Recycling Authorizations: Toward Secondary and Approximate Authorizations Model (SAAM)

Konstantin Beznosov
Laboratory for Education and Research in Secure Systems Engineering (LERSSE)
Department of Electrical and Computer Engineering
University of British Columbia

Technical Report

LERSSE-TR-2005-01

March 2005

Abstract: In large and complex enterprises, obtaining authorizations could be communicationally and/or computationally expensive, and, due to infrastructure failures, some times even impossible. This paper establishes the concept of recycling previously made authorizations for serving new authorization requests. It introduces secondary and approximate authorizations model (SAAM) with the semantics of matching best suitable approximate authorizations.

1 Introduction

Although, every authorization decision to be enforced should ideally have been made specifically for the authorization request in question, it is not always possible due to the failures in the underlying infrastructure. Some times, it is not even desirable due to the prohibitively high cost of making authorizations in the case of complex policies or expensive to obtain information used for policy evaluation. With the business and policy decisions commonly made to be just "good enough" and security mechanisms implemented and configured to be broad, imprecise inexpensive authorizations could provide a viable alternative to the all-or-nothing model supported by conventional authorization systems.

This paper introduces the concept of recycling previously made authorizations for serving new authorization requests. By putting forth Secondary and Approximate Authorizations Model (SAAM), it attempts to define precisely the notions of primary vs. secondary and precise vs. approximate authorizations. Another contribution of this paper is in suggesting a method of selecting the best approximate authorization for a given request from a set of existing authorizations. The method is applicable to a large class of authorization policies that satisfy two basic requirements: monotonic policies, and the abstraction of subjects as sets of name-value pair attributes. By adding another assumption, that an authorization policy could identify those subject attributes that were critical for arriving at the final authorization decision, the method for selecting the best approximate authorization is improved further.

The rest of the paper is organized as follows. Section 2 develops the argument against precise and in favour of secondary and approximate authorizations. Basic model is introduced in Section 3, whereas advanced models are presented in Section 4. Related work is discussed in Section 5. Conclusions are drawn and future work is discussed in Section 6.

2 Authorizations: Primary vs. Secondary And Precise vs. Approximate

As with other types of decisions, one is better off with enforcing authorization decisions that have been made specifically for the authorization request in question and using information from the request about the corresponding subject, object, required permission(s), etc. We refer to such authorizations as primary and precise. An authorization decision is *primary* if and only if it is made for the authorization request in question, i.e., not for some other request. Otherwise an authorization is *secondary*. A primary authorization is also *precise*.

Primary and precise authorizations could simply be unavailable to the enforcement function or other entity at the time, or could take too long to make. In this section, we argue that primary and precise authorizations are not always possible, desirable, or necessary, and that secondary and approximate ones could be used instead.

2.1 An Argument Against Precise Authorizations

We distinguish between secondary and approximate authorizations because a secondary authorization could be either precise or approximate. An authorization (decision) is *precise* if and only if it is made based on the information about the subject, object, permission, and environment exactly matching the corresponding information in the authorization request. Otherwise, such a decision is *approximate*. Note that the same decision could be precise for one authorization request and approximate for another. Primary and precise authorizations are not always possible, desirable, or necessary.

They are not always possible because the authorization engine(s) responsible for making the authorization is/are unavailable due to the failure (transient, intermittent, or permanent) of the network, software located in the critical path (e.g., OS), hardware, or even due to a miss-configuration of the supporting infrastructure. Consider, for instance, a massive-scale enterprise with hundreds of thousands of networked computers. Even if the mean time to failure (MTTF) of each computer (and all its critical software and hardware components) were one year, in an enterprise with half a million of computers, over 1,300 would stop every day.[1] With the average availability of every machine maintained at the level of 99.9% (somewhat unrealistically high expectation for such a large population), 500 computers would be unavailable at any given moment.

Precise authorizations are not always desirable due to the cost of arriving at them. In a large-scale enterprise, making authorizations could be computationally expensive due to the heterogeneity and the large size of the object and subject populations. The overall delay of requesting and obtaining an authorization could be comparable to, or larger than, the time of processing the corresponding application request, making the authorization overhead prohibitively expensive.

