Enabling the Deployment of ABAC Policies in RBAC Systems

Gunjan Batra[1(✉)], Vijayalakshmi Atluri[1], Jaideep Vaidya[1], and Shamik Sural[2]

[1] MSIS Department, Rutgers Business School, Newark, USA
{gunjan.batra,atluri,jsvaidya}@rutgers.edu
[2] Department of Computer Science and Engineering, IIT Kharagpur, Kharagpur, India
shamik@cse.iitkgp.ernet.in

Abstract. The flexibility, portability and identity-less access control features of Attribute Based Access Control (ABAC) make it an attractive choice to be employed in many application domains. However, commercially viable methods for implementation of ABAC do not exist while a vast majority of organizations use Role Based Access Control (RBAC) systems. In this paper, we present a way in which organizations having a RBAC system can deploy an ABAC policy. Thus, we propose a method for the translation of an ABAC policy into a form that can be adopted by an RBAC system. We compare the cost of enforcement in ABAC and RBAC with respect to time taken to evaluate an access request, and experimentally demonstrate that RBAC is significantly better in this respect. Since the cost of security management is more expensive under RBAC when compared to ABAC, we present an analysis of the different management costs and present mitigation approaches by considering various administrative operations.

1 Introduction

Role-Based Access Control (RBAC) has been a well accepted standard for access control for more than three decades. Most businesses today use RBAC to assign access to the network and systems based on job title or defined role. However, a primary limitation of RBAC is its significant dependence on user identity for mapping it to a set of roles. As an alternative, the Attribute Based Access Control (ABAC) model has been developed. In ABAC, subject requests to perform operations on objects are granted or denied based on assigned attributes of the subject, assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions [8]. As such, ABAC can comprehensively handle various factors affecting access control decisions like location, time, server load, etc., and also facilitates inter-domain accesses. Furthermore, use of user and object attributes for defining access control makes ABAC more portable across organizational domains. Indeed the flexibility, portability and *identity-less* access control features make ABAC very

Published by Springer International Publishing AG, part of Springer Nature 2018. All Rights Reserved
F. Kerschbaum and S. Paraboschi (Eds.): DBSec 2018, LNCS 10980, pp. 51–68, 2018.
https://doi.org/10.1007/978-3-319-95729-6_4

attractive to be employed in many application domains, including cloud computing, web services, collaborative and coalition based systems, as it is feasible to make access control decisions without any prior knowledge of the subject. As a result, many organizations are now moving to ABAC, which is a non-identity based model and is highly dynamic and flexible. Indeed, Gartner predicts that by 2020, 70% of enterprises would use ABAC as the dominant mechanism to protect critical assets, up from less than 5% today [8].

ABAC is also advantageous from a security management perspective. Since ABAC allows for the creation of access policies based on the existing attributes of the users and objects, rather than the manual assignment of roles, ownership or security labels, it minimizes the need for manual intervention in configuring and deploying access control. More specifically, if an employee changes roles or leaves the company, an administrator must manually change access rights accordingly perhaps within several systems. As organizations expand and contract, partner with external entities, and modernize systems, this method of managing user access becomes increasingly difficult and inefficient [8]. On the other hand, such organizational changes effectively do not incur any manual cost under an ABAC system as no changes need to be made to the access control configuration. As such, the administrative cost of ABAC is significantly lower as compared to that of RBAC (or even that of discretionary access control (DAC)).

Despite many organizations wanting to adopt ABAC as their method of access control, there do not yet exist many commercial ABAC implementations. Some vendors such as Axiomatics, do offer ABAC implementations as dynamic authorization solutions, however, ABAC implementations have not yet been incorporated into any of the popular operating systems, or applications such as DBMS, etc. As such, organizations wanting to adopt ABAC, need to implement it on their own, which can often be error-prone and unreliable. Since RBAC is widely deployed in almost all commercially available OS and application systems, our basic idea in this paper is to propose an approach that can help realize an ABAC policy using a RBAC system. Essentially, we translate the ABAC policies into an equivalent RBAC configuration so that a user gains access to a resource in RBAC if and only if that user has the specified access under ABAC.

There are a number of benefits for taking this path to enforcing access control. First, our approach is an alternative where ABAC can simply be realized with a readily available RBAC implementation. Second, it is well known that when an access request is submitted by a user, the enforcement in ABAC is much more expensive in terms of time and processing power than that in RBAC. We experimentally show that this is indeed true. As a result, with our approach, one can enjoy the benefits of ABAC (such as flexibility, etc.) as well as the benefits of RBAC (efficient authorization enforcement). Due to this, one may still want to go on our proposed path, even if an ABAC implementation were to be available in future. Third, as ABAC paradigm is more suited for cloud environments due to its fine-grained property. Therefore, our proposed approach is a solution for the organizations that have an RBAC system in place and would like to be a part of cloud or another data sharing environment.

However, while RBAC administration and maintenance are considered less costly when compared to DAC, as mentioned earlier, it is more expensive when compared to ABAC. Recognizing this fact that the maintenance cost in RBAC is significantly higher than that of ABAC, we propose methods to handle such changes effectively by considering the different change scenarios such as addition/deletion of users and objects, changes to ABAC policies including addition/deletion of subject/object attributes, addition/deletion of ABAC rules.

The rest of this paper is organized as follows. In Sect. 2, we provide a brief overview of ABAC and RBAC. In Sect. 3, we discuss the problem of converting an ABAC policy to RBAC. The idea is to cover all the authorizations of ABAC model and build an equivalent RBAC model. We also examine how the number of policy rules in ABAC relates to the number of roles in RBAC. In Sect. 4 we experimentally compare the cost of enforcement in an ABAC system to the cost of enforcement in RBAC once the ABAC policies are implemented in the RBAC system. In Sect. 5, we discuss the management cost by considering the administrative operations in this system and ways to make it more efficient. In Sect. 6, we discuss related work. Finally, in Sect. 7, we conclude the paper and discuss future research directions.

2 Preliminaries

In this section, we briefly present the attribute based access control (ABAC) model [1,12] and the Role Based Access Control model [7], upon which all of the following work is based. In ABAC, the authorization to perform an operation (e.g.,read/write/modify) is granted based on the attributes of the requesting user, requested object, and the environment in which a request is made. In RBAC, the authorization to perform an operation is based on role of a user requesting permission to access and object.

