A Technical Architecture for Enforcing Usage Control Requirements in Service-Oriented Architectures

B. Agreiter, M. Alam, R. Breu, M. Hafner
University of Innsbruck
Austria

A. Pretschner*
Information Security
ETH Zürich
Switzerland

J.-P. Seifert
University of Innsbruck, Austria, and Samsung Information Systems America, San Jose, CA, USA

X. Zhang
Samsung Information Systems America
San Jose, CA, USA

ABSTRACT

We present an approach to modeling and enforcing usage control requirements on remote clients in service-oriented architectures. Technically, this is done by leveraging a trusted software stack relying on a hardware-based root of trust and a trusted Java virtual machine to create a measurable and hence trustworthy client-side application environment. We define a model-driven approach to specifying remote policies that makes the technical intricacies of the target platform transparent to the policy modeler.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; D.2.11 [**Software Engineering**]: Software Architectures; D.3.2 [**Programming Languages**]: Language ClassificationsSpecialized application languages

General Terms

Security, Design, Languages

Keywords

SOA, Policies, Trusted Computing, Usage Control, Access Control

1. INTRODUCTION

A fundamental security issue in service-oriented architectures (SOAs) concerns controlling the usage of data, including sharing of confidential information. In current security models, access control is enforced on the server side only. This means that the data provider looses control over his data once it is given away. The key to controlling access and usage of a specific resource object after delivery to a remote platform is the ability to enforce specific remote policies that are issued by the resource owner on that remote platform. These policies stipulate so-called usage control requirements [20, 22] that restrict future usages of data.

Different business contexts give rise to different trust models. In outsourcing scenarios, service providers typically have a genuine interest in enforcing policies that the data owner has imposed. This is a consequence of the large value (say, hundreds of thousands of Euros) of the respective outsourcing contracts—if the service provider does not adhere to the policies, he looses a customer and a large sum of money. On the other hand, the owners of web services that implement, for instance, the negotiation of health insurances, may have a strong interest in keeping sensitive data, even though this may conflict with respective policies. In these cases, the data provider (that is, the service requester) is likely to have an interest in technological means that enforce the respective policies [23].

Most notably in scenarios of that second kind, the heterogeneity and openness of SOAs pose a significant challenge on the enforcement of remote usage control policies. SOA-based systems are, in general, open and non-proprietary systems. This is in stark contrast with closed systems where vendors and resource providers have some means to enforce their policies on a user's device or client host by keeping design and implementation of their hardware and software security controls secret. This approach, found in many current DRM mechanisms, is supposed to work in practice, at least in the short run [5]. As an alternative, trusted computing technologies [21], like the ones put forward by the Trusted Computing Group (TCG [2]), provide the technical underpinning for the enforcement of a variety of usage control requirements, thereby bringing some of the properties of closed platforms to open systems. However, the great variety of hardware and software configurations of peer hosts in SOA-based settings results in an equally great complexity of the respective configuration task.

Problem Statement. We tackle the question of how to technically enforce usage control policies in SOA-based systems that are not under direct control of a data provider who requests remote services and gives away sensitive data.

Contribution. We propose an approach to model and enforce usage control policies for remote endpoints in SOAs. The granularity of usage control enforcement is that of trusted applications (as opposed to that of operating system calls or a language API).

*Corresponding author: pretscha@inf.ethz.ch. The work of this author was supported by the Bolzano-Innsbruck-Trento Joint School for Information Technology and done while he worked at Innsbruck University.

That is, once an application is considered trustworthy, it has full access to the data object for which usage control policies are defined. All other applications do not have access to the (decrypted) data item. Our architecture is also able to enforce action requirements of the kind "notify owner upon access" or "delete data after 30 days" (§2.3). The complex task of managing security policies for heterogeneous target systems is reduced by model-driven security configuration (§4.4).

Organization. §2 gives an overview of security in SOAs, trusted computing technologies, and usage control requirements. §3 shows how usage control policies can be modeled, and §4 presents our enforcement technology. §3 and 4 form the core of this paper. §5 puts our work in context, and §6 concludes.

2. BACKGROUND

Our work combines techniques and concepts from three different areas, namely those of web services and SOA security, trusted computing and usage control. We briefly present the necessary background in this section.

2.1 Web Services and SOA Security

A comprehensive set of web services security standards has emerged over the past years, catering to some of the major security concerns that hamper the broad adoption of web services by businesses. OASIS has proposed a security extension on top of SOAP [11]. This extension uses the XML encryption and signature mechanism to add security features to SOAP messages [9, 8]. This way, security mechanisms can be integrated into the header and the body of a SOAP message, and be sent via any transport channel without compromising security. Beside transport level security extensions, a variety of standards provides means to manage and exchange security policies. XACML [29] is a standard to define access control for resources in a system. Sun has proposed a specific profile for XACML—called Web Services Policy Language—to define the reconcilement of access rights between partners. SAML [27] is a standard for the exchange of security tokens. WS-Policy [6] allows for the definition of protocol-level security requirements. According to one of the core principles of SOA, the main benefit of these standards lies in the abstraction of the underlying technical implementation of the security controls. However, implementations based on these standards still remain very technical. Other approaches [17, 16] reduce some of the complexity through model based configurations. Nevertheless, this stack of XML-based security standards exclusively supports end-to-end security in terms of, for instance, integrity and confidentiality of data transmitted between peers—as well as access control—based on the traditional model of perimeter security.

