The Why and How of Delegations in Distributed Security Systems

Abstract

In this paper we address the main questions associated with delegations and the role they play in security architectures for distributed systems; *why* they are required and *how* they can be implemented. In distributed and dynamic systems like the Semantic Web or pervasive computing environments, describing comprehensive policies in terms of every possible resource and user is not always possible since neither resources nor users can be completed enumerated a-priori. Even if all users and resources can be pre-determined, it is not feasible to develop complete policies because it is not possible to account for every access right or task required in these dynamic environments. We believe that policies should be as simple as possible and control should be decentralized, that is authorization should be possible by more than just a few key entities and tasks/responsibilities should be delegate-able. We believe that delegations are the key as they allow policies to be less exhaustive and allow policies to be dynamically updated. We address two kinds of delegations: delegation of rights and delegation of responsibility. Delegation of rights allows access rights to be transferred whereas delegation of responsibility allows obligations to be transferred. We describe our policy language, which is based on deontic concepts and speech acts, that allows both these kinds of delegations to be represented and reasoned over. Speech acts like delegation, revocation, request and cancel are an integral part of the design as they provide decentralized control by allowing policies to be dynamically modified. We discuss the delegation protocols that we have developed and describe our integrated approach to policies and speech acts. We describe our prototype implementation of a security system, which supports decentralized control through policies and delegation statements represented in logic and a semantic language, Resource Description Language Schema.

1 Introduction

In distributed and dynamic systems like the Semantic Web and pervasive computing environments, it is not always possible to develop complete policies because it may not be conceivable to enumerate the users and resources of the system. Even in cases where the users and resources are known, developing comprehensive policies may not be achievable because it is not possible to correctly identify every access right or task that will ever be required. We propose the use of delegations to allow security policies in these environments to be less comprehensive, more flexible and easier to manipulate. We present a unified approach to delegations, both of rights and responsibility, in distributed systems using a policy language based on speech acts and deontic concepts of rights, prohibitions, obligations, and dispensations (Dispensation is a waiver from an obligation). We address two kinds of delegations: delegation of rights and delegation of responsibilities/obligations. Delegating a right is giving a user the right to perform a certain action or use a certain a resource that he/she previously did not have the right

to. On the other hand, delegating a responsibility is transferring an obligation to the delegatee, making the delegatee responsible for performing a certain action and allowing the delegator a dispensation. Related research in delegations tends to concentrate either on delegation of responsibility or delegation of rights but not both [16, 3]. Though there has been a lot of research in the area of policies, we believe our policy language and protocols have a unique advantage - they provide a integrated approach to delegations in distributed systems by tightly coupling policies, delegations and delegation protocols. Policies are represented as rights, prohibitions, obligations and dispensations over actions and conditions. Speech acts namely delegations, revocations, requests and cancellations, on actions, rights and obligations are permitted. These speech acts create new policies involving rights, prohibitions, obligations and dispensations. Delegations in particular cause rights, obligations and dispensations. We have developed several different delegation protocols as there are a number of ways in which a delegation can be interpreted. These protocols provide policy makers flexibility in implementing delegations. We have also developed a prototype implementation of a policy engine that interprets policies and speech acts and that incorporates some of these delegation protocols.

In Section 2 we provide an overview of our policy language and we illustrate through examples how policies are represented. Section 3 describes the different kinds of delegations and their representation in our policy language. In Section 4, we briefly recapitulate the importance and the need of delegations. Section 5 discusses details of the protocols developed and our prototype implementation. We discuss some of the related research in delegations and conclude with a summary in Section 7.

2 Policy Language

Our policy language is modeled on deontic concepts of rights, prohibitions, obligations and dispensations [18]. We believe that most policies can be expressed as what an entity can/cannot do and what it should/should not do in terms of actions, services, conversations etc., making our language capable of describing a large variety of policies ranging from security policies to conversation and behavior policies. The policy language has domain independent ontologies but also requires domain specific ontologies. The former includes concepts for permissions, obligations, actions, speech acts, operators etc. The latter is a set of ontologies, used by the entities in the system, which defines domain classes (person, file, deleteAFile, readBook) etc. and properties associated with the classes (age, num-pages, email). This allows entities to describe policy in the terms of the information that they commonly use and are familiar with.

The language includes two constructs for specifying meta-policies that are invoked to resolve conflicts; setting the modality preference (negative over positive or vice versa) or stating the priority between policies [15]. For example, it is possible to say that in case of conflict the Federal policy always overrides the State policy.

