Policy Review in Attribute Based Access Control
A Policy Machine Case Study

Sherifdeen Lawa and Ram Krishnan*
University of Texas at San Antonio, Texas, United State
{sherifdeen.lawal,Ram.Krishnan}@utsa.edu

Abstract

The Next Generation Access Control (NGAC), founded on the Policy Machine (PM), is a robust Attribute Based Access Control (ABAC) framework that enables a structured and flexible approach for the establishment of Discretionary Access Control (DAC) policies, accommodates limited expression of non-confinement Mandatory Access Control (MAC) policies, has shown support for all aspects of the Role Based Access Control (RBAC) standard, and possesses algorithms for both per-user and per-object review. However, NGAC lacks the mechanism for other critical administrative review problems like comprehensive approaches to grant authorization (revoke authorization) for a denied access request (an authorized access request). We proposed approaches to grant authorization of (one of the administrative operations) any denied user assignment access request as our initial work in response to the policy review features not in the PM.

Keywords: Attribute Based Access Control, Policy Review, Policy Machine

1 Introduction

Attribute Based Access Control (ABAC) remains to be a promising form of access control. Unlike traditional access control, ABAC authorization to perform a set of operation is determined by evaluating attributes associated with the subject, object, and in some cases, environmental conditions against policy, rules, or relationships that describe the allowable operations for a given set of attributes [5]. However, one of the open problems of an ABAC system is it auditability(policy review) [11].

The collection and/or organization of protected data for further analysis in accordance with predefined regulations and directives, is refered to as policy review [5]. For instance, a policy review may be the evaluation of users that are denied access to a particular object, an evaluation of access control entry an object attribute acquires, etc. Compare to the likes of Access Control List (ACL) and Role Based Access Control (RBAC), where to determine the set of users who have access to a given resource or the set of resources a given user may have access to (sometimes referred to as a "before the fact audit") is relatively straight forward [11], ABAC is considerably more complicated [5]. For example, in order to evaluate the set of subjects that have access to a given object in an ABAC system, it requires running a simulation of access request for every known subject in the environment, since ABAC is an identity-less access control system.

Furthermore, apart from users and resources, other ABAC elements poses critical policy review questions. For instance, how can we determine the number of users logged in using environmental attributes, and how does an administrator determine the access rights delegated among users in ABAC system? These are some of the critical policy review problems. Without a more complete and efficient

Journal of Internet Services and Information Security (JISIS), volume: 10, number: 2 (May 2020), pp. 67-81
DOI: *10.22667/JISIS.2020.05.31.067*

*Corresponding author: Department of Electrical and Computer Engineering, The University of Texas at San Antonio, One UTSA Circle, San Antonio, TX 78258, USA, Tel: +1-210-458-6293

methods to answer these policy review questions, ABAC will likely be unusable in cases where legal or industry regulations prohibit systems that rely solely on auditing techniques [11].

The NIST Next Generation Access Control (NGAC) is one of the two current implementation of ABAC. The Policy Machine (PM) has been the foundation for the NGAC and can manage policies easily. Rather than the traditional rule based ABAC policy specification approach, the policy machine has a unique and simplified approach that implies policy specification through the RBAC-style relations. Since authorization policies in most existing ABAC models including [6, 10] are expressed in propositional logic, policy review is an NP-complete or even undecidable problem. However, policy machine lend itself to an enumerated approach of specifying authorization and policy review is inherently simple (polynomial time) [3]. Another reason why the policy machine was our pick for this work, is the consistent grow in contribution to the opensource implementations of the policy machine project. At the time of this writing, there are two independent opensource repositories that implement the policy machine in (Java and Ruby) two different programming languages. The NIST owned repository is evolving from its standalone Java program called Harmonia to a web based application through an administrative REST API. The policy machine leverages on its evolution to demonstrate the power of NGAC and the benefits it can bring to the security industry [9]. There has been a considerable amount of research work applying the Policy Machine (PM) [2] - [8] but not so significant contribution has been made to improve the effectiveness and efficiency of the PM. Currently, the NGAC reference model only support users' capabilities and access entries audit. While improvement upon the former by reducing the overall time complexity from cubic to linear run time was done [7], a faster theoretical approach for both per-user and per-object policy review was also proposed [1].

Other policy review problems not supported by the PM framework but are nonetheless critical includes review of authorization and prohibition approaches. This is a question of how does an administrator grant(deny) any user denied(authorized) an access request in the PM without unintended consequence? Section 3 demonstrates the challenge an administrator of the PM may encounter in an attempt to delegate an administrative authority.

To simplify our exploration, in this work we focus on policy review of an authorization approach and further narrow our scope to one of the administrative operation (user assignment) in the PM framework.

In summary, the contribution of this work is:

- The development of an algorithm that generates a collection of one or more access request sets that can authorize a given denied user assignment access request.

This contribution is fundamental to a generalized review of authorization approaches in the PM framework, thus providing the efficient and effective approach to grant authorization to any denied access request.

The reminder of this paper is structured as follows. Section 2 provides overview of the policy machine framework and its components. Section 3 presents a formal problem statement and the scope of this work. Section 4 details on our observations and claim on the PM Models. Section 5 describe the approaches to generate an access request sets that each set grant authorization to a given denied user assignment access request and presents its algorithm. Section 6 is a conclusion of this work.