Currently, there is no way to cope with a non-co-operative client environment for remote policy enforcement. This points to one of the biggest security issues in SOAs: the sharing of secure information. Once a piece of data has left the provider, it is out of control.

2.2 Trusted Computing

Trusted Computing, an initiative pushed by the TCG, refers to a concept of integrating cryptographic controls and mechanisms on a special hardware chip called the *Trusted Platform Module* (TPM) for commodity computing systems (e.g., PCs, consumer electronic devices, PDA's and other mobile devices) in order to mimic some of the security properties of highly-secured closed systems.

The TPM is, firstly, empowered with cryptographic mechanisms to measure the integrity of a software stack running on the device. This is done through binary hashing. The measurements can be reported to third parties, which can decide whether or not the configuration should be trusted (remote attestation). Secondly, by means of *sealed storage*, a TPM is able to protect I/O and storage of data inside the device. Finally, it is able to strictly isolate the data residing inside memory from other potentially malicious applications. A comprehensive introduction to trusted computing is given in [10].

This design can effectively counter some threats to the security of a system, including those caused by malicious code, viruses, Trojans, etc. The conjecture that a trusted and tamper-proof security basis cannot be achieved using software based solutions alone has extensively been confirmed in practice. Our SOA-based architecture leverages TC technologies to establish a runtime environment on a remote host that is trusted by the resource owner to enforce his policy.

Nevertheless, the problem with current attestation mechanisms is that they are static, inexpressive and technically complex. This makes the specification and management of security policies a big issue in distributed and heterogeneous environments. To overcome these shortcomings, we propose a model-driven approach to configure the security components with policies.

2.3 Usage Control Requirements

Access control is concerned with the decision of who is granted access to a data item on the grounds of currently available information. Usage control extends this notion to what may and may not happen to a data item in the future, *after it has been given from a data provider to a data consumer*. Usage of data can broadly be classified into management (storage, deletion), distribution, rendering, processing, and execution [13]. Usage control requirements come in two different forms. *Usage restrictions* prohibit usages under specific circumstances (e.g., "access data at most once" and "view only with a certified viewer"). *Action requirements* express mandatory actions that have to be executed in specific circumstances (e.g., "delete data after thirty days", "notify owner whenever the data is accessed"). Specifications of the circumstances come as time conditions, cardinality conditions, event-defined conditions, purpose conditions, and environment conditions. A detailed discussion of this general class of usage control requirements—that encompasses both data protection and management of intellectual property—as well as a formal specification language is defined in [13].

3. USAGE CONTROL POLICIES FOR SOAS

In order to illustrate some of our framework's functionality, we take an example application scenario from a medical domain. In our case, doctors, nurses and administrators are given restricted access to resources (patient data) at the hospital's main site. Whenever a service requester attempts to access a patient record, the following steps are performed for authentication and authorization:

1. The service requester authenticates herself to the hospital site and is assigned to a role (e.g., surgeon, nurse, general practitioner etc.).

2. After authentication, a component called the *security gateway* evaluates the service requester's eligibility for the requested resource according to her role, as well as static and dynamic constraints (e.g., access on working days only).

3. If access is granted, the security gateway attaches a remote policy to the released data item. This policy is specific to the target.

4. The policy is shipped with the released object/data to the requester.

The policy specifies constraints on the future usage of the object or information on the service requester's platform. The ability to reliably enforce a specific policy is verified through the abovementioned functionality of *remote attestation* [25] that a TPM provides—checking the integrity of the software stack through static binary hashing.

3.1 Security Model

We base our scenario on a distributed peer-to-peer style application model. A peer-to-peer workflow has several peer nodes representing differing security domains. In accordance with our motivating example, we have two nodes: a remote client host RCH_{wfl} that requests a resource, and a remote service provider RSP_{wfl}, who is in possession of the resource.

In this paper we address threats that materialize in a non-cooperative client environment. This is defined as a remote host with a runtime environment that is not under control of the resource owner. We assume that the client cannot by default be trusted to comply with the usage control requirements imposed by the resource owner. Thus, the client host has to be considered as basically non-co-operative or even potentially malicious. This leads to the following basic threat: *The remote object requester may be able to violate usage control requirements imposed by the remote service provider.* For example, the client may access the resource beyond the rights granted, or provide access to unauthorized third parties (further dissemination). This may have the following reasons.

Untrusted Platform. Although based on a measured and therefore trusted hardware and system software stack, the *configuration* of the remote client host's (RCH_{wfl}) runtime system (e.g., the Java Virtual Machine) is not such that the remote service provider (RSP_{wfl}) can be sure that his access control policy will be enforced.

