A novel approach for integrating security policy enforcement with dynamic network virtualization

A novel approach for integrating security policy enforcement with dynamic network virtualization

Cataldo Basile, Antonio Lioy, Christian Pitscheider, Fulvio Valenza and Marco Vallini
Politecnico di Torino, Dip. Automatica e Informatica, Torino, Italy
(e-mail: {cataldo.basile, antonio.lioy, christian.pitscheider, fulvio.valenza, marco.vallini}@polito.it).

***Abstract*—Network function virtualization (NFV) is a new networking paradigm that virtualizes single network functions. NFV introduces several advantages compared to classical approaches, such as the dynamic provisioning of functionality or the implementation of scalable and reliable services (e.g., adding a new instance to support demands). NFV also allows the deployment of security controls, like firewalls or VPN gateways, as virtualized network functions. However, currently there is not an automatic way to select the security functions to enable and to configure the selected ones according to a set of user's security requirements. This paper presents a first approach towards the integration of network and security policy management into the NFV framework. By adding to the NFV architecture a new software component, the Policy Manager, we provide NFV with an easy and effective way for users to specify their security requirements and a process that hides all the details of the correct deployment and configuration of security functions. To perform its tasks, the Policy Manager uses policy refinement techniques.**

I. Introduction

Computer networks continue to grow in size and importance therefore the reduction of deployment and (re)configuration times has become critical. Nowadays, computing services are deployed on virtualized infrastructures but network and security functions still run on dedicated hardware. Network Functions Virtualization (NFV) tries to solve this disadvantage by defining a virtualized infrastructure for network functions. These functions, named Virtual Network Functions (VNF) are implemented as virtual machines, therefore can be dynamically added and removed on-demand reducing administration tasks, response times and costs. Recent works on NFV proposed to allow each tenant to customize its network infrastructure by inserting custom functions. This possibility enables the deployment of security functions as well (such as firewall, logging, proxies, VPN concentrators).

Adopting this NFV-based approach also for managing security functions may drastically reduce the deployment time and costs, but the increase in the overall management complexity may strongly reduce its impact. Therefore, automating the support of security functions is fundamental. This paper modifies the NFV architecture addressing the following challenges: (1) identify security functions to provision, (2) decide where selected security functions will be deployed, and (3) generate the necessary configurations to implement a set of user-defined security policies. At last, these features must be offered to end-users, with low technical skills, and to expert users that typically have specific needs (e.g., configuration tuning).

This paper adds to the NFV architecture a new software component, the Policy Manager. By using a user-oriented approach to specify security requirements, Policy Manager implements an automatic process for generation and deployment of related configurations. Although the proposed approach is suitable for every type of security features, this paper is focused on functions for filtering traffic (including stateful packet filtering, traffic inspection, etc.). These features can be joined to offer an integrated solution, e.g., parental control.

The Policy Manager acts as an intermediary between users and NFV, offering two interfaces. The user's interface supports the definition of the policies and the NFV interface sends configuration commands to the NFV orchestrator. The Policy Manager uses policy refinement techniques to select the VNFs to use and derive their configurations. *Policy refinement* is the process "to determine the resources needed to satisfy policy requirements, to translate high-level policies into operational policies that may be enforced by the system, to verify that the set of lower level policies actually meets the requirements of the high-level policy" [1]. Policy refinement allows the separation of policy specification and VNF configuration. By providing a high-level view of the desired behaviour of the system, the user is not distracted by specific details of a function implementation and can focus on the desired outcome. Policy refinement is well studied in literature for legacy systems to translate user policies into configuration commands, but, to the best of our knowledge, it has never been proposed to configure NFV security functions.

The High-Level Policies language (HLP) is used to capture user's security requirements. HLP is an authorization language whose policies can be represented with sentences close to natural language. Examples of HLP policies are "do not download malware", "do not access blacklisted site". As evident, HLP policies are technology, function and implementation independent.

The Policy Manager refines HLP policies into the concrete configuration of each VNF. Practically, it identifies the VNFs to enforce the user security requirements and derives configurations. When more VNFs are available for satisfy the same requirement (e.g., different types of firewall), an optimization to choose among VNFs will take place. Finally, configuration are passed by the orchestrator to the correct security VNFs. As alternative, VNFs are selected by the user. To avoid errors, the Policy Manager performs non-enforceability analysis to check compatibility between user's requirements and selected VNFs. This analysis informs the user providing indications and remediation tips.

