International Journal of Soft Computing 10 (6): 369-382, 2015
ISSN: 1816-9503

A CIM Based Security Policy Refinement Process from Security Objectives to Concrete Configurations

Anas Abou El Kalam, Jean-Philippe Leroy, Larbi Bessa and Jean-Marie Mahe
IPI-LISER/Propedia, Paris, France

Abstract: Managing security and configuration in a large scale distributed network is a labor-intensive task, error prone and time-consuming. This is mainly due to the large number and the complexity of security mechanisms that need to be enforced in order to meet the security goals. The misconfiguration of a single security component out of hundreds may cause failures, mainly related to availability, integrity, confidentiality and performance. In this study, we introduce a global framework based on Common Information Model (DMTF CIM) and Model-Driven Architecture (MDA) concepts to address the problem of security policy refinement process. The main goal is to automatize, enhance and simplify the different functions related to security configuration management which is generally manually performed and qualified as a hard-task, especially for large scale networks and systems. The proposed framework includes tree levels of abstraction to bridge the gap between high-level security policies and low-level ones that represent concrete configurations. Moreover, it integrates conflicting resolution mechanisms and proposes an open source based implementation.

Key words: Security management, security policy, CIM (Common Information Model), MDA, performance

INTRODUCTION

Managing security in today's information technology infrastructures is quiet hard. From large enterprises to small businesses and from nationwide data centers to personal computers, an inordinate amount of time and effort may be spent in managing IT. The exponential increase in the size of IT infrastructures, coupled with their growing technical complexity has led to a situation where despite some aspects of automation, there are not enough skilled people to ensure seamless security management and accurate enforcement of abstract policies (generally expressed in natural language). Indeed, the complexity and the interdependencies of network resources and processes make the configuration and management of security hard and error-prone task. Moreover, skilled security administrators need to intervene frequently to detect conflicting situations (e.g., between operational and security rules), keep the IT infrastructure running and behaving correctly.

According to Agrawal, the leading cause of 62% of the network downtime incidents is human error in its configuration. Pescatore found that 65% of cyber-attacks exploit configuration errors in the network. Moreover, server misconfiguration and information-leakage vulnerabilities are the most prevalent category of issues identified by OWASP, Web_AppII cation_Vulnerability_ Statistics Report (most of these misconfigurations may lead to unauthorized access to sensitive information). Thus, configuration in security management is a key challenge for system security administrators.

Subsequently, several research and industry works was focused on management frameworks that go beyond the direct human manipulation of network devices and systems (Horn, 2001). One approach toward this aim is to build Policy-Based Management Systems (PBMS). Policy-based management refers to a software paradigm developed around the concept of building autonomous systems or systems that manage themselves with minimum input from human administrators (Strassner, 2003). This paradigm provides system administrators and decision makers with interfaces that let them set general guiding principles and policies to govern the behavior and interactions of the managed systems. Although, some aspects of this paradigm are mature, notably the mechanisms related to QoS provisioning in IP networks and service level agreement in networks, few works address the automatic security management issues. Furthermore, existing studies are still in the research stage and large portions of the security management chores are still carried out manually and in an ad hoc manner or not adapted to complex and large-scale systems.

Corresponding Author: Anas Abou El Kalam, IPI-LISER/Propedia, Paris, France

In this research we deal with security in wide area, covering several aspects related to the operation of distributed applications and networks. Basically, security deals with three basic properties of computer systems covering several aspects related to the operation of operations: confidentiality, integrity and availability. These aspects are often qualified as security services, specified by security policies (e.g., high-level access rules) and implemented by means of security mechanisms (e.g., Access Control Lists "ACL", digital certificates, ASML assertions, etc). Table 1 gives a general overview of the relation between the main security properties and the main security mechanisms that may implement them. More mechanisms are listed in the common criteria, the ISO 27001 (annex A), 27002.

Although, this list is not a complete catalog of all the tasks related to security of computer systems and networks, it provides a general sense for the complexity and magnitude of problems that are associated with the secure operation of systems. In all the mentioned tasks, policies can be used to manage different aspects related to the security of distributed computer systems and networks. Consequently, we need network elements that are consistently and properly configured and well managed in order to meet the security goals.

Dealing with these issues, our work focuses on the refinement and derivation of security goals to consistent and suitable security configuration. To reach this goal, we introduce in this study a global framework based on Common Information Model (CIM) and Model Driven Architecture (MDA) (OMG, 2001). CIM is a standardized information model promulgated and maintained by the DMTF (Distributed Management Task Force) while MDA is a software engineering paradigm that aims at considering "models" as first class products within a software development process.

Moreover, to bridge the gap between high-level security policies (that are derived from security goals) and low-level ones (that represent concrete configurations), we propose in this study a framework based on tree levels of abstraction:

Table 1: Security services vs. the main security mechanisms

Security service/Properties	Related tasks and mechnisms
Confidentiality	Provide support for authentication
	Provide support for access control
	Encryption
Integrity	Malware prevention
	Intrusion detection
	Compliance, access control
	Hash functions, HMAC, encryption
Availability	Prevent Denial of service attacks
	Provide the necessary resources
	Redundancy, replication

- SP-CIM: Security Policy Computation Independent Model
- SP-PIM: Security Policy Platform Independent Model
- SP-PSM: Security Policy Platform Specific Model

This approach brings several benefits. In fact, a significant part of the simplification produced by our approach is obtained by allowing the security administrator to only specify the high-level security objectives or goals rather than specifying the detailed configuration of the different system resources. Consequently, our proposed framework encourages the specification of component-level security policies to move one layer up where they can be written just once for a single type of component and transformed into technology-specific policies for different instances of a particular component type.

Furthermore, our approach provides the capability to analyze a set of security policy rules and to validate the fact that they collectively achieve well-defined security objectives without conflicts or errors.

LITERATURE REVIEW

The main approaches to automate the implementation of security mechanisms: Basically, we identify two main approaches to automate the implementation of security mechanisms and services: the requirement engineering approach and the policy based approach. In requirement engineering approaches, researchers try to add security features to existing modeling languages like UML. These modeling languages are not native security models, they thus often model the security policy locally, not from global view of thinking. Jurjens and Shabalin (2007) propose to use UMLSec which is an extension of UML. UMLSec allows the expression of some security-related information in UML diagrams. Stereotypes and tagged values are used to formulate the security requirements. Two models are proposed: a Security Requirements Model which includes architectural or behavioral system details in a prescriptive manner and a Concretized Model summarizing a concrete architecture which should satisfy the security requirements. Both models appear as UMLSec diagrams. However, the approach by Jurjens and Shabalin (2007) presents some drawbacks. First, no abstract model is employed for modeling the policy. Second, the expressions of security properties are application-dependent: there are no generic properties dealing with the anomalies that could exist within single or multi-component network security policies. Besides, this research did not support automating the translation of high level security policy into low level mechanisms.

