A Holistic Approach to Security Policies – Policy Distribution with XACML over COPS

Jan Peters[a] Roland Rieke[b] Taufiq Rochaeli[c]
Björn Steinemann[b] Ruben Wolf[b]

[a] *Fraunhofer Institute for Computer Graphics Research IGD, Germany*

[b] *Fraunhofer Institute for Secure Information Technology SIT, Germany*

[c] *Technical University of Darmstadt, Department of Computer Science, IT-Security group, Germany*

Abstract

The potentials of modern information technology can only be exploited, if the underlying infrastructure and the applied applications sufficiently take into account all aspects of IT security. This paper presents the platform architecture of the SicAri project, which aims to build a security platform for ubiquitous Internet usage, and gives an overview of the implicitly and explicitly used security mechanisms to enable access control for service oriented applications in distributed environments. The paper will introduce the security policy integration concept with a special focus on distribution of security policies within the service infrastructure for transparent policy enforcement. We describe in details our extensions of the COPS protocol to transport XACML payload for security policy distribution and policy decision requests/responses.

Keywords: service platform, security policies, policy distribution, policy decision, policy enforcement, Web services, XACML, COPS.

1 Introduction

Professional usage of today's communication and collaboration infrastructures requires the consideration of appropriate security measures. In this paper, we introduce the SicAri [4] project – an interdisciplinary approach to information security. The project covers technical, cryptographic and usability issues, as well as various legal issues in information security with respect to legislation and jurisdiction. The overall goal is the conception and realization of a Java-based security platform and its tools for ubiquitous Internet usage.

This platform supplies a bunch of applications and provides various security services to the user in a transparent, seamless and integrated way. It is a modular and integrative platform that allows the connection of various end user devices, such as PCs, PDAs, and ambient intelligence devices and gives support for various network types (e. g., wired and wireless networks) and communication paradigms

This paper is electronically published in
Electronic Notes in Theoretical Computer Science
URL: `www.elsevier.nl/locate/entcs`

(e. g., client-server, peer-to-peer or ad-hoc networks). The behavior of the platform in terms of security related action can be determined by security policies. They provide a well-understood and suitable means to administer security issues. Such policies allow to separate the administration, decision finding, and enforcement of access control. But using policies raise additional questions in distributed environments where applications, services and nodes dynamically join and leave the system. The distribution of policies, especially at bootstrapping time, and their update and synchronization process has to be particularly considered. While open standards and open source solutions for single isolated tasks around security policies exist we have not been aware of any open holistic approach to them.

This paper first presents the SicAri platform from a conceptual, an architectural, and partially from an implementation point of view, but the main focus is on our distribution model of the underlying platform security policy which is based on the *Common Open Policy Service (COPS)* protocol. We extended the COPS protocol with a client type specification to transport *eXtensible Access Control Markup Language (XACML)* policies.

The subsequent paper is structured as follows. Section 2 gives an overview of the layers and the components of the platform. The holistic security policy approach is described in Section 3, while Section 4 presents the policy processing including policy enforcement, policy decision and policy distribution. Section 5 considers some related work. Finally, the paper ends with an outlook in Section 6.

2 The SicAri Platform

The main function of the *SicAri platform* is to provide interfaces to the user's application to access the services provided by the platform, which are basic services and application services (see below). Together with the middleware, the platform also provides the communication infrastructure between distributed components. This is realized through a consistent service management comprising service discovery, service description, and service invocation. If desired, local services are automatically provided as Web services to remote platforms during runtime, which enables the interoperability of our service platform with other service-oriented architectures.

In case of new users requiring new application services, the platform architecture supports modular and extensible building blocks, adding the possibility to incorporate new services. Therefore, reusability of existing building blocks should be as easy as possible for developers.

All security related aspects of a platform instance are regulated by a security policy. In the case of multiple platforms interacting with each other, all platforms running in the same *SicAri infrastructure* share the same security policy. Each platform has its own policy enforcement component. Access attempts to local or remote services and resources are checked against the security policy considering the requester's current session ID with respect to a single-sign-on mechanism, activated roles, the available permissions, and other parameters. In addition to policy decision and enforcement, the platform further covers the aspects of policy generation, administration, and validation (cf. Section 3).

Thereby, the platform's architecture features a holistic approach to security policies based on current standards and the support of implicit and explicit security mechanisms in heterogeneous and distributed service infrastructures. Mobile devices can easily be connected to this infrastructure directly, when running the SicAri middleware, resp. through security-aware gateway components and specific protocols, in case the full platform does not execute on the mobile device due to resource limitations.