To make this complex refinement feasible, our approach divides the process into two steps by passing through an intermediate format, the medium-level policies (MLP). We

define three policy abstraction layers (HLP, MLP and the concrete VNF configurations), and two translation modules. The former translator refines HLP into MLP and the latter translates MLP into concrete VNF configurations.

The rest of this paper is structured as follows: Section II presents the most important concepts used in this paper; Section III gives a general overview of the proposed approach; Section IV describes the refinement process in detail; Section V summarizes the paper.

II. Background

This section gives a short introduction of NFV, policy refinement, and non-enforcability. This introduction may help the reader for better understanding the proposed architecture. Furthermore a general overview of the research results in this areas is presented.

A. Network function virtualization NFV

The new Network Function Virtualization (NFV) concept decouples software implementation of network functions (e.g., router, firewall, NAT) from the compute, storage and networking resources through a virtualization layer. In this context, the ETSI standards organization is working on the definition of an architecture and the requirements for the deployment of any Virtual Network Functions (VNFs). Those VNFs can run on a range of industry standard server hardware, and can be moved and instantiated in any locations in the network, without the need of new equipment installation [2].

Recently a new kind of flexibility is achieved both from the Network Service Provider (NSP) and end-users sides through this new technology, as users traffic can be processed by any NSP-provided VNFs as well by any third-parties VNFs. Unfortunately, even if a great number of research activities analysed in depth the deployment of a generic VNFs in the NSP network [3], [4], at the best of our knowledge, none seems to address which effects could have the virtualization of a network security functions, (i.e., firewall, IDS, etc.) and which aspects must be taken into account.

Other research focuses on function description and how to correctly integrate third-parties VNFs in the NSP network: Koslovski et al. [5] proposes a language to describe storage and computing resources for a given function; wears Spinoso et al. [6] proposes that a VNF programmer must provide a functional description in order to correctly integrate and configure such VNF. However more detailed informations are required to support third-parties security functions in the provider network.

B. Refinement

Policy refinement has been well studied in literature and has been proven to be a mature and efficient process. Although policy refinement can be applied to all kind of policies, this paper only considers security policies.

Bartal et al. propose a solution named Firmato [7], it was one of the first solution proposals in this area and supports only packet filter firewalls. It is based on a entity-relationship model of the security policy and of the network topology. Verma et al. [8] used a similar approach, the authors present a firewall analysis and configuration engine named FACE. It takes as inputs the network topology and a global security policy written in a high-level language. Garcia-Alfaro et al. [9] proposed MIRAGE, a management tool for the analysis and deployment of configuration policies. It is based on the same principles as Firmato [7] and FACE [8], but it is also capable of configuring intrusion detection systems IDS and VPN routers. MIRAGE can also perform conflict analysis on already deployed configurations. Basile et al. [10] proposes to use ontologies for security policy translation. Network filtering rules are derived from security policies that refer to high-level concepts such as users and services. To map these high-level concepts to low-level network concepts such as IP address, port, protocol, an ontology is used. In [11], Guarnieri et al. proposed a model-driven security approach for the design and generation of concrete security configurations for software architectures. In this approach the system architect models the architecture of the system by means of UML class diagrams, and then the security administrator adds security requirements to the model by means of *Security4UML*, a UML profile. From the model enriched with security requirements, the concrete security configuration is derived in a semi-automated way.

According to [12], a good security policy must be *implementable* through system administration procedures (e.g., publishing of acceptable use guidelines) and *enforceable* with security tools or controls, where appropriate, and with sanctions, where actual prevention is not technically feasible. However in a real scenario, some policies may be less precisely enforceable on some systems than others or in worst case, completely non-enforceable. Unfortunately, in literature, non-enforceability analysis has received little or no attention and it has not been investigated in-depth. For example, as suggested by [13], the access control on traditional UNIX systems is much less granular when compared with ACLs on modern implementations and some policies are not fully supported. In particular, two situations can be detected: high-level constrains require a set of functions that are not available (non-enforceable) or only a subset of them is available (partial enforceable). Therefore the policy should be accompanied by an indication of how to handle these situations, e.g., warning the user, suggesting a more relaxed policy, adding a third-party software or install a different VNF to compensate the absent functionality. Verma et al. in [8] propose an iterative process (that includes topological analysis) to identify an unimplementable policy and suggesting how to make it implementable.

