Ontology-based policy translation

Ontology-based Policy Translation

Cataldo Basile, Antonio Lioy, Salvatore Scozzi, Marco Vallini

Politecnico di Torino
Dip. di Automatica ed Informatica
Torino, Italy
{cataldo.basile, antonio.lioy, marco.vallini}@polito.it, salvatore.scozzi@gmail.com

Abstract. Quite often attacks are enabled by mis-configurations generated by human errors. Policy-based network management has been proposed to cope with this problem: goals are expressed as high-level rules that are then translated into low-level configurations for network devices. While the concept is clear, there is a lack of tools supporting this strategy. We propose an ontology-based policy translation approach that mimics the behaviour of expert administrators, without their mistakes. We use ontologies to represent the domain knowledge and then perform reasonings (based on best practice rules) to create the configurations for network-level security controls (e.g., firewall and secure channels). If some information is missing from the ontology, the administrator is guided to provide the missing data. The configurations generated by our approach are represented in a vendor-independent format and therefore can be used with several real devices.

1 Introduction

One major weakness in today security landscape is still an old one: the human factor. Already back in the year 2000, the "Roadmap for defeating Denial of Service attacks"[1] highlighted the general *scarce technical talent* and *decreased competence* of system administrators. Unfortunately, this scenario has not significantly improved, as confirmed by recent reports [1]. It is therefore important to support security managers with automatic tools to minimize human errors.

Presently, policy-based network management (PBNM) [2,3] seems the best approach to cope with system administration because it separates the goals (i.e., the policy) from the mechanisms to achieve them (i.e., the security controls). Policies are expressed as high-level security statements, derived from business goals or best-practice rules. However actual security controls are mostly placed at network level. Firewall and virtual private network (VPN) are ubiquitous in security architectures because they have excellent performance (compared to application-level controls) and do not require changes to business applications. Evidence says that, as the network grows bigger, the configuration of network-level controls becomes exponentially complex. We have therefore a mismatch: policies are expressed at high level while controls are at network level. Bridging this gap is the task of *policy translation* that consists in transforming a policy from the level of abstraction A to the (lower) level B. This procedure is repeated until

[1] http://www.sans.org/dosstep/roadmap.php

Authors' version of the paper published in the proceedings of
CISIS-2009 - Int. Workshop on Computational Intelligence in Security for Information Systems
Burgos (Spain), September 23-26, 2009, pp. 117-126, DOI: 10.1007/978-3-642-04091-7_15
The final publication is available at www.springerlink.com in Vol.63 of the series Advances in Intelligent and Soft Computing

the policy is expressed in a way that can be used to configure a specific control (e.g., IP-level filters for a firewall).

Quite often translation from high-level policies to actual controls is manually performed by security administrators but this is still time-consuming and error prone: if policy translation is not automated, PBNM is simply a problem shift, not a solution. Our approach, called *Ontology-Based Policy Translator* (OPoT), handles this problem using ontology-based reasoning to refine policies into configuration for the actual controls. In particular, OPoT mimics skilled administrators in collecting all the needed information and applying best practice for security configuration. Ontologies are very effective in capturing, defining, sharing, and reusing the knowledge about a specific domain. In particular, they are good at representing relationships between entities and at verifying the correctness of knowledge [4]. Therefore we use an ontology to represent the domain of interest – computer networks – from the topological and functional point of view, and the environmental data necessary to correctly configure them.

OPoT exploits ontology-based reasoning to understand the information necessary for policy translation and to enrich the knowledge base in order to fill the gap between abstraction levels. In other words, automatic reasoning is used to enlarge the domain of knowledge of the ontology without the need to involve human administrators. Nevertheless, the reasoning alone is unable to fully translate a policy because some information cannot be guessed or derived (e.g., the users' roles). To this purpose, OPoT guides the administrator in providing the data necessary to complete the configurations. OPoT also reports about possible anomalies when it discovers that best practice is not followed.