In spite of the platform features that are mentioned above, the platform is not intended to completely replace the existing information infrastructure nor its existing security mechanism, such as firewall, etc. The platform rather gives the opportunity of building distributed and security-centered applications on top of its service infrastructure, which can easily be extended or be plugged into existing infrastructures.

2.1 Platform Architecture

Figure 1 (left-hand side) gives a high-level overview of the generic service architecture. On top of the middleware, there is a service and application layer. A locally authenticated user can directly interact with applications defined on this layer. The applications themselves make use of the basic and application-specific services of the platform's service layer. On the one hand, these services integrate external databases, legacy systems and applications; on the other hand, they provide basic security mechanisms, such as authentication or access control. The middleware layer is responsible for the secure and seamless integration and communication of applications and services, locally and remote.

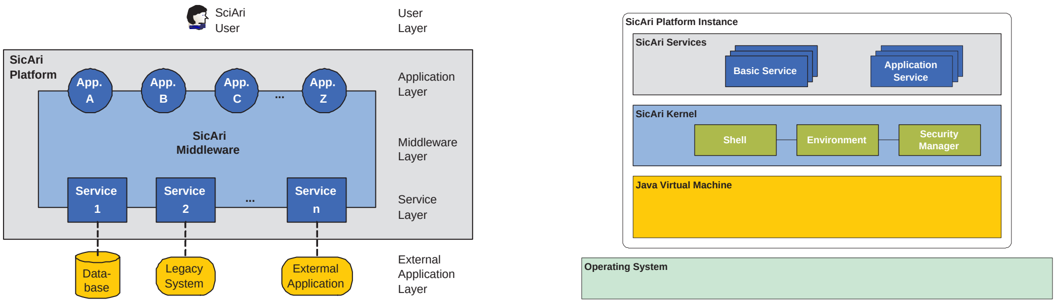


Fig. 1. High-level and layered architecture of the SicAri platform

The platform is based on the *SicAri kernel* (see right-hand side of Figure 1) which runs on top of a Java Virtual Machine and thereby attracts a broad spectrum of potential users and of potential devices, ranging from mobile devices and personal computers to scalable, distributed environments.

The user application that runs on the platform accesses obligatory basic services and optional application services provided by the platform. The entirety of installed services thereby constitutes the characteristic and specific use case of the local platform instance.

Thus, this platform is rather a collection of services which are flexibly loaded and configured within a common environment on top of a Java Virtual Machine, than a monolithic system designed as one static piece of software. Due to this architecture the platform offers the following advantages: minimalist design, modularity and reusability of services, extensibility, maintainability and intrinsic security features of the kernel.

2.2 Platform Components

This sections briefly describes the basic components of the architecture.

Environment. The environment is a hierarchical name space for service object instances. Object instances can be registered, looked up, searched for, and removed. This mechanism for local service management and service discovery is extended by the transparent use of Web services for platform communication among distributed platforms. Services registered within this environment are implicitly encapsulated by a security proxy which enforces the current security policy on a search resp. access request to a service object.

Shell. The shell is a user interface for the environment. It allows administration of and access to the environment. Hence, it supports at least the afore mentioned operations *register*, *lookup*, *search*, and *remove* of object instances. Furthermore it allows navigation in the name space and invoking of Java methods. The SicAri shell compares to a UNIX Shell, where the environment stands for the file systems and services can be compared with files.

Security Manager. The security manager is responsible for policy enforcement. The Java programming language provides a standard security manager [9] which is accessible via an API. SicAri replaces this security manager with an own implementation.

Service. A service is a piece of software which fulfills a very specific task. It provides a small, well-defined programming interface. Typically there is no direct interaction between the user and a service. Every service is published as an object instance in the environment which allows access from other services and applications. Services are retrieved by means of the environment's search and lookup functionality. The service may be separated in a local access stub and one or more remote components which provide the functionality. Further, a service can integrate legacy applications and external data sources into the SicAri architecture, by means of a wrapper or proxy.

SicAri kernel. The SicAri kernel (or just *kernel*) consists of the Java implementations of the environment, the shell, and the SicAri security manager as defined above. Together they provide service bootstrapping and configuration, local service management (registration/searching), and a consistent security context by means of implicit access control.

SicAri platform. The platform consists of the kernel started on top of a Java Virtual Machine, a number of mandatory basic services, and optional application services. Any application can rely upon the availability of the basic services, as there are among others the authentication manager, the identity manager, the cryptographic key master, and the policy service.

SicAri infrastructure. The infrastructure is a compound of several platforms managed by the same security policy. These platforms may be distributed within the infrastructure.