III. Approach

The proposed approach modifies the NFV architecture adding a new component named *Policy Manager*. The integration of this component with NFV architecture is sketched in Fig. 1, where *Policy Manager* transparently enforces user security requirements in agreement with the other network requirements, providing an additional layer between the end-user and the NFV orchestrator. The Policy Manager performs the refinement of security requirements expressed with a high-level policies (HLP) into the concrete configuration of each VNF. Practically, the Policy Manager first identifies the security VNFs that can be used to enforce the user security requirements then derives the needed configurations. These configurations are later passed by the orchestrator to the correct security VNFs. By separating the security requirements from the effective required VNFs and their security configurations

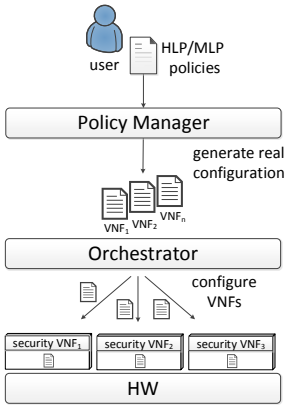

Fig. 1: Architecture

the end-user does not need to take in consideration the aspects related to VNFs configuration, focusing on the overall impact of his policy. To make this refinement feasible, we split the process into two steps by using a intermediate format, the medium-level policies (MLP). Therefore, the Policy Manager adopts three policy abstraction layers (i.e., HLP, MLP and the concrete VNF configurations) and two translation modules (to refine HLP into MLP and to translate MLP into concrete VNF configurations). It is worth noting that our proposal follows the design principles proposed by Strassner for policy-based network management, where the HLP maps to "Business/System View"-layer, the MLP maps to the "Administrative View"-layer and the concrete configurations map to the "Device/Instance View"-layer [14]. By introducing HLP and MLP in the architecture, the refinement process becomes independent from VNF implementations.

A. *Policy abstractions.*

Although the full specification of policy abstraction is out of the scope of this paper, in this section we provide a brief introduction of HLP and MLP. As introduced before, the users specify their security requirements with the HLP. We designed the HLP (starting from our previous works [15], [16]) as an authorization language that follows the subject-action-object-attribute paradigm (also referred to as target-effect-condition) [17]. A security requirement is expressed as a set of sentences close to natural language, e.g., "do not download malware", "do not access gambling sites", "allow Internet traffic from 18:30 to 20:00 for Alice". The elements of a sentence (subject, object, etc.) are chosen by the user from a predefined set and implemented in a GUI editor as different lists, i.e., a list for each element (e.g., action, subject). This approach is transparent for users (avoiding to learn new language) and makes it possible to map each element of a sentence to the related HLP component. It is clear that, users can customize some elements of a sentence, for example to define timing constraints, particular URL, etc.. Again, to simplify the definition of a complex security policy, a template-based approach is provided. A template contains a set of HLP that participate to a common goal. For example, the template "enable parental control" implements simple HLPs as "do not access blacklisted site", "log access to websites" and "permit access to Internet from 20:00 to 22:00". Elements as "blacklisted site" contains a predefined set of URLs initially collected from a list managed by a trusted authority. However, the user can modify that list adding or removing some URLs. As a consequence, HLP policies are technology, function and implementation independent, therefore a HLP can be enforced with different VNFs of different vendors.

MLP has been designed to abstract the configurations of security VNFs. Unfortunately, defining this abstraction is not trivial because each security control has a specific language. To this purpose, MLP follows the approach of [18] and organized by security functions. A security function is a basic feature[1] offered by a VNF (e.g., channel protection, filtering, anti-virus, parental control). Therefore, MLP is composed by a general model that defines the high-level concepts (policies, rules, conditions, actions, etc.) and a set of sub-models to capture the semantics specific concepts as attributes, condition types, methods (e.g., HTTP GET), etc. For instance, MLP supports the configuration of a packet filter, or the options related to the configuration of an anti-virus. Expert users may have specific needs (e.g., fine tuning) for the configuration of security features. To satisfy these requirements, users may directly use statements offered by MLP to write abstract configurations. After, those are passed as input to the Policy Manager as depicted in Fig. 1.