Untrusted Security Services. The required application and security services are not available or not enforced at the remote client's side (RCH_{wfl}).

Malicious Applications. An application running on the remote client host (RCH_{wfl}) can access the resources provided by the service provider (RSP_{wfl}) in an unauthorized manner. This includes applications that have been installed both inadvertently (Trojans) and on purpose (unauthorized readers).

In this threat model, the enforcement of usage control requirements will be facilitated by (1) the specification of a suitable remote (client) host architecture; (2) the definition of suitable attestation protocols which can be used by the remote service provider (RSP_{wfl}) to verify whether his security goals will be met by the remote (client) host(RCH_{wfl}); (3) the leveraging of trusted computing technologies to the host architecture together with domain separation, essentially to protect unauthorized access to the resource by other applications; and finally (4) the application of a model-driven configuration approach to harness the technical complexity of the security controls in order to provide a correct realisation of usage control requirements. The subject of this paper is a respective technical architecture.

3.2 Policy Modelling

SECTET-PL [16] is a predicative language in OCL-like style that allows the specification of fine-grained data-dependent access permissions on the grounds of roles. Originally developed with the goal of integrating aspects of authorization in use case-based development [7], we use this language in our framework to specify permissions and actions for requesting web services. The approach is model-driven in the sense that we generate platform independent XACML policies from the predicative specifications [16, 3]. The SECTET-PL permission predicates are specified according to the general structure given in Figure 1.

```
context class
...
perm[role_i] :      pcondExp_i
...
proh[role_j] :      ncondExp_j
...
oblig[role_k] :     oactExp_k
...;
```

Figure 1: Syntax of SECTET-PL Constraints

The positive rule **perm**$[role_i]$: $pcondExp_i$ describes the condition $pcondExp_i$ under which some role $role_i$ is permitted to access the resource (the class in this case). In contrast, the negative rule **proh**$[role_j]$: $ncondExp_j$ describes the condition $ncondExp_j$ under which some role $role_j$ is prohibited to access the resource. The obligation rule **oblig**$[role_k]$: $oactExp_k$ describes a sequence of actions $oactExp_k$ that some role $role_k$ must perform after the release of the resource.

As an example, Figure 2 shows a security constraint attached to the entity PsychiatricDisease using SECTET-PL. The constraint formalizes a high-level security requirement which states that the information about the psychiatric health of the patient could be disclosed to "Psychiatric Clinic X" only. Moreover, the constraint restricts, by means of obligations, the `PhysicianRole` for the maximum duration that the medical record can be read after its first successful instantiation. The obligation constraint makes use of the external function `duration()`, defined in the interface `ExternalFunctions`. Note that these constraints extend beyond classical access control.

This interface `ExternalFunctions` refers to the security infrastructure in order to verify a certain relationship between the requester of the web service and a particular element of the policy model. The identification variables (e.g. subject) associated with these external functions distinguish different types of requesters. For instance, `subject.map(T)` allows the connection of the calling actor with his/her internal representation to the business logic enabling permissions such as "the actor has access to his/her own data".

Other obligations can be specified in a similar way. For instance, one can specify that the information can only be retrieved in a specific location, or that the latest anti-spy ware software must be installed using the external functions defined in the interface `ExternalFunctions`. Detailed information on obligations and their different types can be found in [13].

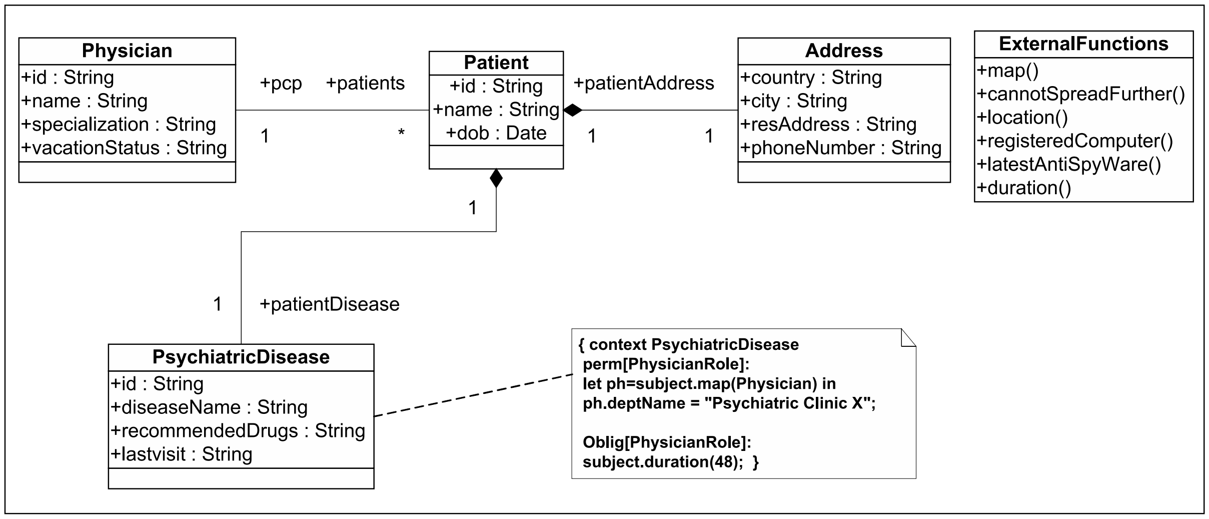
The policy model is then transformed into an XACML policy file to the end of configuring the policy decision point of the *Enforcing Agent* (§4.4).