The advantages of this approach are manifold. First, it addresses policy translation in a way that reduces the impact of administrators' mistakes in the configuration workflow: we just assume that the provided information (e.g., organization charts, tasks) is correct. Second, it strongly decreases the effort and the time to have a set of correct configurations (from days to minutes). Finally, it quickly adapts to changes in the methodologies. The proposed approach can still be improved in several aspects, but our initial experimental results demonstrate its power as well as its extensibility.

2 Background and Related Work

The adoption of security ontologies is continuously increasing for network security and risk management fields. Strassner suggests that the traditional information and data models are not capable to describe detailed semantics required to reason about behaviour. His work [5] modifies the existing DEN-ng policy model to support and generate ontologies for governing behaviour of network devices and services.

Tsoumas et al. [6] develop a security ontology to perform network security management thus modelling assets (e.g., data, network, services), countermeasures (e.g., firewall, antivirus) and the related risk assessment. They also describe a framework for security knowledge acquisition and management.

Fenz et al. [7] define a knowledge model (defining a security ontology) to support risk management domain, incorporating security concepts like threats, vulnerabilities, assets, controls, and related implementation. . Another work [8] proposes an ontology-based approach to model risk and dependability taxonomies. The related framework

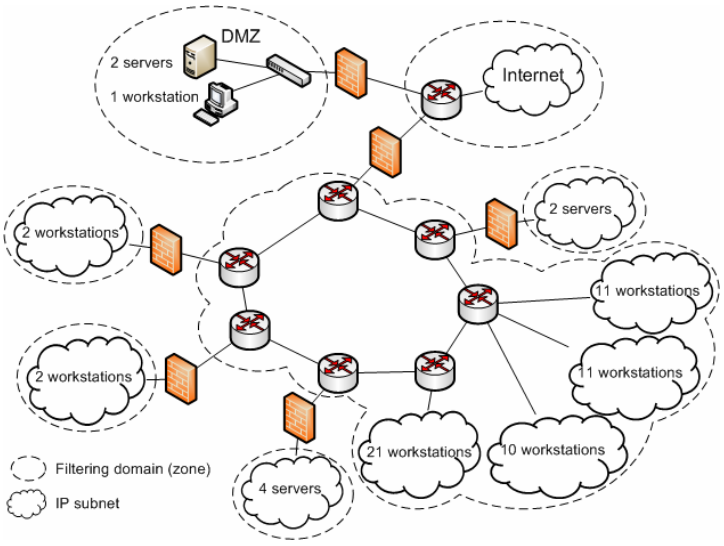

Fig. 1. The reference network used to validate our approach.

contains a tool for simulating threats against the modeled ontology. KAoS represents another approach to policy-based management using the description logic and ontologies from representation to conflict analysis and translation [9,10].

3 Case Study

We present in Fig. 1 the network use as a case study to validate our approach and to present its main results and implementation. The network contains 90 nodes divided into the following categories: workstation, server, router, firewall and switch. A server is a computer that provides a service through a Service Access Point (SAP). The network is organized into different IP subnetworks, according to different company's departments and it includes also a DMZ (De-Militarized Zone). The network is divided into seven filtering domains, called zones [11]. Each zone is delimited by network elements with filtering capability (firewalls in this case study).

In this network we will configure two categories of security controls: traffic filtering and channel protection policies. Enforcing these kind of policies is theoretically easy, nevertheless they are constant sources of problems and mistakes, especially when the size on the network grows and for non-experienced administrators [12].