Authorizations don’t necessarily have to be precise because other (even more critical) decisions in business environments are made to be just “good enough.” Businesses (and

[1] Some enterprises, such as Amazon.com, are not far away from having that many computers with reliability as short as two months [12].

governments, as the recent events in Iraq, showed [2]) often make decisions based on incomplete information for the issue on the table. Furthermore, taking into account that protection mechanisms are commonly designed, implemented, and configured to be *broad* [1], the necessity of precise authorizations is reduced even further.

2.2 An Argument in Favour of Secondary and Approximate Authorizations

Secondary and approximate authorization decisions are expected to be useful mainly when the supporting enterprise infrastructure is partially unavailable for making and delivering primary decisions by a deadline, a case expected to be frequent for massive-scale enterprises with large numbers of computers, objects, and subjects. While waiting for a primary authorization, a system can speculatively compute or otherwise obtain a secondary, and probably approximate, authorization based on prior decisions (made for other authorization requests) cached locally or obtained from its neighbours. By the time the system times out on an authorization request from a remote authorization server, it could continue application processing by enforcing the best approximate authorization.

3 Basic Model – SAAM$_B$

This section defines more precisely elements of the basic model, SAAM$_B$. We first define several essential and auxiliary elements, such as authorization request and decision, request identicity and equivalence. Then, we use them to define primary/secondary and precise/approximate authorizations. We start with a central building block, authorization request:

> **Definition 1** (**authorization request**) An *authorization request* is a five-tuple (s, o, p, e, i), where $s \in S$ is a subject, $o \in O$ is an object, $p \in P$ is a permission, $e \in E$ is environment, and $i \in I$ is identity of the request.

In the basic model, we do not assume any specific structure and/or semantics of any of the five elements that comprise a request. This generality enables the basic model to be universally applicable to any authorization system. In the following sections, however, additional assumptions on the structure and/or semantics of request elements are imposed in order to devise useful semantics for authorization approximation.

We promote request identity to be an explicit element of authorization requests to distinguish between primary and secondary authorizations, as it will become clear in Definition 7. Request identity, however, is not just a convenience abstraction. It is commonly supported by the underlying technologies for matching authorizations to the corresponding requests. For example, in middleware based on the semantics of remote procedure calls (RPC) such as CORBA [10], EJB [11], DCOM [5], and DCE [4], an object request broker (ORB) uses request ids to pair the outgoing requests on remote objects with the incoming replies. Format and representation of request identities are technology-specific, e.g., Universal Unique IDentifier (UUID), Universal Resource Identifier (URI) [3].

Environment is an optional representation of request time and other attributes of the environment provided to the enforcement and/or authorization function(s) and could be used for computing authorizations.

Definition 2 (**authorization**) An *authorization* is a tuple (r, d), where $r \in R$ is an authorization request, as defined in Definition 1, and *decision* $d \in D$ is a result of the authorization granting or denying access.

We leave structure of *decision*s undefined to enable not only binary *grant*/*deny* decisions, but also other types such as conditions and obligations, defined, for example, in UCON [7]. The only assumption about d is that it contains enough information to answer decisively the question if access should be granted or denied:

Invariant 1 For any decision $d \in D$, it is possible to derive if access should be allowed, provided all specified in d obligations and/or conditions, if any, are met.

For simplicity, we will refer to specific elements of authorization requests and decisions by concatenating instance variable with the element name while separating them with a dot.

Example 1 For an authorization request r, $r.i$ is the identity element of r.

Example 2 For an authorization decision *a*, *a.r.o* is the object element of the request, for which the decision was made.

We also need to define what it means for two requests to be equivalent and identical:

Definition 3 (**identical requests**) Authorization requests *r* and *r*' are *identical iff* $r.i \equiv r'.i$.

Definition 4 (**function identical_requests**) Boolean function *identical_requests(r, r')* returns True *iff* requests *r* and *r*' are *identical* according to Definition 3.

Since, more than one request could be made in regards to the same object, subject, permission, and environment, some requests could differ only in their identities. We refer to such requests as equivalent.

