On the Feasibility of RBAC to ABAC Policy Mining: A Formal Analysis

Shuvra Chakraborty[1,2(✉)], Ravi Sandhu[1,2], and Ram Krishnan[1,3]

[1] Institute for Cyber Security, University of Texas at San Antonio, San Antonio, TX, USA
{shuvra.chakraborty,ravi.sandhu,ram.krishnan}@utsa.edu
[2] Department of Computer Science, University of Texas at San Antonio, San Antonio, TX, USA
[3] Department of Electrical and Computer Engineering, University of Texas at San Antonio, San Antonio, TX, USA

Abstract. Given a Role-Based Access Control (RBAC) system along with supporting attribute data, the process of automated migration to an Attribute-Based Access Control (ABAC) system is a particular instance of the ABAC policy-mining problem. In this paper, we formulate and investigate the feasibility problem of RBAC to ABAC policy mining. Specifically, the ABAC RuleSet Existence problem is introduced formally for the first time in RBAC context. In case of infeasibility, the notion of ABAC RuleSet Infeasibility Correction is formalized and a solution developed utilizing role-based attributes.

Keywords: Attribute-Based Access Control · ABAC policy mining · Feasibility · RBAC · Rule mining

1 Introduction

Following its inception in the mid-nineties, Role-Based Access Control (RBAC) [3,10] has achieved clear dominance over other contemporary access control models and remains prevalent. In recent years, the emerging interest in the Attribute-Based Access Control (ABAC) model as an evolution of RBAC motivates the practical problem of migrating to ABAC from existing access control models.

While there have been differing opinions about which one of RBAC or ABAC is more flexible, scalable, auditable, and provides better support for dynamic environments [12], the benefits of ABAC are increasingly evident. ABAC can be configured to do Discretionary Access Control (DAC), Mandatory Access Control (MAC) and RBAC [7]. It is suitable for large enterprises and notably overcomes some limitations of RBAC such as role explosion [5]. Consequently, ABAC has attracted interest across industry, government applications, and is the fastest-growing access control model today [4]. Therefore, converting an already deployed access control system to an ABAC system is an emerging research problem. Based on this context, ABAC policy mining [11,15] is the process of

S. K. Sahay et al. (Eds.): SKM 2019, CCIS 1186, pp. 147–163, 2020.
https://doi.org/10.1007/978-981-15-3817-9_9

automated migration to an equivalent ABAC policy when an existing access control model along with supporting data is given. Such automation reduces the manual effort needed for migration as well as time and possibilities of error [15].

ABAC policy mining problem was first mentioned by Xu and Stoller [15] where, given user-permission relation, consistent ABAC policy rules with generalized constraints are generated. In [1], the ABAC RuleSet Existence problem was introduced where the feasibility of consistent ABAC policy generation was investigated when an enumerated authorization system along with attribute data is given. In this paper, we study another instance of the general feasibility problem: migration to the ABAC system when an RBAC system and accompanying attribute data are provided. To the best of our knowledge, we have formalized this problem in RBAC context for the first time.

Our major contributions in this paper are as follows.

- We have introduced the idea of partition-based ABAC RuleSet Existence problem in the RBAC context for the first time.
- An approach for determining ABAC RuleSet Existence as well as a correction procedure (in the case of infeasibility) has been presented with examples. Role-based attributes are defined in this context to remove infeasibility.
- Some significant directions for future enhancement are identified.

The rest of the paper is organized as follows. Section 2 defines RBAC and ABAC terminologies to facilitate further discussions. Section 3 introduces ABAC RuleSet Existence and infeasibility correction problems along with associated definitions. In Sect. 4, an infeasibility solution approach utilizing role-based attributes is presented, along with associated proofs. Section 5 presents some future work directions. Finally, Sect. 6 gives a brief discussion of related work.

2 RBAC and ABAC Terminologies

This section represents the RBAC and ABAC related terminologies of the current work. Although the ABAC terminologies were defined in our previous paper [1], these are also included here for completeness. They are identified by explicit reference to corresponding definition in our work [1]. There are some other definitions from our previous work [1], which are included in the subsequent sections; they are customized in RBAC context and marked as adapted.

Every access control system should specify and enforce a function checkAccess (Def. 1, [1]) which is typically a logical formula. This function abstracts the evaluation of an access request from underlying implementation details of the system. Given a complete access control system, a user $u \in U$ is allowed to perform an operation $op \in OP$ on object $o \in O$ iff $checkAccess(u, o, op)$ is True where U, O and OP are the sets of users, objects and operations in the system, respectively.

The key component of RBAC system is role [3], an intermediary between user and permissions in the system. For example, all users assigned to a role "manager" may practice all permissions associated with that role. A complete RBAC system is defined as follows:

Table 1. RBAC system of Example 1

Roles	RPA	RUA	authPerm	authUser
r1	{(o1, op1)}	{u1}	{(o1, op1), (o3, op1)}	{u1}
r2	{(o2, op2)}	{u3}	{(o2, op2)}	{u3}
r3	{(o3, op1)}	{u4, u5}	{(o3, op1)}	{u1, u4, u5}
r4	{(o1, op1), (o3, op1)}	{u2}	{(o1, op1), (o3, op1)}	{u2}

Definition 1. ***RBAC system***
An RBAC system $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ *is a tuple where,*