SicAri application. An application is a software which fulfills a complex task. Since it usually interacts with the user, it provides both an interfaces for user-interaction and a programming interface. Applications make use of services in order to fulfill their tasks.

3 Holistic Approach to Security Policies

Policy-based control of networks and computer systems has the benefit that the controlling units of the system are kept decoupled from the management components and the rule base that governs the decisions. This enables the administrator to easily run, manage and change the system's behavior without having to modify the software or the controlled nodes. The system is controlled by policies that specify behavior rules which are interpreted by decision components and are asserted by enforcement components. Hence, if conditions change or new services or applications are added to the system one just adapts the policy rules. Using a central administration component the platform administrator does not have to deal with the multitude of different nodes in the system. This applies to network management issues, e. g. *Quality of Service (QoS)* or resource allocation, as well as to network and service security.

All security related tasks of the platform are controlled by a security policy. The platform covers various aspects of security policies, such as policy specification, policy patterns and policy compiler (see below), policy decision and enforcement, policy negotiation and provisioning, policy administration, and conflict resolution. Thereby, the security policy integration concept is based on manifold requirements with respect to policies, as for example:

- Control of all security related processes and tasks. Impossibility to bypass the policy enforcement component.
- Compatibility of the policy framework with the platform's plug-in approach. No need to change existing or upcoming services in order to enforce the platform's security policy. Transparency of policy control.
- Consideration of trade-off between expressiveness and complexity of the policy description language.
- Support for platform administrators during policy management.

It is another goal of the platform to bridge the gap between the informal specification of security policies (i. e., what the security administrator wants to enforce) and its corresponding machine-readable policy specification (i. e., what the system actually enforces).

3.1 Policy Architecture

The policy architecture comprises the components of the policy framework and their interactions in order to guarantee that all security relevant processes in the platform are fulfilled according to the underlying security policy. Access control policies are based on the *Role-based Access Control (RBAC)* standard. The general concepts of RBAC are well-understood and extensively described in the literature, please refer to [7,13,19]. RBAC is assumed to be policy-neutral. This means that RBAC provides a flexible means to deal with arbitrary security policies. The policy integration concept of the SicAri platform requires the interaction of various components. Our implementation of RBAC uses XACML [12] as its specification language.

XACML thereby serves as "glue" between a couple of policy components: Starting with the policy generation process which leads to XACML-based user-role assignments and XACML-based role-permission assignments, this XACML specification is used as basis for policy validation, afterwards. The validated policy then is distributed to resp. updated at all PDPs, and subsequently used for policy decisions.

Figure 2 depicts the component framework of the SicAri policy architecture. The remainder of this section gives a more detailed overview of all the components involved and their interactions with other components.

