Privacy Enforcement Through Policy Extension

Saritha Arunkumar†, Mudhakar Srivatsa, Berker Soyluoglu‡, Murat Sensoy‡, Federico Cerutti*
IBM Hursley Labs, UK† IBM Research, USA Ozyegin University‡ Cardiff University, UK*
saritha.arun@uk.ibm.com, msrivats@us.ibm.com, murat.sensoy@ozyegin.edu.tr,
CeruttiF@cardiff.ac.uk

***Abstract*—Successful coalition operations require contributions from the coalition partners which might have hidden goals and desiderata in addition to the shared coalition goals. Therefore, there is an inevitable risk-utility trade-off for information producers due to the need-to-know vs. need-to-hide tension, which must take into account the trustworthiness of the other coalition partners. A balance is often achieved by deliberate obfuscation of the shared information. In this paper, we show how to integrate obfuscation capabilities within the current OASIS standard for access control policies, namely XACML.**

I. Introduction

As widely discussed in [1], Information sharing is key to the operational efficiency of a coalition network. From an information provider's perspective, successful decision-making at the consumer using the shared data, indirectly results in utility for the provider and offers incentive for sharing. However, the act of sharing presents risks to strategic assets. The risk arises from the possibility that the data being shared may reveal more information to its recipient than was intended. Policies instituted at the producer to manage this risk often negatively affects the quality of decision-making at the consumer by introducing additional sources of uncertainty, thereby reducing the provider's utility.

One of the ways in which the balancing between risk and utility is achieved is through deliberate obfuscation of data. Obfuscation is a process through which the quality of the shared information is degraded in a controlled manner to protect against sensitive inferences regarding strategic assets [2]. This allows the data to retain utility while lowering the associated risk.

In this paper we demonstrate the capability of the current OASIS standard for access control policies : XACML (eXtensible Access Control Markup Language)to encompass the description of obfuscation methodologies, and we suggests which elements should be particularly involved in this process.

XACML [7] is an OASIS standard that defines an XML-based language for specifying access control policies, requests and responses.

The XACML policy language is designed to support ABAC (Attribute Based Access Control). Decisions can be made according to some conditions on the attributes of the subject, the object of the request or the system environment. This approach makes XACML a flexible authorisation system that allows the specification of context-aware and risk-intelligent access control policies. The standard also defines a RBAC (Role Based Access Control) profile as a particular case of an ABAC system.

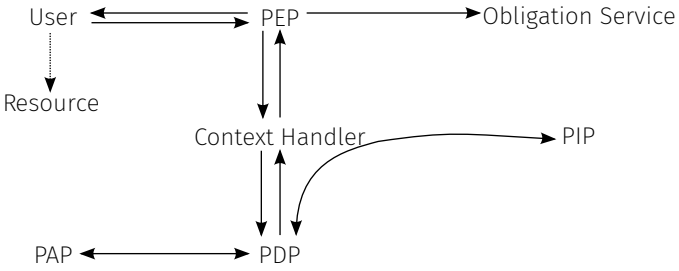

Fig. 1: XACML reference architecture, [7]

XACML separates the *authorisation decision* point from the *decision enforcement point*. Before performing an action, each application needs to ask the XACML decision point for permission. There is therefore, no need to embed the access control rules in the code of different applications. Changing access control rules just need adaptation of the related policy in the XACML authorisation system: this will automatically affect each application.

The rest of the paper is organized as follows. Section II overviews the XACML architecture. Section III introduces an abstract model of permissions and obligations for obfuscation. Section IV demonstrates the capability of XACML to encompass the description of obfuscation methodologies. Section V briefly evaluates our prototype implementation and Section VI concludes the paper with a discussion of our contributions.

II. Preliminaries

This section present preliminary information about policy framework that our approach built upon.

A. XACML Architecture

The reference XACML architecture proposed by OASIS is shown in the data-flow diagram in Figure 1 and it comprises the following elements:

1) PDP (Policy Decision Point): It receive access requests from the context handler (step 3 in Section V) and it queries the Policy Repository to find the applicable policies for an access request (step 4 in Section V). In the case of no (or more than one) policy, the PDP returns an error code. In the case of an applicable policy, the PDP evaluates it and returns the corresponding decision to the context handler (step 9).

To perform its duty, the PDP might need to retrieve additional attributes outside the request context, via the PIP module (steps 5-8 in Section V).