B. *Translation.*

The refinement of HLP policies into MLP policies is a very complex task. First of all, it requires to identify the security functions (i.e., capability) needed to enforce the policy. Then, suitable VNFs to enforce the policy must be selected. However, several VNFs with the same security function are typically available. Therefore, the process must choose among them. Different implementations and combinations of VNFs may have different side-effects on the overall performance, throughput, latency and/or bandwidth. For example, a particular VNF implementation requires more processing resources than the others, or significantly reduces the network throughput. For this reason, the Policy Manager adopts a set of optimization techniques (as presented in Section IV) to choose among alternatives. When the set of optimal VNFs is identified, the HLP are mapped into MLP statements. For example, "allow web traffic" is easily translated into a rule whose action is allow and the condition selects all the IP traffic towards the destination port 80. In the same way, other concepts are expanded with predefined values, like "gambling sites" that can be determined by a set of URLs (also maintained and obtained by third parties) or by a set of DNS servers that do not perform reverse translation of specific URLs (like OpenDNS).

On the other hand, the transformation of MLP policies into VNF configurations mainly involves a change of syntax, as MLP has been designed to share the same semantics as the VNFs. Each VNF implementation typically has a different configuration language and VNF-specific translation module is needed. This actually maps MLP policies into a concrete configuration. For example, the refinement process requires a

[1] A VNF may implement more than a single security feature, e.g., a firewall can implement at the same time stateless packet filter and stateful filtering. The proposed refinement process also support this case. However, for simplicity, we avoid to explicitly distinguish these cases in this paper.

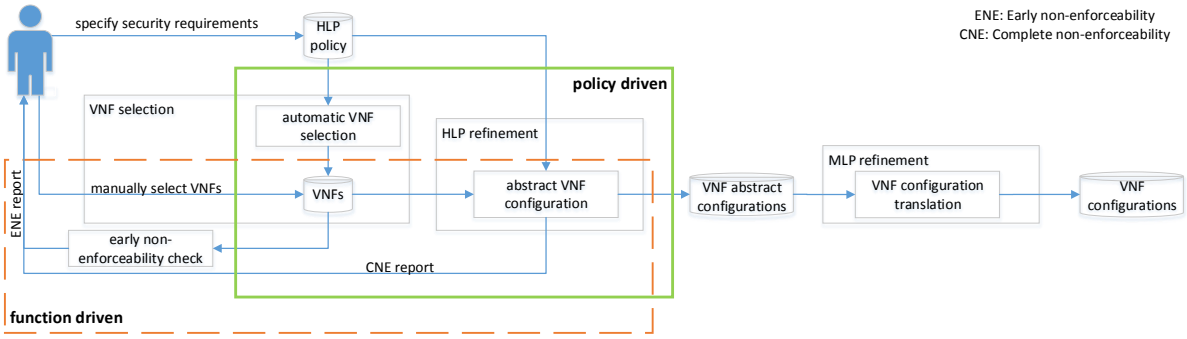

Fig. 2: Policy-driven and Function-driven VNFs selection

firewall VNF and generates the corresponding MLP. Then, the translation module transforms the MLP configuration into the firewall settings.

IV. HLP Refinement

The HLP refinement approach is performed by the Policy Manager and presented in Fig. 2. The VNFs can be manually selected by the user, or automatically by using an ad hoc process. The former is named *function-driven* VNFs selection, the latter *policy-driven* VNFs selection. As introduced before, different VNFs provide different security functions. Examples of such functions are "packet filtering", "deep packet inspection", "signature-based malware detection", "traffic anonymization", "traffic encryption" and "transparent proxying". For instance, the HLP policy "enable parental control" can be implemented by using either a single VNF (that contains all the required security functions) or by using a set of VNFs (e.g., a virtual packet filter, a VNF for logging traffic/sessions, a virtual web proxy). This choice may impact on performance, cost and/or efficiency.

In the function-driven approach (depicted and surrounded by the dashed line in Fig. 2), the user specifies her/his HLP policy and selects the set of VNFs she/he wants to use. Moreover, the user has the opportunity to decide which policy should be enforced with a particular VNF. Before starting the generation of MLP, the Policy Manager checks if the required VNFs support the security functions required to enforce the user's policies. In practice, actions, objects and attributes are statically mapped to a set of security functions that are required to enforce an HLP. Then, each VNF supports a subset of that functions. Therefore, starting from an HLP it is possible to identify which security functions are required and which VNFs satisfy that policy. If the selected VNFs do not satisfy these requirements, the user is warned and a set of remediations strategies is proposed. This step is named *early non-enforceability*.