2.1 RBAC

The basic components of RBAC are as follows:

Users ($\mathcal{U}$): Represents a set of authorized users/subjects. Each member of this set is denoted as u_i, for $1 \leq i \leq |\mathcal{U}|$.

Objects ($\mathcal{O}$): Represents a set of resources to be protected. Each member of this set is denoted as o_i, for $1 \leq i \leq |\mathcal{O}|$.

ROLES ($\mathcal{R}$): Represents a set of roles. Each member of this set is denoted as r_i, for $1 \leq i \leq |\mathcal{R}|$.

$\mathcal{OPS}$: Represents a set of operations. Each member of this set is denoted as op_i, for $1 \leq i \leq |\mathcal{OPS}|$.

$\mathcal{PRMS}$: Represents the set of Permissions $\mathcal{PRMS} \subseteq \{(o\text{-}op) \mid o \in \mathcal{O} \wedge op \in \mathcal{OPS}\}$.

$\mathcal{UA}$: User Role assignment relation, $\mathcal{UA} \subseteq \mathcal{U} \times \mathcal{R}$ is a many-to-many mapping of user to role assignments. We use a $m \times n$ binary matrix to represent $\mathcal{UA}$.

$\mathcal{PA}$: Permission Role assignment relation, $\mathcal{PA} \subseteq \mathcal{PRMS} \times \mathcal{R}$ is a many-to-many mapping of permission to role assignments.

2.2 ABAC

The basic components of ABAC are as follows:

Users ($\mathcal{U}$): Represents a set of authorized users/subjects. Each member of this set is denoted as u_i, for $1 \leq i \leq |\mathcal{U}|$.

Objects ($\mathcal{O}$): Represents a set of resources to be protected. Each member of this set is denoted as o_i, for $1 \leq i \leq |\mathcal{O}|$.

Environment ($\mathcal{E}$): Represents a set of environment conditions, independent of users and objects. Each member of this set is denoted as e_i, for $1 \leq i \leq |\mathcal{E}|$.

$\mathcal{U}_\mathcal{A}$: Represents a set of user attribute names. Members of these sets are represented as ua_i, for $1 \leq i \leq |\mathcal{U}_\mathcal{A}|$, Each ua_i is associated with a set of possible values it can acquire. For instance, if a user attribute *Position* is associated with the values {*Manager*, *Associate*, *Customer*}, then for every $u \in \mathcal{U}$, value of the attribute *Position* can be either *Manager*, *Associate* or *Customer*.

$\mathcal{O}_\mathcal{A}$: Represents a set of object attribute names. Members of these sets are represented as oa_i, for $1 \leq j \leq |\mathcal{O}_\mathcal{A}|$. Each oa_i is associated with a set of possible values it can acquire. For instance, if an object folder with records of customers has object attribute *Region* associated with a set of values {*EastCoast*, *WestCoast*}, then for every $o \in \mathcal{O}$, *Region* can be either *EastCoast* or *WestCoast*.

For the sake of simplicity, in this paper, we ignore environmental attributes.

$\mathcal{U}_\mathcal{C}$: Represents a set of all possible user attribute conditions denoted as uc_j, for $1 \leq j \leq |\mathcal{U}_\mathcal{C}|$. Members of this set are represented as equalities of the form $n = c$, where n is a user attribute name and c is either a constant or *any*. For instance if user attribute *Position* has possible values {*Manager, Associate, Customer*} and user attribute *Region* has possible values as $\{EastCoast, WestCoast\}$, then $\mathcal{U}_\mathcal{C}$ will be a set comprising of $\{Position = Manager\}$, $\{Position = Associate\}$, $\{Position = Customer\}$, $\{Position = any\}$, $\{Region = EastCoast\}$, $\{Region = WestCoast\}$, $\{Specialty = any\}$. Note here, that the condition $n = any$ does not have to be explicitly chosen. It is set only if at least one other condition for n is present. We use the notation $\mathcal{U}_\mathcal{C}.u_i$ to express the user attribute condition set of a user u_i.

$\mathcal{O}_\mathcal{C}$: Represents a set of all possible object attribute conditions denoted as oc_k, for $1 \leq k \leq |\mathcal{O}_\mathcal{C}|$. Members of this set are represented as equalities of the form $n = c$, where n is an object attribute name and c is either a constant or *any*. For instance if object attribute *Region* has possible values {*EastCoast, WestCoast*} and object attribute *RecordOf* has possible values {*Manager, Associate, Customer, Staff*}, then $\mathcal{O}_\mathcal{C}$ will be a set comprising of $\{Region = EastCoast\}$,

$\{Region = WestCoast\}$, $\{Region = any\}$, $\{RecordOf = Manager\}$, $\{RecordOf = Customer\}$, $\{RecordOf = Associate\}$, $\{RecordOf = Staff\}$, $\{RecordOf = any\}$. For an attribute name n, if the value of c is *any*, then the attribute n is not relevant for making the corresponding access decision. Therefore, as above, the condition $n = any$ does not have to be explicitly chosen. It is set only if at least one other condition for n is present. We use the notation $\mathcal{O}_{\mathcal{C}}.o_i$ to express the object attribute condition set of an object o_i. ABAC Policy base Π_A: This represents a set of access rules in the ABAC system. Each member of this set is denoted as π_i, for $1 \leq$ i $\leq |\Pi|$, where π is a quadruple of the form $\langle uc, oc, ec, op\rangle$. If a user makes a request to access an object, the policy base is searched for any rule through which the user can gain access. If such a rule exists, then access is granted, otherwise it is denied.

In $\mathcal{U}_{\mathcal{C}}$ and $\mathcal{O}_{\mathcal{C}}$ we have represented the attribute conditions as equalities, however, our approach is flexible to include the complex attribute condition constructs (inequalities, negation, subset, etc.) by converting them to their corresponding list of attributes conditions. In the following, we define the mapping between users and user attribute conditions as well as objects and object attribute conditions.

Table 1. $\mathcal{UAR}$

User (u)	$Region = EastCoast(uc_1)$	$Position = Manager(uc_2)$	$Region = WestCoast(uc_3)$	$Position = Associate(uc_4)$
u_1	0	1	1	0
u_2	0	0	1	1
u_3	1	1	0	0
u_4	1	0	0	1

$\mathcal{UAR}$: User attribute relation $\mathcal{UAR} \subseteq \mathcal{U} \times \mathcal{U}_{\mathcal{C}}$ is a many-to-many mapping of users and user attribute conditions. We use a $m \times n$ binary matrix to represent $\mathcal{UAR}$, where $\mathcal{UAR}[i,j] = 1$, if user u_i satisfies an attribute condition uc_j. As shown in the example in Table 1, user u_1 is an *Manager* whose region is *WestCoast*.

