore, according to the latest research reports, the global computing market is expected to grow at a 30% CAGR reaching $270 billion in 2020 [2]. Therefore, Cloud system should be carefully designed to be secure since otherwise, the existence of security flaws may lead to a decrement in income of an organization.

In this paper, we are concerned with the development of access control (AC) systems for distributed and collaborative systems, as Cloud systems. Since AC systems are highly complex and of great significance, their right definition, design and development are mandatory for the production of an AC system that corresponds to the initial requirements. To manage the increased complexity of AC systems, system engineering processes are applied. Nevertheless, generic approaches can be enhanced to facilitate the definition of access control requirements in Cloud environments as presented in [3]. Additionally, a common stage in the development of a system is its verification, where a system can be verified for its correctness. Several papers have examined the automated verification of AC systems, and a number of techniques have been proposed to verify them as presented in [4], [5], [6], and [7].

The structure of the remainder of this paper is: Section 2 provides preliminary information regarding AC in the Cloud and systems engineering stages. Section 3 describes our proposed methodology. A use case is given in Section 4. Finally, we conclude this paper in Section 5.

2 Preliminaries

In this section we provide basic preliminary information regarding AC and its role in Cloud systems. Moreover, we briefly refer to two important stages met in any development approach, viz. requirements engineering and verification in the context of AC.

2.1 Access Control

AC in modern distributed systems has become even more challenging since they are complicated and require the collaboration among domains. A domain can be defined as a protected computing environment, consisted of users and resources under a same AC policy. AC is an essential process in all systems. The role of an AC system is to control and limit the actions or operations in a system that are performed by a user on a set of resources. Nevertheless, an AC system is considered of three abstractions of control, namely AC policies, AC models, and AC mechanisms. A policy can be defined as a high-level requirement that specifies how a user may access a specific resource and when. AC policies can be enforced in a system through an AC mechanism that is responsible for permitting or denying a user access upon a resource. An AC model can be defined as an abstract container of a collection of AC mechanism implementations, which are capable of preserving support for the reasoning of the system policies through a conceptual framework. Consequently, the AC model is capable of bridging the existing abstraction gap between the mechanism and the policy in a system [8], [9].

The Cloud is a fairly new and emergent technology and its definition is a topic for discussion in several research papers [10]. Nevertheless, Cloud computing is defined in [11] using five attributes viz. multitenancy, massive scalability, elasticity, pay as you go and self-provisioning of resources. These attributes successfully imprint the distinctive characteristics of the Cloud and differentiate it from similar technologies,

as the Grid computing paradigm. Multitenancy refers to the business model implemented by the Cloud, where a single shared resource can be used from multiple users. Massive scalability refers to the potential of the system to scale (i.e. increase or decrease) in resources. The on-demand and rapid increment or decrement of computing resources is translated as elasticity of the Cloud. Thus, more storage space or bandwidth can be allocated when required, and vice versa. Pay as you go is the process of paying for the resources that are used. Lastly, the users are provided with the ability to self-provision resources, namely storage space, processing power, network resources and so on. An additional characteristic defined in [12] by the National Institute of Standards and Technology (NIST) is Broad Network Access, which states that available capabilities can be accessed using standard mechanisms over the network, and promote their use by heterogeneous clients.

The service model of Cloud computing is based on the SPI framework [11], [12]. SPI stands for Software-as-a-service (SaaS), Platform-as-a-service (PaaS) and Infrastructure-as-a-service (IaaS). Specifically, the SaaS provides software that is used under a business model, namely the usage-based pricing. The PaaS offers the platform for the development of the applications, and lastly, the IaaS handles the provision of the required hardware, software and equipment, in order to deliver a resource usage-based pricing model.