2) PEP (Policy Enforcement Point): It is an application-dependent component. It has to intercept every user access request (step 1 in Section V), forward it to the context handler (step 2 in Section V) and enforce the authorisation decision returned by the context handler (step 10 in Section V) only if all the obligations can be correctly interpreted and fulfilled by the PEP, otherwise the PEP returns an error message. As shown in Figure 1, the system obligation fulfilment is usually performed by a different module then PEP.

3) Context handler: It receives the requests from the PEP in an application-dependent format (step 2 in Section V), convert them to an XACML request and forward them to the PDP adding the request context information (step 3 in Section V). It can be requested—by the PEP—to retrieve some additional attribute from the PIP if these are needed to evaluate a particular policy or rule (steps 5-6-7-8 in Section V). The context handler is also responsible for receiving the XACML response from the PDP, with system obligations, and to convert it to a format understandable for the PEP before forwarding it (steps 9-10 in Section V).

4) PIP (Policy Information Point): It provides the PDP with the information needed to evaluate a request that is not available in the request context (e.g. subject, resource and environment attributes). It is an interface between the PDP and the Data Source of the application, thus it is an implementation-dependent component.

5) PAP (Policy Administration Point or Policy Repository): It stores the XACML policies and is queried by the PDP when it has to find an applicable policy for an XACML access request or to find a specific policy that has been referred by another one. An important feature of a PAP is that it has to allow the definition of a subset of the policy set as top-level policies. These are the only policies that directly applicable for evaluating a request. The policies in the set of non-top-level policies can be used only if referred to by another one. It is important to carefully define these two sets in a way that, for every possible request, only one applicable top-level policy can be found in the PAP.

B. Policy language

The atomic unit of an access control policy set is a `<Rule>`. A XACML `<Policy>` may be composed of a number of individual rules combined together according to a combining algorithm. In the same way a `<PolicySet>` may be composed by a number of individual policies. Three top-level policy elements are defined:

1) `<Policy>`: It contains a set of `<Rule>` elements and a specified procedure for combining the results of their evaluation. It is the basic unit used in the policy specification and so it is intended to form the basis of an authorisation decision. Its main components are:

- A `<Target>`: it specifies for which requests the rule applies via a conjunction of `<AnyOf>` elements. Every `<AnyOf>` element contains a disjuction of `<AllOf>` elements. Every `<AllOf>` element contains a conjunction of `<Match>` elements. A `<Match>` element compares an attribute value with an embedded value applying a specific match function.
- A set of `<VariableDefinition>` elements: these allow the policy writer to associate a `VariableId` to a specific value. The value is enclosed in the tag and has to be a subtype of the `<Expression>` element type. These variables can be referenced using the `<VariableReference>` throughout the whole policy in the definition of rule's condition, obligations or advice.
- A set of `<Rule>` elements.
- A `rule-combining-algorithm` attribute: it specifies the algorithm used by the PDP to combine the results of the different rule components. There are six normative algorithms:
  1) `Extended Indeterminate values`;
  2) `Deny-overrides`;
  3) `Ordered-deny-overrides`;
  4) `Permit-overrides`;
  5) `Ordered-permit-overrides`;
  6) `Deny-unless-permit`.
- A `<ObligationExpressions>` including a set of `<ObligationExpression>` elements that specify obligations to be fulfilled by the PEP when the rule applies and the specified effect is returned. An obligation in XACML is represented as a set of attribute assignments that the PEP has to be able to interpret. In Figure 2 we show an example system obligation that obliges the PEP to send a confirmation email to the user who obtained access to a resource. The `<ObligationExpression>` element must contain a set of `<AttributeAssignmentExpression>` each one assigns a value to a specified `AttributeId`. The value is enclosed in the tag and has to be a subtype of the `<Expression>` element type previously defined.
- A `<AdviceExpressions>`, i.e. a set of `<AdviceExpression>` elements—syntactically analogous to `<ObligationExpression>`—that specify advice that the PEP should fulfil when the rule applies and the specified effect is returned.