4. ENFORCING POLICIES IN SOAS

4.1 Target Architecture

The general idea consists of establishing of a trusted relationship between a service provider, delivering some document or information, and a service requester, where the provider can trust the remote client to enforce his policy on the delivered object.

Figure 3 shows the client-side target architecture. Our approach for remote policy enforcement leverages the trusted platform module (TPM) by providing the hardware based root of trust (A.), a



```
{ context PsychiatricDisease
 perm[PhysicianRole]:
 let ph=subject.map(Physician) in
 ph.deptName = "Psychiatric Clinic X";

 Oblig[PhysicianRole]:
 subject.duration(48);  }
```

Figure 2: Sample policy model with associated usage control constraint

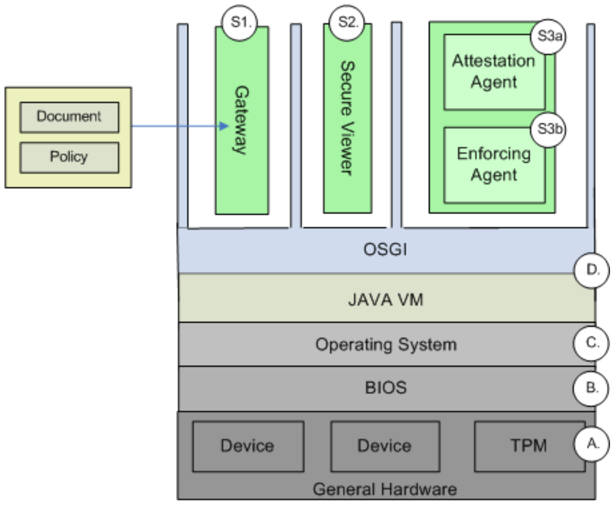


Figure 3: Target Architecture

trusted system software stack, consisting of the BIOS (B.), the Bootloader, and the Operating System (C.). As application runtime we assume a Java virtual machine with the OSGi service platform (D.).

The ***OSGI Framework*** [19] plays a key-role in the architecture. It ensures domain separation for independent Java services, also called *Bundles*. It provides functions for provisioning, administration and life cycle management of services. Together with the hardware components (A.), the two layers of the system software stack (B. & C.) provide the foundations for the trusted run-time environment (D.) that may host a diverse range of applications. We subsequently abstract from these layers and concentrate on the application and security services running ontop of these. A detailed technical account on how the lower layers (especially the OS) may enforce application level security is provided elsewhere [31, 18, 15]; [4] shows how these layers may be configured through models.

The following applications and security services (S1.–S3.) run in strictly separated domains as *OSGI Bundles*. They ensure that the object is used according to the policy of a remote user (policy modeling is treated in §3.2).

Secure Viewer (SV). Whenever a user wants to work on or read a document, the first step is to open the appropriate *SV* (Component S2). The application bundle offers an interface to either browse documents stored locally, or to request documents from a remote resource owner. It can be distributed independently of any other application or document, may even be provided by a third party. The only requirement is that it is signed by a trusted application provider, otherwise it will not be started.

The ***Enforcing Agent (EA)*** acts as a security proxy for applications (e.g., the *SV*) which want to access a protected document. It determines access and usage constraints to be enforced in compliance to a *Usage Policy* which, like the *Application Policy*, is also issued by the object owner. This OSGI bundle comes as a signed JAR file implementing Sun's Policy Decision Point [1], for XACML-policy file evaluation. If the EA denies access, no further action will be performed on the document.

The ***Attestation Agent (AA)*** checks application and security properties of other application bundles (e.g., Viewer and Gateway) according to the *Application Policy* for their ability to enforce a policy (e.g., patient records are only viewable with the appropriate application). This component also implements an XACML-PDP and supports the specification and download of tests via XACML-files.

The ***Gateway (GW)*** acts as a single point of entry into the object requester's domain and has to be trusted by the remote party (achieved by binary attestation as described in the next section). It provides very restricted functionality. The goal is to keep this bundle lightweight and unaware of document contents. Its purpose is to communicate to the remote document providers, to decrypt incoming documents with its own private key (K_{RC}) and to start the appropriate bundles when documents are opened. It additionally offers a notification service for the *EA*. The *GW* is started together with a trusted Java VM and the OSGi framework. *EAs* can register themselves to be invoked at regular intervals. This allows for instance a protected object document to be deleted after 48 hours. This application-level functionality exemplifies behaviour that is checked by the *AA*.