4 Our Approach

The objective of our ontological framework is to derive configurations for security controls - based on ACL (Access Control List) and secure channel mechanisms - from a

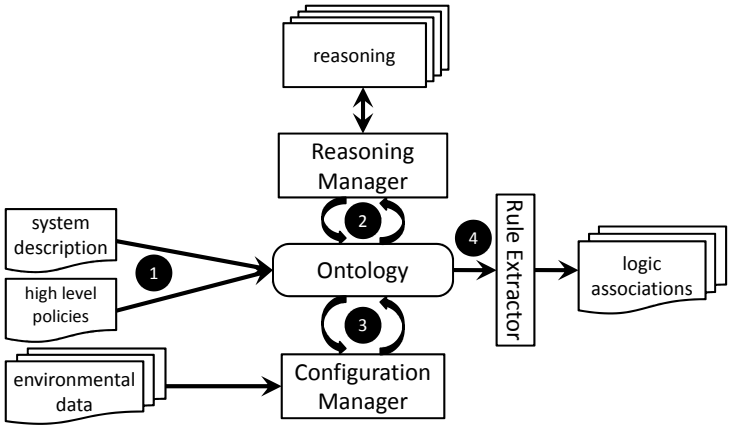

Fig. 2. The logical workflow of the OPoT system.

fixed set of business-oriented policies using automatic reasonings and interactions with the security administrator. The main phases of this process are presented in Fig. 2. The security administrator is the person in charge of configuring network security: we assume that he can access all the information required to generate configurations.

The ACL controls are derived both for devices (e.g., router, managed switch, firewall) and end-nodes. On the end-nodes we can distinguish two classes of controls: OS-level (e.g., local firewall) and application-level (e.g., web server access control). We also aim at configuring secure channels: TLS (end-to-end) and IPsec (end-to-end, gateway-to-gateway and end-node-to-gateway). The output of this process is represented in a vendor-independent format inspired to the CIM Simplified Policy Language [13] and IETF Policy Core Information Model [14] and its extensions [15].

To simplify the problem, the configurations are obtained as translation of a set of twelve basic policies. Every policy is a statement describing a business-level security objective. Examples are "every user must access only the services he is authorized to use", "business services must be connected securely to their components located in other parts of the network at higher security". We analyzed different sources to decide the sample policies to support. First we combined our experience and interviews with the security administrators in our institution (representative of an open environment with several unrelated actors, multiple levels of security and untrusted users) and partners (representative of private networks handling highly sensitive data and hosting some public services). Moreover, we analyzed different policy models (e.g., Role Based Access Control (RBAC) [16] and Mandatory Access Control [17]). Finally, we examined publicly available policy templates, in particular the SANS Security Policy Project [18].

Together with policies, OPoT takes as input the initial security ontology containing the class hierarchy and the system description represented in the P-SDL language [19].

The process of policy translation as knowledge refinement and enrichment has been designed to mimic the behaviour of skilled and experienced administrators in acquiring environmental information and translating it to actual rules to enforce best secu-

rity practices. Knowledge enrichment is performed through two entities, the *Reasoning Manager* (RM) and the *Configuration Manager* (CM). The RM coordinates a set of *reasonings* that apply logical inference on the knowledge base to increment it using a standard ontology reasoner. Example of reasonings are the automatic identification of DMZs, the classification of services according to their scope, the classification of SAP. However, not all the required information can be derived from the inputs. The CM coordinates the import of external/environmental non-deducible information by driving the administrator to collect missing information. For example, OPoT may asked to provide the host-user assignments, the user's task assignment according to the company's organization charts (e.g., expressed using RBAC), or description of business services provided through the network (e.g., the WS-CDL description of services).

The logical workflow of our framework is simple. First of all, the administrator chooses among the available policies. Every policy entails the type of information required to refine it, that is, the reasoning and data to be acquired to be enforced. Then, system description is used to populate the initial ontology. Subsequently, OPoT asks RM to execute the implied reasonings. Reasonings also warn the administrator about possible anomalies. If needed, CM asks the administrator to provide missing data through a graphical user interface. RM runs always before CM because it does not require effort from the administrators. This cycle may run more than once until the policy is enforceable. If in a cycle the ontology is not modified and the goals are still not achieved, the policy is considered not enforceable and a warning message is sent to the administrator.