1. *U, O, and OP are finite sets of users, objects, and operations, respectively.*
2. $P = O \times OP$*, is the set of all possible permissions in the system. A permission* $p \in P$ *is an object-operation pair where ops(p) and obj(p) denote the operation and object associated with p, respectively.*
3. *Roles is a finite set of role names.*
4. *The set of permissions directly assigned to a role* $r \in Roles$ *is given by RPA(r) where,* $RPA\colon Roles \rightarrow 2^P$*. The set of users directly assigned to a role* $r \in Roles$ *is given by RUA(r) where,* $RUA\colon Roles \rightarrow 2^U$*.*
5. *The role hierarchy relation is* $RH \subseteq Roles \times Roles$ *where RH must be acyclic. Here,* $(r, r') \in RH$ *denotes r is a senior role than r'.*
6. *Let reflexive transitive closure of RH be denoted by* RH'*. A role* $r \in Roles$ *acquires the set of permissions associated with all junior roles according to given hierarchy, and denoted by* $authPerm(r) = \{p \in RPA(r') | (r, r') \in RH'\}$*. A role* $r \in Roles$ *inherits all the users associated with seniors roles in hierarchy, and denoted by* $authUser(r) = \{u \in RUA(r') | (r', r) \in RH'\}$*.*
7. $checkAccess_{RBAC}(u\colon U, o\colon O, op\colon OP) \equiv \exists r \in Roles.(u \in authUser(r) \wedge p \in authPerm(r) \wedge (o, op) = (obj(p), ops(p))$*. In simple words, given a role* $r \in Roles$*, a user* $u \in authUser(r)$ *may practice all permissions* $p \in authPerm(r)$*.*

Example 1. The sets of users (U), objects (O), operations (OP) and roles (Roles) are $\{u1, u2, u3, u4, u5\}$, $\{o1, o2, o3\}$, $\{op1, op2\}$, and $\{r1, r2, r3, r4\}$, respectively. Given, $RH = \{(r1, r3)\}$, the user and permission assignment for each $role \in Roles$ is shown in Table 1. Here, user u1 can perform operation op1 on object o3 since $checkAccess_{RBAC}(u1, o3, op1)$ evaluates to True.

The key component of ABAC policy is attribute, which represents characteristics of entities in the system. In ABAC, attribute values of the requesting user as well as the requested object are used to determine whether a particular access request can be granted or denied. To define an ABAC system, ABAC policy is defined first as follows:

Definition 2. ***ABAC policy (Def. 3, [1])***
An ABAC policy, POL_{ABAC} is a tuple $\langle OP, UA, OA, RangeSet, RuleSet\rangle$,

- *OP is a finite set of operations.*
- *UA and OA are finite sets of user and object attribute function names, where for convenience, we assume $UA \cap OA = \emptyset$.*
- *$RangeSet = \{(att, value) | att \in (UA \cup OA) \wedge value \in Range(att)\}$ where, Range(att) specifies a finite set of atomic values.*
- *RuleSet is a set of rules where, for each operation op, RuleSet contains a single rule, $Rule_{op}$. Formally, $RuleSet = \{Rule_{op} | op \in OP\}$.*
- *Each $Rule_{op}$ is specified using the grammar defined below.*
  $Rule_{op} ::= Rule_{op} \vee Rule_{op} \,|\, (Atomicexp)$
  $Atomicexp ::= Atomicuexp \wedge Atomicoexp \,|\, Atomicuexp \,|\, Atomicoexp$
  $Atomicuexp ::= Atomicuexp \wedge Atomicuexp \,|\, uexp$
  $Atomicoexp ::= Atomicoexp \wedge Atomicoexp \,|\, oexp$
  $uexp \in \{ua(u) = value \,|\, ua \in UA \wedge value \in Range(ua)\}$
  $oexp \in \{oa(o) = value \,|\, oa \in OA \wedge value \in Range(oa)\}$
  For a specific operation $op \in OP$, $Rule_{op}$ is specified with user u and object o as formal parameters. The formal semantics of $Rule_{op}$, evaluated for an actual user a and object b is given in Definition 3.

For the ease of further reference, **partially defined ABAC policy** [1] is a tuple, denoted by $POL_{ABAC-RuleSet} \equiv \langle OP, UA, OA, RangeSet\rangle$.

Definition 3. ***ABAC system (Def. 4, [1])***
An ABAC system is a tuple, given by, $\langle U, O, UAValue, OAValue, POL_{ABAC}, checkAccess_{ABAC}\rangle$ where,

- *U and O are finite sets of users and objects, respectively. Here, OP, UA, OA, RangeSet and POL_{ABAC} are defined as in Definition 2.*
- *$UAValue = \{UAValue_{ua} \,|\, ua \in UA\}$ where, $UAValue_{ua} : U \rightarrow Range(ua)$ such that $UAValue_{ua}(u)$ returns the value of attribute ua for user u. For convenience, we understand ua(u) to mean $UAValue_{ua}(u)$.*
- *$OAValue = \{OAValue_{oa} \,|\, oa \in OA\}$ where, $OAValue_{oa} : O \rightarrow Range(oa)$ such that $OAValue_{oa}(o)$ returns the value of attribute oa for object o. For convenience, we understand oa(o) to mean $OAValue_{oa}(o)$.*
- *$checkAccess_{ABAC}$(a:U, b:O, op:OP) $\equiv Rule_{op}$(a:U, b:O) where $Rule_{op}$ is as stated in Definition 2. Given any user $a \in U$ along with attribute value assignments ua(a), where $ua \in UA$ and an object $b \in O$ along with attribute value assignment oa(b), where $oa \in OA$, the expression $Rule_{op}(a, b)$ is evaluated by substituting the values ua(a) for ua(u) and oa(b) for oa(o) in the $Rule_{op}$ expression. User a is permitted to do operation op on object b if and only if $Rule_{op}(a, b)$ evaluates to True.*

A **partially defined ABAC system** [1] is a tuple, $\langle$U, O, UAValue, OAValue, $POL_{ABAC-RuleSet}\rangle$ where U, O, UAValue, OAValue are defined above. It represents an incomplete ABAC system where everything except the ABAC rules is given.

Table 2. ABAC data for Example 2

(a) UAValue

User	uat1
u1	F
u2	F
u3	F
u4	G
u5	G

(b) OAValue

Object	oat1
o1	F
o2	F
o3	G

(c) Range

uat1	$\{F, G\}$
oat1	$\{F, G\}$

Example 2. The set of users (U), objects (O), operations (OP), user attribute names UA and Object attribute names (OA) are $\{u1, u2, u3, u4, u5\}$, $\{o1, o2, o3\}$, $\{op1, op2\}$, $\{uat1\}$, and $\{oat1\}$, respectively. Table 2 shows the user attribute value assignment (UAValue), object attribute value assignment (OAValue), and ranges of the attributes. It can be easily noticed that both Examples 1 and 2 have the same sets of U, O and OP. RuleSet consists of two rules: $Rule_{op1}$ and $Rule_{op2}$, respectively. For instance, if $Rule_{op1} \equiv \langle uat1(u) = G \wedge oat1(o) = G \rangle$, then both u4 and u5 are allowed to perform operation op1 on object o3.