Table 2. $\mathcal{OAR}$

Object (o)	$Region = WestCoast$ (oc_1)	$Region = EastCoast$ (oc_2)	$Recordof = Customer$ (oc_3)
o_1	1	0	1
o_2	0	1	1

Table 3. Policy (Π_A)

Attributes	*Permission*
uc_3, uc_4, oc_1, oc_3	op_1
uc_2, uc_3, oc_1, oc_3	op_1
uc_1, uc_2, oc_2, oc_3	op_1
uc_1, uc_4, oc_2, oc_3	op_1
uc_2, uc_3, oc_1, oc_3	op_2
uc_1, uc_2, oc_2, oc_3	op_2

$\mathcal{OAR}$: Object attribute relation, $\mathcal{OAR} \subseteq \mathcal{O} \times \mathcal{O_C}$ is a many-to-many mapping of objects and the set of all attributes conditions, where we again use a $m \times n$ binary matrix to represent $\mathcal{OAR}$. $\mathcal{OAR}[i,j] = 1$ if an object o_i satisfies an object attribute condition oc_j. Table 2 shows an example where object o_1 is the *recordof Customer* in *WestCoast region*.

3 ABAC to RBAC Translation

This section presents our methodology to translate the ABAC policy configuration to an equivalent one in RBAC. Towards this end, we first formally define the *optimal ABAC to RBAC translation problem* and then present our approach.

3.1 Problem Formulation

Intuitively, our goal is to discover RBAC roles from ABAC policy base in such a way that the set of RBAC roles is minimum and at the same time the authorizations are the same as those under ABAC. In the following, we formalize the definition of the ABAC to RBAC translation problem.

$\mathcal{A}$: An authorization a having the form of $\langle u, o, op \rangle$ denotes that the user u is allowed to perform an operation op on the object o, where $u \in \mathcal{U}$, $o \in \mathcal{O}$, and $op \in \mathcal{OPS}$. We use $u.a$, $o.a$ and $op.a$ to denote the user, object and operation associated with a. We denote the set of all authorizations as $\mathcal{A}$. For each operation $op_i \in \mathcal{OPS}$, we define $\mathcal{A}_{op_i} \subseteq \mathcal{A}$ such that for every $a \in \mathcal{A}_{op_i}$, $op.a = op_i$. For example, if $\mathcal{OPS} = \{\text{read,write}\}$, we have $\mathcal{A}_{\texttt{read}}$ and $\mathcal{A}_{\texttt{write}}$ such that $\mathcal{A}_{\texttt{read}} \cup \mathcal{A}_{\texttt{write}} = \mathcal{A}$.

Given an ABAC policy base Π_A, we say $\mathcal{A}$ covers π if for every user u and object o combination where u is allowed to perform operation op on o, there exists an authorization $a = \langle u, o, op \rangle \in \mathcal{A}$. (In the following subsection, we provide an algorithm on how to derive such $\mathcal{A}$ from Π.) Similarly, given an RBAC policy Π_R, we say $\mathcal{A}$ covers Π_R if for every user u and object o combination where u is allowed to perform operation op on o in Π_R, there exists an authorization $a = \langle u, o, op \rangle \in \mathcal{A}$. Now we are ready to formally define the optimal ABAC to RBAC translation problem.

Problem Statement. Given an ABAC policy Π_A, Users $\mathcal{U}$, Objects $\mathcal{O}$, User Attribute relation ($\mathcal{UAR}$), and Object Attribute relation ($\mathcal{OAR}$), the ABAC to RBAC translation problem is to identify a RBAC policy Π_R that includes a set of Roles $\mathcal{R}$, $\mathcal{PA}$ and $\mathcal{UA}$ such that the set of authorizations $\mathcal{A}$ derived from Π_A and Π_R are equal and the number of roles $|\mathcal{R}|$ is minimum.

3.2 Approach

In this section, we discuss how we develop a system that will translate ABAC policies in a manner that they can be implemented by an RBAC. The $\mathcal{UAR}$, $\mathcal{OAR}$ and ABAC policy base Π_A is fed to an ABAC-RBAC Translator which

Algorithm 1. Generating $\mathcal{A}$ and $\mathcal{UPA}$

Require: $\mathcal{UAR}$, $\mathcal{OAR}$, Π_A
INITIALIZE $\mathcal{A} = \emptyset$
for all $(u_i\text{-}o_j)$ combinations in $\mathcal{UAR}$ and $\mathcal{OAR}$ **do**
  for all π_k in Π_A **do**
    if $\pi_k \subseteq \mathcal{U_C}.u_i \bigcup \mathcal{O_C}.o_j$ **then**
      $\mathcal{A} \leftarrow \mathcal{A} \bigcup (u_i, o_j, op_k.\pi_k)$
    end if
  end for
end for
INITIALIZE $\mathcal{UPA}$ of size $M \times N$ such that $M = 1, \ldots, |U|$; $N = 1, \ldots, |u_i - o_i|$ in $\mathcal{A}$
for all a_l in $\mathcal{A}$ **do**
  $\mathcal{UPA}(u_i.a_l,\ o_j.a_l\text{-}op_k.a_l) \leftarrow 1$
end for

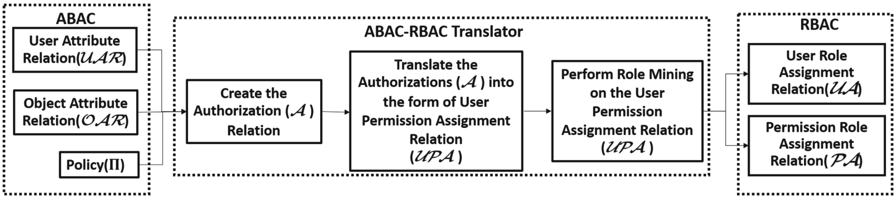

Fig. 1. Approach for Deployment of ABAC in RBAC

generates Π_R, which includes $\mathcal{R}$ and the corresponding $\mathcal{UA}$ and $\mathcal{PA}$ that form the RBAC policy. The detailed process for translation is described below and has been shown in Fig. 1.