The *early non-enforceability* analysis is performed in real-time to identify only the macroscopic errors that lead the refinement process to failure. For example, this analysis is useful when the user specifies a parental control policy but the selected VNFs does not support this security function. In this case, the refinement is aborted after the analysis. If the early enforceability does not detect any lack of functions, the generation of MLP is automatically performed for each security control of a VNF(s) (the "abstract VNF configuration phase" in Fig. 2). During the generation of MLP other cases of non-enforceability may appear. For example, when a policy requires to inspect the content of a HTTP protocol field, but the selected VNF does not support this feature, the policy is not enforceable. Similarly, when it does not support a particular option, the policy is partially enforceable. Let us consider a parental control scenario to protect children access to Internet, where applications with different features are available. Two distinct VNFs, VNF_1 and VNF_2 are available, both capable of enforcing a parental control policy but with different functions. VNF_1 includes a "application content inspection" function and a "URL filtering" function. VNF_2 supports the functions of VNF_1 and a feature to specify time-based policies for "URL filtering". Hence, if a user wants to specify that access to Facebook web site is permitted only after dinner from 20 pm to 22 pm VNF_1, he is not allowed to enforce that policy. Therefore, the abstract VNF configuration phase produces a *complete non-enforceability report* (CNE report) where all types of enforceability errors/issues are shown to the user. The function-driven approach is recommended only for expert users, as it can lead to several issues, e.g., sub-optimal configurations, lack of performance, costs, non-enforceability. Each VNF specifies the set of available functions (e.g., "application content inspection", "URL filtering"), the supported features (e.g., user's defined set of URL, time-based policies) and other tuning options. A unskilled user could select a VNF that does not completely satisfy the required network throughput requirements, or in the worst case cannot satisfy the security policy at all. Therefore a wrong VNF selection leads to a non-enforceable security policy.

The policy-driven approach (depicted and surrounded by a continuous line in Fig. 2) selects the required VNF automatically from a catalogue of available VNFs. Each VNF available in the catalogue is associated to a set of security functions. Therefore, when high-level policy requirements matched the related functions, a set of candidate VNFs is selected. The selection can be straightforward (when only one VNF is available with a required function) or may be based on various criteria (such as cost, performance, reliability or reputation) when multiple VNFs offer a required function. This may result in a trade-off among different criteria and the user must specify its preferences. Examples of these criteria are:

adopt open-source VNF; choose applications with low network latency (e.g., to match QoS requirements); adopt applications that are reliable to faults or that have a better reputation (according to an expert review). Since several VNFs may be identified to enforce the policies a selection criterion (i.e., optimization target functions) must be defined. The user may choose among a set of Policy Manager-provided profiles that specify a predefined set of target functions (e.g., maximize performance, minimize costs). In particular, for performance a set of different categories should be considered: e.g., CPU usage, RAM, network throughput. Once a profile or a criterion is selected by the user, the refinement process: formulates a Mixed integer linear programming (MILP) problem (considering a specific target functions and related constraints derived from selected profile), invokes an external solver to perform the optimization, analyses solver results and identifies the set of VNFs to adopt. For the sake of simplicity, we avoid to present full details on how an optimization problem is formulated.

V. Conclusions

The innovations in NFV made it possible to deploy complex network structures based on virtualized functions with a reduced cost and time. Although the infrastructure is flexible enough to accommodate this improvements and to configure the interconnections between the VNFs, there is no efficient method to select and configure security functions within a dynamic virtualized network. This paper proposes a novel approach to solve this problem by defining an extension, named Policy Manager, for the existing NFV architecture. The Policy Manager introduces an additional layer between the user and the NFV orchestrator. The user defines his security policies with the High-level Policy language (HLP) and the Policy manager refines them into configurations for the required VNFs. The required VNFs are selected either manually by the expert users (function-driven) or automatically by the Policy Manager (policy-driven) for the end-users. The policy-driven approach uses a selection criteria defined by the end-user to find the best possible combination of VNFs. In the function-driven approach the expert users selects his desired VNFs and also specifies which VNF enforces which policy. Both approaches perform an enforceability analysis and warn the user in case of some policies cannot be enforced. The proposed extension has mayor advantages over the current architecture. First, it enables end-users, with low technical skills, to configure the network and related services. Second, the policy definition is independent from VNF implementations and therefore one VNF can be substituted with another without reconfiguring the whole network.

Currently, our approach has been implemented only for a limited set of HLP policies, mainly related to filtering requirements, and only for a very limited set of VNF (packet filters, stateful firewalls, L7 filters, basic content inspection). However the proposed approach can be easily extended, adding new security feature and/or VNFs. Therefore, as future work, we will extend the Policy Manager adding other types of security functions (e.g., VPN, proxy, IPS/IDS) and supporting more VNFs. Other improvements are expected in the optimization process used in the policy-driven approach with the support of more multi-objective target functions.