Moreover, the aforementioned service models are provided under three deployment models viz. public, private and hybrid Cloud [11]. The public Cloud provision resources over the Internet and are accessible via a web application. A third-party operates as the host and performs all the required operations (e.g. management, security). The private Cloud provides the same functionality as the public deployment model within internal and private networks. This model requires the acquisition of the appropriate hardware and software. The hybrid model refers to the combination of the public and private deployment models. Usually, the latter model is used to keep sensitive data in the private network and deploy non-core applications to the public. An additional service model proposed by NIST is the community Cloud [12], which refers to infrastructure exclusive used by a specific community of consumers from organizations that have shared concerns.

In Cloud systems, the main objective of AC is to grant authorized users the right to use a service, and at the same time to prevent access to non-authorized users. Similarly to the Grid paradigm, a Cloud AC policy can be defined as a Cloud security requirement that specifies how a user may access a specific resource and when. Such a policy can be enforced in a Cloud system through an AC mechanism, which is enforced by a Cloud Service Provider (CSP). The latter is responsible for granting or denying a user access upon a service. Therefore, AC in Cloud systems is similar to the Grid. The main difference is mostly subject of the service delivery model that is applied on the Cloud system (SaaS, PaaS, IaaS) [11]. Hence, in the SaaS delivery model, the CSP is responsible for managing all aspects of the network, server and application infrastructure. In the PaaS delivery model, the customer is responsible for AC to the applications deployed in the PaaS platform. Lastly, in the IaaS delivery model, the customers are entirely responsible for managing all aspects of AC. In gen-

eral, AC in the Cloud is not standardized across CSPs, and user AC to Cloud resources is generally weak because of coarse user access management [11].

2.2 Requirements Engineering

This section is intended as an overview of requirements engineering, which is being increasingly recognized as the most critical phase of the systems development process. If the requirements for a system are not right, there will inevitable be problems after the system is delivered. There are immense variations in what is generally understood as a requirement. The term requirement might be used to refer to statements that are clearly at totally different levels of detail. These variations in the statement of requirements arise because of the different ways requirements are used in different organizations. Therefore, a statement of a requirement can range from a high-level abstract statement of a service to be provided or of a system constraint to a detailed mathematical functional specification of a system component [13].

Requirements can be identified into four types: Functional requirements, where define some functionality to be provided by the system (i.e. what the system will do). Non-functional requirements, where define some operational constraints on the behaviour of the system (i.e. how the system should behave). Design requirements, where define constraints on the system design or implementation. Process requirements, where define constraints on the system development process.

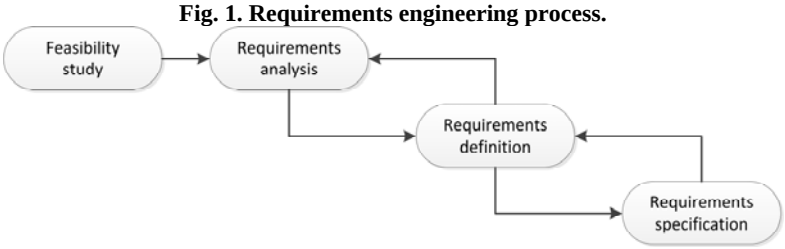
Although it is convenient to classify requirements into different classes, there is not really a strict distinction between these. Specifically, non-functional requirements at one level of abstraction are translated into more detailed functional requirements.

Regarding requirements there is a need for specifications at different levels of detail. The principal reason for this is that these different specifications are designed for different purposes and readers. Requirements definition or specifications can be classified into three types viz., user requirements definition, system requirements specification, and software or hardware specifications.

User requirements definition has to be understandable by potential end-users of the system and their managers as well as the developers of the system. The system requirements specification is a more detailed document that is usually part of the contract for the system. Lastly, software or hardware specifications include a detailed description that can serve as a basis for a design or implementation.