In previous discussions, only atomic-valued attributes [7] are considered while generating ABAC rules. A set-valued attribute [1] is a function which takes an entity (user and object, here) and returns a subset of its range. For instance, given Range(att) $= \{a, b\}$, function att may return only one of $\{\{a, b\}, \{a\}, \{b\}, \{\}\}$. Since a set-valued attribute can be converted to atomic attributes [1], the ABAC rule and evaluation approach discussed earlier in this study are sufficient to manage set-valued attributes by reduction to atomic-valued.

3 Problem Definitions

In this section, ABAC RuleSet Existence and ABAC RuleSet Infeasibility Correction problems are defined in RBAC context. In order to do that, the meaning of equivalency between two access control systems is required. Given two access control systems, stm1 and stm2, with an identical set of users (U), objects (O), and operations (OP), stm1 and stm2 are equivalent iff $\forall (u, o, op) \in U \times O \times OP$. $checkAccess_{stm1}(u, o, op) \iff checkAccess_{stm2}(u, o, op)$. Based on the foregoing, ABAC RuleSet Existence problem [1] is defined in RBAC context as follows:

Definition 4. ***ABAC RuleSet Existence problem (adapted from Def. 6 of [1])***
Given, an RBAC system and a partially defined ABAC system where U, O and OP are identical to the given RBAC system, does there exist a RuleSet so that the resulting ABAC system is equivalent to the given RBAC system? Such a RuleSet, if it exists, is said to be a suitable RuleSet.

To demonstrate the significance of the problem, let's consider the RBAC Example 1 and ABAC Example 2: does there exist a RuleSet so that the resulting ABAC system is equivalent to the given RBAC system? Note that it is always possible to generate equivalent ABAC system when explicit IDs are introduced for both user and object [13]. We strongly believe that the inclusion of such IDs is antithetical to the spirit of ABAC. Hence, we rule out the use of such IDs. For example, in RBAC Example 1, user u1 can perform operation op1 on object o1 whereas user u3, a user with the same attribute value assignment as u1, is not allowed to do so. It is clearly evident that no suitable ABAC RuleSet can exist.

In [1], ABAC RuleSet Existence problem is analyzed when an authorization relation $AUTH \subseteq U \times O \times OP$ and accompanying attribute data are given as input. Given an RBAC system, it is trivial to find an equivalent $AUTH$ relation, such that $(u, o, op) \in AUTH \Leftrightarrow checkAccess_{RBAC}(u, o, op)$. For example, AUTH for the RBAC Example 1 is given as $\{(u1, o1, op1),(u1, o3, op1), (u2, o1, op1), (u2, o3, op1), (u3, o2, op2), (u4, o3, op1), (u5, o3, op1)\}$. Since RBAC system to AUTH conversion takes $O(|U| \times |O|)$ complexity, the partition-based solution from [1] can be reused in RBAC context by simply deriving the equivalent AUTH relation for the given RBAC system. The following relation R generates a partition on the set of all possible user-object pairs:

Definition 5. ***Binary relation R (Def. 7, [1])***
Given a partially defined ABAC system tuple as $\langle U,\ O,\ UAValue,\ OAValue,\ POL_{ABAC-RuleSet}\rangle$*, the binary relation R on set* $UO = U \times O$ *is defined as* $R \equiv \{((u1, o1), (u2, o2)) \mid (\forall ua \in UA.ua(u1) = ua(u2)) \wedge (\forall oa \in OA.oa(o1) = oa(o2))\}$

It is apparent that the binary relation R is an equivalence relation and thereby induces a partition on UO. Let P be the partition on UO induced by R and denoted by, $P = \{P_1, P_2, ..., P_n\}$, where $1 \leq n \leq |UO|$. For convenience, each $P_i \in P$ is called as partition element (or shortly partition) and P is called partition set (Def. 8, [1]) for the rest of the paper. By definition of R, each $P_i \in P$ is identified by a unique collection of (attribute name, value) pairs, given by $PV(P_i)$ where,

$PV(P_i) \equiv (UV(u1) \cup OV(o1))$ for any $(u1, o1) \in P_i$, where
UV(u:U) $\equiv \{(ua, value) | ua \in UA \wedge value = ua(u)\}$
OV(o:O) $\equiv \{(oa, value) | oa \in OA \wedge value = oa(o)\}$
The idea of conflict-free partition is defined in RBAC context as follows:

Definition 6. ***Conflict-free partition (adapted from Def. 9 of [1])***
Given $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ *as an RBAC system and partition set P where U, O and OP are identical, a* $P_i \in P$ *is conflict-free with respect to a specific* $op \in OP$ *iff the following statement is true:*
$\forall (u, o) \in P_i.checkAccess_{RBAC}(u, o, op) = True \vee \forall (u, o) \in P_i.checkAccess_{RBAC}(u, o, op) = False$
P_i has conflict with respect to $op \in OP$ *otherwise. Partition set P is conflict-free with respect to given RBAC system iff for each* $op \in OP$*, every* $P_i \in P$ *is conflict-free. P is called a conflict partition set, otherwise.*