Definition 5 (**equivalent requests**) Authorization requests *r* and *r*' are *equivalent* iff $r.s \equiv r'.s \wedge r.o \equiv r'.o \wedge r.p \equiv r'.p \wedge r.e \equiv r'.e$.

Definition 6 (**function equivalent_requests**) Boolean function *equivalent_requests(r, r')* returns True *iff* requests *r* and *r*' are *equivalent* according to Definition 5.

To ensure that all requests with the same identities are exactly same, we require that if two requests are *identical* they are also *equivalent*:

Invariant 2 $\forall$ *r* and *r*' from *R*, *identical_requests(r, r')* $\Rightarrow$ *equivalent_requests(r, r').*

It follows from definitions 3 and 5, and Invariant 2 that the only difference between equivalence/identicity is the inequality/equality of the identity elements of the requests. Using definitions 1 to 6, we can now define primary, secondary, precise, and approximate authorizations.

Definition 7 (**primary and secondary authorizations**) For an authorization request *r*, an authorization decision *a* is *primary iff* *r* and *a.r* are *identical*. Otherwise, *a* is *secondary* for *r*.

Definition 8 (**function primary_authorization**) Boolean function *primary_authorization (r, a)* returns True *iff* authorization *a* is *primary* for request *r* according to Definition 7.

Clearly, False result of *primary_authorization* means that the authorization is *secondary*. A similar function operating on a set of authorizations could also be defined:

Definition 9 (**function select_primary_authtorization**) For a given authorization request *r* and a set of authorization decisions *A*, function *select_primary_authtorization (r, A)* returns either an authorization *a* from *A* such that *primary_authorization* (*r, a*), or 0.

Definition 10 (**precise and approximate authorizations**) For an authorization request *r*, an authorization decision *a* is *precise iff r* and *a.r* are *equivalent*. Otherwise, *a* is *approximate* for *r*.

Definition 11 (**function precise_authorization**) Boolean function *precise_authorization* (*r*, *a*) returns True *iff* authorization *a* is *precise* for request *r* according to Definition 10.

It follows directly from definitions 3, 5, 7, and 10 that if for request *r* authorization *a* is *primary*, it is also *precise* for *r*.

In our model, we also require that for any request any two *precise* authorizations should have same decisions:

Invariant 3 $\forall\ r \in R,\ \forall\ a \in A,\ \forall\ a' \in A$: $precise\_authorization(\ r,\ a\ ) \wedge precise\_authorization(\ r, a'\ ) \Rightarrow a.d \equiv a'.d$.

The above invariant does not exclude time-sensitive policies because time is part of the request's *environment* element, *e*. Therefore, two requests that differ only in when they are made would have different value of the time component in their environment elements, and hence would differ.

Next, we introduce a convenience function for selecting precise authorizations from a set of candidates:

Definition 12 (**function select_precise_authtorizations**) For a given authorization request r and a set of authorization decisions A, function *select_precise_authtorizations (r, A)* returns a subset $A' \subseteq A$ such that $\forall\ a \in A'$: *precise_authorization* (r, a).

Note that *select_precise_authorizations*(…) might return an empty set when A contains no precise authorizations for r.

3.1 Selecting Approximate Authorizations

Given a set of authorizations A and an authorization request r, one can select some authorization a from A such that a would be the best authorization approximation for r in A. In a general case, a is selected in the following order:

1. If *select_primary_authorization*(r, A) $\neq$ 0, then a = *select_primary_authorization*(r, A). Otherwise,
2. if A' = *select_precise_authorization*(r, A) $\wedge$ $A' \neq \varnothing$, then a is assigned to any[2] authorization from set A'. Otherwise,
3. *best_approximate_authorization*(r, A).

To define function *best_approximate_authorization*(…), we make further assumptions on the structure and semantics of authorization requests and decisions, as described in the following section.

4 Advanced Models

In this section, we make additional assumptions on the structure and semantics of authorization requests, making it possible to define function *best_approximate_authorization*(…).