Steps for ABAC to RBAC translation:

Step 1. Construct the set of Authorizations $\mathcal{A}$ from the User Attribute Relation ($\mathcal{UAR}$), Object Attribute Relation ($\mathcal{OAR}$) and the ABAC policy base (Π_A): For each user(u_i)-object(o_j) combination from $\mathcal{UAR}$ and $\mathcal{OAR}$, we check if their corresponding attribute conditions($\mathcal{U_C}.u_i$ and $\mathcal{O_C}.o_i$) form a superset of any of the given ABAC rules in Π_A. For every such superset occurence, we include the set comprising of user(u_i), object(o_j) and the operation($op_k.\pi_k$) in $\mathcal{A}$. The procedure is automated in the first part of Algorithm 1. As an example, given $\mathcal{UAR}$ in Table 1, $\mathcal{OAR}$ in Table 2 and Π in Table 3, the derived $\mathcal{A}$ is shown in Table 4.

Step 2. Derive User Permission Assignment ($\mathcal{UPA}$) from $\mathcal{A}$: The $\mathcal{UPA}$ is defined as an $M \times N$ matrix, where $M = |\mathcal{U}|$ comprising of a row for each user, and $N = |\mathcal{O}\text{-}op|$, comprising of a column for each object and operation combination in $\mathcal{A}$. Using ($\mathcal{A}$), we derive ($\mathcal{UPA}$) as follows: We consider all the Users in ($\mathcal{A}$) and associate the objects with permissions to form PRMS(o-op) in RBAC.

Table 4. $\mathcal{A}$

User u	Object o	Permission op_i
u_1	o_1	op_1
u_2	o_1	op_1
u_3	o_2	op_1
u_4	o_2	op_1
u_1	o_1	op_2
u_3	o_2	op_2

Table 5. $\mathcal{UPA}$

	o_1-op_1	o_1-op_2	o_2-op_1	o_2-op_2
u_1	1	1	0	0
u_2	1	0	0	0
u_3	0	0	1	1
u_4	0	0	1	0

There is a row in $\mathcal{UPA}$ for each user and a column for each PRMS(o-op). For each row, if the (o-op) is true for that user, the corresponding cell is filled with 1, otherwise with 0. The procedure is automated in the second part of Algorithm 1. Given $\mathcal{A}$ in Table 4, the derived $\mathcal{UPA}$ is shown in Table 5.

Step 3. Derive User Assignment Relation ($\mathcal{UA}$) and Permission Assignment Relation ($\mathcal{PA}$) by performing Role Mining: For the automation of this step, we have used DEMiner algorithm proposed by Uzun et al. [3]. The primary reason to choose this is because it generates a compact set of roles which are disjoint in their permissions. As a result, it makes administration of access requests much easier, which is in sync with the idea of this work. When a user requests for a specific permission, there will be a single role with that specific permission, thus making the access control decision faster and efficient. This is the reason why we choose this algorithm as the benchmark. It reduces the administrative cost, as the roles generated are non overlapping and the access request decision is evaluated faster than any other role mining algorithm that produces overlapping roles.

We performed slight modification to the DeMiner algorithm by sorting the users in the $\mathcal{UPA}$ in decreasing order of the number of $\mathcal{PRMS}$ before applying the algorithm on our dataset. This helped improve the efficiency and effectiveness of the algorithm in terms of time and the number of roles created. Considering our example once again, given $\mathcal{UPA}$ in Table 5, the derived $\mathcal{UA}$ and $\mathcal{PA}$ are shown in Tables 6 and 7, respectively.

Theorem 1. *Let $\mathcal{A}$ be the set of authorizations covered by Π_A and $\mathcal{R}$. If $|\Pi_A|$ is the minimum number of ABAC rules required to cover $\mathcal{A}$ and $|\mathcal{R}|$ is the minimum number of roles required to cover $\mathcal{A}$, then $|\Pi_A| \geq |\mathcal{R}|$.*

Proof. Let 'k' be the minimum number of ABAC Rules $|\Pi_A|$, where $\Pi_A = \{\pi_1, \pi_2, \pi_3 \ldots \pi_k\}$ that cover a set of authorizations $\mathcal{A}$ and let 'n' be the minimum number of RBAC roles $\mathcal{R}$ that cover the same set of authorizations $\mathcal{A}$ is $\mathcal{R} = \{r_1, r_2, r_3, \ldots r_n\}$.

Table 6. $\mathcal{UA}$

	r_1	r_2	r_3	r_4
u_1	1	1	0	0
u_2	1	0	0	0
u_3	0	0	1	1
u_4	0	0	1	0

Table 7. $\mathcal{PA}$

	o_1-op_1	o_1-op_2	o_2-op_1	o_2-op_2
r_1	1	0	0	0
r_2	0	1	0	0
r_3	0	0	1	0
r_4	0	0	0	1

Because 'k' is the minimum number of rules, each rule covers atleast one unique authorization. So, if we map each of the policy rules π_i in ABAC to a role r_j in RBAC (where both π_i and r_j cover same set of authorizations in $\mathcal{A}$), we will get exactly 'k' roles. We have shown the same in Example 1 described below. Therefore, for every rule we can create one corresponding role which will cover same set of authorizations. So, we can infer that in all possible cases, the count of roles to express a set of authorizations $\mathcal{A}$ will never be more than the count of rules. In the worst case, $|\mathit{\Pi}_A|$ and $|\mathcal{R}|$ will be equal.

So far, we know that, for 'n' to be the minimum roles required to express $\mathcal{A}$, 'k' has to be equal to 'n' or greater than 'n'. Else we cannot say that 'n' is the minimum number of roles (i.e. $k \geq n$). To check if 'k' could be less than or equal to 'n', we conjecture that, we can map the authorizations expressed by a single role in RBAC to a single rule in ABAC. We use a simple counter example to disprove the above conjecture. We can see in Example 2 below, that for 2 RBAC roles, we need atleast 6 ABAC rules to express the same authorizations. We need 5 ABAC rules: π_1, π_2, π_3, π_4 and π_5 to describe authorizations of r_1 and one ABAC rule π_6 to describe authorizations of r_2. Note that it is impossible to describe role r_1 by a single ABAC rule as r_1 covers the set users which satisfy no common attribute condition(s).