Figure 1 illustrates the activities of the requirements engineering process. Despite the fact that there is a clear division between process activities, in practice, the process is iterative with the activities interleaved and with very blurred boundaries between them. Specifically, analysis definitions and specification may be seen as a single activity. In more detail, the feasibility study activity is clearly related to the feasibility study activity that takes place during the conceptual design. Requirement analysis is the process of finding out the requirements and requirement's definitions are intended to be a high-level description of the system requirements. Lastly, system requirement's specifications define the requirements in detail as a basis for the contract for the system procurement and for the system developers.

Fig. 1. Requirements engineering process.



2.3 System Verification

Verification is a critical process well separated from the previous stages of requirements engineering and system's design, and concerned with building the system right. Verification is used in the comparison of the initial conceptual system based on defined requirements to the computer representation that implements that conception. Specifically, it must ensure that the system does what it should, does it only the way it should and does not do what is should not do [14].

The principal methods for the verification of complex systems are four [15]:

- Testing: Testing is performed on the system itself. However, testing of distributed systems is not always a cost effective process since it can be performed when an implementation of the system is available. Furthermore, it can only prove the existence of bugs, but not their absence.
- Simulation: Simulation-based approaches ensure that a finite number of user-defined system trajectories meet the desired specification. Even though computationally inexpensive simulation is ubiquitous in system design, it suffers from completeness as it is impossible or impractical to test all system trajectories. Furthermore, simulation-based testing is semi-automatic since the user must provide a large number of test cases [16].
- Deductive verification: Deductive verification is based on manual mathematical proof of correctness of a model of a system. It is a very highly cost process and, furthermore, requires highly skilled personnel.
- Model checking: Model checking performs exhaustive testing of all behaviours of a model of the system. It is not vulnerable to the like-hood that an error is exposed; this contrasts with testing and simulation that are aimed at tracing the most probable defects. Additionally, it provides diagnostic information in case a property is invalidated, which is very useful for debugging purposes. In principle, model checking is an automated process and its use requires neither a high degree of user interaction nor complex test data [14].

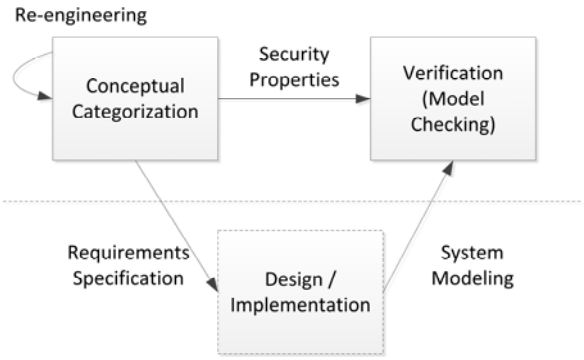
3 Proposed Methodology

In this section, we provide information regarding our proposed methodology for the development of AC systems. The methodology is independent of the applied development model of a system since the stages of requirements engineering and verifica-

tion exist in most of them (e.g. sequential, spiral). Figure 2 illustrates the proposed methodology in a system development process that consists of the stages of requirements engineering, systems' design and verification. We propose for the stage of requirements engineering to apply the Conceptual Categorization (CC) [3] and during verification to apply model checking techniques. CC is capable of tailoring the requirements engineering through a re-engineering process. Furthermore, CC operates as an input for the verification stage by defining security requirements. The security requirements are transformed into security properties in temporal logic (e.g. LTL, CTL). The set of defined security properties can be verified on the systems' transition system (TS) using model checking techniques.

In our proposed methodology, we are mostly concerned in performing security requirements engineering and verification of the system to be developed. Therefore, we are not concerned with other stages of the development process, as the design and implementation of the system. However, we depict in Figure 2 their interaction with our defined stages. Thus, the proposed stages can be used transparently in any development model without breaking it.

Fig. 2. System development process.

