Security Policies in Dynamic Service Compositions

Julian Schütte[1], Hervais Simo Fhom[2] and Mark Gall[1]
[1]*Fraunhofer Institution for Applied and Integrated Security AISEC, Garching near Munich, Germany*
[2]*Fraunhofer Institute for Secure Information Technology SIT, Darmstadt, Germany*

Keywords: Service Composition, Policy Composition, Pervasive Systems.

Abstract: The paradigm of service composition emerged in the context of service oriented architectures, where it mainly referred to creating value-added services by combinitions of individual services. Nowadays, service composition is getting more and more dynamic and becomes part of pervasive systems. One of the major challenges in this context is to fulfill the security requirements of all involved parties without requiring human interaction to negotiate protection level agreements. In this paper, we propose an approach for composing access control decisions and obligations required by equitable policy domains on the fly. We show that our approach allows a policy-compliant collaboration without requiring the peers to reveal their individual rules and confirm its practicability by a prototype.

1 INTRODUCTION

Policies are essential when it comes to controlling the non-functional behavior of a system. Especially in heterogeneous and dynamic infrastructures, where collaborating endpoints as well as their preferences and security requirements are not known in advance, it must be possible to specify the desired behavior in terms of security and quality-of-service at an abstract level. A number of policy frameworks for specifying access control rules and obligations in such systems has been proposed and the usage of semantic web technology for separating domain knowledge from the actual policies has emerged as a promising approach (e.g., (Kagal et al., 2006; Toninelli et al., 2007; Nejdl et al., 2005)).

Most of these frameworks focus on settings where all entities are under control of a single administrative policy domain and fall short of appropriately combine security requirements of composed services, governed by different administrative domains. In general, two ways of combining policy domains are possible: *offline* and *online*. Offline approaches assume that the policies of all involved domains are known beforehand, follow a common specification and can be merged into a single, common policy that complies with each party's requirements. However, these assumptions do not hold in dynamic systems, where services are composed in an ad hoc fashion and parties are not willing to publicly reveal their policies.

In such cases, one needs to follow an online approach which combines policy decisions, rather than the policies themselves, in a way that does not violate the requirements of any domain. Especially for the latter approach, it is promising to include *metapolicies*, i.e. "policies over policies", in order to declare strategies for resolving inter-domain policy conflicts at run time.

In this paper, we describe a composition process for policy decisions from multiple domains. The proposed process resolves conflicts between access decisions and obligations, issued by different domains, and merges them into a single applicable decision which is in line with the policy of each involved domain. The composition process has been implemented as a prototype and integrated into the Apollon policy framework – a modular semantic-based policy framework for pervasive systems. It is based on a combination of Description Logic (DL) and Defeasible Logic (dl) to take into account the semantic policies of Apollon on the one hand, and the requirements of non-monotonic reasoning for conflict resolution on the other hand.

The rest of this paper is structured as follows. In Section 1.1, we give a motivating scenario and review work related to ours. In Section 2, we introduce the policy model and explain how decisions are be annotated. We explain the composition algorithm for resolving inter-domain policy conflicts in Section 3 and provide results of a prototype implementation in Sec-

Schütte J., Simo Fhom H. and Gall M..
Security Policies in Dynamic Service Compositions.
DOI: 10.5220/0004025302330238
In *Proceedings of the International Conference on Security and Cryptography* (SECRYPT-2012), pages 233-238
ISBN: 978-989-8565-24-2

tion 4. Finally, Section 5 concludes the paper.

1.1 Motivating Scenario

The general problem we address can occur in all kinds of distributed systems where different administrative domains are dynamically composed. For the sake of illustration, we have chosen the following e-Health scenario.

Consider a hospital running a virtual medical platform (VMP) which integrates health data about patients from various services, such as from the hospital's own electronic health record (EHR) database, or from external sources. The VMP allows legitimate users (e.g., hospital personnel or the patient herself) to access information that is relevant to them. We further assume that a doctor, Alice, uses the VMP system to collect information about the medical history of a patient Bob. As the VMP is merely a service aggregator, it relies on its backend services to provide the requested data. These services reside in independent administrative domains, so we cannot assume that they are willing to reveal their security policies to each other. As each domain features its own policy, a request from Alice to the VMP will trigger different and possibly conflicting policy decisions by the backend services. For example, while the hospital's EHR database can be accessed by Alice (because as a doctor of the hospital, she is a legitimate user) with the obligation that the access has to be logged and results have to be *AES-encrypted*, Bob might have set up an access control policy that allows only his family members to access his personal health info service, requires a notification email to be sent to Bob whenever somebody accesses his data, and further requires data to be *strongly encrypted* — Note the different abstraction level as opposed to the more specific requirement *AES*.