Model driven policy based approch: Firmato is an interesting contribution on the treatments of security policy in networks with many firewalls and distinct security policies for sub-networks. Basically, Firmato is a firewall management toolkit with a model definition language, a model compiler, translating global knowledge of the model into firewall-specific configuration files and a graphical firewall rules illustrator. An entity-relationship model is used to specify both the access security policy and the network topology, mainly through the concept of roles. In this approach, there is some mixing between the network topology and the access security policy to be enforced, so that the role concept becomes ambiguous. Indeed, the researchers are bounded to introduce the "group" concept with an unclear semantics; sometimes group is used to design a set of hosts and sometimes it stands for a role. Besides, the approach concentrates the efforts in just one security component (firewalls).

Hassan and Hudec have introduced Role-based Network Security (RBNS) Model that can be used as an intermediary level between high-level policy and low-level firewall rule-base. The main concept of RBNS Model is that network services are assigned to roles and hosts are made members of appropriate roles, thereby acquiring the roles' network services. Researchers keep from the RBAC Model only the concept of role. Indeed, the specification of network entities, role and permission assignments are not rigorous and do not precisely fit reality. In particular, all RBNS relations are binary even though an access control security goal and its equivalent filtering rule are always a triple (source, service, target). Other models such only deals with web services or firewalls and does not have the flexibility to support other network security components. Moreover, there is no concrete link between the security policy item and its implementation; consequently, the implemented security mechanisms cannot be reviewed or assessed.

The research done by Laborde *et al.* (2007) presents a different solution to the problem of deploying security policies: the use of Petri Nets as the language to specify the system and CTL (Computational Tree Logic) as the language to express the security properties. Four generic system functionalities are identified and modeled as different Petri Nets which are then interconnected in order to specify the system (i.e., each security device is modeled by a Petri Net): channel (e.g., network links), transformation (IPsec and NAT), filtering (to include the firewalls) and the end flow functionality for the hosts (the active and passive entities in the network). The Petri Nets transitions for each PEPs (firewalls and IPsec tunnels) are guards which actually represent the security rules to be enforced by the PEP. The policy model is RBAC-based. Researchers have also proposed a formal framework that focuses on network security information management refinement. The framework includes three abstraction levels: the network security objectives, the network security tactics and the, network security device configurations. Our work enhances this refinement process by introducing clear relationship between different level of abstraction (inheritance, aggregation, active dependencies) and makes it usable in real distributed systems. This is possible as we used DMTF CIM as the basis of our framework.

Policy specification language: Policy specification languages are an attempt to formalize the intent of the owner into a form that can be read and interpreted by machines. For the study of current policy specification languages, we have considered the following languages:

The method presented by Bandara uses the Goal Oriented Requirement Engineering (GORE) approach (Van Lamsweerde, 2004). Goals are specified in KAOS (Dardenne *et al.*, 1993). Each goal is refined into sets of sub-goals based on predefined formal schemes until they can be directly enforced. Goal refinement formalization is an interesting approach; nevertheless, it is not sufficient to automate the refinement task because it only deals with the behavioral aspect. It does not consider the informational part of the refinement problem.

Some proposals express access control policies as XML documents as exemplified by XACML OASIS (2004). XACML is an XML specification for expressing policies for information access over the internet and is being defined by the organization for the Advancement of Structured Information Standards (OASIS) technical committee. The language provides XML with a sophisticated access control mechanism that enables the initiator not only to securely browse XML documents but also to securely update each document element. As a similar approach (of specifying security policies), Ponder is a declarative and object-oriented language policy language that can define both access control rules (like XACML) called authorization rules, general management rules called obligations and policies related to certain roles or positions.

However, most of formal approaches and some policy languages suffer from being non intuitive and do not easily map to implementation mechanisms. They assume a strong mathematical background that generally makes them difficult to use and understand.

Other interesting policy languages like CIM SPL are designed to work with the CIM Information Model. Each CIM-SPL policy is written under the scope of a single CIM object referred to as the anchor object of the policy. All other CIM objects referenced by a CIMSPL policy should be accessible by traversing CIM associations starting from the anchor object of the policy. In the same category, we identify other languages like the CIM Query Language or CQL. CQL is designed primarily for querying the CIM Information Model and thus is not a real policy language but rather a DMTF preliminary standard that facilitates writing queries for extracting data from a CIM data management infrastructure.

BACKGROUND

The general architecture and components: As pointed out in the introduction, policy based management systems represent an important approach that aim to automate management activities in distributed networks. The IETF architecture consists of four components (Fig. 1): a policy management tool, a policy repository, a Policy Decision Point (PDP) and a Policy Enforcement Point (PEP) (Yavatkar *et al*., 2000). The policy management tool provides a user interface for the creation and definition of policies. The policy repository provides mechanisms for storing policies and retrieving them as needed by the decision points. The policy decision points are modules in the system responsible for making policy decisions. They in fact analyze the policies stored in the repository that would be applicable under a given circumstance and determine what to perform in order to comply with those policies. Policy enforcement points are elements that are responsible for enforcing the outcome of those policy decisions.

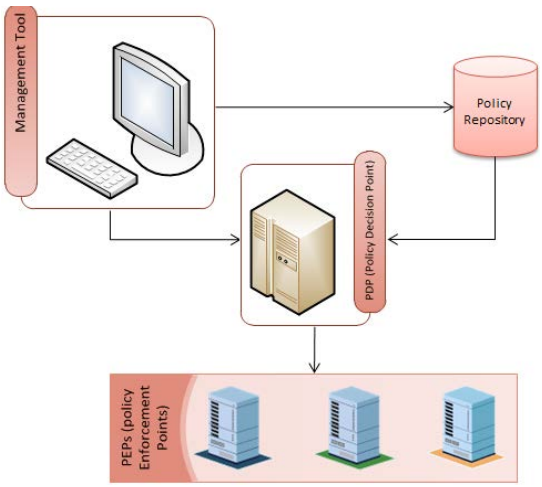

Fig. 1: The IETF policy architecture

This policy based management systems architecture (proposed by the IETF), is a little bit closed to the ones used by security standards such as XACML (Moses, 2009). The later uses in fact the following components: a PAP (Policy Administration Point: manages access authorization policies), PDP (Policy Decision Point: evaluates access requests against authorization policies before issuing access decisions), PEP (Policy Enforcement Point: intercepts user's access requests and enforces the PDP decision), PIP (Policy Information Point: acts as a source of attributes values) and PRP (Policy Retrieval Point: where security policies are stored).