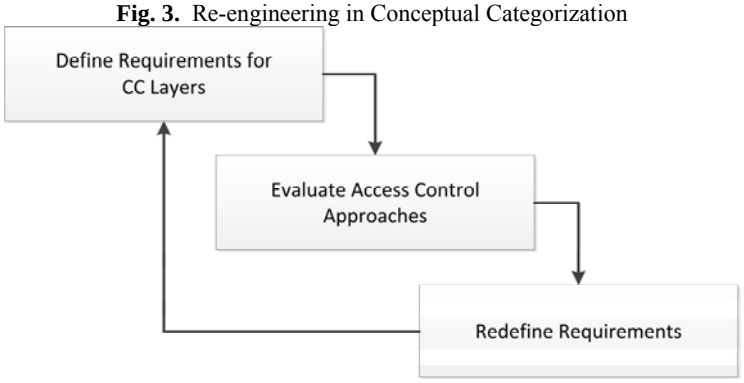


3.1 Access Control Requirements

Concerning the requirements engineering stage, we choose to apply the CC that is proposed in [3], which is a layered approach for the definition and evaluation of security requirements for modern collaborative systems, as the Cloud. In brief, CC is based on four abstraction layers. These are the entropy, assets, management and logic layers. The differentiation from generic security engineering approaches is that, in CC, factors that affect the security of the systems are mainly considered in their categorization. CC is able to identify and group security requirements into discrete layers of different abstraction level. The abstraction level refers to the ability of a layer to identify requirements in different breadth and depth. The entropy layer identifies requirements from the dispersion of the objects in a system and the assets layer from the type of shared objects within the boundaries of the entropy layer. The next layer defines requirements from policy management and the logic layer incorporates requirements that are not handled by the former layers.

In addition to the simple, sequential development process, the CC approach can enhance and be used for tailoring the process of requirements engineering. This is required since in practice new requirements always emerge that inevitably leads to implementation changes. CC can be used for re-engineering existing systems. Therefore, it can be applied on old systems, which usually have a considerable amount of intelligence and experience encapsulated within them. Figure 3 depicts the basic steps that help to re-engineer existing systems using an evolutionary life cycle. The process can be seen as a spiral where cycling through the different processes leads to the desired outcome. Specifically, the evolutionary life cycle includes the definition of requirements in CC layers. In turn, an evaluation of the examined system is performed. This second process, checks the compliance of the already defined requirements. If expectations from the examined systems are low, then new requirements are identified and defined, which when applied will potentially lead to a new system. New requirements are redefined in CC layers and the process is repeated. This evolutionary life cycle helps for faults to be found more quickly and provides the opportunity to include updated technology, and at the same time it facilitates the whole process of delivering a fully functional system [17].

Fig. 3. Re-engineering in Conceptual Categorization



3.2 System Verification

As presented in section 2.3, verification is a process of significant importance. Nevertheless, there are several approaches to perform it, each having its pros and cons. However, in our methodology we choose to apply a model checking technique to verify AC systems for reasons of automation, diagnostic information, integration in existing development cycles and its sound and mathematical underpinning. Model checking requires the definition of a system's model. These models are characterized as transition systems (TS) and are defined to describe their behaviour.

Formally, a TS is a tuple $(S, Act, \rightarrow, I, AP, L)$ where [14]:

- S is a set of states,
- Act is a set of actions,
- $\rightarrow \subseteq S \times Act \times S$ is a transition relation,

- $I \subseteq S$ is a set of initial states,
- AP is a set of atomic propositions, and
- $L : S \to 2^{AP}$ is a labelling function.

We consider the TS to be a finite system. TS is called finite if *S*, *Act* and *AP* are finite. Additionally, given that Φ is a propositional logic formula then $s \in S$ satisfies the formula Φ if the evaluation induced by *L(s)* makes the formula Φ true. After defining a TS, the model of the system can be verified against a number of security properties stated in the initial requirements definition stage. The majority of security requirements regarding AC can be expressed as security properties. Security properties are invariants that are given by a condition Φ for the states and requires that Φ holds for all reachable states [14].