All bundles can be distributed independently of each other. The only requirement for them is to be signed by a trusted application provider, otherwise they will not be started. The application policy, which is used by the *AA* checks whether the *SV* complies to requirements for the respective document (e.g. patient records are only viewable with the appropriate application). The interaction between the different components is described in §4.3.

4.2 Establishing Trust

The establishment of a trusted relationship between a service provider (RSP_{wfl}) who owns the object and a remote client host (RCH_{wfl}) requesting the object is ensured by the following procedure.

1. **Authentication.** The initial authentication of RCH_{wfl} occurs in a traditional way at the server-side (RSP_{wfl}) based on certificates and credentials.

2. **Remote Attestation.** In terms of usage control, the goals of the remote service provider (RSP_{wfl}) are:

   (a) RSP_{wfl} can verify that the RCH_{wfl} is capable to enforce the application's usage control policy.

   (b) RSP_{wfl} can define a policy that will be enforced on RCH_{wfl}.

   Basic trust establishment occurs through remote attestation: two peer node measurements PNM_{H1} and PNM_{H2} capture the configuration values of the complete host software stack. At start up, the system software stack and the runtime environment of the client platform (object requester) are checked to be compliant to a known good configuration value. This is performed by measuring the software stack up to the operating system (including BIOS, Master Boot Record, Operating System) through binary hashing and reporting the value PNM_{H1} to the object owner, thereby leveraging the hardware-rooted trust of the TPM by a procedure called "trusted" bootstrap as described in [18]. In a second step, the run-time environment (Java Virtual Machine and OSGi Framework) that guarantee a strict separation of execution domains for the security services is measured in the same way by the operating system. Again, measurement PNM_{H2} is reported to the remote object owner, who can decide whether to accept the current client configuration as a trusted environment by comparison to a "known good value". After the establishment of a trusted relationship between the object owner and the requester, everything is in place for document and policy delivery. Attestation through binary hashing for the application bundles is not necessary, because the OSGi framework is configured to only execute signed code (including the mechanisms that enforce usage control requirements). This implies that downloaded bundles can be trusted. Furthermore, it is still possible to restrict the permissions of these bundles using the Java `SecurityManager` for an even finer grained access.

3. **Handing over Authorization for Domain Secret.** The Asymmetric Authorization Transfer Protocol (AATP [2]) allows the transfer of authorization data for some kind of object locked in the TPM to a remote entity (which in our case is the object owner) such that the issuing entity has no knowledge of the authorization data. In a first step, the runtime environment of the remote host (OSGi and trusted Java virtual machine) generates a "domain secret" in the form of a secret key. The domain secret will be used to grant or deny access to local objects through en- and decryption. This domain secret is then encrypted with a TPM key and ownership over the encryption key is transferred to the object owner through AATP. The TPM of the object requester will only grant access to the domain secret (stored within the TPM) if ownership of the new authorization data is proven. Authorization information comes with the enforcing component and is performed according to the Object Independent Authorization Protocol [2, 18].

4.3 Document and Policy Delivery Protocol

Document Delivery..

The object owner having decided to trust the remote requester according to the protocols described in §4.2, document delivery is performed according to the following protocol (cf. Figure 4):

1. At first, the party requesting the object has to download and install two security applications called ***Attestation Agent (S3a)*** and ***Enforcing Agent (S3b)***. Both are signed JAR files and may be provided either by the object owner or a third party. The signature guarantees application integrity and guarantees the code's ability to act on behalf of the object owner. In case the document is already available locally (e.g., by restarting a previously ended session) this downloading step is omitted.

2. The document together with a *Usage-* and an *Application Policy* are sent to the object requester. The document and the policy files are encrypted with the remote client's key (K_{RC}), so that it is protected against interception when sent by the object owner. The document and the policies are signed.

3. Upon reception, the client side's ***Gateway (GW)*** checks the integrity of the policies and the document. The *GW* has access to a secure keystore containing its own private and public keys for the document and the applications. If the signature is valid and the *GW* succeeds in decrypting the document, a protected domain is instantiated by the OSGi-Framework as an isolated run-time environment for the *AA* and the *EA*.

4. The *GW* passes the document and two policies in XACML-format to this newly instantiated bundle in plaintext. The ***Usage Policy*** determines access and usage constraints to be enforced by the *EA*, whereas the ***Application Policy*** defines which properties are to be tested by the *AA*.

5. (This step is optional and only applies if the framework is also equipped with remote semantic attestation that allows the host to run certain tests on the client.) The *AA* then runs checks on the status and capabilities of the platform. These tests are based on information contained in the *Application Policy*, an XACML-based file which describes requirements to the applications and the system e.g. check whether the document repository is accessible only as specified, check the capabilities of the secure viewer and testing of security properties of the gateway etc. The tests should help to establish trust in application-level security properties.