2 PM Overview

The rest of this section provides an overview of policy machine core data elements and selected relations pertinent to this work. The assignment, association, prohibition, and obligation are the primary relations while privileges and restrictions are derived relations from associations and prohibitions in the PM respectively. For the scope of this work, we only delved into assignment and association relations

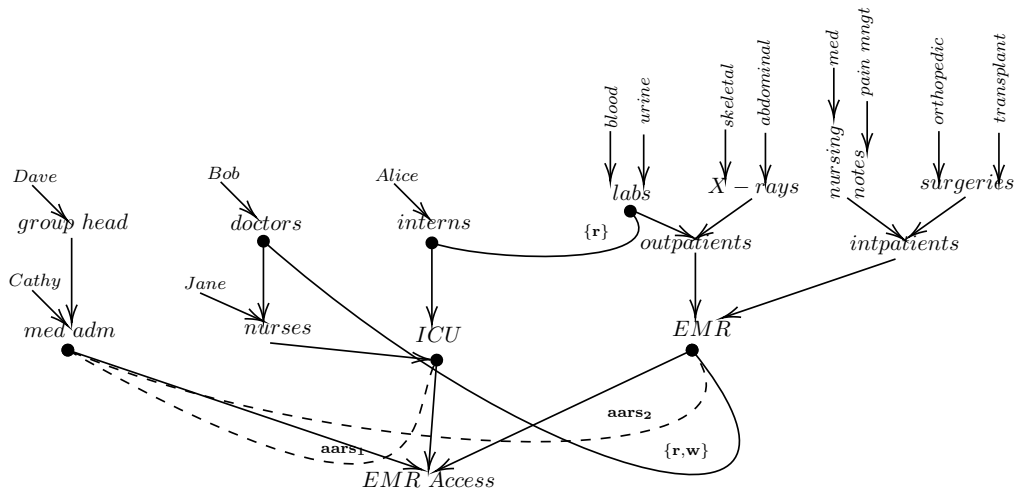


Figure 1: Policy Machine Authorization Graph

and concepts related to them. To ease referencing, we provide the formal definition of PM components discussed in this work in appendix B. Readers with prior knowledge of policy machine fundamentals may skip this section.

2.1 PM Basic Elements

In figure 1, the following sets: {*Alice*, *Bob*, *Cathy*, *Dave*, *Jane*}, {*group head*, *med. adm.*, *doctors*, *nurses*, *interns*, *ICU*}, {*blood*, *urine*, *skeletal*, *abdominal*, *med*, *pain mngt*, *orthopedic*, *transplant*}, {*labs*, *X-rays*, *outpatients*, *inpatients*, *nursing notes*, *surgeries*, *EMR*}, {*EMR Access*}, {$aars_1 \cup aars_2$} and {*r*, *w*} are the instances of the following core elements of the policy machine: finite set of Users, User Attributes, Objects, Object Attributes, Policy Classes, Administrative Access Rights, and Resource Access Rights respectively. The finite set of users, user attributes, objects, object attributes, and policy classes are called the policy elements (PE) of the policy machine. Each element of Administrative Access Right set are denoted with a prefix 'c-'/'d-' (i.e., create or delete) followed by the paired policy elements (e.g., uua or oaoa) that translates to the authority(permission) required to perform an operation creating/deleting a relation between a user and a user attribute or an object attribute and an object attribute. The PM is structured to accommodate multiple policy classes, however, we limit this work to a single policy class.

2.2 PM Relations

The assignment relation represents the allowed hierarchical relations between policy elements. The relation ASSIGN, represents all possible assignments between policy elements in a policy class and an individual assignment can be expressed as either (pe_i, pe_j) $\in$ ASSIGN or pe_i ASSIGN pe_j, on elements pe_i, pe_j of the policy elements. As in figure 1, the tuple (*nurses*, *ICU*) assigns *nurses* to *ICU* and the ordered pair is an element of the assignment relation, ASSIGN. For any pe_i and pe_j that are elements of PE, pe_j is said to contain pe_i if there exist a path from pe_i to pe_j.

The authority(Access right) to perform an operation is granted through associations. The relation ASSOCIATION, defines the set of possible associations within a policy specification. For a policy class

containing user(object) attribute, the association (ua_i, ars_j, ua_k) or (ua_i, ars_j, oa_k) specifies that all users contained by ua_i possess the authority to perform the operations enabled by the access rights in ars_j on ua_k(oa_k) and all user(object)attributes it contains.

Figure 1 is an authorization graph, G = (PE, ASSIGN, ASSOCIATION), where the solid downward-arcing and the dashed downward-arcing connectors are the associations grating resource and administrative access rights respectively. For instance, the triple association, ($doctors$, $\{r, w\}$, EMR) permits any user contained by the user attribute $doctors$ to perform both read and write operations on files and directories (objects and object attributes) contained by EMR.

Access Request, AREQ, is a finite set of possible access requests in a policy specification. A triple of access request relation, (p, op, argseq), denotes a user associated with a process, p, is requesting to perform the operation, op, on the resource or policy elements referenced by the argument sequence, argseq. The Access Decision, Access_Decision, is function that maps an access request, (p, op, argseq), to a grant or deny of the operation, op, on the argument sequence based on authorities and/or prohibitions held by the subject associated with the process.

3 Problem Statement and Scope

In the context of policy review, previous contributions were to improve on an existing algorithm in the policy machine [1, 7]. However, as we have mentioned before, there are other policy review problems that are not addressed in the policy machine framework. In the following subsections we formalize the policy review of authorization approaches and provide details on the extent of this work as the basis for the review of authorization approaches in the policy machine.

3.1 Problem Statement

How do you grant an access if a user is not allowed to perform an operation on an(a) object(subject)? is undeniably an important policy review question. Currently, the policy machine has no mechanism to answer this query. The significance of integrating a review function that answers this question is demonstrated using figure 1.

Assuming the administrative access rights, $aars_1$ = { {'c-uua'}, {'c-uaua'}, {'c-assoc-fr', 'c-assoc-to'}, {'d-uua'}, {'d-uaua'}, {'d-assoc-fr', 'd-assoc-to'} } and $aars_2$ = { {'c-ooa'}, {'c-oaoa'}, {'c-assoc-to'}, {'d-ooa'}, {'d-oaoa'}, {'d-assoc-to'} }. The user contained by the user attribute, *interns* has no access right to read any patient record contained by the *X-rays* object attribute. (i.e., a request by *Alice* to read *skeletal* or *abdominal* images is denied). At the minimum, the powerset of {{'c-uua'}, {'c-uaua'}, {'c-assoc-fr', 'c-assoc-to'}, {'c-ooa'}, {'c-oaoa'}} (i.e., a subset of $\{aars_1 \cup aars_2\}$ without any delete access right) are the possible access request operations on the policy elements of the graph to grant *Alice*'s denied access. However, each of the possible operation can be applied to more than one combination of policy elements to authorize *Alice*'s denied access. For instance, the access request to create an object attribute assignment can be done in two ways to grant *Alice* a read, r access on *abdominal* images (i.e., the object attribute assignment of the *abdominal* to the *labs* and *X-rays* to the *labs* object attribute).