Another important aspect of the language is that it models speech acts like delegation, revocation, request and cancel that allow policies to be less exhaustive and allow for decentralized security control. A delegation causes an access right to be added. A revocation speech act nullifies an existing right (whether policy based or delegation based) of an entity. An entity can request another entity for a right or to perform an action on its behalf and an entity can also cancel any previously made request. Though policy languages like Ponder [7], ASL [8], PolicyMaker [14] and KeyNote [5] support delegation of

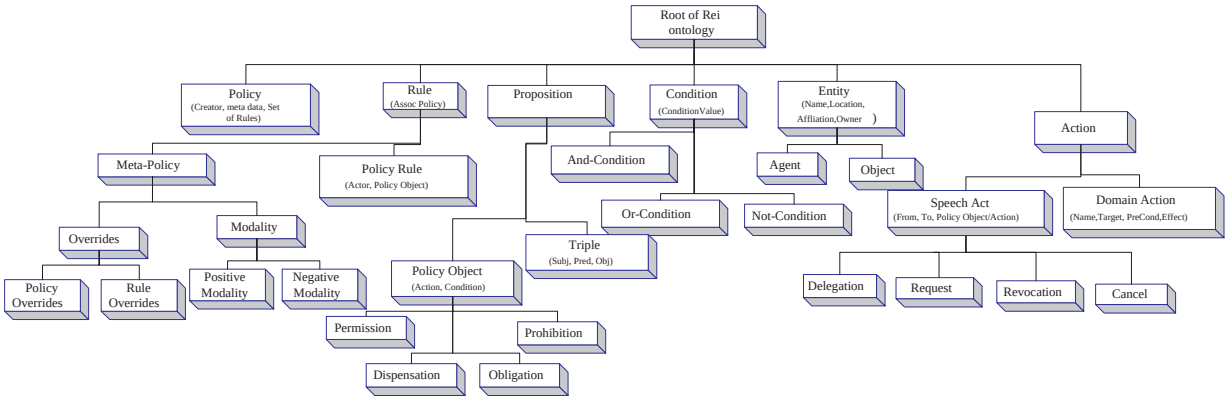

Figure 1: Policy Ontology

rights, they do not take any other speech acts into consideration causing their solutions to be less unified and the control to be more centralized.

The policy language supports individual policies as well as group and role based policies in a uniform manner by allowing domain dependent representations for roles and/or groups to be included in the conditions of the policy rules.

The policy language includes a RDF-S [23] ontology for specifying entities, objects, policies, credentials, and domain specific information. Figure 1 illustrates the ontology in RDF-S for policies, meta policies, actions, and domain specific information.

2.1 Representation of Actions and Conditions

Though the actual execution of services/actions is outside the system, the policy language includes a representation of them that allows more contextual information to be captured and allows for greater understanding of the action and its parameters. This allows policies to be defined over different aspects of an action and not just its identity. For example, it is possible to say 'John' has the right to all services (e.g. print, fax, xerox) on any color printer from ABC Labs. Action specifications include the target resources they act on, pre-conditions and effects. Complex actions can be composed using action operators of *sequence, non-deterministic choice, once* and *iteration*.

Simple conditions are built from domain specific information and complex conditions can be built from simple conditions using *and, or* and *not*. A condition is of the form "entity from ABC Labs" or "in the role of student".

Using these actions and conditions, a policy maker is able to create policy rules. There are four kinds of policy rules, each representing a deontic concept : right, obligation, prohibition, and dispensation. Each policy rule is a tuple of action and associated requirements. In order to associate these policy objects with users, the policy maker uses the *has* predicate in logic or the PolicyRule class in RDF-S creating rights, obligations etc. for the users.

Example 1 : Simple condition. A web service specifies that anyone from ABC Labs can access its service which is identified by 'Service1'.

The policy rule in logic is as follows :

has(X, right(X, 'Service1', origin(X, 'ABC Labs')))

where, service1 is an allowable action, and origin(X, 'ABC Labs') is a domain dependent condition specified by the deployer. It is assumed either the matchmaker that the service is registered with or the service itself it able to verify the origin of an entity using traditional schemes like PKI, SPKI etc.

We are experimenting with ways to encode our simple policy rules in RDF and Notation3 [4]. Eventually, we hope to use whatever standard or standards emerge from the semantic web community for representing rules in RDF. Here is our N3 example in one framework we are using.

```
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix policy: <http://someuri.com/ontologies/policy#>.
@prefix itt:  <http://someuri.com/ontologies/itdomain#>.
@prefix time:  <http://someuri.com/ontologies/time#>.
@prefix : <#>.

:v1
 a policy:Variable;
 policy:PolicyRule
  [a policy:Right;
   policy:actor :v1;
   policy:action [a policy:Action; policy:actionName "service1"];
   policy:condition [a rdf:Statement;
                     rdf:subject :v1;
                     rdf:predicate itt:origin;
                     rdf:object <http://abclabs.com/>]].
```

The rest of our examples in the paper are in logic due to space constraints.

Example 2 : Complex Condition. Consider a right associated with 'John'. *'John'* has the right to either perform action *printBW* followed by repeated executions of *printColor* or perform action *fax* once. He only has the right while he satisfies the associated conditions.

has('John',
right('John', nond(seq(printBW, iteration(printColor)), once(fax)),lab-member('John','AI')))

This work on action operators is based on the ideas discussed by Smith and Cohen [17] and is not supported by the other policy languages in our bibliography.