6. After successful installation of the bundles, the *OSGi - Framework* first generates the *Domain Secret*, which is a symmetric key (K_{Doc}), encrypts the document and stores it into the document repository, and finally transfers control over the *Domain Secret* to the remote object owner according to the AATP and OAIP as described in §4.2. The key K_{Doc}, now protected by the TPM of the object requester's host, is only

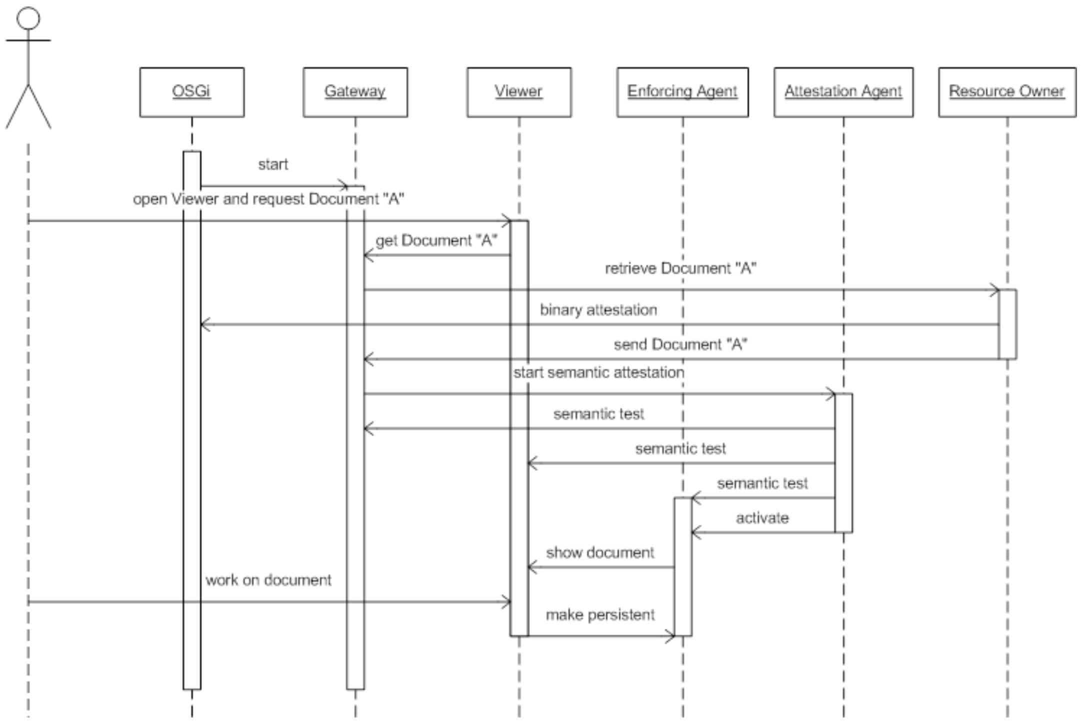


Figure 4: Overview of document and policy delivery

released (by decryption with the key stored within the TPM) when shown the authorization secret chosen by the object owner during AATP. The authorization secret is a 20-bytes long SHA-1 digest passed to the enforcer once trust is established [2].

7. The *EA* is "activated" by getting the authorization secret from the object owner. It can get the *Domain Secret* guarded by the TPM, decrypt the document and grant access to the secure viewer on request. It is a valid assumption to trust the EA to behave in the interest of the document provider, because it is provided and signed by him. In case testing of application properties failed, the document is discarded and the Enforcing and Attestation Agents are deleted. Otherwise, the *EA* now directly communicates with the viewer and sends it the document. On every action the viewer wants to take on the document (e.g. change elements, print etc.) it asks the *EA*, who has to allow all actions.

Usage Control. Usage rules require ongoing checks (e.g., delete the document after 48h) which are enforced through the *EA* on a regular basis. If a document is opened while the enforcer realizes that the policy disallows further usage, it stops the bundle with the according viewer. This is possible because the enforcer is in possession of the reference to this bundle. Usage information is stored with the TPM and only accessible to applications in possession of the domain secret (e.g., the *EA*). In case of a system re-boot, the *EA* will first load usage information and check with the policy.

Trusted Applications. Attestation through binary hashing for the application bundles is not necessary, because the OSGi framework is configured to only execute signed code. This implies that downloaded bundles can be trusted, and it is still possible to restrict the permissions of these bundles using the Java `SecurityManager` for an even finer grained access control.

4.4 Configuring the Enforcing Agent

The EA is configured with XACML policy files which are received by the client along with the document. This *usage policy* is generated from the policy model or, more precisely, from the SECTET-PL constraints attached to a usage control restricted resource. Referring to our example, Figure 5 shows an example policy for a physician who is allowed to only read a document. Note that the example is given in simplified XACML syntax.

The policy generated for the physician contains the authorization and obligation constraints in the form of `<Rule>`s. The `<Target>` element contains the name of the document (line 08) and of the operation (line 11) to which the `<Rule>` applies. A `<Condition>` element additionally specifies an authorization constraint. If the authorization constraint is met by the subject, access will be granted. The `<Condition>` element defines authorization constrains in the form of XACML functions for extended X-Path, X-Query, Date, Time etc.