When it comes to security management, the complexity faced in implementing such system is huge, especially for large scale and distributed systems. In fact, we need (at least) to give precise answers to the following questions: When where and how security policies are created, modified and transformed from higher levels of abstraction to lower level ones and then, to concrete implementation and/or configuration? How security policies are stored, distributed and enforced in a possibly geographically distributed system? When a policy becomes obsolete? How it is retracted throughout a system without disrupting the system operation?

Besides that, another aspect of this complexity comes from possibly conflicting security policies in a system that need to be resolved. Conflicts may arise during design time or at run-time and they need to be detected and resolved. Basically, conflicts become possible for different reasons not only because security rules contains some contradictory modalities (such as permissions or obligations and prohibitions) but also because there are often multiple policy authors responsible for managing different operating behaviors or subdomains of the whole system. Moreover, conflicts may also arise between the system functioning rules (business rules) and security rules. These issues may seem easy but they are in reality more complex car the system, functional, network and security operators/communities/managers (are often different and) have different interests to defend.

In this study, we focus on the first question. In fact, our framework represents a holistic view of how we specify and carry out the refinement process. The answer to this question becomes a challenging task when we know that implementing a security policy involves a

multitude of mechanisms. In the study, we present some technics which can make easier the task of security management in various scenarios.

Policy usages in security: Basically, policy-based management can be used for a variety of purposes when managing security. For example:

- Security policies specify access control guidelines about who can and cannot use the system; besides, the system then enforces the policy controls into the configuration of the security gateways (routers, firewalls) and other components
- Security policies are defined for high-level guidelines on organization of system behavior; these policies are then translated to underlying access control mechanisms on the information system entities (software, hardware, network, sites, organization, users)
- Security policies are someway defined to declare how the system administrators and system elements should satisfy the (security) needs and react to threats; the system then uses these policies to respond automatically to any detected security threats
- Security policies are defined to specify the level of security required for communications among different network elements; the system then automatically enforces such communication when secure information flows need to be created

Consequently, security policies in a context of policy-based management can be used for building a self-protection mechanism in distributed network and systems as well as managing access control and securing communication (e.g., using Ipsec, SSL/TLS, VPNs (Virtual Private Networks), RANs (Remote Access Network), Firewall rules, etc.).

In this context, given that managing all these applications and mechanisms in a comprehensive way requires handling security at a business-level abstraction, we need thus a common model or language that captures the semantics, structure and basic concepts of different security policies. This model should be able to express the concepts for managing devices and services that are involved in security policy specification. Moreover, on the one hand, it should be rich enough to include the complex and abstract relationships between items and on the other hand it should be simple enough for efficient software handling and implementation. To achieve these goals, we base our work on the Common Information Model (CIM).

CIM and policy information model: An information model is a structure for organizing information or data. It typically consists of a representation for data entities and the relationships among those data. The current state of the art in describing information model is to use an object oriented approach, frequently represented using the Unified Modeling Language (UML) (OMG, 2001). The object-oriented model consists of a collection of objects. Each object belongs to a class. A class represents an abstract definition for a group of objects of the same type. A class typically contains fields (which can contain data values) and methods (which define operations allowed on an object of that class). Basically, classes can be associated by specific types of relationships such as inheritance, aggregation, composition or dependencies.

Mainly based on UML, the Common Information Model (DMTF CIM) is a model for describing the overall information management in a network/enterprise environment. CIM is comprised of a specification and a schema. The specification defines details for integration with other management models while the schema provides the current model descriptions. The CIM schema captures notions that are applicable to all common areas of management, independent of implementations. Basically, the CIM can be described in one of the following formats: Unified Modeling Language (UML) diagram, Managed Object Format (MOF) or Extensible Markup Language (XML) (W3C, 2008).

The CIM schema includes the core and common models. We focus on the CIM policy model which represents one of the CIM Common Models. CIM Policy Model is an information model used to represent the structure of policies. It provides an abstract representation of the information needed to define a policy.

More precisely, the CIM Policy Model provides a common framework for specifying system behaviors that are both sufficiently abstract to be independent of implementation-specific details and scalable to configuring large complexes of distributed systems and networks, i.e., the DMTF Policy Model is a specific model for expressing such policies in a general and scalable way.

Figure 2 provides an overview of the classes that comprise the CIM Policy Model, their associations to each other and their associations to other classes in the overall CIM schema. The majority of this model is documented by Moore *et al*. (2003) and in the CIM Policy Model.

Note that as it is in an abstract way, the information model for policies needs to be specified, instantiated and visualized through a policy language or a GUI (Graphical User Interface).

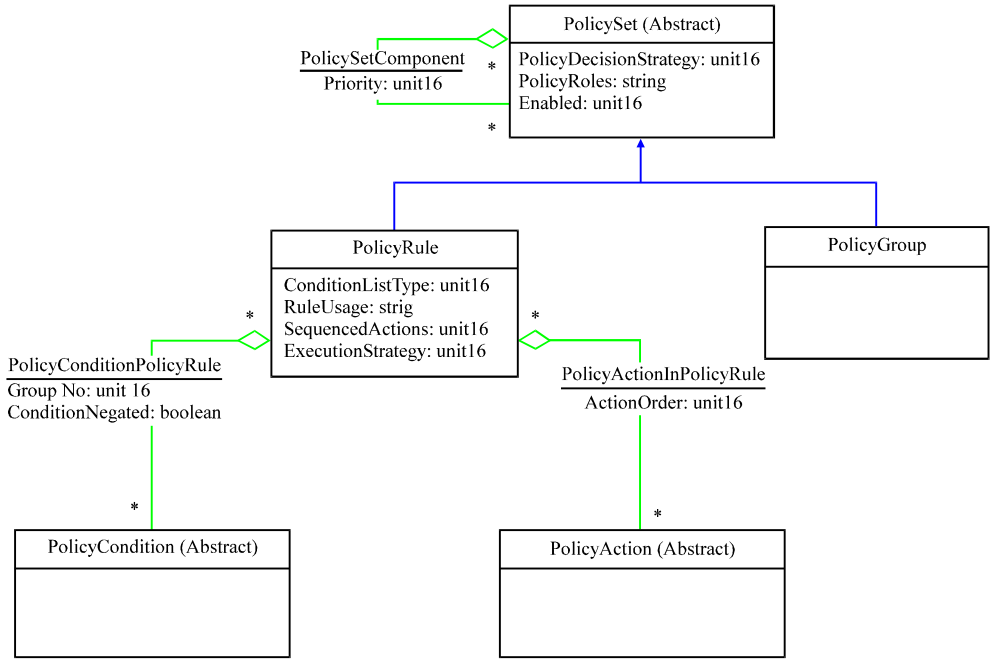

Fig. 2: A global view of CIM Policy Model

THE PROPOSED FRAMEWORK

Our proposed framework is focused on two main steps:

- A methodology for the identification of security needs and objectives
- A CIM-based security policy refinement process

Security objectives identification: In order to specify security objectives that are related to the target system, the tasks of Fig. 3 need to be performed. These tasks are then followed by our security policy refinement process. For compatibility and usability reasons, it is worth nothing that our methodology is based on industry standards (Stoneburner *et al.*, 2002). It is intended to serve as a starting point and to be compatible with existing, detailed security process standards such as the common criteria.

In general, the first analysis task is the identification of information assets that are processed by the system. These assets are then categorized qualitatively (e.g., high, medium and low) with respect to the potential impact to an organization and generally according to the following security properties:

- Confidentiality-protection against unauthorized disclosure of information

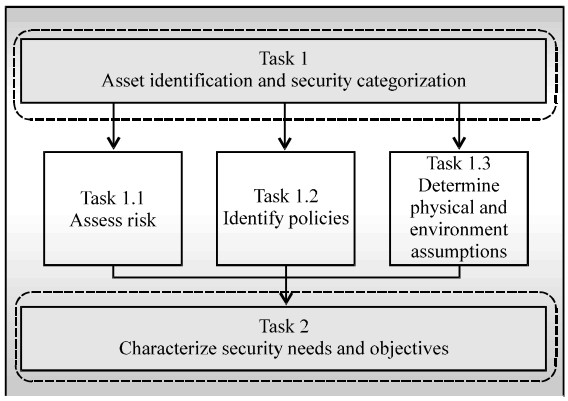

Fig. 3: Security needs and objectives identification methodology

- Integrity-protection against unauthorized modification or destruction of information
- Availability-protection against disruption in access to or use of information

Basically, security categorization assesses the impact of security services/properties loss, irrespective of the causes of loss. It actually reflects the sensitivity/criticity of the asset and provides an initial qualitative measure of the overall security needed to protect information system assets; as such, it can be used as the basis for determining the level of rigor that is applied throughout the remainder of the process.

After that, Task 1.1 "risk assessment", at the most basic level includes: identification of potential threats and threat actions, processes and sources that may result in a loss of security need or a violation of a security policy and analysis of the likelihood and severity of each threat action. This severity reflected to probability/potentiality of occurrence of the attack, the resources and knowledge of the attacker as well as the robustness of the security mechanisms to deploy.

Subsequently, the analysis results help the organizations prioritizing the risks and then selecting the most appropriate and cost-effective security controls and mechanisms that reduce the risks to an acceptable level. These tasks are commonly called risk management; techniques to manage the risks could be: avoidance, reduction, sharing and retention.

In parallel, Task 1.2 is used to identify policies that may have an impact on the selection of security mechanisms for the studied information system. The selected policy may depend on different factors such as the probability of the threat (the greater the likelihood of the threat, the less the attack will require resources and knowledge and the greater the security mechanisms should be robust and reliable). The policy may also depend on the systems constraints. The latest could be technique, organizational, legal, staff-related, etc. For instance, information systems that employ encryption which is a technical security mechanism, may be subject to several restrictions in some countries; consequently, these policies may impact the selection or use of encryption.

Task 1.3 examines the characteristics of the physical and operational environment in which the targeted system operates. That can impact the threat likelihood analysis and the selection of security mechanisms.

The final task (Task 2) is the characterization of security objectives which are overarching security goals that influence the selection of security mechanisms. Typically, security objectives reflect the results of tasks 1.1 through 1.3 and they represent organizational policies and business objectives. Note that the security objectives could be extracted from (or in conformance with) the common criteria, the ISO 27001 (annex A), 27002 and other norms and standards.

Finally, note that the proposed methodology describes the main steps to derive the security objectives. It remains simple and applicable but also compatible with other complex and hardly automatized methods such as OCTAVE, EBIOS, MEHARI and ISO 27005.

Security policy refinement process: As a result of the methodology presented in the previous study, we naturally derive a high level specification of security objectives that need to be implemented. We adopt a model-based approach to define a general framework for specifying, refining and deploying concrete security policies in order to achieve the security objectives. The major aim is to (automatically) convert security policies in terms of high-level goals into policies in terms of low level configuration parameters that could be interpreted by the system components.

To achieve this goal, several steps should be followed. First, we suggest that the transformation of (the high-level) security objectives to policies are done by a security administrator using a security policy-authoring tool and then mapped to CIM abstract models. Afterwards, security policies can be translated into low level platform independent security models and then to platform specific security models. Finally, different forms of security policy rules are generated for individual security components of the system. Consequently, the refinement process bridges the gap between the representation that is more familiar to the security administrator and the format that is easier to process at low-level systems.

Besides that, it is worth nothing that security policies should be stored for persistence and later retrieval. It can also be distributed for execution to various parts of the IT distributed system. Over time, policies may be altered and adapted to the changing IT system environments or to capture the changes in the high level policy set by the organization. Finally, policies that are expired must be disabled in the system.

Figure 4 shows a global overview of our proposed framework. Basically, at the highest level, security policies start as natural language descriptions. From these descriptions, many details need to be sorted out before policies can be implemented in term of security mechanisms. Indeed, the policy authors at this level may not have a technical background they write security policies as the user of an IT infrastructure using a vocabulary that may be ambiguous and therefore, requires further interpretation.