It is shown in [1] that given an AUTH relation and partially defined ABAC system, a suitable RuleSet exists iff partition set P is conflict-free. The overall asymptotic complexity of ABAC RuleSet Existence problem [1] is $O(|OP| \times (|U| \times |O|))$. The construction of AUTH relation by enumerating every possible user-object-operation tuple from an RBAC system takes $O(|U| \times |O|)$ time, thus overall asymptotic complexity of determining ABAC RuleSet Existence in RBAC context remains the same as [1], $O(|OP| \times (|U| \times |O|))$. By definition, suitable RuleSet (Theorem 1, [1]) consists of $|OP|$ rules, one for each $op \in OP$. Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as a conjunctive clause where, $P_i \times \{op\} \subseteq AUTH$. For a specific $op \in OP$, $Rule_{op}$ (Theorem 1, [1]) construction steps are shown below:

$$Rule_{op} = \bigvee_{P_i \times \{op\} \subseteq AUTH} (uexp(PV(P_i)) \wedge oexp(PV(P_i)))$$

$$uexp(PV(P_i)) = \bigwedge_{(ua,value) \in PV(P_i)} (ua(u) = value)$$

$$oexp(PV(P_i)) = \bigwedge_{(oa,value) \in PV(P_i)} (oa(o) = value)$$

Example 3. Using the RBAC tuple in Example 1 and accompanying attribute data in Example 2, partition set P is { {(u1, o1), (u1, o2), (u2, o1), (u2, o2), (u3, o1), (u3, o2)}, {(u4, o1), (u4, o2), (u5, o1), (u5, o2)}, {(u1, o3), (u2, o3), (u3, o3)},{(u4, o3), (u5, o3)} } and PV values are { {(uat1,F), (oat1,F)}, {(uat1,G), (oat1,F)}, {(uat1,F), (oat1,G)}, {(uat1,G), (oat1,G)} }, respectively. An equivalent AUTH is given by $\{(u1, o1, op1), (u1, o3, op1), (u2, o1, op1), (u2, o3, op1), (u3, o2, op2), (u4, o3, op1), (u5, o3, op1)\}$. It is apparent that partition set P is conflicted in this example as shown in Fig. 1. Here, bold user-object pair w.r.t. an operation in a partition represents those user-object-operation tuple belong to AUTH while others do not.

Example 4. In order to show a conflict-free partition set, let's consider the data in Table 3 along with same U, O, OP, Roles and RH in RBAC system of Example 1. Here, the same attribute data as in Example 2 are used. Hence, generated partition set P and PV values are same as Example 3. In this example, an equivalent AUTH $= \{(u1, o1, op1), (u1, o2, op1), (u1, o3, op1), (u2, o1, op1), (u2, o2, op1), (u2, o3, op1), (u3, o1, op1), (u3, o2, op1), (u3, o3, op1), (u4, o3, op2), (u5, o3, op2)\}$. By definition of conflict-free partition set, P is conflict-free in this case and generated RuleSet is $\{Rule_{op1}, Rule_{op2}\}$. Here, $Rule_{op1} = \langle (uat1(u) = F \wedge oat1(o) = F) \vee (uat1(u) = F \wedge oat1(o) = G) \rangle$ and $Rule_{op2} = \langle (uat1(u) = G \wedge oat1(o) = G) \rangle$.

If partition set P is not conflict-free, no suitable RuleSet exists [1]. Hence, in order to make the equivalent ABAC system generation always possible, one possible approach is to ensure that P is always conflict-free. There can be many possible ways to achieve this, either exact or approximate. In this study, ABAC RuleSet Infeasibility Correction problem in RBAC context is defined as follows.

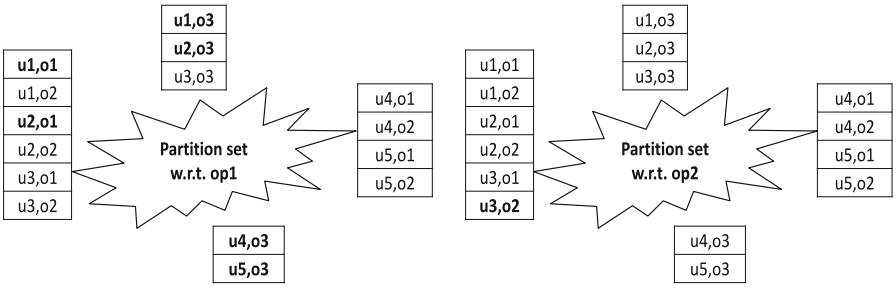

Fig. 1. Conflict partition set for Example 3

Table 3. RBAC system of Example 4

Roles	RPA	RUA	authPerm	authUser
r1	{(o1, op1)}	{u1, u2, u3}	{(o1, op1), (o2, op1)}	{u1, u2, u3}
r2	{(o3, op2)}	{u4, u5}	{(o3, op2)}	{u4, u5}
r3	{(o2, op1)}	{}	{(o2, op1)}	{u1, u2, u3}
r4	{(o3, op1)}	{u1, u2, u3}	{(o3, op1)}	{u1, u2, u3}

Definition 7. ***ABAC RuleSet Infeasibility Correction problem (adapted from Def. 10 of [1])***
Given, RBAC system and partially defined ABAC system with unspecified RuleSet where U, O, and OP are identical to the given RBAC system, and a conflicted partition set P, ABAC Ruleset Infeasibility Correction problem is adding new attributes to (1) only UA or only OA or, both UA, OA, and (2) assign appropriate values to the new attributes, so that suitable RuleSet generation is always possible.

In the next section, an exact solution algorithm is presented for ABAC RuleSet Infeasibility Correction problem with the help of role-based attributes.

4 ABAC RuleSet Infeasibility Correction Solution

It is already established that if partition set P is conflict-free an equivalent ABAC system generation is always possible, since each $P_i \in P$ is uniquely identified by attribute values. Given a conflict partition set P, new role-based attributes are added and values are assigned accordingly so that each conflict partition in P is split into conflict-free fragments uniquely identified by attribute values. Thereby, equivalent RuleSet can be generated. Here, each conflict partition is processed separately to prevent unnecessary split of conflict-free partitions.

According to the construction in [7], an RBAC system can be configured to equivalent ABAC system even if no user, subject and object attributes are provided. The role membership information of an RBAC system can be utilized

Table 4. Role-based attribute values for RBAC system in Example 1

Objects	$oroleAtt_{op1}$	$oroleAtt_{op2}$
o1	{r1, r4}	{}
o2	{}	{r2}
o3	{r1, r3, r4}	{}

Users	*uroleAtt*
u1	{r1, r3}
u2	{r4}
u3	{r2}
u4	{r3}
u5	{r3}

to generate appropriate attribute sets and value assignments. We adapt the construction in [7] to our user-object context as set-valued role membership attributes and omit the subject notion of [7].