In that situation, the access decisions and required obligations of the hospital and Bob are in conflict and the VMP cannot return the requested data without violating at least one of the policies. Because the conflict does not occur between individual rules of a single policy, but between policies of equal and independent domains, it is also not possible to define a single "rule combining algorithm" to simply prefer one of the decisions. Furthermore, domains are not hierarchically structured, so it is not feasible to assume that the hospital's decisions should overwrite Bob's personal security policy, or vice versa. Figure 1 illustrates this conflict situation. In this paper, we will provide an approach to allow domains to annotate their policy decision with meta-information about how it may be combined with that of another domain so that

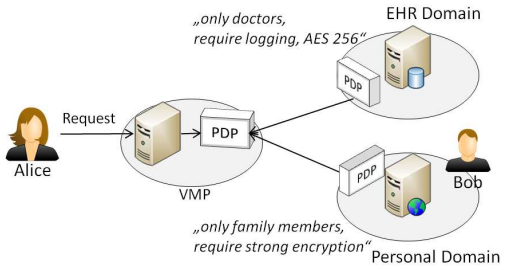

Figure 1: Conflict occurs in the composed VMP service between the decisions of EHR and Personal domain.

the VMP is able to derive an integrated decision that respects the policies of both domains.

1.2 Related Work

In (Cuppens et al., 1998), Cuppens et. al. describe conflict resolution in merged policies by applying "meta-strategies" which create a total preference order over the set of conflicting rules. While our approach is related to theirs, it differs from (Cuppens et al., 1998) in that we regard policies as private to each domain and rather focus at merging decisions than policies. In (Bonatti et al., 2002), an algebraic framework for composing access control policies from multiple domains is described. Our work focuses at merging obligations as a result from individual composition strategies, while the algebra in (Bonatti et al., 2002) is suited to express a composition strategy for authorization rules. Further work tackling the problem of composing policies of different domains has been done in (Wei and Yu, 2010), where the authors describe an approach of using Description Logic for policy representation, and apply reasoning over composed policies in order to detect possible conflicts. This approach is orthogonal to ours, as in (Wei and Yu, 2010), the authors assume that domains are willing to reveal their policies to each other. Also related to our work is (Lee et al., 2006), where WS-SecurityPolicy rules are combined using defeasible logic. As (Lee et al., 2006) focuses on the composition of WS-SecurityPolicy assertions, the approach does not allow each domain to use their own policy language, but rather presumes a common policy representation, which has to be revealed to the collaborating domains. In following, we rely on the description logics notations as used in (Baader et al., 2007) and defeasible logic as described in (Nute, 2003).

2 DECISION AND OBLIGATION MODEL

In contrast to *policy composition* approaches, we focus at *decision composition*. The advantage here is that each domain can foster its own policy model and representation which fit their needs and second that policies can remain private to a domain and do not have to be revealed for the sake of composition.

An access request comprises subject s, resource r and the required action a, whereas in the Apollon prototype, each of them is represented as a description logic class expression. If a value-added service combines multiple domains, this is done by sending requests with a *compose* action to each of them. With respect to the example from section 1.1 above, the request could be denoted as $req_{comp} = < VMP, EHR \sqcup Personal, compose >$.

Policy decisions are of the form $dec_p = \langle e, O \rangle$, where the effect $e \in \{permit, deny\}$ denotes if the access is to be granted or denied, and $O = \{o \,|\, o \sqsubseteq Executable\}$ denotes a set of obligations. By *obligations*, we understand actions which must be enforced before access to the requested resource is granted, so they refer to prerequisites required by protection level agreements, for example. We express an obligation as a DL subclass of the *Executable* concept, which is modeled as $Executable \sqsubseteq \forall hasName.string \sqcap \forall hasLoc.string \sqcap \forall hasType.string \sqcap \forall monitoredBy.Monitor \sqcap \forall hasParameter.Param$, where *hasLoc* defines the location of execution, so that the PEP will either execute the obligation locally or instruct any other peer to execute it.