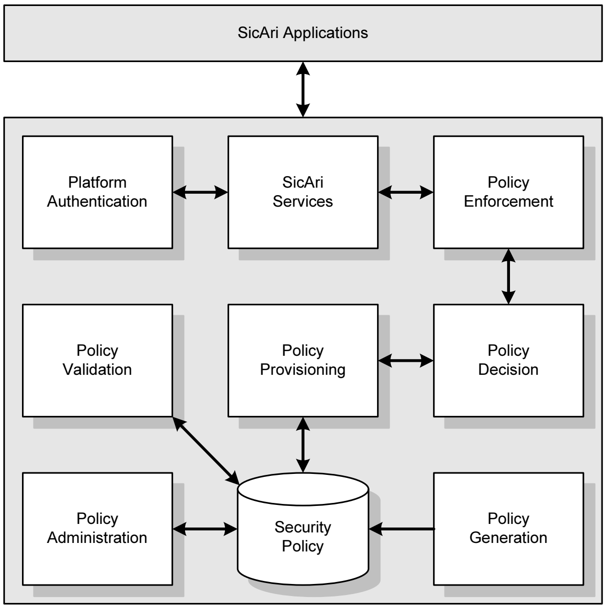


Fig. 2. Policy components of the SicAri platform

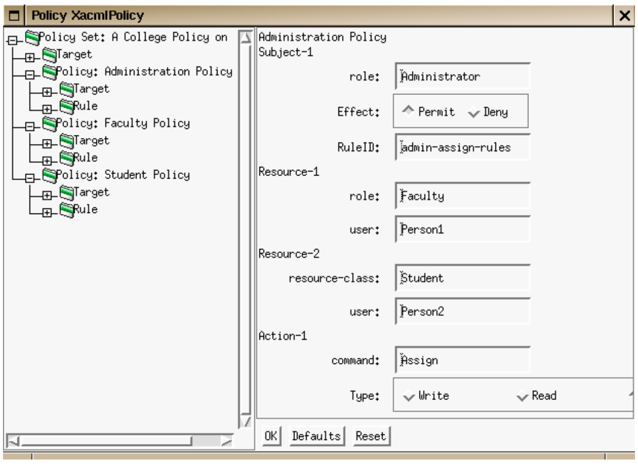
Policy Generation. We currently focus on generation of policies for access control to resources representing the required security properties. Such a policy is generated using *policy patterns* formalized in OWL [5] that allow to specify template policy archetypes for recurring security requirements. Policies are usually derived from security requirements of business or organizational goals. For example, the execution of the tasks "credit request" and "credit approval" with respect to a banking scenario require two different persons to give their affirmation to complete the task. This security requirement can be satisfied by introducing an assignment constraint in the model, for example, separation of duty.

Policy generation leads to a policy specification conforming with the RBAC profile of the XACML 2.0 standard [14].

Policy Validation The task of the policy validation component is to evaluate, whether a policy correctly implements given security goals. We extended the SH verification tool [15,16] to accept a subset of XACML as input and to translate it into transition patterns, which specify the behavior of *Asynchronous Product Automata (APA)*. APA are a class of general communicating automata and provide a means to model arbitrary distributed systems while transition patterns define the possible state transitions of the modeled system. Each policy rule is converted into such a transition pattern which then encodes the action that is controlled by that rule. This in turn results in an operational model of the policy system that can be executed in the SH verification tool. It allows to analyze the policy system's behavior, to simulate its potential information flow and to verify the wanted security goals. For that purpose the system's reachability graph is computed which spans all possible sequences of transition steps that are allowed by the given policy.

We have chosen to support a subset of XACML that comprises the most important elements and attributes of the language. One first goal was to reach the expressiveness that allows to handle one well-known XACML example which has been validated in the literature before [2,8]. Some concepts of XACML like obligations and rule combining algorithms are not yet supported.

Policy Administration. Even if the ability of automated security policy generation is provided by the platform, there may be the need of fine granular policy administration, e. g. a new user needs to be added, or the permissions of a user or role need to be changed. Therefore, we provide an administration API based on the RBAC standard and a corresponding graphical user interface (cf. Figure 3).



Administration Policy
Subject-1
role: Administrator
Effect: Permit Deny
RuleID: admin-assign-rules
Resource-1
role: Faculty
user: Person1
Resource-2
resource-class: Student
user: Person2
Action-1
command: Assign
Type: Write Read

Fig. 3. Policy administration

Policy Provisioning. The components for policy generation, validation, and administration mutually share access to the global security policy database containing the current XACML-based policy specification. This whole policy specification resp. changes of policy subsets are subsequently distributed by policy provisioning components. The component covers distribution of policies, policy updates, as well as transport of policy decision requests and responses. Policy provisioning is outlined in more detail in the next section.

Policy Decision. The policy decision component uses an extended version of SUN's reference implementation of an XACML evaluator [20]. It comprises of library classes that can be used in building a *Policy Enforcement Point (PEP)* or a *Pol-*

icy Decision Point (PDP). Since we use the RBAC profile of the XACML 2.0 standard to specify our policies, some modifications were necessary in order to evaluate these XACML policies.

As described above, the policies in form of an RBAC model are stored into three different categories, namely: *Role Policy Set (RPS)*, *Permission Policy Set (PPS)* and *Role Authorization (RA)*. The RPS and the PPS contain the roles definition and their corresponding permissions, respectively. Therefore, each RPS has a reference to the corresponding PPS. However, the implementation of SunXACML version 1.2 does not support references in policy. Therefore, we have extended this implementation accordingly to actually support policy references.

Policy Enforcement. The policy enforcement component assures that all security relevant tasks can only be fulfilled if they are in accordance with the underlying security policy. The policy enforcement component detects security relevant tasks, consults the policy decision component in order to decide upon a task, and enforces the policy decisions, i. e. allows a platform entity to access a platform resource or not.

SicAri Services. Interaction between applications and services and between one service and another as well as access of local resources within services is implicitly controlled by the policy enforcement component. Except the situation that a service wants to explicitly request the policy decision component, policy processing is done transparently during service execution. That is, SicAri services do not have to be aware of the existence of a security policy. As consequence, there is no need to modify or adapt existing or upcoming services to be compatible with the policy integration concept. The only thing that needs to be done by a security administrator is to configure resp. re-generate the security policy according to the security requirements in the context of new services integrated into the service infrastructure.

Platform Authentication. Finally and as another precondition for policy enforcement e. g. by means of access authorization, every acting entity in the service infrastructure has to be successfully authenticated. Thus, several authentication modules are provided locally on a platform instance to allow different user login procedures according to the specific use case and characteristic of the local platform instance.