Definition 8. ***Role-based user attribute***
Given $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ *as RBAC system tuple, role-based user attribute is a set-valued attribute,* $uroleAtt: U \rightarrow 2^{Roles}$. *For a user* $u \in U$, $uroleAtt(u) = \{r \in Roles | u \in authUser(r)\}$.

Definition 9. ***Role-based object attribute***
Given $\langle U, O, OP, Roles, RPA, RUA, RH, checkAccess_{RBAC}\rangle$ *as RBAC system tuple, role-based object attribute for a* $op \in OP$ *is a set-valued attribute, denoted by* $oroleAtt_{op}: O \rightarrow 2^{Roles}$. *For an object* $o \in O$, $oroleAtt_{op}(o) = \{r \in Roles | p \in authPerm(r) \wedge (o, op) = (obj(p), ops(p))\}$.

Although *uroleAtt* is set-valued by definition, it is treated specially in this study: same as an atomic attribute. In order to generate uexp, "value" is as given in the Definition 8 and to evaluate "uroleAtt(u) = value" in rule expression, "=" is considered as set equality operator. Similarly, each role-based object attribute w.r.t. a $op \in OP$ is treated specially as an atomic attribute. In order to generate oexp, "value" is as given in the Definition 9 and to evaluate "$oroleAtt_{op}$(o) = value" in rule expression, "=" is considered as set equality operator.

Lemma 1. *Given an RBAC system, one user attribute as in Definition 8 and* $|OP|$ *object attributes as in Definition 9 (for each* $op \in OP$*) are sufficient to generate equivalent ABAC system.*

Proof:
Follows from the RBAC to ABAC configuration in [7]. Let the set of user attributes, UA = *uroleAtt* and set of object attributes, OA = $\{oroleAtt_{op} | op \in OP\}$. The attribute value assignments of user and object attributes are as in Definitions 8 and 9, respectively. To generate an equivalent ABAC system, each $P_i \in P$ must be identified by unique PV values as well as partition set P should be conflict-free [1]. It is trivial to show that both conditions are true, thereby, equivalent ABAC system generation is always possible.

Table 5. Partition set in Example 5

Partition set
{(u1,o1)}
{(u1,o2)}
{(u1,o3)}
{(u2,o1)}
{(u2,o2)}
{(u2,o3)}
{(u3,o1)}
{(u3,o2)}
{(u3,o3)}
{(u4,o1), (u5,o1)}
{(u4,o2), (u5,o2)}
{(u4,o3), (u5,o3)}

Example 5. According to Lemma 1, the set of attributes and corresponding value assignment of RBAC system in Example 1 are shown in Table 4. Here, partition set P is shown in Table 5. For instance, PV($\{u1, o1\}$) is given by $\{(uat1, F), (oat1, F), (uroleAtt, \{r1, r3\}), (oroleAtt_{op1}, \{r1, r4\}), (oroleAtt_{op2}, \{\})\}$. In this example, partition set P is conflict-free and for each $P_i \in P$, PV(P_i) is unique.

This unique property of role membership in RBAC system makes it independent of supporting attribute data. It is a significant difference as compared to given authorization relation in [1] where, a user and an object attributes are added to the attribute sets and unique random values are assigned to resolve infeasibility issue. The unique random value generation can be considered as an additional task whereas role membership attributes eliminate the need for such values and promotes self-sufficiency. Although Lemma 1 specifies the sufficiency of the role-based attributes to make an equivalent ABAC system generation, a more practical scenario is where supporting attribute data are provided. Therefore, the following definitions and proofs are presented to resolve ABAC Infeasibility Correction problem when supporting attribute data are provided; so that the resulting partition set becomes conflict-free where each partition element is uniquely identified by attribute values.

Definition 10. ***Binary relation*** $\mathbf{R_{Pi}}$ ***on*** $\mathbf{P_i} \in \mathbf{P}$ *([1])*
$R_{Pi} \equiv \{((u1, o1), (u2, o2))|\forall o \in O.\forall op \in OP.((u1, o, op) \in AUTH \Leftrightarrow (u2, o, op) \in AUTH) \wedge \forall u \in U.\forall op \in OP.((u, o1, op) \in AUTH \Leftrightarrow (u, o2, op) \in AUTH)\}$

By inspection, R_{Pi} is an equivalence relation (Lemma 2, [1]). Let, R_{Pi} induces a partition on P_i, say $S_i = \{S_{i1}, S_{i2}, ..., S_{im}\}$, where $1 \leq m \leq |P_i|$. Each $S_{ik} \in S_i$ is called a partition element (or shortly partition) and S_i is called partition

set. By definition, S_i further refines the partition P_i. Given a partition $P_i \in P$, let $uList_i$ and $oList_i$ denote the sets of users and objects present in P_i. By inspection of definition of R, $P_i = uList_i \times oList_i$. Let $uList_i$ be further partitioned as follows: any two users $u1, u2 \in ulist_i$ belong to same partition iff $\forall op \in OP. \forall o \in O.(u1, o, op) \in AUTH \iff (u2, o, op) \in AUTH$. Let this assumption split $uList_i$ into q partitions, denoted by $\{ul_{i1}, ..., ul_{iq}\}$. Similarly let $oList_i$ be partitioned as follows: any two objects $o1, o2 \in olist_i$ belong to same partition iff $\forall op \in OP. \forall u \in U.(u, o1, op) \in AUTH \iff (u, o2, op) \in AUTH$. Let this assumption split $oList_i$ into r partitions, denoted by $\{ol_{i1}, ..., ol_{ir}\}$.

Lemma 2. *$S_i = \{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$ and it is conflict-free.*

Proof: Trivial [1].

Given a conflict partition $P_i \in P$, S_i has to be conflict-free and each $S_{ik} \in S_i$ should be identified uniquely by attribute values. The given set of attributes are not sufficient to serve this purpose unless there is some change in given attribute value assignments. The following definition adds the already defined role-based attributes to the given attribute set:

Definition 11. ***Add new role-based user and object attributes***
Given ABAC RuleSet Infeasibility Correction instance, the following steps are proposed.