In case we have common attributes between users or objects in the role, for example in role r_2, user u_1 and u_4 have a common attribute uc_4, then one ABAC rule could cover the same authorizations of r_2, i.e. π_6 (this will give access to both u_1 and u_4 to o_3 to perform op_1 as both u_1 and u_4 satisfy user attribute condition uc_4). Hence, we need at least 6 ABAC Rules to express the authorizations covered by 2 roles. Thus, Example 2 is a testimony to that fact that it is possible to have an RBAC role where no single ABAC rule can express the authorizations of that particular single role.

To conclude, the number of Policy Rules in ABAC is always greater than or equal to the number of Roles in RBAC, i.e., $|\mathit{\Pi}_A| \geq |\mathcal{R}|$. □

Example 1: An ABAC rule π_1: $\langle uc_1, oc_1, read\rangle$ gives users u_1 and u_2 (both having attribute uc_1), *read* access on object o_1(having attribute oc_1); i.e. two authorizations $\mathcal{A}_1$ and $\mathcal{A}_2$, where $\mathcal{A}_1 = \langle u_1, o_1, read\rangle$ and $\mathcal{A}_2 = \langle u_2, o_1, read\rangle$. The corresponding role r_1 will be assigned to users(u_1,u_2) and will be granted permission (o_1,*read*).

Table 8. $\mathcal{A}$

User	Object	Permission
u_1	o_1	op_1
u_1	o_2	op_1
u_1	o_3	op_1
u_2	o_1	op_1
u_2	o_2	op_1
u_3	o_1	op_1
u_3	o_2	op_1
u_4	o_3	op_1

Table 9. $\mathcal{UA}$

	r_1	r_2
u_1	1	1
u_2	1	0
u_3	1	0
u_4	0	1

Table 10. $\mathcal{PA}$

	o_1-op_1	o_2-op_1	o_3-op_1
r_1	1	1	0
r_2	0	0	1

Table 11. $\mathcal{UAR}$

$User$	uc_1	uc_2	uc_3	uc_4
u_1	1	0	0	1
u_2	0	1	0	0
u_3	0	0	1	0
u_4	0	0	0	1

Table 12. $\mathcal{OAR}$

$Object$	oc_1	oc_2	oc_3
o_1	1	0	0
o_2	0	1	0
o_3	0	0	1

Example 2: An RBAC system which has two roles r_1 and r_2 giving authorizations $\mathcal{A}$ (Table 8) to four users(u_1, u_2, u_3, u_4). The $\mathcal{UA}$ relation is given in Table 9 and $\mathcal{PA}$ relation is in Table 10. The users and objects satisfy the attribute conditions as shown in the User Attribute Relation $\mathcal{UAR}$ (Table 11) and Object Attribute Relation $\mathcal{OAR}$ (Table 12). In total, atleast 6 ABAC policy rules are required to cover the authorizations of both the roles. They are as follows:

π_1: $\langle\ uc_1\ \rangle$, $\langle$ Any $\rangle$ π_4: $\langle\ uc_3\ \rangle, \langle\ oc_1\ \rangle$
π_2: $\langle\ uc_2\ \rangle, \langle\ oc_1\ \rangle$ π_5: $\langle\ uc_3\ \rangle, \langle\ oc_2\ \rangle$
π_3: $\langle\ uc_2\ \rangle, \langle\ oc_2\ \rangle$ π_6: $\langle\ uc_4\ \rangle, \langle\ oc_3\ \rangle$

4 Experimental Comparison of Access Request Evaluation Cost in ABAC and RBAC

In order to compare the time taken for *access request* (AR) evaluation, the same ABAC and RBAC policy, we need to first create two equivalent policies and compare the time taken to evaluate the same set of access requests. This is done as follows. First, a synthetic ABAC policy base ($\mathit{\Pi}_A$) is created. For creating synthetic ABAC Policies we used the data generator used by Talukdar et al. [12]. Next, using the ABAC policy base and the User Attribute relation ($\mathcal{UAR}$) and Object Attribute Relation ($\mathcal{OAR}$), the ($\mathcal{UPA}$) relation is created, on which

Role Mining is done on the ($\mathcal{UPA}$) relation to create the User Assignment ($\mathcal{UA}$) and Role Assignment ($\mathcal{PA}$) relation. Any Role Mining algorithm could be used, as long as it completely covers the given $\mathcal{UPA}$. In this particular case, we use the DEMiner algorithm proposed by Uzun et al. [3].

For each set of experiments, we have compared the access request evaluation time for both ABAC and RBAC. The experiments are performed on a Intel Core i7 2.60 GHz machine with 8.00 GB memory running 64-bit Windows 10. Since we are interested in seeing how the access request evaluation cost changes with respect to different parameters, we run fours sets of experiments where one parameter is varied while keeping the rest constant. Specifically, we examine the following four different scenarios: (1) increasing the rule size, (2) increasing the number of attributes in ABAC rules, (3) increasing the number of users and objects, and (4) increasing the count of positive authorizations. Here positive authorizations imply access requests that should be granted, while negative authorizations imply access requests that should be rejected. To compare the efficiency of ABAC and RBAC, we have evaluated the time taken to evaluate access requests for 100 user-object pairs. For the first three cases, we take 50 random positive authorizations and 50 random negative authorizations. For the last case, we have increased the count of positive authorizations and reduced the count of negative authorizations by keeping total access requests at 100. Further these access request evaluations were run three times and the time was averaged over all of these runs.

The key parameters are the number of users ($\mathcal{U}$), objects ($\mathcal{O}$), user attributes ($\mathcal{U_C}$), object attributes ($\mathcal{O_C}$), number of rules given (Π_A) to the ABAC system. In Tables 13, 14 and 15, the first column $|\mathcal{U}|$ is count of users, the second column $|\mathcal{O}|$ is count of objects, third column $|\mathcal{U_C}|$ is count of user attribute conditions, fourth column $|\mathcal{O_C}|$ is count of object attribute conditions, fifth column $|\Pi_A|$ is count of ABAC policy rules, $|\mathcal{R}|$ is the number of RBAC roles discovered after role mining, $AvgRT_{\texttt{ABAC}}$ is the average run time for ABAC and $AvgRT_{\texttt{RBAC}}$ is the average run time for RBAC. In Table 16, there are two additional columns for count of Positive Authorizations and Negative Authorizations.