Basically, to write policies that have an unambiguous meaning, a glossary (or even an ontology) of terms that precisely describes various components in the distributed IT system, their functionality and concepts of system management needs to be agreed upon. We believe that the Common Information Model (CIM) provides such standard glossary of terms for systems' management. Moreover, the security policy creation module needs to include a user interface. The user interface for security policy creation is likely to be a panel embedded within a larger management system for networks and distributed applications. Common guiding principles of good

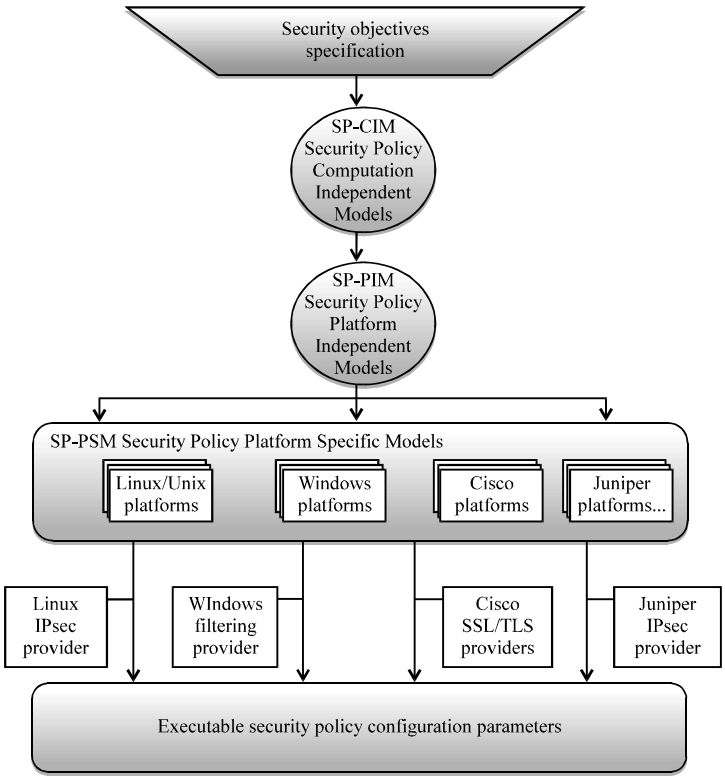

Fig. 4: An overview of the proposed framework

Graphical User Interface (GUI) design should be applied to the design of these panels: it should be simple, intuitive, flexible, consistent, etc.

In addition, the user interface may not allow the natural language description of security policies unless the system is designed to handle a limited vocabulary and syntax for the specific domain of security, so that the high-level security policy descriptions can be mapped to CIM abstract models and then to low-level ones without ambiguity.

In practice, the user interface will support writing security policies in a well-defined syntax but also should propose templates that can be filled to define policies or at least constrain the specification of security policies so that policies are syntactically correct. Moreover, ontologies may be user to help the administrator respecting global semantics. Examples of ontologies could be found by Cherdantseva and Hilton.

Typically, the user interface would have constraints in place so that only a few standard terms from the agreed-upon glossary can be placed in each blank position in a template. We refer to this layer as security objectives specification layer.

Subsequently, a more precise and unambiguous interpretation of the security policies authored at the security objectives specification layer is done with the help of modeling approaches that create abstract models of IT infrastructure and processes. As opposed to the previous layer this layer requires a technical background to be able to translate high-level security objectives using a formal specification such as Unified Modeling Language (UML) (OMG, 2001).

Note that using UML could necessitate a great effort to model system components and processes and then mapping security policies as constraints on the behavior of abstract system components and processes. Our approach, attempts to overcome these difficulties by using CIM DMTF standard (that is based on UML). CIM has the ability to represent (in high-level of abstraction) system components and processes and there complex relationships as discussed earlier. In fact, CIM seems suitable as it contains several specific abstract standard models (dedicated to the system components, among others) such as: CIM_Network, CIM_Device, CIM_System and CIM_Database.

To capture the semantics as well as the basic concepts of different security policies from the security objectives specification layer, we believe that the CIM_Policy Model is very suitable. Indeed, this CIM Model is based on one of the most widely used policy

information models describing a policy using condition-action rules (which means: if the condition is true, then perform the action). This information model is global enough to be able to specify more specific versions such as the event-condition-action rule (which means: upon occurrence of the event, if the condition is true then perform the action).

Note that in order to express the concepts for managing devices and services that are involved in security policy specification, the model should on the one hand be rich enough to include the complex and abstract relationships between items and simple enough for software to handle efficiently on the other hand.

The interaction between high level and this level will be handled by mean of CIM operations which can invoke one or more methods to enforce different management activities. These methods are either intrinsic operations defined by the specification to model general CIM operations or extrinsic operations defined as methods of specific CIM classes. The intrinsic operations can be grouped by their functionalities as showed in Table 2.

To improve the richness of the model at this layer, some other security models can be integrated to CIM. In our framework, we suggest using the Organization-Based Access Control (OrBAC) Model as it allows specifying several security policies independently of the implementation. Actually, OrBAC distinguish between the (abstract level) of the security policy (specified by abstract entities only) and concrete access (deriving the access decision (e.g., allow or deny the access) according to abstract security rules and the current situation of the system.

To achieve that ObBAC uses the concepts of organization, role, activity and view as abstract entities of the security rule. In fact, OrBAC abstracts subjects into roles. Consequently by means of roles, we are able to structure the subjects and to update easily the security policy when new subjects are added to the system. Similarly, a view is a set of objects that satisfy a common property. They characterize the way objects are used in the organization. For example, the "medical record" view abstracts files f1.txt, f2.xml, etc. containing medical data in hospital_1 (having a file management system) while the same view corresponds to Table 1 and 2 in hospital_2 (having a relational database). In the same way, OrBAC uses the "activity" entity to abstract actions. An activity joins thus actions that partake of the same principle or carry out the same objective/privilege. For example, the consulting activity will corresponds to the openf () action in hospital_1 while it corresponds to the select action in hospital_2.

Table 2: CIM intrinsic operations grouped by their functionalities

Functional group	CIM intrinsic operations
Basic "read"	GetClass, EnumerateClasses, EnumerateClassNames, GetInstance, EnumerateInstances, GetProperty EnumerateInstanceNames
Basic "write"	SetProperty
Schema manipulation	CreateClass, ModifyClass, DeleteClass
Instance Manipulation	CreateInstance, ModifyInstance DeleteInstance
Association traversal	Associators, AssociatorsNames, References, ReferencesNames
Query	ExecQuery
Qualifier declaration	GetQualifier, SetQualifier, DeleteQualifier, EnumerateQualifier

Using these concepts, a security policy that applies to a given organization is defined as a collection of permissions, prohibitions, obligations and recommendations. For example the rule "Permission (org, r, v, a, c)" specifies that in the org organization, the r role has the permission to carry out the a activity on the v view in the c context. We refer to this layer as security policy computation independent model as policies are mapped to the CIM Abstract Model which is implementation and security mechanisms independent. Indeed, the role, view, activity are indeed abstract entities that could be instantiated later according to the organization platform (the next level).