Firstly, considering an enterprise scale, there are combinatorial explosion of possible approaches to authorize(deny) a denied(authorized) access request as demonstrated in the preceding example. Since access is granted(denied) dynamically through attributes in ABAC, another nontrivial problem is how to determine the scope of other users granted(denied) capabilities and access entries granted(denied) for each possible approaches? Apparently, a manual approach to answer these questions is not only time consuming but also susceptible to unintended authorization.

To address this lacking feature of the policy machine, we proposed an algorithm for a user assignment authorization, a foundational algorithm for a generalized policy review of authorization approaches. In a more general term, the policy review of authorization approaches can be stated as – given a denied access request, find all the possible sets of one or more access request sets that can authorize the denied access request such that no strict subsets of any access request set can grant the denied access.

Formally, given $\text{AREQ}_D : \forall areq_d \in \text{AREQ}_D \cdot \big(\text{Access\_Decision}(areq_d) = \text{deny}\big)$, find AREQ_{auth} :

$$\forall areq_d \in \text{AREQ}_D, \forall areq_A \in \text{AREQ}_{auth} \cdot \Big(\big(\forall areq \in areq_A \cdot \text{Access\_Decision}(areq) = \{\text{accept}\}\big) \longrightarrow \text{Access\_Decision}(areq_d) = \{\text{accept}\}\Big)$$

$$\wedge \Big(\forall areq_s \subset areq_A \cdot \Big(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\text{accept}\}\big) \longrightarrow \text{Access\_Decision}(areq_d) = \{\text{deny}\}\Big)\Big)$$

3.2 Scope

The PM model has multiple facets – the policy elements and the assignments that make up a policy element diagram, the association and prohibition that apply the policy element diagram to form the authorization graph, and obligation that are carried out when access related event occur [4]. However, in this work, we restrict ourselves to a minimal PM composed of data elements required in the policy review of a user assignment authorization approach in a single policy class (i.e., finite sets of Users and User Attributes in a policy class). As a result, applicable classes of administrative access rights for the authorization of a denied user assignment access request that we considered are:

1. Authority to create an assignment between a user and a user attribute.
2. Authority to create an assignment between user attribute and user attribute.
3. Authority to create an association between user attribute and user attribute.

We assumed privileged users that can grant authorization to a denied user assignment access request excludes the administrative super user of the PM system.

4 Observations, Derived Components, and Claim

In the following subsections, the distinction of a Principle Administrator and required condition for policy element deletion were alluded to, and derived notations for a fluid logical expression of our assertion is enumerated.

4.1 Observations

1. The Principal Authority (PA), also known as the super user, is a compulsory predefined entity of the PM. The PA is responsible for creating and controlling the policies of the PM in their entirety and inherently holds universal authorization to carry out those activities within the PM framework. The access rights held by the principal administrator can be delegated to domain and/or subordinate administrators except the following:
   (a) The access right to create and delete policy class.
   (b) The access right to create and delete assignment of attributes to the policy class.

2. In order to preserve the properties of the policy machine model, a policy element targeted by a delete operation must not be involved in any defined relation. For example, if a user attribute is involved in an assignment, association, or prohibition relation, the user attribute cannot be deleted until it is no longer involved in the relation

4.2 Derived Components

For the simplicity of our formal assertion that all types of delete operations have no effect on any authorization approach, we define a derived function, Authorizing Access Request, and notations.

Definition 1. [Authorizing Access Request] An Authorizing Access Request is a function that takes a set of denied access requests as input and return a set of one or more access request sets as output for each denied access request such that the following conditions are satisfied:

1. A given denied access request is authorized if all access requests of the associated access request set is authorized.

2. A given denied access request cannot be authorized by any strict subsets of the associated access request set.

A formal expression for this function and its conditions are given as **func_aareq** : $\text{AREQ}_D \longrightarrow 2^{2^{AREQ}}$,

$$\text{Where } \forall areq_d \in \text{AREQ}_D, \forall areq_A \in \text{func\_aareq}(areq_d) \cdot \Big(\big(\forall areq \in areq_A \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \longrightarrow \text{Access\_Decision}(areq_d) = \{\text{accept}\} \Big) \wedge \Big(\forall areq_s \subset areq_A \cdot \Big(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \longrightarrow \text{Access\_Decision}(areq_d) = \{\text{deny}\} \Big) \Big)$$

Derived Notations

In order to demonstrate the relationship of a user and a process, simplify the representation of any administrative access right, and reference any element of an access request we adapt the following notations:

- **User's Process:** A user can have a one to many relation with process, while a process can have a one to one relation with user. We denote a user u_s associated with a process p as $\boldsymbol{p}_{u_s}$.

- **Generalizing Access Rights:** Every administrative access right for the creation and deletion of policy element is prefixed by 'c-' and 'd-' respectively, followed by a policy element or a pair of policy elements (e.g., c-uapc). To generalize, we denote any create and delete administrative access right as **'c-*'** and **'d-*'** respectively.

- **Access Request Element:** Let areq = (p_{u_r}, aop, argseq) be a tuple of an administrative access request. For simplification of expression, we adopt the dot notation to reference the access request variables. For example, the administrative operation aop in the access request areq is denoted as **areq.aop**

4.3 Claim

The create and delete access rights are the only types of administrative access rights that applies to policy elements and their relations. However, deletion of any policy element and/or relations can not authorize any denied access request

Claim: The deletion of any policy element and/or relations can not authorize any denied access request.