2) `<Rule>`:

- A `<Target>`.
- a boolean `<Condition>` that refines the applicability of the rules beyond the predicates implied by its target. This is an expression that apply functions chosen from a standard set to one or more attribute variables. A `<Condition>` element must contain one element of a type that extends the `<Expression>` type, such as:
  - `<Apply>` and `<Function>`, that apply a specified function to a number of members;
  - `<AttributeDesignator>` and `<AttributeSelector>` that allows the retrieval of the value of a named attribute from the request context and from a given xml path respectively;
  - the `<VariableReference>` element that al-

```
<ObligationExpression
 ObligationId="send-confirm-email"
 FulfillOn="Permit">
  <AttributeAssignmentExpression
   AttributeId="mail"
   DataType="http://www.w3.org/2001/XMLSchema#string">
    <AttributeDesignator
     AttributeId="subject-category:mail"
     Category="urn:oasis:names:tc:xacml:1.0:
           subject-category:access-subject"
     DataType="http://www.w3.org/2001/XMLSchema#string"
     MustBePresent="false" />
  </AttributeAssignmentExpression>
</ObligationExpression>
```

Fig. 2: Obligation attribute definition in XACML 3.0

```
<Request ...>
  <Attributes Category="...:
   subject-category:access-subject">
     <Attribute IncludeInResult="false"
      AttributeId="...:subject:subject-id">
      <AttributeValue
       DataType="urn:oasis:names:tc:xacml:1.0:
         data-type:rfc822Name">
        bs@simpsons.com</AttributeValue>
      </Attribute>
   </Attributes>
   <Attributes...
```

Fig. 3: Example of request context in XACML 3.0

lows reference to a variable defined using the `<VariableDefinition>`.

- An `Effect` attribute: the decision value $d \in \{$"Deny", "Permit"$\}$ that has to be returned if both the rule and target conditions evaluates to true. If neither the condition nor the target apply, the return value is "Not applicable."

3) `<PolicySet>`: It contains a set of `<Policy>` or other `<PolicySet>` elements and a specified procedure for combining the results of their evaluation. The main components are:

- A `<Target>`.
- A set of `<Policy>` or `<PolicySet>` elements. An external `<PolicySet>` or `<Policy>` can be referred by its id using respectively the `<PolicySetIdReference>` and the `<PolicyIdReference>`.
- A `Policy-combining algorithm`.
- A set of `<ObligationExpression>`.
- A set of `<AdviceExpression>`.

C. Contexts

When the PEP sends a requests to the context handler, the latter creates a *request context* that includes one or more sets of attribute elements, each of which is associated with one of the supported attribute categories. See Figure3 for an example.

An XACML *response context* includes one or more results, each of which is comprised of a decision and, optionally, obligations and advice.

III. Abstract Model of Permissions and Obligations for Obfuscation Purposes

From a formal point of view, in order to reuse the large corpus of studies in deontic reasoning [4], we refer to a formal

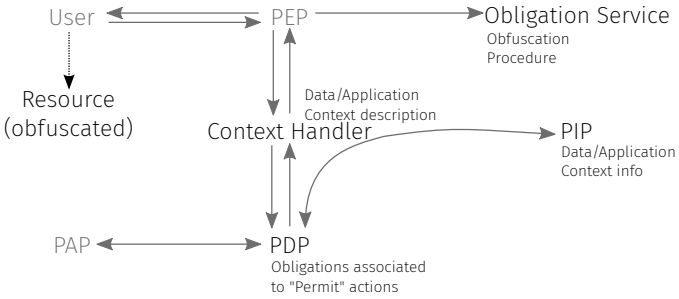

Fig. 4: Obfuscation-specific intervention on the XACML reference architecture

syntax using a logic with modalities. These modalities are: $\mathbf{OB}$ — *obligation* — and $\mathbf{PE}$ — *permission*.

The *obligation* activities can be seen as a strict order, something that we assume will be surely enforced: to this aim, we assume the existence of a robust component in each node of the network, as well as in any element at its edge which guarantees the applicability of obligations.

The *permission* activities represent alternatives that can be exploited by the autonomous risk/benefit assessment that takes place at the edge of the network regarding suitability to share.

Therefore, the expression $\mathbf{OB}\alpha \cdot \gamma$ should be read as: *the action α must be executed under the condition γ* — similarly for the *permission* modality $\mathbf{PE}$. Adopting an expressive language like this will allow us to deal with inconsistencies among policies which might arise [6].

The action can be of two types:

1) not share (ω), i.e. sending no information at all;
2) share to a level x ($\sigma(x)$), i.e. anything between sharing forged data, to sharing the legitimate data.

Instead, γ can have several components which are dependent on the requested resource.

There are then domain independent conditions: for instance, this is the list of conditions that are considered by IpShield:

- `time` (e.g. day, week);
- `location`;
- `activities`;
- `app name`;
- `sensor type`.

We therefore assume to be able to uniquely identify the various fields that can be used for expressing the conditions.