For all the experiments, we observe that the count of roles $|\mathcal{R}|$ discovered after role mining is much less than the count of ABAC policy rules $|\Pi_A|$ for the same set of authorizations. We can also observe that the run time for access request evaluation for ABAC is significantly greater than the run time for access request evaluation for RBAC. Next we see the individual effects of varying the parameters while keeping all others constant.

Varying Number of ABAC Rules: Table 13 and Fig. 2 show the results obtained for access request evaluation time of ABAC and RBAC, while increasing the count of ABAC Rules, but keeping all other parameters constant. We have varied the ABAC rule count between 500, 1000, and 2000. We observe that the count of RBAC roles discovered was 200 in all the three cases. The average access request evaluation time for RBAC remains roughly the same, whereas the access request evaluation time for ABAC increases linearly. This is due to the fact that

Table 13. Increasing rule size

$\|\mathcal{U}\|$	$\|\mathcal{O}\|$	$\|\mathcal{U}_C\|$	$\|\mathcal{O}_C\|$	$\|\Pi_A\|$	$\|\mathcal{R}\|$	$AvgRT_{\texttt{ABAC}}$ (in ms)	$AvgRT_{\texttt{RBAC}}$ (in ms)
200	200	500	500	500	200	19.385	0.032
200	200	500	500	1000	200	35.227	0.032
200	200	500	500	2000	200	69.108	0.032

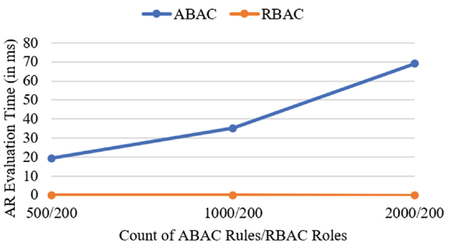

Fig. 2. Increasing rule size

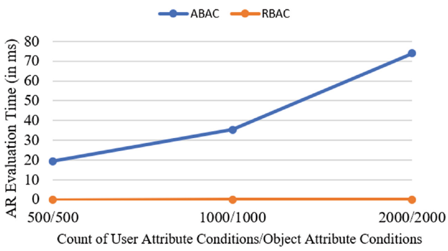

Fig. 3. Increasing attribute size

Table 14. Increasing attribute size

$\|\mathcal{U}\|$	$\|\mathcal{O}\|$	$\|\mathcal{U}_C\|$	$\|\mathcal{O}_C\|$	$\|\Pi_A\|$	$\|\mathcal{R}\|$	$AvgRT_{\texttt{ABAC}}$ (in ms)	$AvgRT_{\texttt{RBAC}}$ (in ms)
200	200	500	500	500	200	19.385	0.032
200	200	1000	1000	500	200	35.381	0.032
200	200	2000	2000	500	200	73.894	0.033

Table 15. Increasing User/Object Size

$\|\mathcal{U}\|$	$\|\mathcal{O}\|$	$\|\mathcal{U}_C\|$	$\|\mathcal{O}_C\|$	$\|\Pi_A\|$	$\|\mathcal{R}\|$	$AvgRT_{\texttt{ABAC}}$ (in ms)	$AvgRT_{\texttt{RBAC}}$ (in ms)
300	300	150	150	50	41	0.656	0.008
400	400	150	150	50	41	0.705	0.008
500	500	150	150	50	41	0.658	0.009

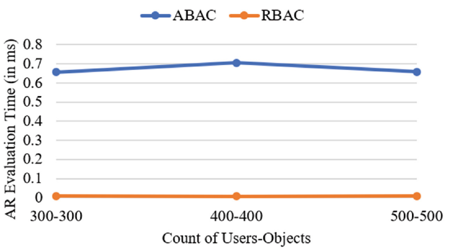

Fig. 4. Increasing User Object Size

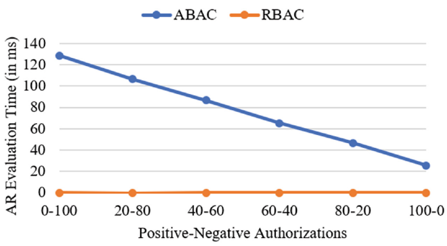

Fig. 5. Increasing Positive Authorizations

Table 16. Increasing Positive Authorisations

$\|\mathcal{U}\|$	$\|\mathcal{O}\|$	$\|\mathcal{U}_C\|$	$\|\mathcal{O}_C\|$	$\|\Pi_A\|$	$\|\mathcal{R}\|$	Positive accesses	Negative accesses	$AvgRT_{\texttt{ABAC}}$ (in ms)	$AvgRT_{\texttt{RBAC}}$ (in ms)
200	200	2000	2000	500	200	0	100	128.640	0.032
200	200	2000	2000	500	200	20	80	106.464	0.031
200	200	2000	2000	500	200	40	60	86.615	0.032
200	200	2000	2000	500	200	60	40	65.242	0.032
200	200	2000	2000	500	200	80	20	46.610	0.031
200	200	2000	2000	500	200	100	0	25.495	0.032

the size of $\mathcal{UA}$ and $\mathcal{PA}$ remain the same for the three cases, whereas the count of ABAC rules to be checked for granted access doubles each time.

Varying Number of User and Object Attributes: Table 14 and Fig. 3 show the results obtained for access request evaluation time of ABAC and RBAC, while increasing the count of Users Attributes and Objects Attributes for ABAC policy rules, while keeping all other parameters constant. We have increased both user and object attribute counts for ABAC rules using values 500, 1000 and 2000 for both. We observe that the count of RBAC roles discovered was 200 in all three cases. The average access request evaluation time for RBAC remains roughly the same, whereas the access request evaluation time for ABAC increases linearly. This is because of the fact that the size of $\mathcal{UA}$ and $\mathcal{PA}$ relation remains the same for the three cases, whereas the count of attributes to be checked for granting access in each rule increases.

Varying Number of Users and Objects: Table 15 and Fig. 4 show the results obtained for access request evaluation time of ABAC and RBAC, while increasing the count of Users and Objects, but keeping all other parameters constant. Again, we observe that the average access request evaluation time in ABAC is almost 75 times that of RBAC.