That is to say, given an access request set that authorizes a denied access, if there exist an operation requesting the deletion of any policy element in the access request set and no strict subset of the access request set can grant the denied access, then the denied access is authorized without the operation requesting the deletion of a policy element and no strict subset of the access request set can grant the denied access. In a more general term, we can express this claim formally as follows:

$$\forall areq_d \in \text{AREQ}_D, \forall areq_A \in \text{func\_aareq}(areq_d) \cdot \Bigg(\Big(\big(\forall areq \in areq_A : \exists d\text{-}* \in areq.aop \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \wedge \text{Access\_Decision}(areq_d) = \{\text{accept}\} \Big) \wedge \Big(\forall areq_s \subset areq_A \cdot \Big(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \wedge \text{Access\_Decision}(areq_d) = \{\text{deny}\} \Big) \Big) \longrightarrow \Big(\big(\forall areq \backslash areq_{dd} \text{: } areq_{dd}.aop = d\text{-}* \in areq_A \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \wedge \text{Access\_Decision}(areq_d) = \{\text{accept}\} \Big) \wedge \Big(\forall areq_s \subset areq_A \cdot \Big(\big(\forall areq \in areq_s \cdot \text{Access\_Decision}(areq) = \{\text{accept}\} \big) \wedge \text{Access\_Decision}(areq_d) = \{\text{deny}\} \Big) \Big) \Bigg)$$

Proof. Let $areq_d = (p_{u_s}, Op, argSeq)$be a denied the access request of user u_s, PRIV'_{u_s} be the finite set of derived privileges of the user, u_s, before any access request that deletes policy element and relation, PRIV''_{u_s} be the finite set of derived privileges of the user, u_s, after any access request that deletes policy element and relation, (u_s, ar, at) be the privilege required by the user, u_s, in order for the denied access request $areq_d = (p_{u_s}, Op, argSeq)$ to be authorized

$$(u_s, ar_i, at_i) \in PRIV''_{u_s} \longrightarrow \nexists (ar_i = ar \wedge at_i = at)$$

□

5 Approaches

This section, we describe the two scenarios that determine all the possible approaches to grant authorization to a denied request. The relations created through user assignment, an attribute assignment, an association between user attributes, and their combinations may be applicable in both scenarios. Also, we proposed an algorithm, UUAReview, that takes a denied request and its authorization graph as an input and generates (Authorizing Access Requests) a set of access request sets that can authorize the denied request as output.

Assume $areq_d$ = (p_{u_s}, Op, $argSeq$) is a denied user assignment access request in an authorization graph G_{auth} = (PE, ASSIGN, ASSOCIATION). We refer to the user, u_s, that the process p_{u_s} is acting on his/her behalf as *source user*, and the user and user attribute of the argument sequence, *argSeq*, as *target user* and *target user attribute* respectively.

The following are the possible assignment scenarios that exist between the *source user*, and *target user attribute*:

I The *source user*, u_s is contained by the *target user attribute*, or some other user attribute contains both *source user* and *target user attribute* in the authorization graph.

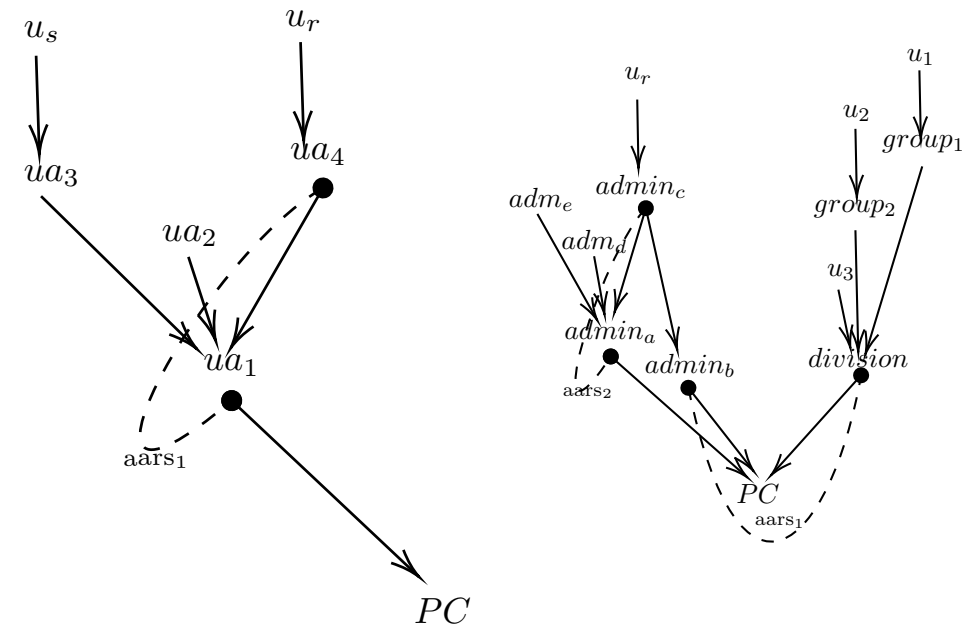
II The *source user* is not contained by the *target user attribute* and no user attribute contains both *source user* and *target user attribute*.

5.1 Scenario I: Source User Contained by Target User Attribute

In order to generate the Authorizing Access Requests for $areq_d$, there must exist an association, (ua_i, ars_j, ua_k), such that the *target user attribute*, is contained by ua_k and the authority to create a user assignment, a user attribute assignment, or an association between user attributes is an element of the set, ars_j. The operations of all possible Authorizing Access Requests is the powerset of the access right set, ars_j. However, the resulting number of approaches to authorize the denied access is more than the number of possible operations, since some operation may be applied to multiple policy elements. Using figure 2a as an example for this scenario, where the *source user*, u_s is contained by *target user attribute*, ua_1, or *source user*, u_s and *target user attribute*, ua_2 are contained by the user attribute ua_1.
Supposing $aars_1 = \big\{\{c\text{-}uua_{ua_1}\}, \{c\text{-}uua_{ua_1}\}, \{c\text{-}assoc\text{-}fr_{ua_1}, c\text{-}assoc\text{-}to_{ua_1}\}\big\}$, then approaches to authorize the denied request are the conforming operations, $2_1^{aars_1}$, on the policy elements, u_s and ua_1 - ua_4, by user, u_r.