IV. XACML-compatible Obfuscation Mechanisms

From previous Sections, an XAMCL-compatible obfuscation mechanism requires specific implementations within:

1) the policy structure;
2) the Obligation Service;
3) the Context Handler;
4) the PIP.

Those specific implementations are summarised also in Figure 4.

A. Policy Structure

In Section III, two possible actions are identified:

- not share;
- share to a level x.

Instead, the XACML standard allows only for two decision values:

- Deny;
- Permit.

While "not share" is logically analogous to "Deny," "Permit" identifies only the case "share to the level $x = 0$." Therefore, in order to encompass obfuscation levels, each `Policy` must consider an `ObligationExpression` such that, in the case the evaluation of the rules for a given `Target` returns "Permit," the `ObligationExpression` should provide information regarding the level of obfuscation, depending on contextual information.

B. Obligation Service

The chosen nature of obfuscation mechanisms to be represented as an obligation requires the obligation service to be both application-dependent and data-dependent.

Let us consider the IPShield [3] case. In this context, obfuscation can apply to a plethora of sensors: therefore the Obligation Service must be in the position of knowing the data domain and the methodology to apply for guaranteeing an obfuscation of a given degree. For instance, obfuscating with degree $x = 1$ a GPS location might suggests to add a significant amount of Gaussian noise on top of the original stream.

The Obligation Service, therefore, is the component that needs to implement the actual obfuscation techniques for privacy enforcement.

C. Context Handler

As noticed in Section I, the Context Handler receives the requests from the PEP in an application-dependent format, converts them to an XACML request and forwards them to the PDP. The application-dependent nature of this component makes necessary for it to react to specific stimuli. In particular, since XACML3.0 gives freedom in defining attribute elements in request contexts, the Context Handler must be enriched to be able to handle application-specific privacy enforcement attribute requests, such as those identified in Section III in the context of IPShield, such as:

- `time`;
- `location`;
- `activities`.

In particular, what we identified as `sensor type` is already managed by the standard, as the `Target`, while `app name` is the `Subject`.

D. PIP

PIP is the other implementation-dependent component, and it is the interface between the PDP and the Data Source of the application. It therefore must handle context requests application and data dependent.

```
<Response>
  <Result ResourceID="patient-list">
    <Decision>Permit</Decision>
    <Status>
      <StatusCode Value="urn:oasis:names:tc:xacml:1.0:status:ok"/>
    </Status>
    <Obligations>
      <Obligation ObligationId="obligation" FulfillOn="Permit">
        <AttributeAssignment AttributeId="com.ozu.obligationClass"
        DataType="http://www.w3.org/2001/XMLSchema#string">
        com.berkersoyluoglu.filters.TestFilter</AttributeAssignment>
      </Obligation>
    </Obligations>
  </Result>
</Response>
```

Fig. 5: Example of response returned by the PDP

E. PDP

PDP is where the policies are evaluated and the response is returned to PEP. So the implementation is as follows, when the user sending the request it sends a class that will be used to filter the result if the response is "Permit". Doing so enable us to have more control over the obligations. The filter class can be implemented as long as it implements the IFilter interface.

Figure 5 demonstrates an example response returned by the PDP.

V. Experimentation

We also had an experimentation setup to prove the concepts put forward in this paper. For this purpose, we implemented a prototype system whose architecture is shown in Figure 6. The setup consists of two pieces:

1) mobile applicaiton.
2) server containing the XACML logic

Mobile application represents a thin client that is only responsible for providing the input that is needed for the PDP to work e.g. location information. Server on the other hand is made up of several components depicted in Figure 6.

If we were to go over the steps of a request and a response in order of their execution.

1) Mobile application launch
2) Application starts up broadcasting a name for ZeroConf Networking.
3) Send that name and the location of the current device to the Server
4) User logins
5) User requests data
6) Application searches for devices near by sends their names with the data requests.
7) Server executes the policies with the data provided.
8) If the PDP returns access granted PAP calls the filter if there exists a filter provided by the user.
9) The result is returned to the user.