1. *$UA_{new} = UA \cup uroleAtt$ and $OA_{new} = OA \cup \{oroleAtt_{op} | op \in OP\}$. Hence, total $1 + |OP|$ attributes are added.*
   Note: Initially, all new attributes are assigned UND which specifies "Unknown" attribute value assignment.
2. *To ensure clarity, $PV_{new}(S_{ik} \in S_i)$ is introduced.*
   $PV_{new}(S_{ik}) \equiv (UV_{new}(u1) \cup OV_{new}(o1))$ for any $(u1, o1) \in S_{ik}$ where
   $UV_{new}(u{:}U) \equiv \{(ua, value) | ua \in UA_{new} \wedge value = ua(u)\}$
   $OV_{new}(o{:}O) \equiv \{(oa, value) | oa \in OA_{new} \wedge value = oa(o)\}$

Lemma 3. *Given a conflict partition $P_i \in P$ w.r.t. a $op \in OP$, $PV_{new}(S_{ik})$ is unique.*

Proof:
By inspection of definition of R, for each $P_i \in P$, $PV(P_i)$ is unique. By definition, S_i further refines the partition P_i. Hence, if it is proved that, given a conflict partition P_i w.r.t. a $op \in OP$, new user attribute can uniquely identify each element of $\{ul_{i1}, ..., ul_{iq}\}$ and similarly, $|OP|$ object attributes can do the same for $\{ol_{i1}, ..., ol_{ir}\}$, then $PV_{new}(S_{ik})$ is unique.

If $u1 \in ul_{im}$ and $u2 \in ul_{in}$ where $m \neq n$, let $uroleAtt(u1) = uroleAtt(u2)$. If $uroleAtt(u1) = uroleAtt(u2)$ then u1 and u2 cannot belong to two different partitions of $uList_i$ since it ensures uroleAtt(u1) and uroleAtt(u2) derives the exactly same set of permissions. Hence, $uroleAtt(u1) \neq uroleAtt(u2)$ proves. Thereby, each element of $\{ul_{i1}, ..., ul_{iq}\}$ is uniquely identified by uroleAtt value. However, given u3, u4$\in ul_{im}$, it is possible that $uroleAtt(u3) \neq uroleAtt(u4)$,

Algorithm 1. confRefine

Require: Conflict partition P_i and corresponding ABAC Ruleset Infeasibility Correction instance

Ensure: Refined partition set S_i where each $PV_{new}(S_{ik} \in S_i)$ is unique

uL := $\{ul_{i1}, ..., ul_{iq}\}$
oL := $\{ol_{i1}, ..., ol_{ir}\}$
//If checked for Lemma 4
if $\exists u \in uList_i.uroleAtt(u) = UND$ **then**
  while $\exists partu \in uL$ **do**
    For all $u1 \in partu$, $uroleAtt(u1) := uroleAtt(u2)$ //where $u2 \in partu$ and $\forall u3 \in partu.\ |uroleAtt(u2)| \leq |uroleAtt(u3)|$
    $uL := uL \setminus partu$
//If checked for Lemma 4
if $\exists (o, op) \in oList_i \times OP.oroleAtt_{op}(o) = UND$ **then**
  while $\exists parto \in oL$ **do**
    for each $op \in OP$ **do**
      For all $o1 \in parto$, $oroleAtt_{op}(o1) := oroleAtt_{op}(o2)$ //where $o2 \in parto$ and $\forall o3 \in parto.|oroleAtt_{op}(o2)| \leq |oroleAtt_{op}(o3)|$
    $oL := oL \setminus parto$
return S_i //$\{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$

although the resulting permissions are the same. By inspection, Algorithm 1 picks the minimum cardinality role set as role-based attribute value for every user in ul_{im}. Similarly, it can be proved that, If $o1 \in ol_{im}$ and $o2 \in ol_{in}$ where $m \neq n$, $\exists op \in Op.oroleAtt_{op}(o1) \neq oroleAtt_{op}(o2)$. Thereby, $PV_{new}(S_{ik})$ is unique.

Lemma 4. *Given $P_i = uList_i \times oList_i$ and $P_j = uList_j \times oList_j$, if $u1 \in uList_i$ and $u1 \in uList_j$, then $uList_i = uList_j$.*

Proof:
Follows from definition of R, it is trivial. Similarly, it can be proved that, if $o1 \in oList_i$ and $o1 \in oList_j$, then $oList_i = oList_j$.
Note: In Algorithm 1, Lemma 4 is used to prevent repeated role-based attribute value assignment of users and objects. Based on the foregoing, the following theorem states and proves the solution of ABAC RuleSet Infeasibility Correction problem.

Theorem 1. *Given an ABAC RuleSet Infeasibility Correction problem instance as in Def. 7, it is always possible to find a suitable RuleSet such that the resulting ABAC system is equivalent to given RBAC system (adapted from Theorem 2 of [1]).*

Proof:
Given an RBAC system, equivalent AUTH relation is generated first. Given a $op \in OP$, the $Rule_{op}$ construction procedure is described below. Here, partition set P construction entirely depend on the given attribute set only (no role-based attributes).

1. Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as conjunctive clause where, $P_i \times \{op\} \subseteq AUTH$. For a $op \in OP$, such $Rule_{op}$ is defined in Sect. 3.
2. After applying Definition 11, each conflict partition $P_i \in P$ is further refined by Algorithm 1. By using Lemma 3, $\forall S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique where each $S_{ik} \in S_i$ is conflict-free. A conjunctive clause is included in $Rule_{op}$ only if $S_{ik} \times \{op\} \subseteq AUTH$ where $S_{ik} \in S_i$. The following shows $Rule_{op}$ construction procedure for conflict partitions in P only:

$$Rule_{op} = \bigvee_{P_i \in CFP(P)} (uexp(PV_{new}(S_{ik})) \wedge oexp(PV_{new}(S_{ik})))$$

where CFP(P) consists of all conflict partitions in P with respect to $op \in OP$, $S_{ik} \in confRefine(P_i)$, and $S_{ik} \times \{op\} \subseteq AUTH$.