Example 3 'John' has the right to access all actions that act on 'HPPrinter021'.

has('John', right('John', Action, action(Action, 'HPPrinter021', PreCond, Effects)))

Example 4 : Request. Continuing with Example 2, 'Jane' is not a lab member of 'AI' and she requests John to give her the right to perform *printBW*.

requestSpeechAct('Jane', 'John', right('Jane', printBW,Conditions))

If accepted and if 'John' has the right to delegate, 'John' sends a delegation to 'Jane' giving her the right to use printBW as long as she is an employee of 'ABC Labs'.

delegateSpeechAct('John', 'Jane', right('Jane', printBW,employee('Jane','ABC Labs')))

2.2 Meta Policies

As there is a potential for conflicts, the language also contains meta-policy specifications for conflict resolution. Meta-policies are policies about how policies are interpreted and how conflicts are resolved dynamically. Conflicts only occur if the policies are about the same action, on the same target but the modalities (right/prohibition, obligation/dispensation) are different. Meta policies in our system regulate conflicting policies in two ways by specifying priorities and precedence relations [13]. Using a special construct, *overrides*, the priorities between any two rules or two policies can be set. In case of conflicts, the rule/policy with the higher priority is chosen. It also possible to indicate the order in which the priorities are checked; policy level or rule level first. The other way of resolving conflicts is by specifying which modality holds precedence over the other in the meta-policies. The policy maker of the system can associate a certain precedence for all policies associated with a set of actions or a set of entities satisfying a certain set of conditions.

3 Delegations

- *The act of delegating, or investing with authority to act for another; the appointment of a delegate or delegates.*
- *One or more persons appointed or chosen, and commissioned to represent others, as in a convention, in Congress, etc.; the collective body of delegates; as, the delegation from Massachusetts; a deputation.*
- *(Rom. Law) A kind of novation by which a debtor, to be liberated from his creditor, gives him a third person, who becomes obliged in his stead to the creditor, or to the person appointed by him.*

- Webster's Revised Unabridged Dictionary

In short, a delegation from one entity to another either gives the later some rights or obligation. A delegation of a right in security systems causes the delegatee to gain some additional access rights, whereas a delegation of responsibility causes the delegatee to gain some new responsibility. Both these kinds delegations can also be one of two types : while delegation and when delegation. In the following subsections, we discuss these types of delegations in greater details.

3.1 Delegation of Right

When a right is delegated from one entity to another, the latter is given the power to act on the formers behalf. Delegations of rights are very important to distributed systems like the internet because web entities may not be able to project who will use them or pre-establish all the desirable requirements of the entities who should use them. Delegations allow access rights of an entity to be propagated to a set of trusted entities, without explicitly changing its policy or requirements. Consider a DAML-S web service that is only accessible by users of a certain role in a certain organization. These users also have the right to delegate the ability to use this service to other users in the organizations for a certain time period. A delegation from one of these users to another employee of the organization permits the latter to use the web service without any explicit modification of the policy or code of the service, while the constraints associated with the delegation are met. The ability to make delegations makes it possible to change access rights of users dynamically.

Representation A delegation of a right allows an entity to transfer its right to another entity or group of entities that satisfy a certain set of conditions. A delegation of a right, if valid, causes an access right to be created. Only an entity with the right to delegate can make a valid delegation.

delegateSpeechAct(Sender, Receiver, right(Receiver, Action, Condition)) AND
Receiver satisfies Conditions $\longrightarrow$
has(Receiver,right(Receiver, Action, Condition))

Example 5. Assuming that John has the right to delegate, he delegates to Mark the right to print on the lab printer as long as he is working on the same project as John.

delegateSpeechAct('John', 'Mark',
right('Mark', printLabPrinter,(project('John', SomeProject), project('Mark', SomeProject))))

action(printLabprinter, [laserPrinter123], queue(laserPrinter123,0), queue(laserPrinter123, 1))

The policy language defines three types of inter-related rights associated with each action, out of which the last two give certain delegation rights.

- Right to execute : Possessing this right allows the entity to perform the action.
  has(Entity, right(Entity, Action, Condition))
  where Action is the action and Condition are the conditions on execution
- Right to delegate execution : If an entity possesses the right to delegate the execution of an action, it can delegate to other entities the right to perform the action but the entity cannot perform the action itself.
  NOT has(Entity, right(Entity, Action, ECondition))
  has(Entity, right(Entity, Action1, Condition1))
  where, *Condition1* are the conditions on the *Entity*
  Action1 is *delegate(right(SomeEntity, Action, Condition))*
  This right gives the possessor the right to delegate the previous right, the right to execute.
- Right to delegate delegation right : The entity can delegate to another entity or a group of entities the right to further delegate the right to perform the action and delegate this delegation right as well. Though at this point the right should have been divided into right to delegate the right to execution and the right to delegate the right to delegation, we decided to combine them as we expect that the conditions on every right will take care of the propagation of the delegation. This right gives the possessor the right to delegate the previous right, the right to delegate execution and the right to delegate the delegation itself.
  NOT has(Entity, right(Entity, Action, ECondition))
  has(Entity, right(Entity, Action1, Condition1))
  where, *Condition1* are the conditions on the delegator, *Entity*
  Action1 is delegate(right(SomeEntity, Action2, Condition2))
  Condition2 are the conditions on the delegatee
  Action 2 is delegate(right(SomeOtherEntity, Action, Condition))

These three rights force conditions on the executor of the action, the delegator of the action and whom the right can be delegated to. An entity has the right to a certain action (including speech acts) if it possesses the right or if it has been delegated the right. It should satisfy the conditions associated with the innermost right of execution of every delegation in the chain. Each delegator should satisfy the condition on the delegation of

the delegation before it in the chain and the delegatee conditions of all previous delegations. If any entity fails any delegator condition, the delegations from that point on are invalid. The policy engine ensures that circular delegations are not allowed, i.e., an entity cannot delegate a right to itself or to another entity who delegates it back to the delegator or to a previous delegator in the delegation chain.