Besides by means of inheritance and different CIM relationships, models for common security mechanisms and services can be derived from the abstract models of the security policy (SP-CIM) layer. As pointed out in the previous study, our general framework supports automated mapping of any policy specified in the security objectives specification layer to the security mechanisms as represented in the DMTF CIM. Looking at this from the network element perspective, the highest specification layer (GUI) should be able to specify any reasonable security policy which can be modeled in the DMTF CIM. The DMTF CIM currently provides abstractions of common security mechanisms and services (e.g., authentication policies, authorizations policies, filtering policies, network communications security mechanisms). Figure 5 shows an overview of the common information model related to some security services. Note that we also extended this work to address some other security services and mechanisms in order to ensure security completeness.

Besides the concepts of inheritance and classical CIM relationships (normal associations, aggregations) that can be used to make security policies derivation and transformation, we distinguish CIM active dependencies. A dependency relationship links together two object classes at least: one of them is identified as antecedent and the other as dependent. The main goal of the concept is to be able to express the link between an event

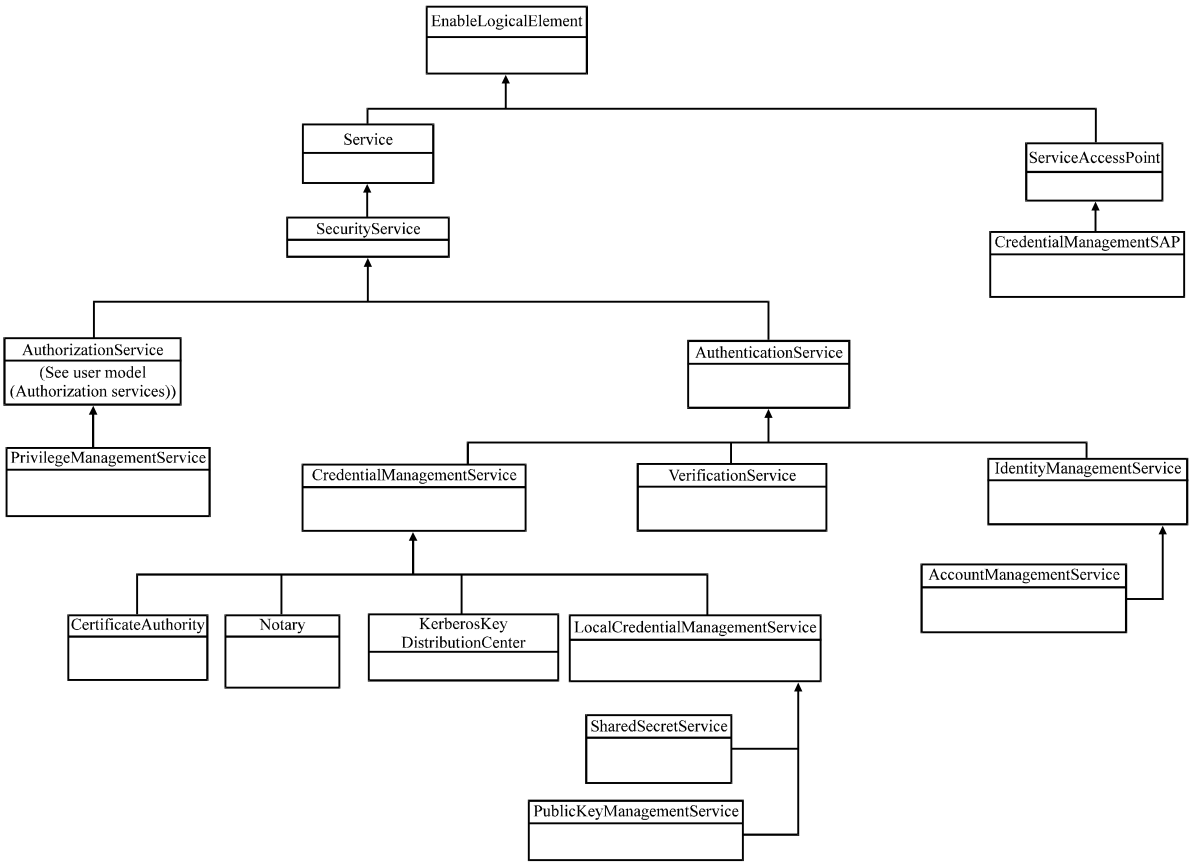

Fig. 5: Overview of some security services in the CIM

occurring on an antecedent instance and an action to be taken on the "dependency" instance. To achieve this, CIM specifies different kinds of actions to be taken depending on the "target" element of the antecedent instance. An action may for example ensure a mapping of property values between antecedent and dependent.

In addition to its use in our top-down derivation and transformation process, this concept of active dependencies is interesting as it can be used as a mean to make an adaptive security management. Indeed, over time, security policies may be altered to adapt to the changing IT system environments or to capture the changes in the security objectives specification layer set by the organization.

From an informational management point of view, a class schema as specified in CIM is a static representation as it is not able to specify the behavior of the managed system elements.

It is worth nothing that this layer offers models for security mechanisms and services instead of global models that capture security needs; we called this layer Security Policy Platform Independent Model (SP-PIM), as models are still platforms independent even if they specify several and specific security mechanisms.

Subsequently, as the mechanisms that can govern and constrain the behavior of a distributed system are provided by the capabilities, it is necessary to take security policies specified for the whole system and refine them into policies for individual components that must be upheld to meet the overall system wide policies specified at the highest layer. Security policies at this layer will be specified in different technology-specific models for different components and will include details of how an abstract system is implemented in a given IT infrastructure. At this layer, policies are no longer abstract but they are tied to a specific and concrete implementation of a system. Should the system or its components change, policies at this layer may need to be rewritten to take the implementation changes into account. We refer to this layer as security policy platform-specific model.

Finally, the platform-specific security policies are turned into configuration parameters, rules, constraints and application deployment descriptors that influence the behavior of system components at the runtime. We refer

to this layer as the executable security policy layer because the security policies at this layer can be directly consumed by the individual system security component.

Providers as showed in Fig. 4, act as drivers and interfaces between the abstract world of the CIM Models and the messy characteristics of real network elements (hardware and software). The providers supply the intelligence to translate abstract configuration specifications into specific commands that are adapted to the reality of the system and its technologies.

As we adopted a layered approach for security policy specification and implementation, at different layers, we expect to identify the need for additional specifications to ensure completeness as well as specific considerations which include: schemes to check whether the specified security policies are syntactically correct, schemes to detect conflicts among security policies, schemes to determine whether a set of defined security policies provides sufficient coverage, schemes to assess the impact of defining alternative security policies and finally, schemes for easing the process of defining security policies.