$$uexp(PV_{new}(S_{ik})) = \bigwedge_{(ua,value) \in PV_{new}(S_{ik})} (ua(u) = value)$$

$$oexp(PV_{new}(S_{ik})) = \bigwedge_{(oa,value) \in PV_{new}(S_{ik})} (oa(o) = value)$$

Here, $Rule_{op}$ is the disjunction of all the conjunctive clauses generated in step 1 and 2. By definition, RuleSet consists of total $|OP|$ rules, one for each $op \in OP$. Hence, a RuleSet can be constructed. To prove equivalency between the resulting ABAC system with constructed RuleSet and RBAC system, it is necessary and sufficient to show that, for a op in OP, $checkAccess_{RBAC}text(c, d, op) =$ True $\iff$ $Rule_{op}(c, d)$ where $c \in U$, $d \in O$ which implies $(c, d, op) \in AUTH \iff Rule_{op}(c, d)$.

The proof is divided into two parts: (i) only if and (ii) if. To prove (i): by inspection of partition and related definitions, $(c, d) \in U \times O$ belongs to only one partition in P. Let, $(c, d) \in P_i$ where $P_i \in P$. If P_i is conflict-free with respect to op then $\forall (u, o) \in P_i.(u, o, op) \in AUTH$ holds (step 1 in $Rule_{op}$ generation). If P_i is a conflict partition then step 2 is followed. Let, (c,d)$\in S_{ik}$ where $S_{ik} \in S_i$. Hence $\forall (u, o) \in S_{ik}.(u, o, op) \in AUTH$ holds. As a result, S_{ik} is included in $Rule_{op}$ as conjunctive clause (as per step 2 in $Rule_{op}$ construction procedure). Since $Rule_{op}$ consists of disjunction of all the conjunctive clauses generated in step 1 and 2, $Rule_{op}(c, d)$ evaluates to true and (i) is proved.

The part (ii) of the proof: by inspection of $Rule_{op}$ construction stated above, if $Rule_{op}(c, d)$ evaluates to true then there exists a conjunctive clause of $Rule_{op}$ which turned into true. By $Rule_{op}$ construction procedure, each such conjunctive clause in $Rule_{op}$ is presenting a particular partition where for all $(u, o) \in partition.(u, o, op) \in AUTH$. Thereby, the statement $(c, d, op) \in AUTH$ is true and (ii) is proved. Hence, the constructed RuleSet completes the ABAC system, and equivalent to given RBAC system (proved by construction).

Example 6. Given RBAC Example 1 and ABAC Example 2, partition set P is conflicted. The corresponding role attribute values are shown in Table 4. According to Theorem 1, {u1, u2, u3, u4, u5} is split into {{u1, u2}, {u3}, {u4, u5}}. Similarly, {o1,o2, o3} is split into {{o1},{o2},{o3}}. Table 6 shows the role-based attribute values after applying algorithm 1. Figure 2 shows the refined partition set where dotted rectangle represents the initial partition before correction. The resulting $Rule_{op1} = \langle\!\langle(\text{uat1(u)=F} \wedge \text{oat1(o)= F} \wedge uroleAtt(u) = \{r4\} \wedge oroleAtt_{op1}(o) = \{r1, r4\} \wedge oroleAtt_{op2}(o) = \{\}) \vee (\text{uat1(u)=F} \wedge \text{oat1(o)= G} \wedge uroleAtt(u) = \{r4\} \wedge oroleAtt_{op1}(o) = \{r1, r3, r4\} \wedge oroleAtt_{op2}(o) = \{\}) \vee (\text{uat1(u)=G} \wedge \text{oat1(o)= G})\rangle\!\rangle$. Similarly, $Rule_{op2} = \langle(\text{uat1(u)} = \text{F} \wedge \text{oat1(o)= F} \wedge uroleAtt(u) = \{r2\} \wedge oroleAtt_{op1}(o) = \{\} \wedge oroleAtt_{op2}(o) = \{r2\})\rangle$. In Fig. 2, the refined partition set is shown where each bold user-object pair belongs to the AUTH w.r.t. some $op \in OP$ while others are not.

Figure 3 shows the steps of ABAC Infeasibility Correction solution for Example 6. If partition set P is conflicted, one viable approach is to stop right there considering suspicious assignment, denoted by a "cross". Otherwise, the subsequent steps are followed to get an exact solution. One notable optimization at this point is: role-based attributes should be used only when they are needed. For example, if each user in the given user set is represented by a unique user attribute value assignment then there is no need to introduce a role-based user attribute, even if the partition set is conflicted. The same strategy can be applied for role-based object attribute: if each object is represented by unique attribute value assignment, role-based object attributes are unnecessary even if partition set is conflicted. If both of the cases do not hold, still role-based attributes can be removed while generating a conjunctive clause for a particular conflicted partition. For a conflict partition $P_i \in P$ where $P_i = uList_i \times oList_i$, if $|uList_i| = 1$ then role-based user attribute can be avoided while generating conjunctive clauses for P_i. Similarly, role-based object attribute can be ignored when $|oList_i| = 1$. For instance, Example 6 uses conjunctive clause $(\text{uat1(u)} = \text{F} \wedge \text{oat1(o)= G} \wedge uroleAtt(u) = \{r4\} \wedge oroleAtt_{op1}(o) = \{r1, r3, r4\} \wedge oroleAtt_{op2}(o) = \{\})$ while generating rule for partition $\{\{u1, o3\}, \{u2, o3\}\}$. In this case, role-based object attributes can be omitted as unnecessary! Thus, the resulting conjunctive clause should be $(\text{uat1(u)=F} \wedge \text{oat1(o)= G} \wedge uroleAtt(u) = \{r4\})$.

According to the approach presented, the solution is found with 9 partitions only where 15 partitions were possible in the worst case (if user and object IDs are introduced).

One significant point is: the partition-based ABAC rule generation proposed in this study is free of unrepresented partition gproblem in [1]. The asymptotic complexity of ABAC RuleSet Infeasibility Correction in RBAC context is given by $O(|OP| \times (|U| \times |O|)^3)$, same as [1].