Policy decisions can be annotated with composition constraints. Our assumption is that users are willing to relax the constraints of a policy for the sake of collaboration with other services. The acceptable level of relaxation will however depend on conditions such as the specific services involved in the composition. Therefore, we propose to use metapolicies in order to allow users to express their domain composition strategy at a more detailed level. Just as for policies, the Apollon framework is agnostic about the actual representation of metapolicies, but assumes the result of their evaluation (the *metapolicy decision*) to be in the format $dec_m = \langle \{\emptyset, permit, deny\}, \mathcal{C}, \mathcal{F}, \mathcal{A} \rangle$ where $\mathcal{C} \subseteq (Executable)^I$ denotes *compulsory* obligations, $\mathcal{F} \subseteq (Executable)^I$ denotes *forbidden* obligations and $\mathcal{A} \subseteq (Executable)^I \times (2^{(Executable)^I})$ denotes a set of *alternative* obligations. Function $(\cdot)^I$ relates to the *interpretation function* in Description Logic which maps individuals to classes. Compulsory obligations are actions which must be enforced by the PEP in every case and must not be overwritten by the decision of another policy domain. In contrast to that, set $\mathcal{F}$ of *forbidden* obligations comprises actions which must not under any circumstances be executed. Alternatives $\mathcal{A}$ to an obligation o have to be executed if the execution of o has been prevented by another policy domain. So, using metapolicies it is possible to express statements like "*access to the service is only allowed for family members and must be AES-encrypted, but when composed with a certified hospital service, doctors may also access the service and any encryption algorithm is acceptable*".

3 COMPOSITION OF ACCESS CONTROL DECISIONS

Policies in Apollon are modeled to a great extend in Description Logics. While this comes with a number of advantages, such as the ability to reason over policies, Description Logics are monotonic and thus it is not possible to defeat or inferred facts. However, this is required when combining decisions from multiple domains, so we use defeasible logic (Governatori, 2004) to extend it by non-monotonic features.

3.1 Transforming Decisions into Defeasible Logic

As a first step of the composition process, policy and metapolicy decisions from each PDP are transformed into defeasible logic (dl). A policy decision $dec_p = \langle e, O \rangle$ is transformed into a dl theory D where the effect and all obligations are denoted as defeasible rules (i.e., rules which may be overwritten later on). The effect is represented as a single variable e_p with $e_p = allow$ if $e = permit$ and $e_p = \neg allow$ if $e = deny$ and all obligations in O are represented as individual variables o_i:

$$\begin{aligned} [e_p]: \quad & \Rightarrow \quad e_p \\ & \Rightarrow \quad o_1 \\ & \vdots \quad \vdots \\ & \Rightarrow \quad o_n \end{aligned}$$

Then, the metapolicy decision is transformed into dl as follows:

1. If the metapolicy returns a *weak effect* ($\emptyset$), add the defeasible rule $[e_m] : \Rightarrow e_m^{weak}$ and a superiority relation $e_m^{weak} > e_p$ to D.
   Thereby, effect e_m^{weak} overwrites e_p, but is still marked as defeasible and thus allows other domains to overwrite it.

2. If the metapolicy requires the effect to be *strictly* enforced (*permit* or *deny*), add the strict rule $[e_m] :\rightarrow e_m^{strict}$ to D.
   This way, effect e_m^{strict} will be definitely provable and cannot be overwritten by another policy domain.
3. For all compulsory obligations $c \in \mathcal{C}$, add strict rules $\rightarrow c$ to D.
   This makes c definite provable and thus a compulsory requirement that cannot be overwritten.
4. For all obligations $a \in \mathcal{A}$ acting as alternatives to an obligation o_i, add strict rules $\neg o_i \rightarrow a$ to D.
   This allows a to be an alternative in case that o_i could not be derived, because it was defeated by a collaborating domain. Alternatives, in contrast to o_i, are written as strict rules and will thus be definitely provable so they cannot be overwritten by another domain.
5. For all forbidden obligations $f \in \mathcal{F}$, add a strict rule $\rightarrow \neg f$ to D.
   This way, f is definite not provable and it is avoided that a collaborating domain can set it as a strictly required obligation.