CONFLICTING RESOLUTION

Several definitions of the "policy conflict" have been given. The RFC3198 states that a policy conflict occurs when the actions of two rules contradict each other. Moffett and Sloman (1994) and Lupu and Sloman (1999) categorise conflicts into modality conflicts (contradiction between types of rules) and application-specific conflicts (due to model-specific inconsistencies or limitations). Strembeck has studied the problem of "conflict detection" in RBAC (Strembeck, 2004) in databases, etc.

In this study, we only deal with modality conflicts. In fact, we consider a conflicting situation when a user simultaneously has the:

- Prohibition and the obligation to carry out the same action on a specific object
- Prohibition and the permission to carry out the same action on a specific object
- Prohibition and the recommendation to execute the same action on a specific object

Such a situation is possible as at a given time, we could have several rules (permissions, prohibitions, obligations, recommendations) for the same (organization, role, activity, view, context). The conflict can occur:

- Between existing security rules
- Between existing rules and a temporary intervention of the administrator, e.g. to add or modify a rule or an entity
- Between existing rules and a rule resulting from the application of the inheritance mechanism (spreading by inheritance)

To prevent conflicting situations, we should first verify the coherence of the security policy (off-line verification). In other words, we need to verify that the system is in a secure state. We should also make sure that every intervention of the security administrator keeps the system in a secure state (online verification).

In this study, we suggest using Logical Programming by Constraints (LPC). LPC is a combination of the logical deduction process and a set of constraint resolving algorithms.

Basically, we affect priorities (an integer) to each security rule, e.g., Permission (Org, R, A, V, C, P): in the organization Org, the role R has the permission to perform activity A on view V with the "P" priority. The access decision is deduced according to the following axiom.

As for a particular (Org, R, A, V, C) we can have different security rules (permissions, prohibitions) with different priorities, it is possible to have conflicting decisions (with different priorities) for the same (s, o, a). To resolve these situations, we first assume that the system is in a secure state. Every intervention (e.g., to add a rule) of the administrator activates the following actions:

- The new rule (e.g., Permission (Org, R, A, V, C, $Priority_a$)) is first kept in a temporary base
- The authorization server invokes the process that extracts the relevant rules; this is achieved by extracting the rules that have the same attributes R, V, A, C, for example Interdiction (Org, R, V, A, C, $Priority_b$)
- If a conflict is detected (e.g., a permission and a prohibition), the system identifies, extracts and displays the rules responsible for this situation
- For an (s, a, o), we consider (among all the is_permited (s, a, o, P)), the one that has the greatest priority. We do the same for all the is_prohibited (s, a, o, P) is_recommanded (s, a, o, P) and is_obliged (s, a, o, P). Then, among these four predicates (decisions), we recuperate the one that has the greatest priority. If there is equality, we consider that is_prohibited is more critical than is_obliged, more critical than is_recommanded, more critical than is_permited

```
Permission (Org, R, A, V, C, Priority)^
Play (Org, s, R)^
Correspond_to (Org, o, V)^
Belong_to (Org, α, A)^
Is_true (C)
→Is_permitted (s, α, o, Priority)
```

Fig. 6: Rules with priorities

- The system suggests relaxations (corrections, alternatives) by inviting the user to accept the decision calculated in the previous step or to reformulate or to change the priority of one or several rules involved in the conflict
- The system takes into account these corrections, re-checks the coherence of the security policy and if the problem is solved- saves the new rule in the base that contains the security rules

Figure 6 gives examples of rules implemented with the Prolog language):

IMPLEMENTATION

We implement a global framework for the configuration and the monitoring of security mechanisms in the Linux environment using the standardized architecture (Fig. 7). To achieve this goal, we extend OpenPegasus (Mishra and Bedi, 2014), an open-source implementation of the DMTF CIM and WBEM (Web-Based Enterprise Management) standards (OpenPegasus, http://www.openpegasus.org.). OpenPegasus is designed to be portable and highly modular. It is coded in C++ so that it effectively translates the object concepts of the CIM objects into a programming model but still retains the speed and efficiency of a compiled language. WBEM defines:

- What is an entity (it sets the global architecture of the system, see following schema (Fig. 8)
- How the different entities communicate (XML through HTTP)
- A model to represent real life entities (such as a server, a printer, etc.)

Basically, as explained in Fig. 9, the GUI sends CIM-XML queries and supervises the state of the server by displaying the configurations. We implements providers for the main security mechanisms (filtering, virtual private networks) located in the server the language. CIMPLE providers works transparently

```
/*Examples of rules describing the system*/
org (hospital).
user (jean).
/*Examples of general relationships*/
subject (X):- org (X).
subject (X):- person (X)
/*Examples of rules OrBAC relationships*/
Play (Org_A, David, Physician).
Consulting_Physician (Jean, David).
age (Jean, 2).
/*Examples of inheritance rules*/
Play_inheritance (Z, A, X):- sub_role (Y, Z), Play (Y, A, X)
Play_inheritance (Y, A, X):- sub_org (B, A), Play (B, X, Y).
/*Adding an entity or a predicate */
Add (org (A)):- \+org (A), asserta (org (A)),!.
Add (org (A)):-
org (A), nl, write ('No:'), nl, write ('org ('), write (A),
write (')'), tab (1), write ('already'), tab (1), write ('exist.').
```

Fig. 7: Examples from the implementation of our conflicting resolution

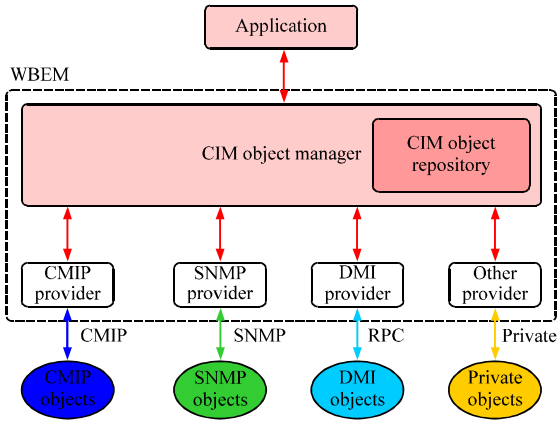

Fig. 8: The global architecture of our providers into the WBEM architecture

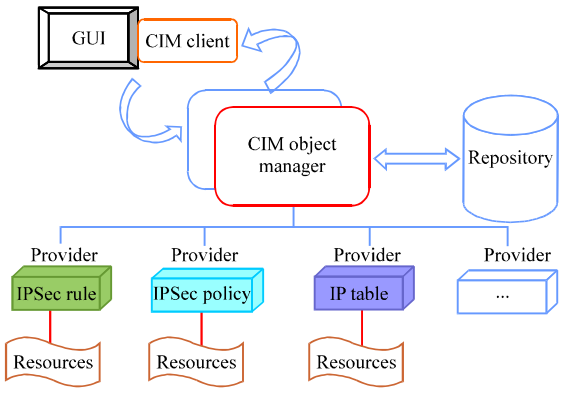

Fig. 9: The main architecture of our implementation

under several prominent provider interfaces including: OpenGroup CMPI specification, OpenPegasus C++ provider interface, OpenWBEM C++ provider interface and Microsoft WMI provider interface.