Definition 1 does not impose any structure or semantics on *subject*, *object*, *permission*, *environment*, or *identity*. To refine the semantics of authorization approximation further, we will make an assumption in the following sub-section that a

[2] Due to Invariant 3, it does not matter which one from A' is chosen.

subject is a set of attributes. In this section, the following generic structure for attributes and a convenience function for retrieving them are defined:

Definition 13 (**attributes**) Attribute $attr \in ATTR$ is a tuplet of name and value: (n, v), where $n \in N$ and $v \in V$.

Definition 14 (**attribute subsets**) For two sets of attributes A and B, $A \subseteq B \Leftrightarrow \forall\, a \in A\ \exists\, b \in B\colon a.n \equiv b.n \wedge a.v \equiv b.v$.

Definition 15 (**function attributes**) For any subject $s \in S$, object $o \in O$, permission $p \in P$, and environment $e \in E$, function $attributes(\ ) \subseteq ATTR$ returns a set of attributes of that element.

We do not assume any specific format or structure of *name* and *value* elements of attributes, although some architectures refine attribute formats further. For instance, a subject attribute in CORBA Security [6] is defined as a triplet of *type*, *authority* (who issued the attribute), and *value*. A simple translation of CORBA subject attributes into SAAM could be done by concatenating authority and value to represent value in our model and using CORBA's attribute *type* as *name*. In XACML [13], subject attribute is a triplet of attribute id, data type, and value. Again, without loss of expressiveness, one could translate XACML's subject attributes into SAAM by combining *id* and *data type* of the attribute into SAAM's *name*. Having defined the notion of attributes, we are ready to introduce other SAAM models.

Also, Invariant 1 allows us to introduce a convenience function that reduces a decision into a binary form:

Definition 16 (**function access_allowed**) For any decision $d \in D$, function *access_allowed*(d) returns True *iff* access should be allowed, provided all specified in d obligations and/or conditions, if any, are met. Otherwise, it returns False.

Subsequently, we define another convenience function also used in SAAM_B refinements:

Definition 17 (**function authorized**) For some subject s, object o, permission p, and environment e, Boolean function $authorized(\ s, o, p, e) \Rightarrow$

($\exists$ authorization a: $a.r.s \equiv s \wedge a.r.o \equiv o \wedge a.r.p \equiv p \wedge a.r.e \equiv e \Rightarrow$ *access_allowed*($a.d$)).

This function returns True if the access control policy would allow access if the corresponding authorization request were submitted. Note that here we assume that the authorization engine can be queried, whereas before our reasoning was limited only by existing authorization requests and decisions.

4.1 SAAM$_{SM}$—Subject Attributes with Monotonic Policies

In this section, we define *best_approximate_authorization*(…) introduced in Section 3.1 by making two assumptions. Firstly, we assume that subjects are sets of attributes that have structure according to Definition 13. Secondly, we assume that policies used for making authorization decisions are monotonic, i.e., consist of only positive rules. We refer to the refinement of SAAM$_B$ based on these two assumptions as SAAM$_{SM}$.

In order to further refine *best_approximate_authorization*(…) for SAAM$_{SM}$, we define *dominates* partial order relation on subjects:

Definition 18 (**dominates relation**) For any two subjects s and s', s *dominates* s' *iff* $\forall$ o, p, and e: *authorized*(s', o, p, e) $\Rightarrow$ *authorized*(s, o, p, e).

That is, s is authorized whenever s' is authorized. Observe that *dominates* relation is reflexive, asymmetric, and transitive. We use '≥' as a shorthand for '*dominates*', e.g., $s \geq s'$. Also note that $\forall$ s and s': $s \geq s' \wedge s' \geq s \Rightarrow s \equiv s'$.

It might seem that the only way to determine if two subjects are in *dominates* relationship is to check their rights for every combination of object, permission, and environment. However, since in SAAM$_{SM}$ monotonic policies and subject representation as a set of attributes are assumed, then for any two subjects s and s': $s' \subseteq s \Rightarrow s \geq s'$. This observation allows us to have an efficient and reliable, albeit conservative, way of determining if two subjects are in *dominates* relationship: if one is a superset of the other than the former dominates the latter. From now on, we will assume that *dominates* relationship is tested using this method.