Acknowledgment

The research described in this paper is part of the SECURED project, co-funded by the European Commission (FP7 grant agreement no. 611458).

References

[1] J. Moffett and M. Sloman, "Policy hierarchies for distributed systems management," *Selected Areas in Communications, IEEE Journal on*, vol. 11, no. 9, pp. 1404–1414, December 1993.

[2] "Network function virtualization - White Paper 2," The European Telecommunications Standards Institute, Tech. Rep., October 2013.

[3] J. Martins, M. Ahmed, C. Raiciu, V. Olteanu, M. Honda, R. Bifulco, and F. Huici, "Clickos and the art of network function virtualization," in *NSDI 14: 11th USENIX Symposium on Networked Systems Design and Implementation*, Seattle, Washington, April 2014, pp. 459–473.

[4] M. Scholler, M. Stiemerling, A. Ripke, and R. Bless, "Resilient deployment of virtual network functions," in *ICUMT13: 5th International Congress and Workshops on Ultra Modern Telecommunications and Control Systems*, Brno, Czech Republic, September 2013, pp. 208–214.

[5] G. P. Koslovski, P. V. Primet, and A. S. Charão, "VXDL: virtual resources and interconnection networks description language," *Networks for Grid Applications*, vol. 2, pp. 138–154, October 2009.

[6] S. Spinoso, M. Leogrande, F. Risso, R. Sisto, and S. Signgh, "Automatic configuration of opaque network functions in cms," in *NVSDN 2014: 1st International Workshop on Network Virtualization and Software-Defined Networks for Cloud Data Centres*, London, United Kingdom, December 2014, pp. 750–755.

[7] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," *ACM Transactions on Computer Systems*, vol. 22, no. 4, pp. 381–420, November 2004.

[8] P. Verma and A. Prakash, "FACE: A Firewall Analysis and Configuration Engine," in *SAINT05: Symposium on Applications and the Internet*, Trento, Italy, February 2005, pp. 74–81.

[9] J. Garcia-Alfaro, F. Cuppens, N. Cuppens-Boulahia, and S. Preda, "Mirage: A management tool for the analysis and deployment of network security policies," *Data Privacy Management and Autonomous Spontaneous Security*, vol. 6514, pp. 203–215, 2011.

[10] C. Basile, A. Lioy, S. Scozzi, and M. Vallini, "Ontology-based security policy translation," *Journal of Information Assurance and Security*, vol. 5, no. 1, pp. 437–445, 2010.

[11] M. A. Neri, M. Guarnieri, E. Magri, S. Mutti, and S. Paraboschi, "A model-driven approach for securing software architectures." in *SECRYPT2013: 10th International Conference on Security and Cryptography*, Reykjavík,Iceland, July 2013, pp. 595–602.

[12] J. Weise and C. R. Martin, "Developing a Security Policy," SANS Institute, Tech. Rep., April 2003.

[13] M. Bishop and S. Peisert, "Your Security Policy is *What*??" University of California, Tech. Rep., March 2006.

[14] J. Strassner, "DEN-ng: achieving business-driven network management," in *NOMS2002 : Network Operations and Management Symposium*, Florence, Italy, April 2002, pp. 753 – 7661.

[15] "PoSecCo deliverable D2.2 - it policy meta-model and language," February 2013. [Online]. Available: http://www.posecco.eu/fileadmin/POSECCO/user_upload/deliverables/d22.pdf

[16] "PoSecCo deliverable D3.5 - models to refine the it policy at service level," September 2012. [Online]. Available: http://www.posecco.eu/fileadmin/POSECCO/user_upload/deliverables/D3.5_Models_to_refine_the_IT_policy_at_service_level_01.pdf

[17] S. Godik, A. Anderson, B. Parducci, E. Damiani, P. Samarati, P. Humenn, and S. Vajjhala, "eXtensible Access Control Markup Language (XACML) Version 3.0," Organization for the Advancement of Structured Information Standards, Tech. Rep., January 2013.

[18] "PoSecCo deliverable D3.3 - configuration meta-model," March 2012. [Online]. Available: http://posecco.eu/fileadmin/POSECCO/user_upload/deliverables/D3.3_Configuration_Meta-Model.pdf