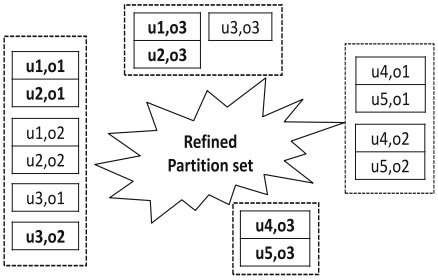

Fig. 2. Refined partition Set in Example 6

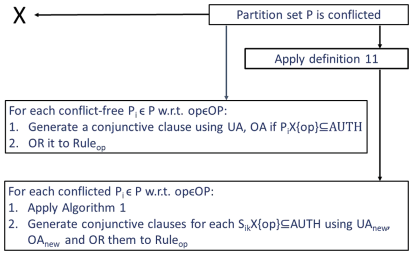

Fig. 3. Steps in Theorem 1

Table 6. Role-based attributes in Table 4 after applying Algorithm 1

Objects	$oroleAtt_{op1}$	$oroleAtt_{op2}$
o1	{r1, r4}	{}
o2	{}	{r2}
o3	{r1, r3, r4}	{}

Users	$uroleAtt$
u1	{r4}
u2	{r4}
u3	{r2}
u4	{r3}
u5	{r3}

5 Future Enhancements

According to the solutions provided in Sect. 4, it is obvious that each conflict partition with respect to an $op \in OP$ must be split (by inspection, at least two parts) unless there is any change in given attribute value assignment.

Now, question is, can we do better? We leave such questions to be addressed in future work. Some specific questions are listed below:

1. Can we find an approach so that it is always possible to split a conflict partition into two, even if more attributes are needed?
2. Can we propose different combinations of refinement approaches apart from current partition refinement so that asymptotic complexity can be improved?
3. Only positive rule generation has been considered here. The impact of using positive and negative rules together may or may not improve the current procedure.
4. By partition definition, each user object pair present in that partition is represented by the same attribute value combinations. What if there is an exception?
5. Although we had the independence of adding any number of extra attributes to resolve the infeasibility, further analysis is required to get the optimal outcome.
6. Algorithm 1 selects the minimal size of role set while choosing values for role-based attributes. It might not provide the optimal outcome; the system administrator and expert inputs could be considered for further optimization.

7. The generated RuleSet is free of unrepresented partition problem in [1]. However, there is scope to optimize the RuleSet.

6 Related Works

The ABAC RuleSet Existence and correction problem [1] are mentioned for the first time by Chakraborty et al., where the primary aim was feasibility analysis of ABAC rule generation. The proposed work [1] generates partition based consistent ABAC rule with respect to given authorizations relation and accompanying attribute data. Apart from that, this work [1] focuses on unrepresented partitions on ABAC policy mining context.

To mention about some other closely related works: ABAC policy mining problem [11,15] forms the background of feasibility analysis proposed in this paper. There have been a good number of works on ABAC policy mining, such as from Authorization [11,15], RBAC [13], log data [9,14], sparse log [2], etc. An evolutionary computation based solution towards ABAC policy mining is described in [8].

All the works noted so far deals with positive ABAC rules only. However, there is a ABAC policy mining approach [6] by Iyer and Masoumzadeh which deals with both positive and negative ABAC rules. In the proposed work [6], an existing rule mining algorithm called PRISM is used to generate consistent ABAC rules with respect to input log: it is assumed that a complete log is given as input or denied otherwise.

Acknowledgement. This work is partially supported by NSF CREST Grant HRD-1736209, CNS-1423481, CNS-1538418 and DoD ARL Grant W911NF-15-1-0518.

References

1. Chakraborty, S., Sandhu, R., Krishnan, R.: On the feasibility of attribute-based access control policy mining. In: IRI. IEEE (2019)
2. Cotrini, C., Weghorn, T., Basin, D.: Mining ABAC rules from sparse logs. In: EuroSP, pp. 31–46. IEEE (2018)
3. Ferraiolo, D., et al.: Proposed NIST standard for role-based access control. ACM TISSEC **4**(3), 224–274 (2001)
4. Hu, V.C., Kuhn, D.R., Ferraiolo, D.F.: Attribute-based access control. IEEE Comput. **2**, 85–88 (2015)
5. Hu, V., et al.: Guide to attribute based access control (ABAC) definition and considerations. NIST Spec. Publ. **800**, 162–800 (2014)
6. Iyer, P., Masoumzadeh, A.: Mining positive and negative attribute-based access control policy rules. In: SACMAT, pp. 161–172 (2018)
7. Jin, X., Krishnan, R., Sandhu, R.: A unified attribute-based access control model covering DAC, MAC and RBAC. DBSec **12**, 41–55 (2012)
8. Medvet, E., et al.: Evolutionary inference of attribute-based access control policies. In: Gaspar-Cunha, A., Henggeler Antunes, C., Coello, C. (eds.) EMO 2015. LNCS, vol. 9018, pp. 351–365. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-319-15934-8_24

9. Mocanu, D., Turkmen, F., Liotta, A.: Towards ABAC policy mining from logs with deep learning. In: IS 2015 (2015)
10. Sandhu, R.S., et al.: Role-based access control models. IEEE Comput. **2**, 38–47 (1996)
11. Talukdar, T., et al.: Efficient bottom-up mining of attribute based access control policies. In: IEEE CIC 2017, pp. 339–348 (2017)
12. Weil, T.R., Coyne, E.: ABAC and RBAC: scalable, flexible, and auditable access management. IT Prof. **15**(03), 14–16 (2013)
13. Xu, Z., Stoller, S.: Mining attribute-based access control policies from RBAC policies. In: CEWIT, pp. 1–6. IEEE (2013)
14. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies from logs. In: Atluri, V., Pernul, G. (eds.) DBSec 2014. LNCS, vol. 8566, pp. 276–291. Springer, Heidelberg (2014). https://doi.org/10.1007/978-3-662-43936-4_18
15. Xu, Z., Stoller, S.: Mining attribute-based access control policies. IEEE TDSC **12**(5), 533–545 (2015)