4 Policy Distribution with COPS and XACML

This sections takes on the policy distribution issue from the introduction. The protocol framework for *Policy Based Network Management (PBN)* which has been defined by the *IETF Resource Allocation Protocol (RAP)* work group offers a good solution to those questions.

4.1 Policy Distribution with COPS

The core of the RAP framework is the COPS [6] protocol. It provides a means to communicate policies and policy decisions in a distributed system. The main characteristics are

(i) the logical and architectural separation of policy enforcement and policy decision components, (ii) a client/server model of PEP and PDP, (iii) reliable transport of messages between PEP and PDP via TCP, (iv) a flexible and extensible framework through self-identifying objects that allow to define arbitrary protocol payload, and (v) a stateful communication between PEP and PDP which share request/decision states that allow the PDP to asynchronously update decisions and configuration information at the PEP.

COPS is designed to be used in two basic scenarios – outsourcing and configuration. In the *configuration scenario* a *local Policy Decision Point (LPDP)* is available and in the *outsourcing scenario* there is none. In the first case the PEP asks the LPDP for local policy decisions and in the latter case the PEP delegates all policy decisions to the remote PDP. Since the configuration scenario has already been implemented in SicAri its concept is described in more detail in the next paragraph. Section 4.2 explains how both scenarios integrate into the platform architecture.

COPS in Configuration Mode

In configuration mode the PEP requisitions a whole configuration for a component. Because COPS is policy independent the configured component can be such different things as router hardware or Web services.

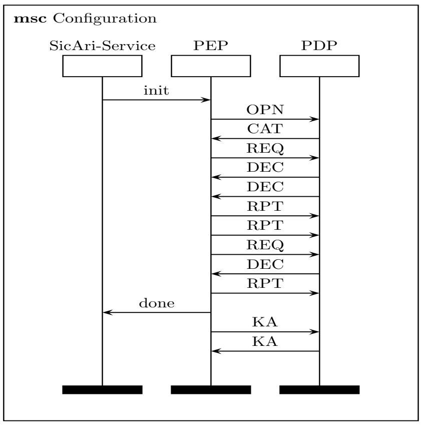


Fig. 4. Configuration request

Figure 4 shows the schematic sequence of the COPS configuration procedure in form of a Message Sequence Chart (MSC). When a service is started for the first time it contacts the PEP. The PEP sends a client open message (`OPN`) to the corresponding PDP. This message contains a unique ID that identifies the PEP to the PDP and it also contains a client specific information (`ClientSI`) object. This object is necessary to enable the PDP to relay the `OPN` message to a PDP module that can handle the requests for the incoming type of policy.

When the PDP is capable to serve the client type it answers with a client accept (`CAT`) message and expects incoming requests. In the configuration scenario the PEP sends one or more request messages (`REQ`) that contain context objects which identify the message as configuration requests. The request messages also comprise

ClientSI objects that carry client specific information on the requested configuration data.

Each configuration request may be answered with a single decision message (`DEC`) or a stream thereof. On reception and successful installation of the configuration data the PEP acknowledges this to the PDP with a report state message (`RPT`) for each of the `DEC` messages. When the PEP finally has received all configuration data from the PDP it signals the installation back to the SicAri service which now can rely on the LPDP to decide access requests.

From now on the PDP proactively keeps the policy at the PEP side up to date. Whenever a change to the master policy at the PDP side is made, it passes it on to all PEPs that make use of this policy. For that purpose both parties regularly exchange keep alive (`KA`) messages to assure that the PEP always uses a policy that is up to date

4.2 Platform Integration

This section describes how the two policy distribution approaches can be integrated into the platform architecture.

In contrast to the COPS specification, the definition of PEP and LPDP in SicAri are slightly different: Whereas the PEP in COPS is defined as a local client component communicating with a global PDP, in the COPS policy configuration scenario this component corresponds best to the LPDP, as defined in SicAri (cf. Figure 4 vs. Figure 5 (a)).