The last entity, the *Extractor*, derives the logical associations from the ontology and then it completes the translation. A logical association expresses an interaction between parties (typically one-to-many) and the related communication properties (filtering constraints, e.g., need-to-access condition). But it is not our desired output, yet. The translation process distinguishes between topological-independent and topological-dependent logical associations. In the first case (e.g., end-to-end interactions such as TLS protected channel) the process directly generates the configurations for the end nodes. In the second case, the network topology needs to be analyzed. All paths between the source and destination nodes are identified and rules are generated for the involved devices. For example, when a company client need to reach an internal service, OPoT configures all the firewalls in the paths between them.

4.1 The Security Ontology

Our security ontology is structured in three levels (Fig. 3). The first one contains classes whose instances cannot be derived from other classes. First level instances are created gathering information contained in external files and running algorithms to manipulate it (e.g., filtering zones). The next levels are generated through either the usage of a standard reasoner (in our case Pellet [20]) or using the reasonings. The higher the level, the more detailed the information; for example, in the second level computers are classified in workstations or servers and in the third level the workstations are further classified in shared or personal. The security ontology contains several properties linking the instances of different classes. This permits to navigate and analyze the ontology

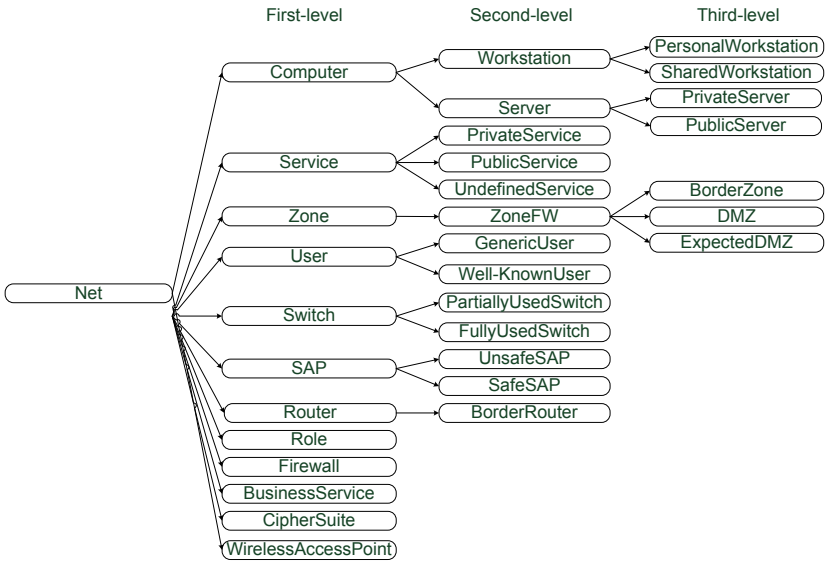

Fig. 3. Security ontology.

for deduction purposes. Thus, for example, an instance in the server class has a property that permits linking it to the instances of the SAP and it is linked to the Ciphersuites.

The ontology has been defined to perform a first automatic classification of second/third level classes. For instance in Fig. 4 is presented the OWL class definition that classifies zones as DMZ if they contains public server and they are linked to a border zone (zone including an access to Internet). If a zone satisfies only the first condition, it is considered as expected DMZ and it will be analyzed to check its characteristics. After system description is scanned, a reasoner is run to perform the automatic classification.

Every reasoning is characterized by a set of input ontology classes and a set of output ontology classes, subclasses of the input ones. The reasoner tries to "move" the instances of input classes towards higher levels of details, that is, to output classes. A set of SPARQL [21] queries permits to retrieve the instances to be analyzed together with all the data useful for the decision.

For instance, the ServerReasoning takes as input the instances of Service classes, tries to understand if they provide functionalities that should be visible from the outside. In order to do this, the reasoning collects information about the firewall zones and possibly the potential DMZ, if already identified. For example, a service that is reachable from the Internet without encountering a firewall is considered public, but the administrator is warned about a possible anomaly.