Example 6. This example demonstrates the working of cascaded delegations of rights.

- Amy has the right to delegate the right to execute printOnePageHP and she delegates this right to Tim. Tim can only execute printOnePageHP if he satisfies both the condition associated with the speech act (employee('Tim', 'ABC Labs') and the condition associated with Amy's delegation right (group-member(X, Group)).

  has('Amy', right('Amy', delegate(right(X, print, group-member(X,Group))), employee('Amy', 'ABC Labs')))

  delegateSpeechAct('Amy', 'Tim', right('Tim', print, employee('Tim', 'ABC Labs')))

- John has the right to delegate the right to delegate the right to execute printOnePageHP and he delegates this right to Tim. Tim can delegate as long as he is satisfies group-member(X, Group) from John's right to delegate and employee(tim, 'ABC Labs') from the delegation. However, John's right and the delegation also place conditions on whom Tim can delegate to, in this case to lab members of 'AI' and employees of 'ABC Labs'.

  has('John', right('John', delegate(right(X, delegate(right(Y, print,lab-member(Y, 'AI'))), group-member(X,Group))),employee('John','ABC Labs')))

  delegateSpeechAct('John', 'Tim', right('Tim', delegate(right(Y, delegate(right(Z, print, employee(Z,'ABC Labs'))), employee(Y, 'ABC Labs'))),employee('Tim', 'ABC Labs')))

- Tim delegates to Jane the right to delegate the right to execute printOnePageHP. Jane must satisfy the conditions associated with the earlier delegations and rights in the delegation chain in order to be able to delegate the right to printOnePageHP and the delegation will only be valid if the entity she delegates to satisfy all the associated conditions as well.

  delegateSpeechAct('Tim', 'Jane', right('Jane', delegate(right(X, print, group-member(X, Group))),(employee('Jane', 'ABC Labs'), time-now(morning))))

3.2 Delegation of Responsibility

Delegation of responsibility or obligations is considered the transfer of obligation from the delegator to the delegatee. In the real world, delegation of responsibility is usually carried out down the management chain. However, this is not always the case. The policy language allows different delegation rights to be assigned to obligations. In the absence of delegation rights on obligations, a default case is assumed; either all delegations are prohibited or permitted.

Representation Obligations are represented as policy rules of the following format

has(X, obligation(ToWhom, Action, Condition))

Unlike axioms of deontic logic, possessing an obligation does not guarantee a right to perform the task. There are two rights associated with delegation of obligations.

- Permitted to delegate the obligation : If an entity is permitted to delegate an obligation, the entity may delegate the obligation.
  has(Entity, right(Entity, Action1, Condition1))
  where, *Condition1* are the conditions on the *Entity*
  Action1 is *delegate(obligation(ToWhom, Action, Condition))*
- Prohibited from delegating the obligation : If an entity is prohibited from delegating an obligation, the entity may not delegate the obligation.
  has(Entity, prohibition(Action1, Condition1))
  where, *Condition1* are the conditions on the *Entity*
  Action1 is *delegate(obligation(ToWhom, Action, Condition))*

Example 7. John requests Marty to write a report and allows Marty to delegate this responsibility. Marty accepts, but then delegates this obligation to Amy.

requestSpeechAct('John', 'Marty', writeReport) AND
delegateSpeechAct('John', 'Marty', right('Marty', delegate(obligation('John', writeReport, Condition1)))) $\longrightarrow$
has('Marty', obligation('John', writeReport, Condition)) AND
has('Marty', right('Marty', delegate(obligation('John', writeReport, Condition1))))

delegateSpeechAct('Marty','Amy', obligation('John', writeReport, Condition1))

3.3 When and While delegations

We also identify two types of delegation, while-delegations and when-delegations. A *while-delegation* forces all following delegators to satisfy its conditions in order to be true. Whereas a *when-delegation* requires the immediate delegator to satisfy its conditions only at the time of the delegation and not after. For example, consider a when-delegation giving Jane the right to delegate when she's an employee. All the delegations that Jane made while she was an employee hold even after she leaves. On the other hand, a similar while-delegation will fail once the delegator leaves the company. The while delegation is the default delegation type.

Example 8. The following represents the difference between a while and a when delegation.

- When-Delegation
  delegateWhenSpeech('Mark', 'Matthew', right('Matthew', Action, Condition)) AND
  Matthew satisfies Condition $\longrightarrow$
  has('Matthew', right('Matthew', Action, Condition))

  Matthew no longer satisfies Condition $\longrightarrow$
  has('Matthew', right('Matthew',Action, Condition))
- While-Delegation or Default Delegation
  delegateSpeech('Mark', 'Matthew', right('Matthew',Action, Condition)) AND
  Matthew satisfies Condition $\longrightarrow$
  has('Matthew', right('Matthew', Action, Condition))

  Matthew does not satisfy condition $\longrightarrow$
  NOT has('Matthew', right('Matthew', Action, Condition))

4 *Why* are Delegations Required ?