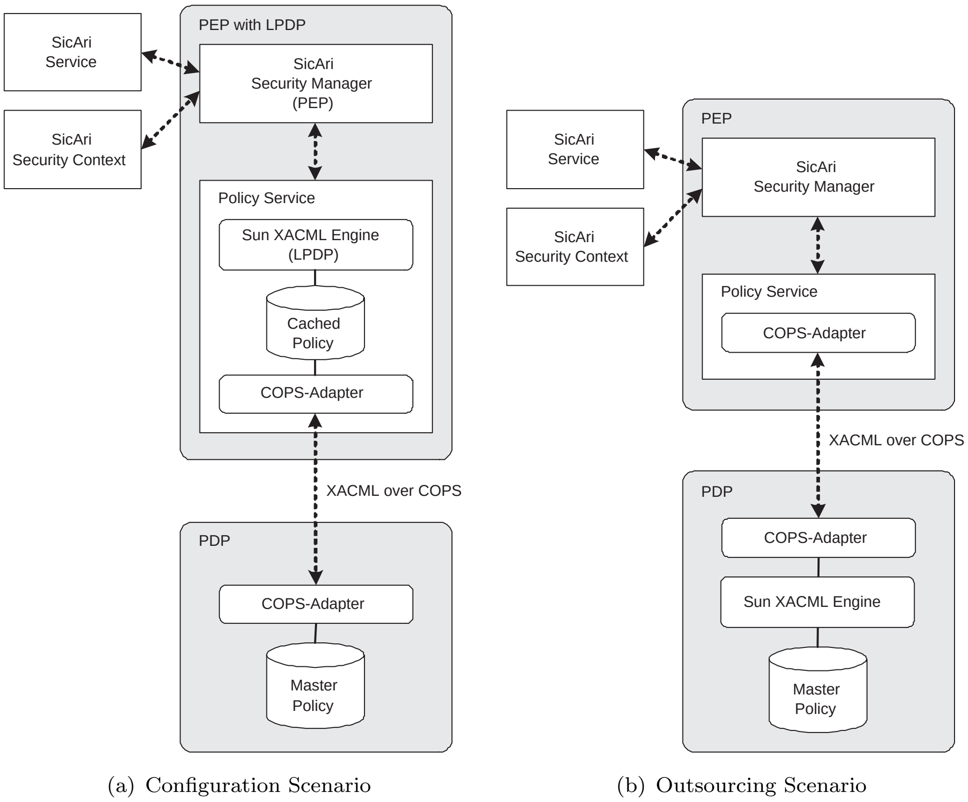
COPS Policy Configuration

The main characteristics of this scenario are the local PEP and LPDP (cf. Figure 5 (a)). The PEP interacts with the local policy service, which mainly consists of the following components: LPDP, cached policy, and COPS adapter. The LPDP is responsible for making policy decisions based on the input from the SicAri security manager (PEP) and a locally cached version of the master security policy. The LPDP is realized by an extended version of Sun's XACML reference implementation (see below). The PEP uses the Java-API of Sun's XACML engine in order to communicate with the LPDP. A (potentially remote) policy provisioning component provides a copy of the latest master policy to the LPDP using the COPS protocol.

COPS Policy Outsourcing

The main characteristic of the second scenario is that a local PEP delegates all policy decisions to a remote PDP. This scenario is not yet implemented.

The policy service mainly consists of a COPS adapter which transforms the policy decision request of the PEP into an XACML policy request. The COPS adapter sends this request to the remote PDP which is responsible for providing the policy decision based on the master security policy. The XACML policy decision response is sent back from the remote PDP via COPS to the local PEP which enforces the policy decision.



(a) Configuration Scenario

(b) Outsourcing Scenario

Fig. 5. Policy provisioning

4.3 XACML over COPS

We extended the COPS framework with an XACML client type. All COPS messages start with a common header that determines the message type and the payload type. Figure 6 shows the schema of this header whose relevant fields are described below.

0	1	2	3
Version | Flags	Op Code	Client Type	
Message Length			

Fig. 6. Common COPS header

The `Op Code` indicates the type of the message, e. g. `REQ` or `KA`. The `Client Type` field provides a code that uniquely identifies the payload carried in the message. For example client-type number 1 is a published *Internet Assigned Numbers Authority* (IANA) number assigned to RSVP policy data [10].

Each COPS message may consist of different COPS objects. The message content is wrapped with the help of 16 different predefined COPS objects. Some of these objects provide fields to carry client-type specific data.

The most important object is the afore mentioned `ClientSI` object that has variable length and transports the client-type data. Figure 7 depicts the generic COPS object structure. Depending on the type of COPS message that is signaled zero, one or more COPS objects may follow the COPS header.

The `C-Num` and `C-Type` fields determine the class and the characteristics of the object. For `ClientSI` objects the `C-Num` field is 9 and the variable length field for

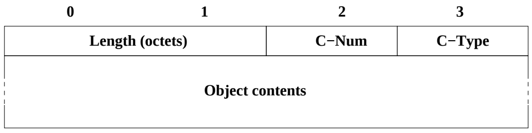


Fig. 7. Generic COPS object

the object content carries the policy data. This data has to be processed by special COPS modules that can interpret the corresponding client type specific information. Our implementation bases on an open source implementation of the COPS protocol from the University of Waterloo [1,3]. We extended their PBN code with several classes to multiplex incoming COPS messages at the PDP to modules which handle client-specific content like XACML.