Varying Number of Positive Authorizations: Table 16 and Fig. 5 show the results obtained for access request evaluation time of ABAC and RBAC, while varying the count of positive authorizations, but keeping all other parameters constant. Out of the 100 random user-object access requests we predetermine the number of accesses that would evaluate to be positive (granted). These positive access requests were varied between the values 0, 20, 40, 60, 80, and 100, with the remaining requests used being negative requests. We observe that the average access request evaluation time in RBAC is roughly the same as earlier, however the average access request evaluation time in ABAC has reduced linearly. An ABAC system checks each policy rule, one by one, to see if it can grant the access. When an access request is granted, no further policy rules need to be checked; whereas, when an access request is denied the ABAC system keeps on checking all the policy rules it has.

The overall results indicate the fact that evaluation of access requests in RBAC is significantly faster across the board in all cases than those of ABAC.

5 Maintenance Cost Comparison in ABAC and RBAC

In this section, we discuss the configuration and the maintenance cost while dealing with various changes to the ABAC policy and the cost of translating them into an equivalent RBAC policy. The list of operations that can be performed on the original ABAC policy system is as follows:

1. Addition/Deletion of Rules
2. Addition/Deletion of Users/Objects
3. Addition/Deletion of User/Object Attributes
4. Addition/Deletion of Attributes in ABAC Rules

We know that in ABAC, the initial configuration cost is the sum of number of attributes of users, objects and the policy rules, i.e., $|\mathcal{U}_\mathcal{C}|$ + $|\mathcal{O}_\mathcal{C}|$ + $|\Pi_A|$. Whereas, when we implement ABAC in an RBAC system, the initial configuration cost will be the sum of the number of user role assignments and role permission assignments, i.e., $|\mathcal{UA}|$ + $|\mathcal{PA}|$. The maintenance cost of the above mentioned operations will be negligible in case of an ABAC system as every access request is evaluated at the time of enforcement. However, if we wish to deploy the ABAC policies using an RBAC system with our approach, the maintenance cost for some operations vary from making changes to the RBAC system directly to performing the entire ABAC-RBAC translation again. In the following, we have identified the maintenance cost associated with each change operation. While Fig. 6 provides the overview of how these changes are handled, the exact approach for each change is discussed in detail. To discuss the way these change operations are handled, we have divided them into two types based on the type of effort required. Specifically, some changes require the ABAC-RBAC translation to be done all over. On the other hand, due to the additional information that we maintain, they do not require such translation to be done again, but lend itself to make the relevant changes to the RBAC policy directly. In the following subsections, we elaborate on these cases, and discuss what additional information need to maintained. It turns out that, very few cases require performing the ABAC-RBAC translation all over again.

5.1 Changes Requiring Direct Modification to the RBAC Policy

Addition of Users: When a new user is added to the system, the $\mathcal{UPA}$ changes which would require performing role mining all over again. However, if we keep the user attributes required for that role, we can avoid this expensive step, as we can simply derive which role to assign to the user. Therefore, we create a Role User Attribute Assignment Relation $\mathcal{RUA}$, which is a many-to-many mapping of roles to user attribute conditions, i.e., $\mathcal{RUA} \subseteq \mathcal{R} \times \mathcal{U}_\mathcal{C}$. We use a $m \times n$

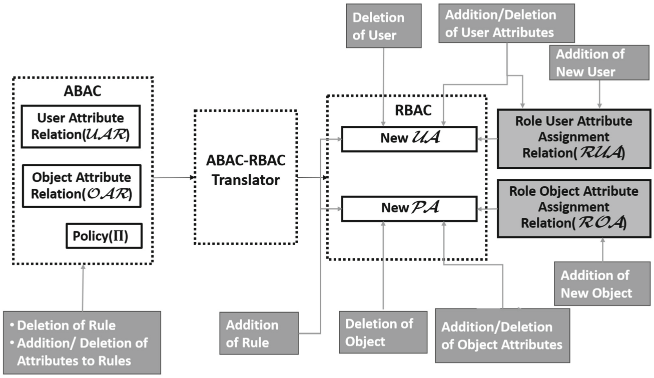

Fig. 6. Management of Administrative Operations on the system

binary matrix to represent $\mathcal{RUA}$, where $\mathcal{RUA}[i,j] = 1$, if user attribute condition uc_j is present in all the users assigned to a role r_j.

For example, we created a $\mathcal{RUA}$ in Table 17 using Tables 1 and 6. Notice that role r_1 in $\mathcal{RUA}$, has $uc_3 = 1$ as uc_3 is present in both the users assigned to r_1 (u_1 and u_2). Basically, attribute conditions assigned to a role are the set of maximum possible common attribute conditions of users in a given role. When a new user is added we can now simply select the roles to be assigned to this user by checking if user has the user attributes necessary for the role. A new row will be added to $\mathcal{UA}$ to reflect this.

Table 17. $\mathcal{RUA}$

Role	uc_1	uc_2	uc_3	uc_4
r_1	0	0	1	0
r_2	0	1	1	0
r_3	1	0	0	0
r_4	1	0	0	1

Table 18. $\mathcal{ROA}$

Role	oc_1	oc_2	oc_3	op_1	op_2
r_1	1	0	1	1	0
r_2	1	0	1	0	1
r_3	0	1	1	1	0
r_4	0	1	1	0	1

Addition of Objects: Similar to the case of adding a new user, in this case we maintain a Role Object Attribute Assignment Relation $\mathcal{ROA}$ which is a many-to-many mapping of roles to object attribute conditions (O_c) and operations ($\mathcal{OPS}$), i.e., $\mathcal{ROA} \subseteq \mathcal{R} \times (\mathcal{O_C} \bigcup \text{OPS})$. We again use a $m \times n$ binary matrix to represent $\mathcal{ROA}$, where $\mathcal{ROA}[i,j]$=1, if object attribute condition oc_j (or op) is present in all the objects (or operations) assigned to a role r_j. Table 18 shows $\mathcal{ROA}$ created using Tables 2 and 7. The attribute conditions assigned to a role are the set of maximum possible common set of object attribute conditions and permissions in a given role. When a new object is added, we select the roles that contain the permissions to perform an operation on the object by checking if the object has the object attributes necessary for the role. $\mathcal{PA}$ relation will be updated with this $\mathcal{PRMS}(o\text{-}op)$ by adding a corresponding column to it.

Deletion of Users/Objects: When a user is removed, the row corresponding to that user is deleted from $\mathcal{UA}$. Similarly, on deletion of an object, all the permissions associated with the object will be deleted from $\mathcal{PA}$.