The resulting defeasible logic theory D of a single policy domain can then be written as:

$$
\begin{array}{lcl|rcl}
[e_p]: & \Rightarrow & e & & \rightarrow & c \\
[e_m]: & \rightarrow & e_m^{strict} & \neg o_i & \rightarrow & a \\
 & \Rightarrow & e_m^{weak} & & \rightarrow & \neg f \\
 & \Rightarrow & o_1 & e_m^{weak} & > & e_p \\
 & \vdots & \vdots & & & \\
 & \Rightarrow & o_n & & &
\end{array}
$$

Further, as each obligation refers in fact to a class expression in Description Logic, we need to add some rules in order to ensure that obligations which are already subsumed by other obligations do not occur twice in the final decision. For instance, in the example from section 1.1, one party requires *AES* while the other one requires *strong Encryption*, and we assume that the former is modeled as a subclass of the latter. In that case, we do not want to apply both obligations but rather the minimal subset satisfying all requirements. Therefore, for each obligation o_x subsumed by another obligation o_y (i.e., $o_x \sqsubseteq o_y$) we add rules to D to ensure that (1) the existence of o_x removes the need for another o_y (i.e., $o_x \rightarrow \neg o_y$) and (2) forbidding o_y also forbids o_x (i.e., $\neg o_y \rightarrow \neg o_x$). Note that this transformation from Description (DL) into Defeasible Logic (dl) differs from the one in (Governatori, 2004), where the author aims at using dl inference to reason over DL.

3.2 Evaluation of a Composed Decision

The defeasible logic theory from the previous section can be thought of as a policy decision where certain parts are compulsory while others may be overwritten, if necessary. If a PEP is part of multiple two domains A and B and thus receives two theories D_A and D_B, it will derive the final applicable decision as follows:

1. Rename labels in D_A and D_B to ensure uniqueness
2. Merge the theories from A and B into one: $D' = D_A \cup D_B$
3. Reason over D' and retrieve the set Q of conclusions. For the sake of readability we will use $\mathcal{P} \subseteq Q$ to denote provable facts, i.e. facts f which are definite ($+\Delta f$) or defeasibly ($+\delta f$) provable in D', and $\mathcal{N} \subseteq Q$ to denote non-provable facts $-\Delta f$ or $-\delta f$, respectively.
4. Test if access rights of A and B are compatible:
   If $allow \notin \mathcal{P}$ and $\neg allow \notin \mathcal{P}$, both domains strictly require conflicting access rights and are thus not compatible with each other.
   If $\exists o_i \in D'$ s.t. $o_i \in \mathcal{N}$ and $\neg o_i \in \mathcal{N}$, the required obligations of both domains conflict with each other, e.g., because A states a compulsory obligation which is forbidden by B.

If the test in (4) fails, the decisions of domain A and B cannot be combined with each other, because both domains strictly require conflicting access rights. In that case, it is not possible to apply both domain's policies likewise and the composed service cannot process the request without violating the security policy of one of the back-end services. As a consequence, access can be denied, or the user can be notified about the event and asked for taking a manual decision. Which of these actions is suitable depends on the actual application and has to be agreed between PDPs before composing their domains. Otherwise, if the decisions are compatible, all obligations denoted by literals in $\mathcal{P}$ will be executed and the respective access right enforced.

3.3 Properties of a Composed Decision

We will now show that the composition of policy decisions does not violate the constraints set by each individual decision: the composed decision must not require forbidden obligations, it must contain all compulsory obligations and it must contain at least alternative for a defeated obligation.

Given a defeasible logic theory $D = (F, R, >)$ and $\rightarrow \neg f \in R_s$ and assuming that $D \nvdash +\Delta f$, we can conclude that $D \vdash -\Delta f$.

As we cannot conclude $D \vdash +\Delta f$, according to the aforementioned assumption, there can be no fact f, as otherwise (1.1) would let us conclude $+\Delta f$. So, as $f \notin F$, rule (2.1) is fulfilled. Further, in order to fulfill (2.2), we need to show that for all rules r enabling f the antecedent is not definite provable. This is obviously the case, as otherwise (1.2) would lead to the conclusion, $+\Delta f$ which we excluded in our assumption. So, under the assumption that $D \nvdash +\Delta f$ we can conclude $-\Delta f$, according to (2.1) and (2,2).