The CM is organized in a set of modules, each one is able to insert information into the ontology. For this reason each configurator module is characterized by the classes whose instances may be inserted, the classes whose instances can be modified and the properties that can be added. For instance, the UsersRoleConfigurator module manages

the RBAC information and it only inserts instances of the RBAC classes while the ServiceConfigurator, able to parse WS-CDL information, may move instances of the Service classes to PublicService or PrivateService classes and it can add the properties NeedtoAccess, a property we introduced to represent the network connections that must be guaranteed for the correct policy enforcement. Additionally, both the configurator modules and reasonings contain precedences, a list of modules that must be run before.

Every policy is characterized by its "goal", that is, the properties that must be derived to implement it, and a set of subsidiary classes that indicate from which information the properties must be derived. In the next section will be presented the case of the "Every user must access only the services he is authorized to use" policy, having as goal the NeedToAccess property and auxiliary classes the User class and the Service class.

The OPoT tools uses information about the single components to decide the ones to execute and the order. Starting from a policy, the tool tries to identify the modules able to derive the goal properties of the policy, then it identifies the dependencies, e.g., which components involve classes that are used as input for detected modules. The result is a dependency graph, later processed by RM and CM. This approach allows the maximum flexibility. In fact, every time a new module is added it is enough to specify which classes it uses and OPoT can automatically decide when they can be used. Nevertheless, a convergence problem arises, indeed, modules may work on the same data originating a loop in the dependency graph. In our example, the convergence is assured because existing modules have been designed to reduce it and because the entire cycle is repeated until the policy is correctly implemented. The solution of this problem will be improved in future versions of the tool.

4.2 An Example of Policy Translation

As an example we present here the case of derivation of filtering rules associated to the policy "Every user must access only the services he is authorized to use". This represents a typical goal in many organizations, implicitly adopted during network configuration. Translating this policy means, as a minimum, to insert all the necessary NeedToAccess properties inside the ontology from components that manage the Service class and the User class. The policy codes in term of ontology classes the best practice stating that a user is supposed to be authorized to access all the private services he needs to use for performing his work, all the company's public services and the Internet, according to the "Remote Access Policy" [18]. A particular attention is devoted to shared workstation. They are nodes usable by several types of users and therefore must be able to access all the services needed by these users to perform their work.

OPoT must define the NeedToAccess associations, thus it "understands" that the first component to run is the UsersRoleConfigurator. In fact, in principle it is not possible to deduce which services a user needs to use, the CM asks administrator to include the explicit user-role RBAC description also containing the remote access policy.

OPoT maps users and services to IP nodes. Service information are present in the ontology in form of SAPs obtained from the system description. The association between users and IP addresses is a typical information that cannot be derived by the system description. For this reason, OPoT asks the administrator to provide the workstation mapping: a user is assumed to use his workstation or a shared one.

```
<owl:Class rdf:about="#DMZ">
        <owl:equivalentClass>
            <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#contains"/>
                        <owl:someValuesFrom rdf:resource="#PublicServer"/>
                    </owl:Restriction>
                    <owl:Restriction>
                        <owl:onProperty rdf:resource="#isLinkedTo"/>
                        <owl:someValuesFrom rdf:resource="#BorderZone"/>
                    </owl:Restriction>
                </owl:intersectionOf>
            </owl:Class>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#ZoneFW"/>
</owl:Class>
<owl:Class rdf:about="#ExpectedDMZ">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#contains"/>
                <owl:someValuesFrom rdf:resource="#PublicServer"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#ZoneFW"/>
</owl:Class>
<owl:Class rdf:about="#PublicServer">
        <owl:equivalentClass>
            <owl:Restriction>
                <owl:onProperty rdf:resource="#provides"/>
                <owl:someValuesFrom rdf:resource="#PublicService"/>
            </owl:Restriction>
        </owl:equivalentClass>
        <rdfs:subClassOf rdf:resource="#Server"/>
</owl:Class>
<owl:Class rdf:about="#PublicService">
        <rdfs:subClassOf rdf:resource="#Service"/>
</owl:Class>
```