An `Obligation Constraint` constitutes the main extension in the XACML policy. Figure 5 partially describes an example `Obligation Constraint` described in Figure 2. According to this obligation, *the Hospital restricts the service requesters for reading the document for 48 hours only*. The extended obligation function `SECTET: obligation:function:duration` obligates the *EA* to make the document inaccessible after 48 hours have passed.

5. RELATED WORK

When compared with approaches that focus on the enforcement of usage control policies on remote client platforms, our work essentially differs in that we apply a model-driven engineering framework advancing aspects of usage control. In the following we give an overview of related work.

In order to improve on the expressiveness of the attestation procedure, [24] proposes a framework for property-based attestation.

```
<PolicySet PolicySetId="DocumentAccess"
                    Combining Algorithm =„deny-overrides">
<Policy PolicyId ="Permissions:for:Physician"
                    CombiningAlgorithm=„deny-overrides">
<Target>
 <Subjects> <AnySubject/> </Subjects>
<Resources>
        http://exampleHealthCare.com/record/patient/PatientA
</Resources>
<Actions>
        http://wsserver.medsys.xsoap:writePatientRecord
</Actions>
</Target>
   <Rule Effect=“permit“>
      <Condition>
       Authorization Constraint
      </Condition>
   </Rule>
   <Obligations>
          Obligation Constraint
   </Obligations>
</Policy>
</PolicySet>
```

```
<Condition  FunctionId="string-is-in">
            <Apply  FunctionId= "custom-xpath-set-values">
            <SubjectAttributeDesignator AttributeId=/Hospital/Physician [id='/
Request/Subject/Attribute[@AttributeID=phyId]']/deptName/text() />
            </Apply>
            <Apply  FunctionId= "custom-xpath-one-and-only-one">
            <ActionAttributeDesignator AttributeId=/Request/Action/Attribute
            [@AttributeId=deptName]/text()/>
            </Apply>
</Condition>
--------------------------------------------------------------
<Obligations>
<Obligation
  ObligationId="SECTET:obligation:function:duration"
                                                  FulfillOn="Permit">
  <AttributeAssignment  AttributeId="SECTET:1.0:attribute:time"
            DataType="http://www.w3.org/2001/XMLSchema#time">
    48h
  </AttributeAssignment>
</Obligation>
</Obligations>
```

Figure 5: Example policy with generated authorization and obligation constraints

The approach is a good starting point for understanding the concept and the motivation for semantically-enriched attestation. However, one of the major drawbacks of the approach is that property-based attestation is delegated to a trusted third party, which is then responsible for setting up an attestation communication with the client platforms. In our approach, we elaborate on an efficient property-based attestation without a third party in the context of SOAs.

Franz et al. [12] propose semantic remote attestation to encapsulate program behaviour and make attestation more dynamic, flexible and expressive. The approach attests the high-level properties of Java programs, e.g., which class can inherit from a given class, and how the behaviour of a given class can be restricted. While the focus of this work is on Java programs only, we leverage the idea to SOAs.

A distributed usage control policy language and its enforcement requirements are presented in [14, 22]. Similar to our objective, this work targets control over data after its release to third parties. The significant difference between this and our work is that our work relies on the underlying trusted computing services of a platform.

Jaeger et al. [15] have recently proposed a framework which is an extension to IMA [26]. Their framework focuses on the improvement of efficiency during the integrity measurement of the platform by limiting the number of measured entities. Compared to our approach, PRIMA does not discuss the specification of TC-related requirements at all. Their model can be incorporated within the Trusted SECTET for efficient integrity measurement.

Remote Attestation for web services, called *WS-Attestation*, has been proposed in [30] and with a slight variation in [28]. However, both approaches directly bind Web services standards to the TC technologies to increase trust and confidence in integrity reporting. In our case, shipping XACML policies with the protected object/information are better suited for secure information sharing and leverages policy oriented TC approach which is a novel aspect.

6. CONCLUSION

We have presented a technical architecture for the enforcement of usage control policies in SOAs. Before a service provider sends data to a requester, the former checks if the latter has sufficiently technological means to enforce a usage control policy and if this technology has not been modified. The usage control policy is then tied to the resource and shipped to the requester. At the requester's side, a dedicated enforcement component is then configured with this policy, and this ensures that the policy will be adhered to. Methodologically, our approach makes use of model-driven engineering technologies that make it possible to graphically specify policies that, after the necessary transformations, are used to configure the enforcement component of our architecture. Technologically, our solution leverages trusted platform technology to SOAs. We have described the threat model that our architecture is able to counter, and described the system configurations and protocols that are necessary to transfer data items and subsequently enforce the usage control policies. As far as we know, there is no general technical architecture for the enforcement of usage control requirements, which we consequently see as the main contributions of this paper.

There are some limitations of our approach. We restrict the general concept of usage control in the following way: only trusted applications are run—and the underlying assumption is that these applications will not violate the policies. That is, we do not implement the usage restrictions at the level of Java API or operating systems calls but rather assume that this is done by the trusted applications (this assumptions can clearly be justified in the running example of this paper). In other words, the enforcement component restricts usages by providing a key to trusted applications, in addition to implementing required actions [13] (e.g., deleting a data object).