Given a defeasible logic theory $D = (F, R, >)$ and $\rightarrow c \in R_s$ and assuming that $D \nvdash -\Delta c$, we can conclude that $D \vdash +\Delta c$.

As $\rightarrow c \in R_s$ and the empty antecedent can always be concluded, (1.2) can be fulfilled and $+\Delta c$ follows.

Given a defeasible logic theory $D = (F, R, >)$ and $\neg o \rightarrow a_i \in R_s$ and assuming that $D \nvdash -\Delta \neg a$, and $+\Delta \neg o$ we can conclude that $D \vdash +\Delta a$.

With $\neg o \rightarrow a_i \in R$ and $+\Delta \neg o$, (1.2) is fulfilled. Further, as we assumed that $D \nvdash -\Delta \neg a$, there are no conflicting definite conclusions and we can conclude $D \vdash +\Delta a$.

Obviously, showing that the desired properties hold in a merged dl theory is straightforward and by checking the merged dl theory for ambiguous conclusions for any literal in $\mathcal{F}$, $\mathcal{C}$ or $\mathcal{A}$, we are able to detect unresolvable conflicts between the policy decisions of different domains.

4 PROTOTYPE EVALUATION

In order to test the applicability of the proposed policy decision composition, we implemented a prototype in form of a module for the Apollon policy framework (Schütte, 2011) and set up a test scenario which corresponds to the use case from section 1.1, in order to validate the functionality of the prototype and investigate its runtime behavior.

Then we measured the average run time of the prototype implementation in the Apollon policy framework, including the evaluation of the access requests, transformation into the dl theory, as well as the actual composition process. As a platform we used a Pentium i7 2.7 GHz with 4 GB of RAM, hosting PEP and PDP locally, so that we can leave network latencies aside. Two factors are of interest when evaluating the run time of our approach: the number of metapolicies and the number of obligations in an annotation. The former is relevant for the PDP when annotating the policy decision, because the more metapolicies the PDP must evaluate, the slower the annotation process will be. In contrast to that, the number of obligations influences the time for merging two annotated decisions, as with the number of obligations, the size of the defeasible logic theory which is derived from the annotations increases.

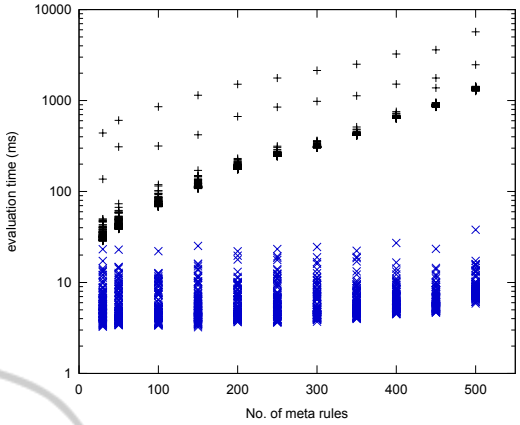

Figure 2: Performance of decision and annotation (+) and combination (x) with increasing number of meta rules.

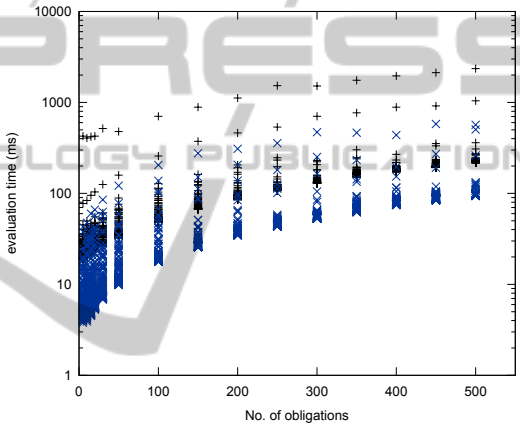

Figure 3: Performance of decision and annotation (+) and combination (x) with increasing number of obligations.

We investigated the run time with 30 to 500 metapolicies. Whether these numbers are realistic depends of course on the use case, but as metapolicies are intended to be more generic than the actual access control rules, we deem the lower end of the range as more likely. When looking at Figure 2, it becomes obvious that the time for deciding an access request and annotating a policy decision increases about exponentially (note the logarithmic scale), while the time for merging the annotated decisions remains more or less constant.