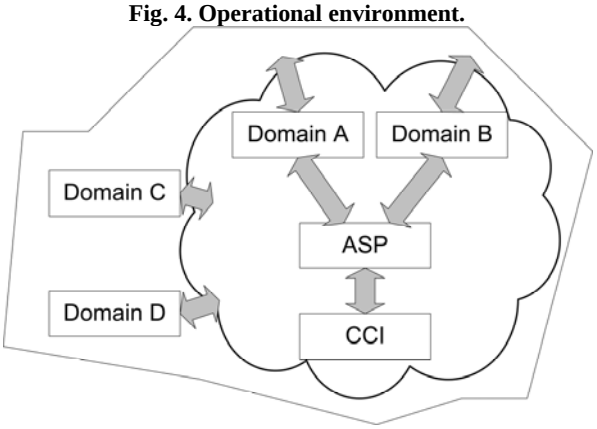
4 Use Case

In this section, we demonstrate the proposed methodology by recalling the application of CC on a generic Cloud environment to evaluate and enhance the Role-Based Access Control (RBAC) model [3]. A new produced AC model entitled domRBAC [18] is evaluated in the context of CC. Also, selected requirements were verified in domRBAC using a model checking approach by NIST.

The operational environment is illustrated in Figure 4. The Cloud environment is composed of individually administered domains, which can dynamically join in or quit the collaboration. Users from the participating domains can request on demand usage of services. Specifically, the environment is composed of companies A and B, represented by domains A and B, respectively. An Application Service Provider (ASP) is a corporate organization that can share a number of pay-per-use services. A complementary entity provides a computational computing infrastructure (CCI). Users from companies A and B can request capabilities from their local domain, collaborate with other users, manage their data and request on demand the use of services. The application of CC concludes to the following [3]:

Entropy requirements: The virtual distribution level of the system is low since there is only one formatted organization. The geographic distribution level that depends on the number of the participating domains can be high and entails heterogeneity issues. In order for the AC system to limit access to participating objects, it must be able to successfully authenticate them since domains might make use of different authentication protocols. Furthermore, since the organization's formation is not static, the AC system must continually observe all kinds of modifications.

Fig. 4. Operational environment.



Assets requirements: AC must be enforced on different types of assets. The scenario considers fine-grained AC on data since it requires sending for computation only segments of users' data. The ASP provides a number of services and the CCI a number of hardware resources. AC for both service and hardware level can be characterized as coarse-grained. The AC system must be able to enforce fine-grained AC on data and coarse-grained on services and hardware resources, respectively.

Management requirements: A number of services uses segments of users' data and submits them at the CCI. This requires a delegation mechanism. Thus, the AC system must be able to support delegation of access rights from domain users to the ASP and CCI. A security issue is that of delegated rights revocation. We assume that delegated rights must be revoked after the completion of a job or on demand by the user. The former requirement demands from the AC system an automation level and the latter to apply changes dynamically. Furthermore, trust relationships must exist between the involving parties. Furthermore, policy conflict resolution must also be examined when composite services exist.

Logic requirements: During user collaboration, their access at the CCI might be restricted by the owner of the resource. This requires an AC system that should support dynamic collaborations. Furthermore, support of stateful sessions because of long lived transactions and decomposition of composed services are required.

Based on the aforementioned requirements, we further proceed with the evaluation of the RBAC model in the context of CC. RBAC handles better centralized architectures where participants are known a priori. Additionally, RBAC is rather coarse-grained approach when it comes to assets definition. Assets, in RBAC, are grouped under roles and to become more granular, the assignments must be split into more. However, the use of context variables in known RBAC variations [19] overcomes such limitations. Delegation of rights and trust relationships are supported, and policy conflict resolution can be tackled in RBAC. Revocation of user assignments, hierarchies and temporal constraints are some of RBAC's. However, RBAC cannot support interactive environments, and also lacks an obligation mechanism, which usually corresponds to business requirements. Table 1 summarizes the evaluation of RBAC in the context of CC based on [3].