Another auxiliary term to be used in defining *best_approximate_authorization*, is of maximal set of suitable authorizations:

> **Definition 19** (**suitable authorization**) For a request *r* and authorization *a*, *a* is *suitable* for *r iff* $a.r.o \equiv r.o \wedge a.r.p \equiv r.p \wedge a.r.e \equiv r.e$.

> **Definition 20** (**maximal suitable authorizations**) For a request *r* and a set of authorizations *A*, *A'* is a *maximal set of suitable authorizations* for *r* on *A iff* ($\forall\ a' \in A'$: a' is *suitable* for *r*) $\wedge$ ($\forall\ a'' \in (A\text{-}A')$: a'' is not *suitable* for *r*)

We can now define a simple version of *best_approximate_authorization*(…), in which, for a given authorization request *r*, the function returns any *suitable* authorization whose subject is dominated by *r*'s subject. However computationally efficient implementations of such semantics could be, the result returned by this version of the function depends not only on the request *r* and set *A* of known authorizations, but also the order of iterating through elements of *A*.

To define another version, although more computationally expensive, of this function with implementation-independent semantics, we introduce the notion of least dominated element of a set of subjects/requests/authorizations. Such an element is dominated by the subject in question and it also dominates any other element in the set dominated by that subject:

> **Definition 21** (**least dominated element of set**) For a given subject *s* and a set *M of subjects/requests/authorizations*, subject/request/authorization m $\in$ *M* is *least dominated subject*/request/authorization in regards to *s iff* $\forall\ m' \in M$: s *dominates* subject element of m' $\Rightarrow$ subject element of m *dominates* subject element of m'.

Now, we define semantics of *best_approximate_authorization*(*r*, *A*) as follows. It first selects set *A'* of *maximal suitable authorizations* for *r* on *A*. Then, the *least dominated authorization* in *A'* in regards to *r.s* is used as the result returned by the function. The following is a more precise definition:

> **Definition 22** (**function best_approximate_authorization**) For a set *A* of authorizations and a request *r*, function *best_approximate_authorization*(*r*, *A*)

returns the *least dominated authorization* in regards to *r.s* of the *maximal subset of suitable authorizations* for *r* on *A*.

With the above definition, SAAM_{SM} completes the semantics of selecting approximate authorization for a given request (Section 3.1) by defining *best_approximate_authorization*(). To do so, this advanced model adds the assumptions about subjects represented as sets of name-value attributes and close-world policies with only positive rules to SAAM_{B}. In other words, SAAM_{SM} is SAAM_{B} extended with Definition 13 through Definition 22.

The efficiency of SAAM_{SM} can still be improved without imposing over-constraining assumptions on the model. We discuss these improvements in the next section.

4.2 SAAM_{SMS} – Significant Attributes

To improve SAAM_{SM}, we take advantage of the fact that not all subject attributes influence authorization decisions. For example, subject id and group membership attributes are not used in authorizations made against pure RBAC [8] policies. Therefore, if two subjects have same roles activated, then they would have same rights even if their id and group membership attributes were different. Due to the monotonic policy assumptions in SAAM_{SM}, additional attributes could only increase the rights of a subject, which allows us to care only about *significant attributes*—those attributes that did influence a particular authorization:

> **Definition 23** (**significant subject attribute**) For an authorization *a*, a subject attribute $attr \in a.r.s$ is *significant iff* authorization decision for otherwise same request but with the subject lacking *attr* would result in a decision different from *a.d.*

Now, we extend SAAM_{SM} into SAAM_{SMS} by assuming that authorization engines mark *significant* subject attributes in the authorizations they return. One possible marking option is to leave only significant attributes in *a.r.s* and remove others. With this option, no changes to *best_approximate_atuhroization()* function defined in SAAM_{SM} are necessary in order to take advantage of significant attributes. For example, with pure RBAC policies, only role attributes would constitute subject elements of authorizations, and any subject whose roles are a superset of the ones present in an authorization would *dominate* the subject element of that authorization.

5 Related Work

To the best of our knowledge, no prior work on the topic of recycling authorizations has been published. The literature considers only the issues related to arriving at and enforcing access control decisions. Although one can argue that selecting the best approximate authorization out of existing ones is just another form of making authorizations, the key difference here is that $SAAM_{SM(S)}$models assume no knowledge about the details of the authorization policies, except that they are monotonic. In some sense, the authorization logic treaded in SAAM as an oracle, which is only known to return same authorization in response to same request and that its policies are monotonic.