4.4 XACML Client Type for the COPS Protocol

As the next step a concept to extend the COPS protocol to transport XACML polices as payload has been developed. For any extension to the COPS protocol one has to take the peculiarity of the target policy language into account. The structure of the client-type specific objects and the protocol extensions should be specified in a supplementary document that defines how the PEP and the PDP interpret and handle the policy specific payload.

Any XACML policy document is structured according to the respectively effective XACML schema. The XACML data flow model defines that a *Policy Administration Point (PAP)* provides the PDP with XACML documents that contain sets of `policy` and `policyset` elements. It is specified in the XACML schema that `policy` and `policyset` elements can be nested inside a `policyset`. The distribution of XACML client-type data in the configuration mode will base on this tree structure of XACML documents. Any leaf and any node of such a document will be encapsulated in a single COPS object and send alone or in a group in a `decision` message from the PDP to PEP. Our concept considers the `policy` and `policyset` elements as such leaf and node objects since they define logical building blocks of a policy. The PEP acknowledges any such COPS message with XACML content with a COPS `report` message. On COPS' PEP side the `policy` and `policyset` building blocks are assembled back into a copy of the XACML master policy which is passed on to the LPDP as defined in SicAri.

The XML structure provides another advantage with respect to the proactive update and delete mechanism of COPS PDPs. A PDP can address any `policy` element in the XACML document using XPATH. Any administrative task modifying the master policy triggers a COPS message for the corresponding `policy` and `policyset` element that has been changed. This COPS message will transport one or more COPS `decision` objects containing `replacement data` that uniquely identifies the processed policy elements using XPATH objects. This way the PDP can generate fine-granular updates at the XML element level. The COPS protocol assures that both, PEP and PDP, always work on the same XACML document. Any policy document is definitely identified by the TCP connection between PEP and PDP together with the client handle that the PEP uniquely assigns to each request that it sends out.

5 Related Work

In [18], Ponnappan *et al.* describe a policy based QoS management system for IntServ/DiffServ networks. This design uses COPS for interfacing with the network devices and CORBA as middleware for component interaction.

An approach presented in [21] by Toktar *et al.* proposes an XACML-based framework for distributing and enforcing access control policies to RSVP-aware application servers. Access control policies are represented in an extension of XACML (based on Sun XACML) which is an alternative to the IETF Policy Core Information Model (PCIM) [11] based approach. The authors use COPS in outsorcing mode to distribute policy requests and decisions between the policy server and the RSVP server that is responsible to enforce the QoS measures.

In his dissertation [17], K. Phanse proposes a management framework for policy based ad hoc network management. He builds on a distributed, hybrid architecture that combines the outsourcing and provisioning models of COPS and COPS-PR to provide an efficient and flexible solution for policy distribution in wireless ad hoc networks. To translate the policy specification into device-specific configuration, the management framework must be aware of the various resources available in the system. Policy provisioning occurs after policies are distributed, and consists of installing and implementing the policies using device specific mechanisms.

Since COPS seems to be the only open service for policy distribution of notable propagation we find it hard to compare our approach to others. While it is difficult to make a quantitative statement on the efficiency to transport XACML policies via COPS it is easier to give a qualitative predication. COPS realizes some design aspects that improve efficient distribution and maintenance of distributed policies. After provisioning the initial policy in configuration mode to the LPDPs the central PDP keeps them up-to-date with unsolicited decision messages whenever some part of the policy changes. Since COPS allows to transport policy parts of arbitrary size it is up to the developer of the payload extension to optimize the communication overhead. COPS only demands that the transported pieces are uniquely addressable. This perfectly fits to XACML because XPATH and unique element identifiers allow to address and transport only those pieces of the master policy that have really changed. Furthermore, COPS in configuration mode promises to economically use network resources because access control policies are not very likely to be changed frequently. This enables the administrator to choose a higher value for the KA timeout thus reducing the communication overhead when there are no updates. The only drawback lurks in the fact that XACML is an XML-based language which are inherently wordy.

6 Outlook

It is planned to implement the mechanism to encapsulate XACML payload in COPS messages as described in Section 4.4. The open source JDOM (`http://www.jdom.org/`) package for parsing and representing XACML documents as objects seems a viable basis to build a solution upon.

We will furthermore develop a concept to use the policy administration, policy validation, and policy provisioning mechanisms implemented in the platform, to manage other policy domains such as network security policies (e. g. to configure external PEPs such as firewalls).