Fig. 4. An extract of the OWL ontology that tries to automatically classify the DMZ

OPoT then deduces from auxiliary classes as well as input classes that services must be classified. As a pre-requisite the DMZ must be identified. The objective of a DMZ is to provide services to a zone considered at a lower security level. Usually a public DMZ is a subnetwork directly connected to the border firewall (if there is one) or to a firewall connected to a border router where services are located. The Internet is supposed at security level 0, while the internal network and the DMZ are at level 1. Then, the reasoning looks for other internal DMZs, it identifies the separation domains, and assigns security levels accordingly. The assumption is that a service in DMZ at security level l must be available from zones having security level $l-1$ but not from the ones at level $l-2$. For services for which it is not possible to identify the scope, administrator is asked to provide the classification. This reasoning reports a set of anomalies to the administrators: the DMZ should not contain workstations and services in zones at security level l should not be accessible to hosts in zones at level $l-2$ or smaller creating breaches in the security "compartments". After CM and RM have run all the identified components and possibly repeated the workflow, all the needed properties are derived.

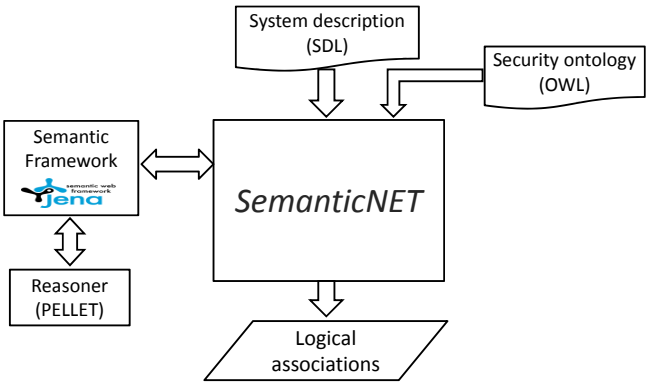

Fig. 5. Architecture.

5 Implementation

OPoT is implemented in Java (Fig.5) using the Jena API [22] to manage the security ontology. The system description and the environmental data are expressed in XML using standards when available (e.g., RBAC profile of XACML [23]). Our security ontology is written in OWL-DL [24] and the software used to edit it is Protégé [25]. A set of classes parses the system description and runs a series of graph-based algorithms to extract additional information (e.g., the filtering zones). The policies are represented as Java classes. A policy links the reasonings to gather the information needed to its translation. Also reasonings are implemented as Java classes that interact with the ontology through the Jena framework and the Pellet reasoner [20] using the Description Logic Interface (DIG) [26].

While deriving the logical associations, OPoT performs several controls, according to best practice, to find ambiguities or conflicts. In case of anomalies, it shows to administrator the list of the problems and suggests him the possible solutions. Finally, the logical associations are extracted using SPARQL [21] to query ontology.

Considering the network in Fig.1 and seven out of twelve policies, the tool spends about two minutes of CPU time from the policy selection to the translation. This test was performed using a PC with a 2 GHz CPU and 1 GB of RAM. The max memory utilization was about 30 MB.

6 Conclusion and Future Work

This paper presented OPoT, an ontology-based approach to policy translation. It uses a security ontology to drive the administrator from high-level policy specification downto system configuration, mainlt at IP level. In principle, OPoT can be extended to use other network security controls, such as IEEE 802.1x. Moreover, automatic reasoning is currently able to cope only with a pre-defined set of policies. In future, when the automatic semantic analysis of text will came of age, we hope to derive the policies directly from a high-level textual specification.