Future work includes an implementation of the architecture on the grounds of existing trusted computing technology as well as case studies that will allow us to better assess benefits and shortcomings of our approach. We will also have to understand precisely in which contexts the enforcement of usage control at the level of trusted applications is sufficient, and in which contexts operating systems or language API calls are the more appropriate level of granularity.

7. REFERENCES

[1] SUN XACML Implementation . Available at sunxacml.sourceforge.net.

[2] Trusted computing group (tcg). https://www.trustedcomputinggroup.org/specs/.

[3] M. Alam, R. Breu, and M. Breu. Model Driven Security for Web Services (MDS4WS). In *Proc. INMIC*, 2004.

[4] M. Alam, M. Hafner, J.-P. Siefert, and X. Zhang. Extending SELinux Policy Model and Enforcement Architecture for Trusted Platforms Paradigms. Accepted for Annual SELinux Symposium.

[5] R. Anderson. Security in open versus closed systems — the dance of Boltzmann, Coase and Moore. In *Open Source Software Economics 2002*, 2002.

[6] S. Bajaj. Web services policy framework (wspolicy). March 2006, Version 1.2.

[7] R. Breu and G. Popp. Actor-centric modelling of access rights. In *FASE 2004*. Springer LNCS Vol. 2984, p. 165-179, 2004.

[8] D. Eastlake and J. Reagle. XML Encryption Syntax and Processing. W3C Rec. 10/12/2002.

[9] D. Eastlake and J. Reagle. XML-Signature Syntax and Processing. W3C Rec. 12/02/2002.

[10] D. Grawrock. *The Intel Safer Computing Initiative Building Blocks for Trusted Computing*. Intel Press, http://www.intel.com/intelpress/sum_secc.htm, 2005.

[11] M. Gudgin, M. Hadley, N. Mendelsohn, J. Moreau, and C. Nielsen. Soap version 1.2 part 1: Messaging framework. W3C Recommendation 24 June 2003.

[12] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing.

[13] M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. Enforcement for Usage Control: A System Model and a Policy Language for Distributed Usage Control. Technical Report I-ST-20, DoCoMo EuroLabs, 2006.

[14] M. Hilty, A. Pretschner, C. Schaefer, and T. Walter. Usage control requirements in mobile and ubiquitous computing applications. In *Proc. of Intl. Conf. on Systems and Networks Communications*, 2006.

[15] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: Policy-reduced Integrity Measurement Arch. In *Proc. 11th ACM symp. on Access Control Models and Technologies*, pages 19–28, 2006.

[16] M. Alam et al. Modeling Permissions in a (U/X)ML World. In *IEEE ARES*, 2006.

[17] M. Hafner, M. Alam, R. Breu. A MOF/QVT-based Domain Architecture for Model Driven Security . In *IEEE/ACM Models 2006 LNCS 4199*.

[18] H. Maruyama, F. Seliger, N. Nagaratnam, T. Ebringer, S. Munetho, and S. Yoshihama. Trusted platform on demand. Technical report, IBM Research, 2006.

[19] OSGI Alliance. OSGi—The Dynamic Module System for Java. www.osgi.org.

[20] J. Park and R. Sandhu. The UCON ABC Usage Control Model. *ACM Transactions on Information and Systems Security*, 7:128–174, 2004.

[21] S. Pearson. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[22] A. Pretschner, M. Hilty, and D. Basin. Distributed usage control. *Communications of the ACM*, 49(9):39–44, September 2006.

[23] A. Pretschner, F. Massacci, and M. Hilty. Usage Control in Service-Oriented Architectures. In *Proc. TrustBus*, 2007. To appear.

[24] A. Sadeghi and C. Stï¿$\frac{1}{2}$ble. Property-based attestation for computing platforms. Caring about properties, not mechanisms. In *Proceedings of the workshop on new security paradigms*, pages 67–77, 2004.

[25] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *Proc. CCS '04*, pages 308–317.

[26] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symp.*, 2004.

[27] SAML 2.0 Specification. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=security.

[28] Z. Song, S. Lee, and R. Masuoka. Trusted web service. In *The Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*, 2006.

[29] XACML 2.0 Specification Set. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml.

[30] S. Yoshihama, T. Ebringer, M. Nakamura, S. Munetoh, and H. Maruyama. WS-Attestation: Efficient and Fine-Grained Remote Attestation on Web Services. In *Proceedings of the IEEE Int. Conf. on Web Services (ICWS'05)*, pages 743–750, Washington, DC, USA, 2005. IEEE Computer Society.

[31] X. Zhang, F. Parisi-Presicce, and R. Sandhu. Towards remote security enforcement for runtime protection of mobile code using trusted computing. In *Proc. of the 1st Int. Workshop on Security (IWSEC)*, 2006. LNCS Kyoto, Japan.