5.2 Scenario II: Source User Not Contained by Target User Attribute

In this scenario, the associations, (ua_i, ars_j, ua_k) and (ua_p, ars_q, ua_r) are required such that ua_r contains *target user attribute*, ua_t, ua_i contains ua_p, ua_k contains *source user*, u_s and ua_i, and the authority to create a user assignment, attribute assignment, or association between user attributes is an element of the access right set, ars_q. A subset of the powerset of the access right sets, ars_j and ars_q provides the operations to authorize the denied user assignment access request. The partial authorization graph (with no object and object attributes) of figure 2b illustrates this scenario. For instance in figure 2b, a denied user assignment access request, $areq_d$ = (p_{u_s}, $\{c\text{-}uua_{division}\}$, $\langle u_t, group_1\rangle$), has the *source user* u_s, not contained by *target user attribute*, $group_1$, and both *source user* and *target user attribute*, are not contained by the same user attribute. The operations for the approaches that authorize the denied user assignment, $areq_d$, are subset of the powerset, $2_1^{aars_1 \cup aars_2}$, on subset of policy elements in the authorization graph of figure 2b.



(a) Denied user is contained by target user attribute

(b) Denied user not contained by target user attribute

Figure 2: Partial Authorization Graphs

5.3 Algorithm Specification

Let G_{auth} = (PE, ASSIGN, ASSOCIATION) be an authorization graph of a policy machine model, where PE (policy elements) is set of nodes, such that the source user u_s is an element of PE, ASSIGN is set of directed edges, and ASSOCIATION is set of annotated arcs. Suppose, $areq_d = (p_{u_s}, Op, argSeq)$ was a denied access request for the assignment of a *target user*, to the *target user attribute*. The algorithm, UUAReview, generates a set of one or more access request sets that authorize $areq_d$. The following constitute the specification of the UUAReview algorithm:

- **Inputs:** The algorithm has two input parameters, the denied access request $areq_d$ and its authorization graph G_{auth}. It is assumed an association in the set ASSOCIATION that grants some privileged user attribute ua_p the access right to create the denied relation and the privileged user attribute ua_p can create assignment and/or association on policy elements that grant the authorization of the denied access request.
- **Output:** Whenever the the precondition is satisfied, a returned set $AREQ_A$ by the algorithm is a set of one or more access request sets that grant the authorization of the denied access request and no strict subsets of any access request set in the set $AREQ_A$ can grant the authorization of the denied access request.
- **Function Header:** The UUAReview algorithm is initialized by the function `mainUUAReview` that takes the graph G_{auth} and the denied access request $areq_d$ as inputs. The `mainUUAReview` has helper functions and are contained in appendix A. The `mainUUAReview` program runs by first checking for the two possible scenarios, as previously discussed. If the *source user* is contained by '*target ua*' then the program searches for four different sets of user attributes that create relations to authorize the denied access. The function, `targetContainsSource` gets called by `mainUUAReview` with the denied access request, its authorization graph, the sets of user attributes, and a privileged user set as parameters.

Algorithm 1: An algorithm to generate an Authorizing Access Request set for a denied user assignment access request

Inputs : $areq_d$ = (p_{u_s}, Op, $argSeq$), $\mathbf{G}_{auth}$ = (PE, ASSIGN, ASSOCIATION)
Output : $AREQ_A = [AREQ_{A1}, AREQ_{A2}, ..., AREQ_{An}]$
Function `mainUUAReview(`$areq_d$, G_{auth}`)`**:**

$AREQ_A \longleftarrow \{\ \}$
if $\exists\ (ua_i, ars_j, ua_k) \in$ G$_{auth}$.ASSOCIATION : $(areq_d.user, ua_t) \in$ ASSIGN$^+ \wedge\ (areq_d.user, ua_k) \in$ ASSIGN$^+ \wedge\ (ua_t, ua_k) \in$ ASSIGN$^+ \wedge\ \{areq_d{\cdot}Op\} \in ars_j$ **then**
SET-I(userAttrBtwSourceAndTarget) $\longleftarrow$ get ua such ua contains *source user* and '*target ua*' contains ua
SET-II(privilegedUA) $\longleftarrow$ get set of privilege ua (i.e., ua that have authority to assign '*target user*' to '*target ua*')
SET-III(UABtwTargetAndPc) $\longleftarrow$ get ua that contains elements of privilegedUA
SET-IV(UANotInPrivAndSource) $\longleftarrow$ get ua not in any of the previous sets
SET-PU(privUser) $\longleftarrow$ get users contained by user attributes in privilegedUA
SET-ARS(assocSubset) $\longleftarrow$ get subsets of ars_j such that $\{areq_d{\cdot}Op\}$ is an element of the set
return `targetContainsSource(`$areq_d$, G_{auth}, $AREQ_A$, *SET-I, SET-II, SET-III, SET-IV, SET-PU*`)`
else if $(areq_d.user, ua_t) \notin$ ASSIGN$^+ \wedge \exists\ (ua_i, ars_j, ua_k), (ua_p, ars_q, ua_r) \in$ G$_{auth}$.ASSOCIATION : $(ua_t, ua_r) \in$ ASSIGN$^+ \wedge\ (ua_p, ua_i) \in$ ASSIGN$^+ \wedge\ (areq_d.user, ua_k) \in$ ASSIGN$^+ \wedge\ (ua_i, ua_k) \in$ ASSIGN$^+ \wedge\ \{areq_d{\cdot}Op\} \in ars_q$ **then**
UAContainedByUa$_i$(**SET-1**) $\longleftarrow$ get the set of user attributes contained by ua_i.
UAContainsU$_s$ (**SET-2**) $\longleftarrow$ get the set of user attributes that contain *source user*
UANotInUa$_i$Ua$_s$(**SET-3**) $\longleftarrow$ get the set of user attributes contained by ua_k and are not elements of the sets SET-1 and SET-2
UAContainedByUa$_r$(**SET-4**) $\longleftarrow$ get user attributes contained by ua_r%= $\{ua_4 \mid (ua_4, ua_r) \in$ ASSIGN$^+\}$
UACanContainTarget(**SET-5**) $\longleftarrow$ get user attributes contained by ua_r and not contained by '*target ua*' and does not contain '*target ua*'
UAContainedByTarget(**SET-6**) $\longleftarrow$ get the set of user attributes contained by '*target ua*'
privUser (**SET-PU**) $\longleftarrow$ get the set of users contained by ua_i and ua_p.
assocSubset (**SET-ARS**) = $\{ars_n \mid ars_n \in 2_1^{ars_q} \wedge areq_d.op \in ars_n\}$
return `sourceNotContained(`$areq_d$, G_{auth}, $AREQ_A$, *SET-1, SET-2, SET-3, SET-4, SET-5, SET-6*`)`
else
return *Authorization cannot be generated*
end
end