Delegation has usually been seen as a concept useful in security and access control because it allows policies to be relatively simple, and allows the rights and responsibilities of entities to be configured dynamically. A policy for all printers in a lab could be defined so that managers have the right to delegate the right to print and the right to re-delegate this right to any employee of the company. However, if any employee that they delegate to, misbehaves in any way, the system will hold the associated manager responsible. This forces the managers to be careful with their delegations, while at the same time allowing the rights on the printers to propagate. Consider another example, if a task needs to be performed the policy designates a manager to be responsible for it. However, under certain circumstances she can delegate the task to her subordinate changing his obligations on the fly.

Delegations are also very natural and useful in modeling the related topics of trust and privacy as well as some aspects of cooperative behavior. Consider a multiagent system (MAS) composed of autonomous, but (mostly) cooperative, agents. In a MAS, as in human society, permissions and obligations are part of the glue that hold things together. They are readily transferred from one agent to another, either implicitly via the conventions of the protocols associated with communication (e.g., accepting a question obligates one to respond with an answer), or explicitly through delegations. When transferred, their meaning can be narrowed in some cases and widened in others.

5 *How* can Delegations be implemented ?

Clearly a policy language that is capable of expressing delegations and rights about delegations is required, however there are some other requirements for integrating delegations into a security system: delegation management, how delegations are stored and managed, and protocols for delegations, how delegations are interpreted. In Section 2 we describe the syntax of our policy language and show through examples how it can be used to specify delegation speech acts, delegation rights and other deontic concepts that are modified by delegations. In this section, we discuss different schemes for delegation management and the protocols we implement. We assume that different systems will incorporate protocols based on their requirements or will allow a delegator to specify which protocol he/she is using allowing the system to interpret the delegation correctly.

5.1 Delegation Management

Though the delegation model is very important to security in distributed systems, managing delegations in a widely distributed and dynamic environment is rather difficult as certificate chains can be arbitrarily long.

We suggest four schemes for managing delegations

Chained Delegation : Every entity stores a chain of delegation certificates leading to its own delegation, in order to validate its delegation and attaches this chain to any delegation it makes. This is not feasible because each chain could be very long and there could be several delegations for every entity. To reduce the number of certificates in a chain, certificate reduction could be used [2], but the original delegator may not be accessible.

Centralized : To avoid handling and processing chains of delegations, all delegations can be addressed to the resource/service or the entity responsible for the resource/service. However this scheme has two problems; it is rather centralized and the delegator may not be able to access the appropriate entity at the time of the delegation.

Decentralized : In decentralized management, the entity has to go out and collect relevant delegations itself in order to validate a request based on delegations.

Delegations on the Web : The last scheme is to continue using delegation chains, but instead of storing

the chains within the entity, the chains could be stored on web pages. In order to prove it has a certain ability, an entity could point to a certain delegation on its delegation page. This delegation in turn would refer to a delegation on the page of the entity who made the delegation. By traversing these delegation links, the requested entity would be able to verify the delegation and decide whether or not to authorize the request.

Choosing the right delegation management scheme depends on the system requirements including the kinds of web entities involved and their capabilities. For example, in a pervasive system where devices tend to be handheld and resource poor, it is probably better to have delegations on the web and have the device store a URI to its delegations. However, in ad-hoc environments where access to the internet is not always guaranteed, it is advisable to use a centralized method where devices store all delegations locally.

5.2 Protocols

Delegations can be used in different ways and we have developed a couple of protocols for each delegation type. These protocols can be further specialized for a certain system by incorporating domain knowledge into the constraints. For example, by adding restrictions on 'roles' for the protocols for delegation of obligations, it is possible to restrict delegations of this sort to flow down through the management chain. Schaad and Moffett [16] include a protocol for delegation, but other researchers depend only on a simple implicit protocol (delegation without request) for delegation of rights.

Protocols for delegation of rights A delegation of a right is an authorization. An authorization is either given when required by the delegatee or when deemed necessary by the delegator.

- Delegation by Request: In this protocol, an entity needs to access a certain resource or perform a certain action and does not have the right to do so. The entity requests an authorized entity for the right. If the authorized entity agrees, it makes a delegation. If the authorized entity has the right to delegate and if the delegatee meets the conditions associated with the delegation, a new access right is created. However, it is possible to have a default (implicit) policy that allows all entities that have a certain right to delegate it.
  requestSpeechAct(Delegatee, Delegator, right(Delegatee,Action,NoCondition)) $\longrightarrow$
  delegateSpeechAct(Delegator, Delegatee, right(Delegatee, Action,ECondition)) AND
  has(Delegator, right(Delegator, delegate(right(Delegatee, Action, DCondition)))) AND
  Delegator meets DCondition AND
  Delegatee meets ECondition and all rights in the delegation chain $\longrightarrow$
  has(Delegatee, right(Delegatee, Action, Condition))
- Delegation without Request : In certain situations, an authorized delegator delegates certain rights to the latter without the latter asking for it. As an example, if a visitor arrives, it is common courtesy to provide internet access to him/her without his/her explicit request.
  delegateSpeechAct(Delegator, Delegatee, right(Delegatee, Action,ECondition)) AND
  has(Delegator, right(Delegator, delegate(right(Delegatee,Action, DCondition)))) AND
  Delegator meets DCondition AND
  Delegatee meets ECondition and all rights in the delegation chain $\longrightarrow$
  has(Delegatee, right(Delegatee, Action, Condition))