Addition of User/Object Attributes: Addition of new attributes to a user requires updates to $\mathcal{UA}$. Essentially, we need to delete the earlier record of this user from $\mathcal{UA}$ and find the new roles to be assigned from $\mathcal{RUA}$ based on this new set of attributes. Similarly, when new attributes to an object are added, the row pertaining to the object needs to be deleted from $\mathcal{PA}$, and a new row need to be added based on this new set of attributes after checking eligibility using $\mathcal{ROA}$.

Addition of Rules: Upon adding an ABAC rule, since the $\mathcal{UPA}$ changes accordingly, we need to redo the ABAC-RBAC translation step to generate the new $\mathcal{UA}$ and $\mathcal{PA}$. However, there is an alternative to avoid this expensive step. Instead, one can add a new role corresponding to this new ABAC rule by examining the users and objects satisfying this new rule and reflect that in $\mathcal{UA}$ and $\mathcal{PA}$. While this is somewhat a manual process, this avoids redoing the translation every time a new ABAC rule is added. However, in this case, the translation step can be performed after a batch of ABAC rules are added. Note, however, that this action might create redundant roles in the system.

5.2 Changes Requiring Redoing of ABAC-RBAC Translation

Deletion of Rules: On deleting an ABAC rule, the $\mathcal{UPA}$ changes, and as a result the step of ABAC-RBAC translation has to be redone, which generates new $\mathcal{UA}$ and $\mathcal{PA}$.

Addition/Deletion of Attributes to ABAC Rules: Since addition or deletion of attributes to a ABAC rule essentially creates a new rule, it results in a new $\mathcal{UPA}$. Therefore, it requires redoing of the ABAC-RBAC translation step.

6 Related Work

There have been attempts in past to integrate ABAC and RBAC. Authors have proposed methods to unify both the models to get benefits of both. Kuhn et al. [10] discussed incorporating attributes into roles to combine the best of ABAC and RBAC and provide an effective access control. Also, Al-Kahtani et al. [11] proposed a model to dynamically assign users to roles using attribute based rules. Further, Jin et al. [9] proposed RABAC: Role-centric Attribute based Access Control where they extend RBAC with user and object attributes and also add a Permission Filtering Policy (PFP) to their model. All these focus on extending RBAC rather than using the basic RBAC that is available in most commercial implementations today.

Huang et al. [5] have proposed a model to integrate ABAC and RBAC at two levels: aboveground and underground. The aboveground level is RBAC model with environment constraints added to it and the underground level uses attribute-based policies for user-role assignment and role-permission assignment. Their work is different from that of ours as they focus on a top-down model to integrate ABAC and RBAC.

To the best of our knowledge, there have been no attempts to address this problem of deployment of an ABAC policy in a RBAC system. The NIST report on ABAC [1] mentions that "while it is possible to achieve ABAC objectives using ACLs or RBAC, demonstrating access control (AC) requirements compliance is difficult and costly due to the level of abstraction required between the AC requirements and the ACL or RBAC model. Another problem with ACL or RBAC models is that if the AC requirement is changed, it may be difficult to identify all the places where the ACL or RBAC implementation needs to be updated." Our approach attempts to draw the benefits from ABAC as well as RBAC by automatically translating a ABAC policy into RBAC.

7 Conclusions and Future Work

In this paper we demonstrate how ABAC can be deployed using an RBAC system. Our evaluation shows that the access request evaluation cost of RBAC is always less than the cost of the ABAC system implementing the same policy. However, since RBAC's maintenance cost is higher than that of ABAC, we also discuss several mitigation strategies to minimize the cost of various administrative operations that cause changes to ABAC. In future, we plan to implement this deployment approach while enforcing segregation of duty constraints [2]. In this work, we assumed there were no environmental conditions in ABAC. In future, we would like to include the environmental conditions as well and see how they translate into a context aware RBAC model.

Acknowledgments. Research reported in this publication was supported by the National Institutes of Health under award R01GM118574, by the National Science Foundation under awards CNS-1564034 and CNS-1624503 and by the National Academies of Sciences, Engineering, and Medicine under the PAK-US Science and Technology Cooperation Program. The content is solely the responsibility of the authors and does not necessarily represent the official views of the agencies funding the research. We would also like to thank Dr. Emre Uzun for his comments.

References

1. Hu, V.C., Ferraiolo, D., Kuhn, R., Schnitzer, A., Sandlin, K., Miller, R., Scarfone, K.: Guide to Attribute Based Access Control (ABAC) definition and considerations. In: NIST Special Publication, 800-162 (2014)
2. Jha, S., Sural, S., Atluri, V., Vaidya, J.: Enforcing separation of duty in attribute based access control systems. In: ICISS, pp. 61–78 (2015)
3. Uzun, E., Lorenzi, D., Atluri, V., Vaidya, J., Sural, S.: Migrating from DAC to RBAC. In: DBSec, pp. 69–84 (2015)
4. Vaidya, J., Atluri, V., Guo, Q.: The role mining problem: finding a minimal descriptive set of roles. In: SACMAT, pp. 175–184 (2007)
5. Huang, J., Nicol, D., Bobba, R., Huh, J.: A framework integrating attribute-based policies into role-based access control. In: SACMAT, pp. 187–196 (2012)
6. Hu, V., Kuhn, R., Ferraiolo, D.: Attribute-based access control. In: IEEE, pp. 85–88 (2015)
7. Sandhu, R., Coyne, E., Feinstein, H., Youman, C.: Role-based access control models. In: IEEE Computer, pp. 38–47 (1996)
8. Fisher, B., Brickman, N., Burden, P., Jha, S., Johnson, B., Keller, A., Kolovos, T., Umarji, S., Weeks, S.: Attribute Based Access Control. In: NIST Special Publication 1800-3 (2017)
9. Jin, X., Sandhu, R., Krishnan, R.: RABAC: role-centric attribute-based access control. In: MMM-ACNS, pp. 84–96 (2012)
10. Kuhn, D., Coyne, E., Weil, T.: Adding attributes to role-based access control. IEEE Comput. **43**, 79–81 (2010)
11. Al-Kahtani, M., Sandhu, R.: A model for attribute-based user-role assignment. In: IEEE, pp. 353–362 (2002)
12. Talukdar, T., Batra, G., Vaidya, J., Atluri, V., Sural, S.: Efficient bottom-up mining of attribute based access control policies, pp. 339–348. IEEE (2017)