Due to the generality of the model and little or no assumptions about the structure and format of authorization elements (Definition 2), SAAM seems to be compatible with most traditional and modern access control models, including UCON [7].

6 Conclusions and Future Work

This paper explores the question of recycling existing authorizations (a.k.a., access control decisions) for serving new authorization requests. Scalable and efficient recycling of authorizations is critical when the authorization logic is not available (e.g., due to infrastructure failures), or caching is employed to optimize performance. We introduce the notions of primary vs. secondary as well as precise vs. approximate authorizations. Another significant contribution of this paper is an initial version of secondary and approximate authorizations model. We define its basic ($SAAM_B$) and advanced ($SAAM_{SM}$ and $SAAM_{SMS}$) variations that take advantage of additional assumptions about the representation of subjects as sets of simple attributes as well as the policies being monotonic to define suitable semantics for selecting the best approximate authorization for a given request.

Further improvements of SAAM are certainly possible. One is to take into account the partial order relations defined over subject attributes in some access control models, e.g., lattice-based MAC [9] and RBAC with role hierarchies. This could make testing of subjects for *dominates* relationship more effective and less conservative. Another possible improvement is to assume an attribute-based structure of the request environment and to take advantage of the distinction between significant and insignificant environment attributes (similarly to *significant* subject attribute, Definition 23). For example, if time of the request is insignificant attribute of the environment, then more authorizations could qualify as *suitable* for a given request. One more candidate for future improvement is studying partial order relation(s) among objects (and possibly permissions), where rights over one object could (perhaps transitively) imply rights over other objects. Yet another opportunity for extending $\text{SAAM}_{\text{SM(S)}}$ is to drop the assumption about the policies being monotonic. SAAM variations presented in this paper implicitly assume one policy authority therefore avoiding conflicts or inconsistencies among authorizations. An interesting generalization would be to support multiple authorities.

7 Acknowledgements

The author is thankful to Frank Piessens for providing helpful feedback on the initial ideas presented in this paper. This work has been supported in part by NSERC Discovery Grant.

References

[1] Bishop, M. *Computer Security: Art and Science*. Pearson Education, Inc., Boston, 2003.

[2] Diamond, J. Senators' report faults CIA on Iraq, *USA Today*, 2004.

[3] IETF. RFC 1630, Universal Resource Identifiers in WWW. Berners-Lee, T. ed., Internet Engineering Task Force, 1994.

[4] Jindal, A., Distributed Computing Environment. in *Proceedings. SHARE Europe Spring Meeting: Managing Communications in a Global Marketplace, 30 March-3 April 1992*, (Cannes, France, 1992), SHARE Europe (SEAS), 385-401.

[5] Kindel, Charlie and Brown. Distributed Component Object Model Protocol (DCOM/1.0), Microsoft Corporation, 1998.

[6] OMG. CORBAservices: Common Object Services Specification, Security Service Specification v1.8, Object Management Group, document formal/2002-03-11, 2002.

[7] Park, J. and Sandhu, R. The UCONabc usage control model. *ACM Transactions on Information and System Security*, *7* (1). 128-174.

[8] Sandhu, R., Coyne, E., Feinstein, H. and Youman, C. Role-Based Access Control Models. *IEEE Computer*, *29* (2). 38-47.

[9] Sandhu, R.S. Lattice-Based Access Control Models *IEEE Computer*, 1993, 9-19.

[10] Siegel, J. *CORBA 3 Fundamentals and Programming*. John Wiley & Sons, 2000.

[11] Sun. Enterprise JavaBeans Specification, Version 2.0, Sun Microsystems Inc., Palo Alto, CA., 2000, 554.

[12] Vogels, W. How Wrong Can You Be? Getting Lost on the Road to Massive Scalability, International Middleware Conference, 2004.

[13] XACML-TC. OASIS eXtensible Access Control Markup Language (XACML) version 1.0, OASIS, 2003.