Protocols for delegation of obligations In most organizations, delegation of responsibility flows down the management hierarchy. However, as role is considered domain specific information, we classify delegation of obligations in two classes: firstly based on how it is done, whether it is a command or a request, and the second based on how the obligation is affected, whether the obligation is transferred entirely to the delegatee, transformed into a set of obligations, or shared between the delegator and the delegatee. A delegation protocol is usually composed of one type from each class. For example, one possible protocols could be interpreted

as command and transfer of obligations or as request and shared obligations. Usually the right to delegate an obligation has to be present before a delegation can be made. However, it is possible to have a default (implicit) policy that allows all entities that have a certain obligation to delegate it.

1. How is the obligation done : Whether the delegatee is asked to take on the obligation or commanded to.

    - Command : In the default case, when a delegation of an obligation is made, the delegatee is forced to take on the obligation.
      has(Delegator, obligation(ToWhom, Action, Condition1)) AND
      has(Delegator, right(Delegator, delegate(obligation(ToWhom, Action, Condition2)))) AND
      delegateSpeechAct(Delegator, Delegatee, obligation(ToWhom, Action, Condition3)) $\longrightarrow$
      One of the protocols affecting the obligation
    - Request : In this protocol, the delegator asks the delegatee whether the delegation can be made before actually carrying out the delegation. This is similar to a request for a right.
      has(Delegator, obligation(ToWhom, Action, Condition1)) AND
      has(Delegator, right(Delegator, delegate(obligation(ToWhom, Action, Condition2)))) AND
      requestSpeechAct(Delegator, Delegatee, delegate(obligation(ToWhom, Action, Condition3))) $\longrightarrow$
      ACCEPT or REJECT
      If ACCEPT, then one of the protocols affecting the obligation

2. Affect on Obligation : If a delegation protocol is a command or request, it can be further interpreted based on how the associated obligation is affected.

    - Transfer of Obligation : In this protocol, when an obligation is delegated, the delegator is no longer responsible (has a dispensation from the obligation) and the entire responsibility is transferred to the delegatee.
      has(Delegator, dispensation(ToWhom, Action, Condition1)) AND
      has(Delegatee, obligation(ToWhom, Action, Condition3))
    - Transform of Obligation : When a delegator uses this protocol, the obligation is transferred to the delegatee as before, however, the delegatee is now obliged to the delegator and the delegator is still obliged to the entity he/she was previously obliged to. This is similar to the work by Schaad and Moffett [16].
      has(Delegator, obligation(ToWhom, Action, Condition31)) AND
      has(Delegatee, obligation(Delegator, Action, Condition3))
    - Sharing of Obligation : In this protocol, the obligation is transferred to the delegatee as before and the delegatee is obliged to the entity that the delegator was previously obliged to. However, the delegator is now responsible for informing the original obliged entity that the responsibility has been transferred to the delegatee.
      has(Delegator, obligation(ToWhom, InformAction, Condition4)) AND
      has(Delegatee, obligation(ToWhom, Action, Condition5))

5.3 Prototype Implementation

Using the policy engine and a monitoring service that provides one of the delegation management schemes, it is possible to implement a policy based framework that supports decentralized control through delegations. Following are the steps in establishing such a security system

- Set up the policy engine and develop policies based on the domain knowledge
- Decide on a default policy, right to delegate is implicit or explicit

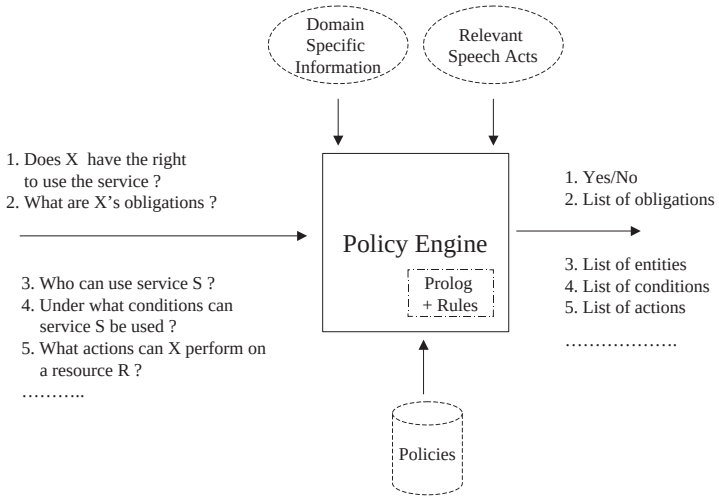

Figure 2: Structure of the Policy Engine

- Set up a monitoring service that decides which delegations are relevant to a security decision and retrieves them based on the delegation management scheme
- Decide which protocols will be supported for delegation of rights and obligations and specialize them for the system under consideration using domain knowledge. Or provide an option by which an entity can make a delegation and select the appropriate protocol

We have developed a policy engine in Prolog [21] that interprets and reasons over the policies, related speech acts and domain information, expressed either in Prolog or RDF-S, to make decisions about applicable rights, prohibitions, obligations and dispensations. It incorporates all the protocols for delegation of rights and we are currently developing the protocols for delegation of obligations. It is possible to extend the policy ontology of Figure 1 with other ontologies, which will be accepted and reasoned over by the policy engine. All domain specific information can be specified as separate ontologies and domain specific actions can be subclasses of DomainAction of the policy ontology.