If the second condition is true, then the main function, `mainUUAReview` calls the function `sourceNotContained`. The denied access, its authorization graph, six different sets of user attributes, and a set of privileged user set are passed as parameters to the `sourceNotContained`. Both functions called by `mainUUAReview` utilize the `getAssignArgSeqSet` and `getAssocArgSeqSet` to create argument sequence of assignment and association operation sets respectively. Similarly, `getAssignAccessReq` and `getAssocAccessReq` generate assignment and association access request sets that may grant authorization or may be elements of the cartesian product returned to `mainUUAReview`

5.4 Discussion

Computations in the algorithm UUAReview are heavily cartesian product of sets. For computations related to creating assignment relations, the cardinalities of the sets of user attributes influence the overall number of possible approaches to grant a denied access. In other words, the more the depth of a policy element (a node) in the authorization graph, the more the number of assignment approaches that can authorize the denied access. Furthermore, associations have higher tendency of producing explosive approaches to authorize a denied access if both user attribute sets and access right sets are of high cardinalities.

In an enterprise with deep/lengthy hierarchies, it may quickly become unwieldy for an administrator to specify an approach to best grant(deny) an access. We intend to expand our algorithm to other types of access requests in the policy machine. We also identify the need to integrate a mechanism that intelligently provides recommended approach or approaches to grant(deny) an access based on provided constraints.

6 Conclusion And Future Work

In an effort to establish a policy review of authorization approaches in the PM models, we present this preliminary work with an algorithm for the review of user assignment authorization approaches. We intend to expand on our algorithm to capture all other types of access rights.

References

[1] R. Basnet, S. Mukherjee, V. M. Pagadala, and I. Ray. An efficient implementation of next generation access control for the mobile health cloud. In *Proc. of the 3rd International Conference on Fog and Mobile Edge Computing (FMEC'18), Barcelona, Spain*, pages 131–138. IEEE, April 2018.

[2] S. Bhatt, F. Patwa, and R. Sandhu. Abac with group attributes and attribute hierarchies utilizing the policy machine. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC'17), New York, New York, USA*, pages 17–28. ACM, March 2017.

[3] P. Biswas, R. Sandhu, and R. Krishnan. Label-based access control: An abac model with enumerated authorization policy. In *Proc. of the 5th ACM International Workshop on Attribute Based Access Control (CODASPY'16), New York, New York, USA*, pages 1–12. ACM, March 2016.

[4] D. Ferraiolo, S. Gavrila, and W. Jansen. Policy machine: features, architecture, and specification. Technical Report 7987 Rev. 1, National Institute of Standards and Technology, Internal Report, 2015.

[5] V. C. Hu, D. Ferraiolo, R. Kuhn, A. R. Friedman, A. J. Lang, M. M. Cogdell, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. Guide to attribute based access control (abac) definition and considerations. Technical Report 800-162, National Institute of Standards and Technology, Special Publication, 2019.

[6] X. Jin, R. Krishnan, and R. S. Sandhu. A unified attribute-based access control model covering dac, mac and rbac. In *Proc. of the 26th IFIP Annual Conference on Data and Applications Security and Privacy*

(DBSec'12), Paris, France, volume 7371 of *Lecture Notes in Computer Science*, pages 41–55. Springer, July 2012.

[7] P. Mell, J. M. Shook, and S. Gavrila. Restricting insider access through efficient implementation of multi-policy access control systems. In *Proc. of the 8th ACM CCS International Workshop on Managing Insider Security Threats (MIST'16), New York, New York, USA*, pages 13–22. ACM, October 2016.

[8] V. Pagadala and I. Ray. Achieving mobile-health privacy using attribute-based access control. In *Proc. of the 11th International Symposium on Foundations and Practice of Security (FPS'18), Montreal, Quebec, Canada*, volume 11358 of *Lecture Notes in Computer Science*, pages 301–316. Springer, November 2018.

[9] J. Roberts, G. Katwala, S. Gavrila, and D. Ferraiolo. Web based implementation of policy machine. https://pm-master.github.io/pm-master/ [Online: accessed on May 20, 2020], May 2020.

[10] D. Servos and S. L. O. HGABAC. Hgabac: Towards a formal model of hierarchical attribute-based access control. In *Proc. of the 7th International Symposium on Foundations and Practice of Security (FPS'14), Montreal, Quebec, Canada*, volume 8930 of *Lecture Notes in Computer Science*, pages 187–204. Springer, November 2014.

[11] D. Servos and S. L. Osborn. Current research and open problems in attribute-based access control. *ACM Computing Survey*, 49(4):1–45, January 2017.

Author Biography

Sherifdeen Lawal received the Bachelor of Technology in Electrical and Computer Engineering from University of Technology, Minna, Nigeria in 2007, Master of Science in Computer Engineering from University of Texas at San Antonio in 2015. He is a doctral candidate at the ECE Computer Security Research Group (CSRG) and he is working on policy review issues in the Attribute Based Access Control.