References

1. Agrawal, D.: Business impact of research on policy for distributed systems and networks. In: IEEE POLICY-2007, Bologna, Italy (June 2007)
2. Westerinen, A., Schnizlein, J., et al., J.S.: Terminology for Policy-Based Management. RFC-3198 (November 2001)
3. Strassner, J.C.: Policy Based Network Management. Morgan Kauffman Publishers (2004)
4. Gruber, T.R.: Toward Principles for the Design of Ontologies Used for Knowledge Sharing. Int. Journal Human-Computer Studies **43**(5-6) (1995) 907–928
5. Strassner, J., Neuman de Souza, J., Raymer, D., Samudrala, S., Davy, S., Barrett, K.: The design of a new policy model to support ontology-driven reasoning for autonomic networking. In: LANOMS 2007, Rio de Janeiro, Brasil (Sept. 2007) 114–125
6. Tsoumas, B., Gritzalis, D.: Towards an ontology-based security management. In: Int. Conf. on Advanced Information Networking and Applications, Vienna, Austria (2006) 985–992
7. Fenz, S., Ekelhart, A.: Formalizing information security knowledge. In: ASIACCS, Sydney, Australia (2009) 183–194
8. Ekelhart, A., Fenz, S., Klemen, M., Weippl, E.: Security ontologies: Improving quantitative risk analysis. In: Hawaii Int. Conf. on System Sciences, Big Island, Hawaii (2007) 156a
9. Uszok, A., Bradshaw, J.M., Johnson, M., Jeffers, R., Tate, A., Dalton, J., Aitken, S.: KAoS policy management for semantic web services. IEEE Intelligent Systems **19**(4) (2004) 32–41
10. Uszok, A., Bradshaw, J., Lott, J., et al.: New developments in ontology-based policy management: Increasing the practicality and comprehensiveness of KAoS. In: IEEE POLICY-2008, Palisades (NY, USA) (June 2008) 145–152
11. Mayer, A., Wool, A., Ziskind, E.: Offline firewall analysis. Int. J. Inf. Secur. **5**(3) (2006) 125–144
12. Al-Shaer, E., Hamed, H.: Modeling and Management of Firewall Policies. IEEE Transactions on Network and Service Management **1**(1) (April 2004) 2–10
13. Agrawal, D., Calo, S., Lee, K.W., Lobo, J.: Issues in designing a policy language for distributed management of it infrastructures. In: IFIP/IEEE Int. Symp. on Integrated Network Management, Munich, Germany (2007) 30–39
14. Moore, B., Ellesson, E., Strassner, J., Westerinen, A.: Policy core information model (RFC-3060) (February 2001)
15. Moore, B.: Policy core information model (PCIM) extensions (RFC-3460) (January 2003)
16. NIST: Role based access control. http://csrc.nist.gov/groups/SNS/rbac/
17. Loscocco, P.A., Smalley, S.D., Muckelbauer, P.A., Taylor, R.C., Turner, S.J., Farrell, J.F.: The inevitability of failure: The flawed assumption of security in modern computing environments. In: National Information Systems Security Conf., Crystal City (VA, USA) (1998) 303–314
18. SANS: The SANS Security Policy Project. http://www.sans.org/resources/policies/
19. POSITIF Consortium: The POSITIF system description language (P-SDL). http://www.positif.org/ (2007)
20. Clark&Parsia: Pellet: The open source OWL DL reasoner. http://clarkparsia.com/pellet
21. Clark, K.G., Feigenbaum, L., Torres, E.: SPARQL protocol for RDF. http://www.w3.org/TR/rdf-sparql-protocol/
22. HP-Labs: Jena a semantic web framework for java. http://jena.sourceforge.net/
23. OASIS: Core and hierarchical role based access control (RBAC) profile of XACML v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf
24. Smith, M.K., Welty, C., McGuinness, D.L.: OWL web ontology language guide. http://www.w3.org/TR/owl-guide/ (2004)
25. Stanford: Protégé. http://protege.stanford.edu/
26. Bechhofer, S.: The DIG description logic interface: DIG/1.0