The server contains several XACML extensions for geographical operations we also ran several tests to see the performance impact of those extensions. Over 100000 runs

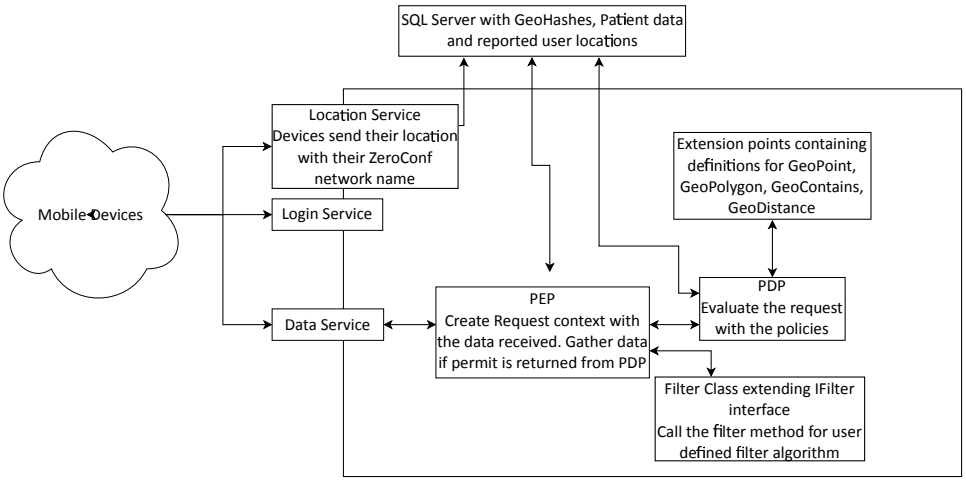

Fig. 6: Implementation architecture

the average time spent on executing policies containing geographical extensions are 6608 nanoseconds and policies that do not contain those extensions run on average 3109 nanoseconds. The object creation overhead should also be accounted for and those add up to 286 nanoseconds. To provide more security server checks for the nearby device list provided by the user while requesting the data. This list is checked against the nearby devices server knows and/or assumes are in the vicinity of the supposed location provided by the user.

VI. Conclusion

In this paper we identified a clear methodology for describing and implementing effective mechanisms for privacy enforcement via obfuscation within the XACML framework. We built this methodology upon our previous experience within the ITA project, notably [1] and [5].

Our methodology requires to adapt:

1) the policy structure;
2) the Obligation Service;
3) the Context Handler;
4) the PIP;

to encompass specific application domains and data sources.

In particular, whenever a policy can returns a "Permit" decision, an obligation must be issued and it must contains the relevant pieces of information necessary for performing obfuscation techniques: in the context of this paper we considered a simple metric value between 0 and 1, as in [5], but more articulated procedure can be easily envisaged.

Moreover, since the Obligation Service, the Context Handler and the PIP are defined as application-dependent, as since XACML3.0 allows arbitrary sets of attributes, no further modifications are needed to the standard. Any XACML-compliant policy is automatically valid within our methodology provided the requirement of specifying an obligation any time a "Permit" decision is issued.

Acknowledgements

The research described in this article was sponsored by US Army Research laboratory and the UK Ministry of Defence and was accomplished under Agreement Number W911NF-06-3-0001. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the US Army Research Laboratory, the U.S. Government, the UK Ministry of Defense, or the UK Government. The US and UK Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

References

[1] F. Cerutti, S. Chakraborty, G. R. de Mel, L. M. Kaplan, T. J. Norman, N. Oren, S. Pipes, M. Sensoy, M. B. Srivastava, and P. Sullivan. Managing information sharing in coalitions through credible obfuscation. Technical report, ITA, 2014.

[2] S. Chakraborty, N. Bitouzé, M. Srivastava, and L. Dolecek. Protecting data against unwanted inferences. In *Information Theory Workshop (ITW), 2013 IEEE*, pages 1–5. IEEE, 2013.

[3] S. Chakraborty, C. Shen, K. R. Raghavan, Y. Shoukry, M. Millar, and M. Srivastava. ipshield: A framework for enforcing context-aware privacy. In *11th USENIX Symposium on Networked Systems Design and Implementation (NSDI 14)*, pages 143–156, Seattle, WA, Apr. 2014. USENIX Association.

[4] P. McNamara. *Logic and the Modalities in the Twentieth Century*, volume 7 of *Handbook of the History of Logic*. Elsevier, 2006.

[5] S. Pipes, F. Cerutti, and S. Chakraborty. Initial Realization of Inference Management in Information Fabric. Technical report, ITA, 2014.

[6] L. van der Torre and Y. Tan. Contrarytoduty reasoning with preferencebased dyadic obligations. *Annals of Mathematics and Artificial Intelligence*, 27(1-4):49–78, Apr. 1999.

[7] XACML. eXtensible Access Control Markup Language (XACML) Version 3.0. 22 January 2013. OASIS Standard. http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html, 2013.