Ram Krishnan received the Bachelor of Technology in Computer Science and Engineering from Pondicherry University in 2001, Master of Science in Computer Engineering from New Jersey Institute of Technology in 2004, and Ph.D. in Computer Science from George Mason University in 2010. He is an Associate Professor in the Department of Electrical and Computer Engineering at the University of Texas at San Antonio where he hold the Microsoft President's Endowed Professorship. His research interest is in the general area of computer security and he directs the ECE Computer Security Research Group (CSRG).

A Appendix: UUAReview Algorithm Helper Functions

Function `getAssignArgSeqSet(`$SETua_a$, $SETua_c$`):`
1 **return** the set of the cartesian product of sets SETua_a and SETua_c.
2 **end**
Function `getAssocArgSeqSet(`$SETua_a$, $SETop_b$, $SETua_c$`):`
3 **return** the set of the cartesian product of sets SETua_a, SETop_b, and SETua_c.
4 **end**
Function `getAssignAccessReq(`$SET_{privUser}$, $SETOp$, $SETua_a$, $SETua_c$`):`
5 **return** the set of the cartesian product of sets $\text{SET}_{\text{privUser}}$, SETOp, and `getAssignArgSeqSet(`$SETua_a$,$SETua_c$`)`
6 **end**
Function `getAssocAccessReq(`$SET_{privUser}$, $SETOp$, $SETua_a$, $SETop_b$, $SETua_c$`):`
7 **return** the set of the cartesian product of sets $\text{SET}_{\text{privUser}}$, SETOp, and `getAssocArgSeqSet(`$SETua_a$,$SETop_b$, $SETua_c$`)`
8 **end**
9 **Function** `targetContainsSource(`$areq_d$, G_{auth}, $AREQ_A$, *SET-I, SET-II, SET-III, SET-IV, SET-PU*`):`
10 `/* If any of the following if statement is true, the function getAssignArgSeqSet and/or getAssocArgSeqSet are(is) called with the parameters to form a set of assignment and/or association access request */`
11 Add the set of `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uua_{au_k}\}$, *{source user}, SET-II*`)` to $AREQ_A$
12 **if** $\{c\text{-}uaua_{au_k}\} \in ars_j$ **then** add the set of `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uaua_{au_k}\}$, *SET-I, SET-II*`)` to $AREQ_A$
13 **if** $\{\{c\text{-}uua_{au_k}\}, \{c\text{-}uaua_{au_k}\}\} \subseteq ars_j$ **then** add the sets of the cartesian product of the access request sets returned by `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uua_{au_k}\}$, *{source user}, SET-IV*`)` and `getAssignAccessReq(`*SET-PU,*$\{c\text{-}uaua_{au_k}\}$, *SET-IV, SET-II*`)` to $AREQ_A$
14 **if** $\{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\} \in ars_j$ **then** add the set of `getAssocAccessReq(`*SET-PU,* $\{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}$, *SET-I, SET-ARS, SET-III*`)` to $AREQ_A$
15 **if** $\{\{c\text{-}uua_{au_k}\}, \{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}\} \subseteq ars_j$ **then** add the sets of the cartesian product ofthe access request sets returned by `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uua_{au_k}\}$, *{source user}, SET-IV*`)` and `getAssocAccessReq(`*SET-PU,* $\{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}$, *SET-IV, SET-ARS, SET-III*`)` to $AREQ_A$
16 **if** $\{\{c\text{-}uaua_{au_k}\}, \{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}\} \subseteq ars_j$ **then** add the sets of the cartesian product of the access request sets returned by `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uaua_{au_k}\}$, *SET-I, SET-IV*`)` and `getAssocAccessReq(`*SET-PU,* $\{c\text{-}assoc\text{-}fr_{ua_k}$, $c\text{-}assoc\text{-}to_{ua_k}\}$, *SET-IV, SET-ARS, SET-III*`)` to $AREQ_A$
17 **if** $\{\{c\text{-}uaua_{au_k}\}, \{c\text{-}uaua_{au_k}\}, \{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}\} \subseteq ars_j$ **then** Add the sets of the cartesian product of the access request sets returned by `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uua_{au_k}\}$, $\{areq_d.user\}$, *SET-IV*`)`, `getAssignAccessReq(`*SET-PU,* $\{c\text{-}uaua_{au_k}\}$, *SET-I, SET-IV*`)`, and `getAssocAccessReq(`*SET-PU,* $\{c\text{-}assoc\text{-}fr_{ua_k}, c\text{-}assoc\text{-}to_{ua_k}\}$, *SET-IV, SET-ARS, SET-III*`)` to $AREQ_A$
18 **return** $AREQ_A$
19 **end**

20 **Function** `sourceNotContained`(*areq$_d$, G$_{auth}$, AREQ$_A$, SET-1, SET-2, SET-3, SET-4, SET-5, SET-6, SET-PU*)**:**

21 /* The following create sets of access request set that grant *source user*the denied access for any of the if statement that is true. */

22 Add the set of `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, {source user}, SET-1*) to *AREQ$_A$*

23 **if** {*c-uaua$_{au_k}$*} $\in$ *ars$_j$* **then** add the set of `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-2, SET-1*) to *AREQ$_A$*

24 **if** {*c-assoc-fr$_{ua_k}$*} $\in$ *ars$_j$* $\wedge$ {*c-assoc-to$_{ua_r}$*} $\in$ *ars$_q$* **then** add the set of `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-2, SET-ARS, SET-4*) to *AREQ$_A$*

25 **if** {{*c-uua$_{au_k}$*}, {*c-uaua$_{au_k}$*}} $\subseteq$ *ars$_j$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, {source user}, SET-3*) and `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-3, SET-1*) to *AREQ$_A$*

26 **if** { {*c-assoc-fr$_{ua_k}$*}, {*c-uua$_{au_k}$*}} $\subseteq$ *ars$_j$* $\wedge$ {*c-assoc-to$_{ua_r}$*} $\in$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, {source user}, SET-3*) and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-3, SET-ARS, SET-4*) to *AREQ$_A$*