As our providers follow the WEBEM/CIM Model, all names of methods are standardized. Here is an example for IPSecPolicy.

```
Load_Status load();
Unload_Status unload();
Get_Instance_Status get_instance(
      const IPSecPolicy* model,
      IPSecPolicy*& instance);
Enum_Instances_Status enum_instances(
      const IPSecPolicy* model,
      Enum_Instances_Handler<IPSecPolicy>* handler);
Create_Instance_Status create_instance(
      IPSecPolicy* instance);
Delete_Instance_Status delete_instance(
      const IPSecPolicy* instance);
Modify_Instance_Status modify_instance(
      const IPSecPolicy* model,
      const IPSecPolicy* instance);
```

CONCLUSION

In distributed computer system, a security policy defines, among others, the constraints on the behavior of its elements, constraints on the offered functions and constraints on the data flow so that security needs are met (regarding the confidentiality, integrity and availability). Enforcing such constraints and configurations in various network elements is generally problematic, complex and error prone. Indeed, from security objectives we need to consistently and properly configure a large number of devices and applications within the distributed computer system. The complexity of this task comes from the fact that there are a large number of security mechanisms that need to be enforced using several technologies. The existence of reliable automatic means can inevitably assist security administrator to manage such a complex task and thus, keep track of system security configuration.

In this study, we established a global framework based on DMTF CIM for developing such means. Our proposal starts with a methodology for the identification of security objectives and then provides high-level abstractions that facilitate the job of the security administrator. In fact, it allows specifying the security policies as closer as the business needs of an organization (security objectives), rather than in language that is closer to the specifics of the technology underlying the systems to which the security policies apply. Therefore, the security administrator does not have to know all the details of the technology and how these business needs can be met, concentrating instead on 'what' should be done. Accomplishing this vision is done by introducing tree levels of abstraction that are based on the CIM Model: SP-CIM, SP-PIM and SP-PSM. Refinement relationships between deferent layers are based on CIM associations (inheritance and aggregation) and also the concept of active dependencies which is a recent contribution to CIM. We can thus translate and transform the general descriptions of security policies (mapped to SP-CIM layer by means of CIM operations) to the specific forms needed in different parts of the overall system.

RECOMMENDATIONS

The presented research, breaking the security policy refinement problem into functional layers, constitutes one more step towards autonomic management of security in term of configuration and effective monitoring. The benefits of our approach are:

- Using business rules to drive network configuration
- Managing security devices from different vendors
- Rapid deployment and reconfiguration of security policies which impact the productivity and deployment costs
- Efficiency and lower risk of human error with automated processes
- Seamless integration into complex network management frameworks as we adopted standard technologies
- Automatic and consistent enforcement of known best practices and security compliance
- Maximized reliability through increased prediction and control of network behavior

We are currently studying how to integrate the OrBAC formalism in our layered framework in order to enhance the ability of SP-CIM to capture several security policies types and consequently, improve its completeness. In the same way, we should study what kind of active dependencies actions we will use to enhance the refinement process.

Also, we expect specifying different management architecture options that are suitable for an effective implementation of our approach in distributed system context.

Finally, we will study how we can dynamically switch from a security policy refinement strategy to another, as security objective can be refined in different ways.

REFERENCES

Dardenne, A., A. Van Lamsweerde and S. Fickas, 1993. Goal-directed requirements acquisition. Sci. Comput. Programming, 20: 3-50.

Horn, P., 2001. Autonomic computing: IBM's perspective on the state of information technology. IBM Research.

Jurjens, J. and P. Shabalin, 2007. Tools for secure systems development with UML. Int. J. Software Tools Technol. Transfer, 9: 527-544.

Laborde, R., M. Kamel, F. Barrere and A. Benzekri, 2007. Implementation of a formal security policy refinement process in WBEM architecture. J. Network Syst. Manage., 15: 241-266.

Lupu, E.C. and M. Sloman, 1999. Conflicts in policy-based distributed systems management. Software Eng., IEEE Trans., 25: 852-869.

Moffett, J.D. and M.S. Sloman, 1994. Policy conflict analysis in distributed system management. J. Organizational Comput. Electron. Commerce, 4: 1-22.

Moore, B., L. Rafalow, Y. Ramberg, Y. Snir and A. Westerinen., 2003. Policy core information model extensions. http://www.priorartdatabase.com/IPCOM/000011355/.

Moses, T., 2009. eXtensible access control markup language (XACML) version 2.0. http://docs.oasis-open.org/xacml/2.0/access control-xacml-2.0-core-spec-os.pdf

OMG., 2001. OMG unified modeling language specification. Version 1.4, February 2001.

Stoneburner, G., A.Y. Goguen and A. Feringa, 2002. Risk management guide for information technology systems. file:///C:/Documents%20and%20Settings/Administrator/My%20Documents/Downloads/NIST-800-30%252f800-66+Summary.pdf.

Strassner, J., 2003. Policy-Based Network Management: Solutions for the Next Generation. 1st Edn., The Morgan Kaufmann Series in Networking, USA., ISBN-10: 1558608591, pp: 516.

Strembeck, M., 2004. Conflict checking of separation of duty constraints in RBAC implementation experiences. Proceedings of the Conference on Software Engineering, March 1-3, 2004, IEEE, pp: 21-27.

Van Lamsweerde, A., 2004. Goal-oriented requirements enginering: A roundtrip from research to practice. Proceedings of the 12th IEEE International Requirements Engineering Conference, Sept. 6-10, IEEE Computer Society, Washington, DC. USA., pp: 4-8.

W3C, 2008. Extensible Markup Language (XML) 1.0 W3C Recommendation. http://www.w3.org/TR/2008/REC-xml-20081126/.

Yavatkar, R., D. Pendarakis and R. Guerin, 2000. A framework for policy-based admission control. RFC 2753. http://www.faqs.org/rfcs/rfc2753.html.