If, while processing a request, the policy engine comes across a conflict, it uses associated meta-policies to resolve it. For example, 'John' wants to print to 'HPPrinter021', he sends a request to 'HPPrinter021'. "HP-Printer021' finds that 'John' has both a right to use it and is prohibited from using it. However, the right has been given higher priority than the prohibition and 'HPPrinter021' permits 'John' to print. The policy engine is able to answer questions about whether a request is valid, what the pending obligations of an entity are, under what conditions a request for a certain action is authorized etc. Figure 2 illustrates the functionality of the policy engine. In order to make correct policy decisions, we assume the presence of a monitoring service that sends all relevant speech acts to the policy engine. The policy engine is implemented in Prolog with a Java interface.

The engine is used as a reasoning engine and is consulted before every request for an action/service and before taking any secure action. Based on the answer of the engine, the controller of the entity (could be the entity itself) decides whether to allow or deny the request or go ahead with the action. The engine is also capable of answering other queries related to policy making: who can perform a certain action, who can perform any action on a certain resource, what conditions must an entity possess in order to use a certain action/resource, what unfulfilled obligations does entity X currently have, etc. These queries can be used to decide whether the policy does what it is supposed to or whether there are any inconsistencies in it.

We have implemented two security frameworks based on an earlier version of the policy language and a few of the delegation protocols, (i) a security architecture for pervasive environments [19], and (ii) a secure

e-healthcare system in pervasive computing environments [20], where access rights of hospital personnel to patient information change based on their credentials (including their role in the hospital, certificates etc.), their relationship to the patient and any delegations they may possess.

6 Related Research

In this section, we compare and contrast our system with other research in this area with reference to flexibility in representation of policies and delegations, types of delegations, delegation management schemes and delegation protocols.

A logical approach for authentication, access control and trust is defined by Abadi et al [1, 10], that focuses on delegation as a representation of the 'speaks for' relation. The system is based on a formal semantics that explains how delegations interact with various combination operators for principals. However, delegations are unconstrained, i.e. a delegation gives the delegatee the right without any conditions attached. This is not very practical and may cause security breeches. This approach also has no support for delegation of obligations.

Jajodia et al. describe the specifications of a language based on stratified logic that tries to support different access control policies [8]. Their Authorization Specification Language (ASL) allows users to not only specify authorization policies, but also specify the way the decisions over these policies are made. ASL depends heavily on the authors' understanding and interpretation of groups and roles, whereas in our policy language, these concepts belong entirely to the domain in which it is being used, and can be interpreted as required by the domain. This language, though a step in the right direction, is complicated because it consists of a large set of interdependent rules that the user has to fully understand in order to use. ASL does not provide explicit protocols for delegation or address delegation of obligations.

Li et al [11, 12] define a language and protocol for delegation based access control, which tends to focus on authorization based on properties of entities. It specifies a language for authorization in open systems, which allows policies, credentials and requests to be represented uniformly. This logic language includes a delegation clause that has a delegation depth to control how many re-delegations are possible, and allows delegations to complex entities. However, delegations are not fully integrated into policies and it does not take into consideration the delegation of obligations. It also does not support speech acts other than delegation.

Blaze et al. [6, 14] define trust management as creating policies, assigning credentials to users, and checking if the credentials of the keyholder conform to the policy before granting it access. They use a programming language to define a policy based on certificates, and a query engine, PolicyMaker. PolicyMaker is a centralized component which is able to interpret policies and answer questions about access rights. Though delegation is addressed, only one implicit protocol is used (delegation without request) and delegation of obligations is not addressed. The Strongman project uses KeyNote [5], the next generation of PolicyMaker, to build secure information systems by automating key and policy management [9]. KeyNote is a trust management system, which binds public keys to access rights. The system is versatile but rather too closely bound to digital certificates as it cannot handle other domain information or link into non PKI systems. It includes a the same notion of delegation as PolicyMaker

Though, sophisticated RBAC models include work on delegation between roles [3], they tend to allow more coarse grained delegations, for example delegation of all rights associated with a role. Whereas, we allow role to be domain specific information and permit delegations to be as fine grained as required. It is possible to delegate a single right or a set of rights that satisfy a certain set of conditions. This encompasses the delegation provides by RBAC, as it is possible to delegate all rights associated with a role to another set of entities that belong to some other role.

Schaad and Moffett discuss an interesting protocol for delegation of obligations [16], where an obligation when delegated can be shared between the delegator and the delegatee. The delegatee is responsible for the task,

but the delegator is responsible for making sure that the delegatee performs the task.

Ponder [7] allows general security policies to be specified. Ponder is a declarative, object oriented language for specifying security and management policies. It allows policy types to be defined to which any policy element can be passed to create a specific instance. This seems to be a useful ability and our policy language allows this to be done naturally. Our language allows actions and policy objects to be defined separately and allows them to be linked dynamically to subjects. Ponder includes a notion of delegation as authorization but does not discuss the right to delegate or the ability to delegate the right to delegate or the delegation of obligations.