AC Model	CC Layers			
	Entropy	Assets	Management	Logic
RBAC	Low / Medium	Low / Medium	Medium / High	Medium
domRBAC	High	Medium	High	Medium/ High

Table 1. Comparison of AC models.

Therefore, after evaluating RBAC, we proceeded with the design of a new access control model entitled domRBAC [18] that is suitable for modern collaborative environments. Table 1 illustrates the evaluation of domRBAC compared to RBAC, where it is shown that domRBAC strengthens the RBAC approach in all layers. Specifically, the features in domRBAC that strengthens RBAC are: i) In entropy layer, domRBAC is capable of supporting secure inter-operation and has the ability to scale well. ii) In assets layer, we identify that domRBAC is able to support basic usage control and therefore usage restrictions can be introduced. Moreover, it is possible to define resources that are being shared by multiple stakeholders. iii) In management layer, using a small administrative overhead it is able to automate the management of policies in an easy and efficient way. iv) In logic layer, domRBAC strengthens RBAC since it provides features such as autonomy and security.

We further refer to the verification technique for the requirements of autonomy and security, which are identified in the logic layer of CC and supported in domRBAC. Secure inter-operation in collaborative systems is required for secure collaboration among participating parties such that the principles of autonomy and security can be guaranteed [20]. The principle of autonomy states that if an access is permitted by an individual system, it must also be permitted under secure inter-operation. The principle of security states that if an access is denied by an individual system, it must also be denied under secure inter-operation. Both principles can be characterized as security properties of a system, which should be preserved during collaborations since their enforcement means that something bad never happens [14].

For the verification of secure inter-operation properties, we applied the technique proposed in [21], which focuses on the verification of generic properties for AC models. The technique is able to cope with various types of AC properties including static, dynamic and historical. It also supports the generation of test cases to check the conformance between models and policy rules through combinatorial test array [22], and optionally generate the verified AC policies in eXtensible Access Control Markup Language (XACML) version 2.0 or 3.0, which is becoming the de facto language for the specification of policy rules in modern collaborative systems such as the Cloud. We adopt the finite state machine to describe the transitions of the authorization states, and the usage of static constraints so to adequately cover the verification of secure inter-operation properties in RBAC. The technique is to verify specified AC properties against AC models using a black-box model checking method [6]. An implementation -- Access Control Policy Tool (ACPT) [23] is developed by NIST Computer Security Division in corporation of North Carolina State University.

ACPT provides graphical user interface templates for composing AC policies and properties. Checking for conformance of AC properties and models is through the SMV (Symbolic Model Verification) model checker. In addition, ACPT provides a complete test suite generated by NIST's combinatorial testing tool ACTS [22] and an XACML policy output for the verified model. Through these four major functions, ACPT performs syntactic and semantic verifications as well as the interfacing for composing and combining AC policies. ACPT assures the efficiency of specified AC policies, and detects policy faults that leak or prohibit legitimate access privileges.

5 Conclusions

In this paper, we proposed a methodology for the development and verification of AC systems. The development stage is based on CC and the verification stage on a sound and mathematical underpinning technique (i.e. model checking). The latter is feasible via the definition of the system's TS and security properties, which have to be verified. The proposed methodology can be applied on any existing development process since it does not break the development model. Through an example, we demonstrated the proposed methodology, which resulted in a critique of existing AC approaches in [3] and the development of a new AC model for collaborative systems (e.g. Cloud environments) in [18]. Furthermore, we referred to the application of a model checking approach proposed by NIST, which resulted in the verification of the principles of security and autonomy in our defined AC model.

6 Acknowledgement

This work has been (partially) funded by the Research Committee of the University of Macedonia, Greece.