Figure 3 shows the results of a second test, in which we kept the number of metapolicies constant, but continuously increased the number of obligations within the annotated policy decisions. In this test, run time increases about linearly for both, the decision taking and the merging phase (note again the logarithmic scale).

These tests confirm the theoretical claim that reasoning over defeasible logic requires linear time, while satisfiability checking in Description Logic is a significantly harder problem and can require up to NEXPTIME, and that this is in fact noticeable for the input dimensions we are dealing with. In the context of our service composition approach, this means that the merging of multiple annotated decisions can be done quite efficiently and scales well with the number of services (i.e., policy decisions), but deciding access requests by reasoning over Description Logic has limits in size of the rule set and might not fulfill the performance requirements of each scenario. Nevertheless, there are many optimization possibilities, ranging from pre-computation and caching, on to reducing the complexity of the problem.

5 CONCLUSIONS

We have tackled the problem of composing policy decisions of collaborating, but independently managed policy domains. We proposed an approach for annotating policy decisions with information from metapolicies and using it in a policy composition process to automatically derive a decision that suits the policies of all involved domains. In contrast to other approaches, we do not assume policies of each domain to be publicly available, but rather allow each domain to use its own policy model and language, as long as the actual decision annotation can be mapped to the format we use. The composition process is based on defeasible description logic and has been implemented in form of a module for the Apollon policy framework. By means of the prototype implementation, we were able to show that proposed solution is applicable and show satisfying performance.

As part of our future work we further investigate ways of handling and possibly predicting unresolvable conflicts between policy domains. Moreover, extending the proposed solution in a way that it supports chains of composed policy domains is currently in progress.

REFERENCES

Baader, F., Horrocks, I., and Sattle, U. (2007). *Handbook of Knowledge Representation*, chapter 3 Description Logics, pages 135–180. Elsevier. ISBN 0444522115.

Bonatti, P., De Capitani di Vimercati, S., and Samarati, P. (2002). An algebra for composing access control policies. *ACM Transaction on Information System Security*, 5:1–35.

Cuppens, F., Cholvy, L., Saurel, C., and Carrère, J. (1998). Merging security policies: Analysis of a practical example. In *Proc. of the 11th IEEE Computer Security Foundations Workshop (CSFW)*, pages 123–137. IEEE Computer Society Press.

Governatori, G. (2004). Defeasible description logics. In Antoniou, G. and Boley, H., editors, *Rules and Rule Markup Languages for the Semantic Web*, volume 3323 of *Lecture Notes in Computer Science*, pages 98–112. Springer Berlin / Heidelberg.

Kagal, L., Berners-Lee, T., Connolly, D., and Weitzner, D. (2006). Using semantic web technologies for policy management on the web. In *Proc. 21st National Conf. on Artificial Intelligence (AAAI)*.

Lee, A., Boyer, J. P., Olson, L. E., and Gunter, C. A. (2006). Defeasible security policy composition for web services. In *Proc. of the fourth ACM workshop on Formal methods in security*, FMSE '06, pages 45–54, New York, NY, USA. ACM.

Nejdl, W., Olmedilla, D., Winslett, M., and Zhang, C. C. (2005). Ontology-based policy specification and management. In *In 2nd European Semantic Web Conference (ESWC)*, pages 290–302. Springer.

Nute, D. (2003). Defeasible logic. In Bartenstein, O., Geske, U., Hannebauer, M., and Yoshie, O., editors, *Web Knowledge Management and Decision Support*, volume 2543 of *LNCS*, pages 151–169. Springer Berlin / Heidelberg.

Schütte, J. (2011). Apollon: Towards a modular semantic policy framework for pervasive systems. In *Int'l Conf. Security and Cryptography (SECRYPT)*.

Toninelli, A., Montanari, R., Kagal, L., and Lassila, O. (2007). Proteus: A semantic context-aware adaptive policy model. In *Proc. Int'l Workshop on Policies for Distributed Systems and Networks (POLICY)*.

Wei, W. and Yu, T. (2010). The design and enforcement of a rule-based constraint policy language for service composition. In *Social Computing (SocialCom), 2010 IEEE Second International Conference on*, pages 873–880.