An additional research aspect with respect to a holistic policy approach in distributed environments will be policy negotiation in case services from two different security domains, enforcing security upon two different security policies, have to interact with each other.

Acknowledgement

This paper was written while the authors were working within SicAri, a project funded by the German Ministry of Education and Research

References

[1] R. Boutaba, A. Polyrakis, and A. Fernandez Casani. Active Networks as a Developing and Testing Environment for Networks Protocols. In *Annals of Telecommunications*, volume 59, pages 495–514, 2004.

[2] Jery Bryans. Reasoning about XACML policies using CSP. In *SWS '05: Proceedings of the 2005 workshop on Secure web services*, pages 28–35, New York, NY, USA, 2005. ACM Press.

[3] Alvaro Fernandez Casani. Implementation of a Policy Based Network Framework using Metapolicies, December 2001.

[4] SicAri Consortium. SicAri – A security architecture and its tools for ubiquitous Internet usage, October 2003. URL: http://www.sicari.de/.

[5] World Wide Web Consortium. OWL Web Ontology Language – Overview, February 2004. URL: http://www.w3.org/TR/owl-features/.

[6] D. Durham, J. Boyle, R. Cohen, S. Herzog, R. Rajan, and A. Sastry. The COPS (Common Open Policy Service) Protocol. RFC 2748 (Proposed Standard), January 2000. Updated by RFC 4261.

[7] David F. Ferraiolo, D. Richard Kuhn, and Ramaswamy Chandramouli. *Role-Based Access Control.* Computer Security Series. Artech House, Boston, 2003.

[8] Kathi Fisler, Shriram Krishnamurthi, Leo A. Meyerovich, and Michael Carl Tschantz. Verification and change-impact analysis of access-control policies. In *ICSE '05: Proceedings of the 27th international conference on Software engineering*, pages 196–205, New York, NY, USA, 2005. ACM Press.

[9] Li Gong. *Java^{TM} 2 Platform Security Architecture, Version 1.2.* Sun Microsystems Inc., 2002.

[10] S. Herzog, J. Boyle, R. Cohen, D. Durham, R. Rajan, and A. Sastry. COPS usage for RSVP. RFC 2749 (Proposed Standard), January 2000.

[11] B. Moore, E. Ellesson, J. Strassner, and A. Westerinen. Policy Core Information Model, Version 1, February 2001.

[12] Tim Moses. eXtensible Access Control Markup Language (XACML), Version 2.0. Technical report, OASIS Standard, 2005.

[13] National Institute of Standards and Technology (NIST). Role-Based Access Control. URL: http://csrc.nist.gov/rbac/.

[14] OASIS Open. Core and Hierarchical Role Based Access Control (RBAC) Profile of XACML v2.0, February 2005. URL: http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rba%c-profile1-spec-os.pdf.

[15] P. Ochsenschläger, J. Repp, and R. Rieke. The SH-Verification Tool. In *Proc. 13th International Florida Artificial Intelligence Research Society Conference (FLAIRS-2000)*, pages 18–22, Orlando, FL, USA, May 2000. AAAI Press. ISBN: 0-1-57735-113-4.

[16] P. Ochsenschläger, J. Repp, R. Rieke, and U. Nitsche. The SH-Verification Tool – Abstraction-Based Verification of Co-operating Systems. *Formal Aspects of Computing, The International Journal of Formal Method*, 11:1–24, 1999.

[17] Kaustubh S. Phanse. Policy-Based Quality of Service Management in Wireless Ad Hoc Networks. Dissertation, Virginia Polytechnic Institute and State University, August 2003.

[18] A. Ponnappan, L. Yang, R. Pillai, and P. Braun. A Policy Based QoS Management System for the IntServ/DiffServ Based Internet. In *Proc. of the IEEE 3th International Workshop on Policies for Distributed Systems and Networks (POLICY 2002)*, page 159ff, Los Alamitos, CA, USA, 2002. IEEE Computer Society.

[19] Ravi Sandhu. Role activation hierarchies. In *Proceedings of the third ACM workshop on Role-based access control*. ACM Press, 1998.

[20] Sun Microsystems, Inc. *Sun's XACML Implementation, Version 1.2*. URL: `http://sunxacml.sourceforge.net/`.

[21] Emir Toktar, Edgard Jamhour, and Carlos Maziero. RSVP Policy Control using XACML. In *Proc. of the IEEE 5th International Workshop on Policies for Distributed Systems and Networks (POLICY 2004)*, page 87ff, Los Alamitos, CA, USA, 2004. IEEE Computer Society.