In summary, our research has the following contributions : (i) flexible representation of policies and delegations in both logic and RDFS, (ii) unified approach to both delegation of rights and delegation of obligations, (iii) policies and delegations are tightly coupled and are affected by each other, (iv) formal protocols for delegation of rights and delegation of obligations, and (v) a prototype implementation that validates our claims.

7 Discussion and Summary

We have currently designed the policy language and have developed a policy engine that accepts and reasons over policies in logic and RDF-S. We are currently working on an OWL [22] ontology to allow policies to be described in OWL. Based on a centralized scheme for delegation management, the protocols for delegation of rights are already incorporated into the policy engine and we are in the process of developing the protocols for delegation of obligations. The policy engine is in Prolog and has a Java interface, allowing it to be used as a reasoning engine.

We have designed and developed two similar policy based security frameworks using the delegation schemes discussed in the paper, i) a security architecture for pervasive environments [19], and (ii) a secure e-healthcare system [20].

In this paper, we discussed the importance of delegations in security for distributed systems and described the two main types; delegation of rights and delegation of responsibilities. We described our policy language, which allows delegations to be represented and reasoned over. We discussed the types of delegations and illustrated how our delegation protocols work. Our policy language and delegation protocols provide a comprehensive and unified approach to using delegations, both of rights and responsibilities, in distributed security systems.

References

[1] Martín Abadi, Michael Burrows, Butler Lampson, and Gordon Plotkin. A calculus for access control in distributed systems. *ACM Transactions on Programming Languages and Systems*, 15(4):706–734, September 1993.

[2] Tuomas Aura. Distributed Access-Rights Managements with Delegations Certificates. In *Secure Internet Programming*, pages 211–235, 1999.

[3] Ezedin Barka and Ravi Sandhu. Framework for role-based delegation models. In *Annual Computer Security Applications Conference*, 2000.

[4] Tim Berners-Lee. Notation 3. http://www.w3.org/DesignIssues/Notation3, 2001.

[5] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The keynote trust management system version, 1999.

[6] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The role of trust management in distributed systems, 1999.

[7] N. Damianou, N. Dulay, E. Lupu, and M. Sloman. The ponder policy specification language. In *The Policy Workshop 2001, Bristol U.K.* Springer-Verlag, LNCS 1995, Jan 2001.

[8] S. Jajodia, P. Samarati, and V. S. Subrahmanian. A Logical Language for Expressing Authorizations. In *IEEE Symposium on Security and Privacy. Oakland, CA*, 1997.

[9] A Keromytis, S Ioannidis, M Greenwald, and J Smith. The strongman architecture. In *Third DARPA Information Survivability Conference and Exposition (DISCEX III), Washington, D.C. April 22-24*, 2003.

[10] Butler Lampson, Martn Abadi, Michael Burrows, and Edward Wobber. Authentication in Distributed Systems: Theory and Practice. In *ACM Transactions on Computer Systems, November, 1992*. 1992.

[11] Ninghui Li, Joan Feigenbaum, and Benjamin Grosof. A Logic-based Knowledge Representation for Authorization with Delegation. In *PCSFW: Proceedings of The 12th Computer Security Foundations Workshop*. IEEE Computer Society Press, 1999.

[12] Ninghui Li, Benjamin N. Grosof, and Joan Feigenbaum. A practically implementable and tractable delegation logic. In *In Proceedings of IEEE Symp. on Security and Privacy, held Oakland, CA, USA, May 2000*, 2000.

[13] Emil C. Lupu and Morris Sloman. Conflicts in policy-based distributed systems management. *IEEE Transactions on Software Engineering*, 1999.

[14] M.Blaze, J.Feigenbaum, and J.Lacy. Decentralized Trust Management. *Proceedings of IEEE Conference on Privacy and Security*, 1996.

[15] Jonathan Moffett and Morris Sloman. Policy conflict analysis in distributed systems management. *Journal of Organizational Computing*, 1993.

[16] Andreas Schaad and Jonathan Moffett. Delegation of obligations. In *IEEE Policy Workshop*, 2002.

[17] Ira A. Smith and Philip R. Cohen. Toward a semantics for an agent communications language speech-acts. In *Thirteenth National Conference on Artificial Intelligence and the Eighth Innovative Applications of Artificial Intelligence Conference, Vol. 2*, 1996.

[18] Author suppressed. Title suppressed. In *Fourth IEEE International Workshop on Policies for Distributed Systems and Networks*, 2003.

[19] Author suppressed. Title suppressed. *ACM - Kluwer Mobile Networks and Nomadic Applications(MONET) : Special Issue on Security in Mobile Computing Environments*, 2003.

[20] Author suppressed. Title suppressed. In *Fifth International Workshop on Enterprise Networking and Computing in Healthcare Industry, Santa Monica, June*, 2003.

[21] Swedish Institute of Computer Science Swedish Institute. SICStus Prolog. http://www.sics.se/ sicstus/, 2001.

[22] Frank van Harmelen, Jim Hendler, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. Owl web ontology language reference - w3c working draft 31 march 2003. http://www.w3.org/TR/owl-ref/, 2003.

[23] W3C. Resource Description Framework. W3C Recommendation, http://www.w3.org/TR/rdf-schema/, 2002.