7 References

1. Mahowald, R.P., C.G. Sullivan, and A. Konary. *Market Analysis Perspective: Worldwide SaaS and Cloud Services, 2012 — New Models for Delivering Software*. 2012; Available from: http://www.idc.com/getdoc.jsp?containerId=238635#.UM4QauTqmuM.
2. Media, M.R. *Global Cloud Computing Market Forecast 2015-2020*. 2012; Available from: http://www.marketresearchmedia.com/?p=839.
3. Gouglidis, A. and I. Mavridis, *On the definition of access control requirements for grid and cloud computing systems.* Networks for Grid Applications, 2010: p. 19-26.
4. Hansen, F. and V. Oleshchuk, *Conformance checking of RBAC policy and its implementation.* Information Security Practice and Experience, 2005: p. 144-155.
5. Jayaraman, K., et al. *Automatic error finding in access-control policies*. in *Proceedings of the 18th ACM conference on Computer and communications security*. 2011: ACM.

6. VINCENT, C., et al., *Model checking for verification of mandatory access control models and properties.* International Journal of Software Engineering and Knowledge Engineering, 2011. **21**(01): p. 103-127.
7. Fisler, K., et al. *Verification and change-impact analysis of access-control policies*. in *Proceedings of the 27th international conference on Software engineering*. 2005: ACM.
8. Capitani di Vimercati, S., S. Foresti, and P. Samarati, *Authorization and Access Control*, in *Security, Privacy, and Trust in Modern Data Management*, M. Petković and W. Jonker, Editors. 2007, Springer Berlin Heidelberg. p. 39-53.
9. Ravi S. Sandhu, P.S., *Access Control: Principles and Practice.* IEEE Communications Magazine, 1994. **32**(9): p. 40-49.
10. Foster, I., et al. *Cloud computing and grid computing 360-degree compared*. in *Grid Computing Environments Workshop, 2008. GCE'08*. 2008: Ieee.
11. Mather, T., S. Kumaraswamy, and S. Latif, *Cloud Security and Privacy: An Enterprise Perspective on Risks and Compliance*. 2009: O'Reilly Media, Inc. 292.
12. Mell, P. and T. Grance, *The NIST definition of cloud computing (draft).* NIST special publication, 2011. **800**: p. 145.
13. Sommerville, I. and G. Kotonya, *Requirements engineering: processes and techniques*. 1998: John Wiley & Sons, Inc.
14. Baier, C. and J.P. Katoen, *Principles of model checking*. Vol. 26202649. 2008: MIT press.
15. Heljanko, K. *Model Checking based Software Verification*. 2006; Available from: http://iplu.vtt.fi/digitalo/modelchecking.pdf.
16. Girard, A. and G. Pappas, *Verification using simulation.* Hybrid Systems: Computation and Control, 2006: p. 272-286.
17. Stevens, R., *Systems engineering: coping with complexity*. 1998: Pearson Education.
18. Gouglidis, A. and I. Mavridis, *domRBAC: An access control model for modern collaborative systems.* Computers & Security, 2012.
19. Tolone, W., et al., *Access control in collaborative systems.* ACM Computing Surveys (CSUR), 2005. **37**(1): p. 29-41.
20. Gong, L. and X. Qian, *Computational issues in secure interoperation.* Software Engineering, IEEE Transactions on, 1996. **22**(1): p. 43-52.
21. Hu, V.C., D.R. Kuhn, and T. Xie. *Property verification for generic access control models*. in *Embedded and Ubiquitous Computing, 2008. EUC'08. IEEE/IFIP International Conference on*. 2008: IEEE.
22. Kuhn, R., Y. Lei, and R. Kacker, *Practical combinatorial testing: Beyond pairwise.* IT Professional, 2008. **10**(3): p. 19-23.
23. Hwang, J.H., et al. *ACPT: A tool for modeling and verifying access control policies*. in *2010 IEEE International Symposium on Policies for Distributed Systems and Networks*. 2010.