27 **if** { {*c-assoc-fr$_{ua_k}$*}, {*c-uaua$_{au_k}$*}} $\subseteq$ *ars$_j$* $\wedge$ {*c-assoc-to$_{ua_r}$*} $\in$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-2, SET-3*) and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-3, SET-ARS, SET-4*)

28 **if** { *c-assoc-fr$_{ua_k}$*} $\in$ *ars$_j$* $\wedge$ {{*c-assoc-to$_{ua_r}$*}, {*c-uaua$_{au_r}$*}} $\subseteq$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_r}$}, SET-5, SET-6*) and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-2, SET-ARS, SET-5*) to *AREQ$_A$*

29 **if** { {*c-uua$_{au_k}$*}, {*c-uaua$_{au_k}$*}, { *c-assoc-fr$_{ua_k}$*} } $\subseteq$ *ars$_j$* $\wedge$ {*c-assoc-to$_{ua_r}$*} $\in$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, source user, SET-3*), `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-2, SET-3*), and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-3, SET-ARS, SET-4*)

30 **if** { {*c-uua$_{au_k}$*}, { *c-assoc-fr$_{ua_k}$*} } $\subseteq$ *ars$_j$* $\wedge$ { {*c-assoc-to$_{ua_r}$*}, {*c-uaua$_{au_r}$*} } $\subseteq$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, {source user}, SET-3*), `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_r}$}, SET-6, SET-5*), and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-3, SET-ARS, SET-5*) to *AREQ$_A$*

31 **if** { {*c-uaua$_{au_k}$*}, { *c-assoc-fr$_{ua_k}$*} } $\subseteq$ *ars$_j$* $\wedge$ { {*c-assoc-to$_{ua_r}$*}, {*c-uaua$_{au_r}$*} } $\subseteq$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-2, SET-3*), `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_r}$} }, SET-6, SET-5*), and `getAssocAccessReq`(*SET-PU, SET-3, SET-ARS, SET-5*) to *AREQ$_A$*

32 **if** { {*c-uua$_{au_k}$*}, {*c-uaua$_{au_k}$*}, { *c-assoc-fr$_{ua_k}$*} } $\subseteq$ *ars$_j$* $\wedge$ { {*c-assoc-to$_{ua_r}$*}, {*c-uaua$_{au_r}$*} } $\subseteq$ *ars$_q$* **then** add the cartesian product of the access request sets returned by `getAssignAccessReq`(*SET-PU, {c-uua$_{au_k}$}, {source user}, SET-3*), `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_k}$}, SET-2, SET-3*), `getAssignAccessReq`(*SET-PU, {c-uaua$_{au_r}$}, SET-6, SET-5*), and `getAssocAccessReq`(*SET-PU, {c-assoc-fr$_{ua_k}$, c-assoc-to$_{ua_r}$}, SET-3, SET-ARS, SET-5*) to *AREQ$_A$*

33 **return** *AREQ$_A$*

B Appendix: PM Core Elements, Relations, and Functions Formal Definitions

Definition 2. [Core Data Elements] The Core Data Elements is the finite set of users, processes, objects, user attributes, object attributes, policy classes, operations, and access rights represented as **U**, **P**, **O**, **UA**, **OA**, **PC**, **Op**, **AR** respectively. where Op $\subseteq 2_1^{AR}$ and ARs = 2_1^{AR} is the non-empty finite set of all subsets of access rights called access right set, ARs.

Definition 3. [Attributes] The finite set of user and object attributes is called the Attributes, AT.
AT = UA $\cup$ OA

Definition 4. [Policy Elements] The finite set of Core Data Elements without the Attributes is called the Policy Elements, PE
PE $=$ U $\cup$ UA $\cup$ OA $\cup$ PC, where O $\subseteq$ OA.

Definition 5. [Assignment] An Assignment is a binary relation ASSIGN on the set PE and the relation must satisfy the given properties [4].
ASSIGN $\subseteq$ (U $\times$ UA) $\cup$ (UA $\times$ UA) $\cup$ (OA $\times$ OA) $\cup$ (UA $\times$ PC) $\cup$ (OA $\times$ PC)

Definition 6. [Association] An Association is a ternary relation ASSOCIATION that represent the authorization of an access rights between policy elements.
ASSOCITION $\subseteq$ UA $\times$ 2_1^{AR} $\times$ AT

Definition 7. [Transitive closure] The transitive closure of the binary relation ASSIGN on the set PE is the transitive relation ASSIGN$^+$ on the set PE such that ASSIGN$^+$ contains ASSIGN and ASSIGN$^+$ is minimal.
ASSIGN$^+$ $\supseteq$ ASSIGN

Definition 8. [Containment] For any x and y in PE, x is said to be contained by y, or y is said to contain x, iff x ASSIGN$^+$ y, i.e. (x, y) $\in$ ASSIGN$^+$ $x, y \in PE : ((x,y) \in ASSIGN^+ \iff \exists s \in iseq_1 PE : (\#s > 1 \wedge \forall i \in \{1,....,(\#s-1)\} : ((s(i),s(i+1)) \in ASSIGN) \wedge x = s(1) \wedge y = s(\#s))$

Definition 9. [Privileges] The Privilege is a ternary relation from a user, an access right, and an attribute.
PRIVILEGE $\subseteq$ U $\times$ AR $\times$ (PE\PC)

Definition 10. [Access Request] An administrative Access Request AAREQ is a ternary relation from a process, an administrative operation, and a conforming non-empty argument sequence.
AREQ $\subseteq$ P $\times$ Op $\times$ Seq$_1$Arg, where Arg = {x |x $\in$ PE $\vee$ x $\in 2^{PE}$ $\vee$ x $\in 2_1^{AR}$}

Definition 11. [Access Decision] An Access Decision is a function Access_Decision to return an accept or a deny for any element in the finite set of administrative access request as its input. Formally, the function can be expressed as:
Access_Decision : AAREQ $\longrightarrow$ {accept, deny}