On the Feasibility of Attribute-Based Access Control Policy Mining

Shuvra Chakraborty[1], Ravi Sandhu[2] and Ram Krishnan[3]
[1,2,3]Institute for Cyber Security
[1,2]Department of Computer Science, [3]Department of Electrical and Computer Engineering
[1]shuvra.chakraborty@utsa.edu, [2]ravi.sandhu@utsa.edu and [3]ram.krishnan@utsa.edu
University of Texas at San Antonio, San Antonio, Texas, USA

Abstract

As the technology of attribute-based access control (ABAC) matures and begins to supplant earlier models such as role-based or discretionary access control, it becomes necessary to convert from already deployed access control systems to ABAC. Several variations of this general problem can be defined, some of which have been studied by researchers. In particular the ABAC policy mining problem assumes that attribute values for various entities such as users and objects in the system are given, in addition to the authorization state, from which the ABAC policy needs to be discovered. In this paper, we formalize the ABAC RuleSet Existence problem in this context and develop an algorithm and complexity analysis for its solution. We further introduce the notion of ABAC RuleSet Infeasibility Correction along with an algorithm for its solution.

1 Introduction

The attribute-based access control (ABAC) policy mining problem [9, 12] requires the discovery of ABAC policy from a given authorization state and given attribute values for users and objects. We use the terms ABAC policy and ABAC rule set interchangeably. Various means of providing authorization input data have been considered, including RBAC [10], log data [11] and sparse log [2]. The previous literature has primarily focused on the detailed mechanics of generating ABAC policy rules from the provided input. Here we develop a more abstract and general characterization of this problem. To the best of our knowledge, this is the first such study of ABAC policy mining.

The fundamental question investigated in this paper concerns the feasibility of finding a suitable ABAC policy, assuming there is no limit to the sophistication of the ABAC policy language. Consider two extreme cases. In the first, all users have identical attribute values and likewise for all objects. Rules based on attribute values thereby cannot distinguish any two user, object pairs and can only give uniform authorization for all such pairs, which is hardly useful in practice. At the other extreme assume user identity and object identity are included as attribute values for users and objects respectively, where identity is globally unique. Every user, object pair is thereby distinguishable from every other pair, so authorization for each pair can be differentiated. In general, the inclusion of identity attributes will guarantee the existence of ABAC policy rules even if all other attributes are ignored. We believe that identity attributes are antithetical to the spirit of ABAC and we disallow them, which makes the feasibility question germane.

Rest of the paper is organized as follows. Section 2 formalizes the ABAC RuleSet Existence problem. Section 3 develops a solution for this problem with associated proofs and analysis. Section 4 discusses the treatment of unrepresented attribute value combinations. Section 5 introduces the ABAC RuleSet Infeasibility Correction problem and provides a solution. A brief discussion of related works is presented in Section 6.

2 ABAC RuleSet Existence Problem

We consider access control systems that mediate access of users to objects. We specifically omit the user-subject distinction, e.g. as in [4]. Given that a user requests to perform an operation on an object, every access control system must define a checkAccess function to decide whether or not this operation should be permitted or denied.

Definition 1 checkAccess
checkAccess: $U \times O \times OP \rightarrow \{True, False\}$ where, U, O, and OP are finite sets of users, objects, and operations, respectively. A user $u \in U$ is allowed to perform operation $op \in OP$ on object $o \in O$ if and only if checkAccess(u, o, op) is True.

In general, the checkAccess function changes with the

system state. Since our focus is on a single state we omit explicit mention of the state. The specification of checkAccess, typically as a logical formula, depends upon the details of the underlying access control model. A simple authorization system, where user-object-operation tuples are used directly to control access is as follows.

Definition 2 Enumerated Authorization System (EAS)
An EAS is a tuple $\langle$U, O, OP, AUTH, checkAccess$_{\text{AUTH}}\rangle$ where, U, O, and OP are finite sets of users, objects and operations, respectively. Here, AUTH $\subseteq$ U $\times$ O $\times$ OP, is a specified authorization relation and checkAccess$_{\text{AUTH}}$(u, o, op) $\equiv$ (u, o, op) $\in$ AUTH.

For instance, user Paul can read F if and only if (Paul, F, read)$\in$AUTH, whereby checkAccess$_{\text{AUTH}}$ is True. We require that the authorization state in the ABAC policy mining problem be given as an EAS. Note that however checkAccess is specified in an access control system, an equivalent AUTH relation can be computed for finite sets of users, objects and operations. So this is a reasonably general assumption.

Our ABAC model is adapted from [4] with two major deviations. Firstly, as mentioned above we omit the user-subject distinction. Secondly, attributes in [4] can be atomic-valued or set-valued. For example, the age attribute of a user is atomic valued. On the other hand a user's department attribute could be atomic valued if only one department is permitted or set-valued if a user can be in multiple departments. Note that set-valued attributes can be replaced by atomic-valued attributes by simply enumerating all combinations and assigning a symbol for each. For simplicity we assume all attributes are atomic valued.

In ABAC, authorization of whether a user can do an operation on an object is decided using the attributes value assignments of both user and object. (Many ABAC systems also include contextual attributes, which we ignore in this paper.) The core of ABAC is a set of rules, which constitute the ABAC policy.

Definition 3 ABAC policy
An ABAC policy, POL$_{\text{ABAC}}$ is a tuple, $\langle$OP, UA, OA, RangeSet, RuleSet$\rangle$ where,

- OP is a finite set of operations, and UA and OA are finite sets of user and object attribute function names respectively. 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$_{\text{op}}$. Formally, RuleSet = $\{$Rule$_{\text{op}}$ | op $\in$ OP$\}$. Each Rule$_{\text{op}}$ is specified using the grammar below.

  Rule$_{\text{op}}$ ::= Rule$_{\text{op}}\vee$ Rule$_{\text{op}}$ | (Atomicexp)
  Atomicexp::=Atomicuexp$\wedge$Atomicoexp|Atomicuexp|
  Atomicoexp | True | False
  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)$\}$

Each Rule$_{\text{op}}$ is specified with user u and object o as formal parameters. The semantics of Rule$_{\text{op}}$, evaluated for an actual user a and object b is given in Definition 4. For example, ABAC rule for read operation is denoted by, Rule$_{\text{read}}$ and given by (rank(u) = manager $\wedge$ type(o) = attendance log) $\vee$ (rank(u) = manager $\wedge$ type(o) = Annual report). Here any user in U with rank = manager can read both types of objects, attendance log and annual report.

A complete ABAC system defines authorization based on ABAC policy as follows.

Definition 4 ABAC system
An ABAC system is a tuple, given by, $\langle$U, O, UAValue, OAValue, POL$_{\text{ABAC}}$, checkAccess$_{\text{ABAC}}\rangle$ where,

- U and O are finite sets of users and objects, respectively. OP, UA, OA, RangeSet and POL$_{\text{ABAC}}$ are defined as in Definition 3.
- UAValue = $\{$UAValue$_{\text{ua}}$ | ua $\in$ UA$\}$ where, UAValue$_{\text{ua}}$: U $\rightarrow$ Range(ua) such that UAValue$_{\text{ua}}$(u) returns the value of attribute ua for user u. For convenience, we understand ua(u) to mean UAValue$_{\text{ua}}$(u).
- OAValue = $\{$OAValue$_{\text{oa}}$ | oa $\in$ OA$\}$ where, OAValue$_{\text{oa}}$: O $\rightarrow$ Range(oa) such that OAValue$_{\text{oa}}$(o) returns the value of attribute oa for object o. For convenience, we understand oa(o) to mean OAValue$_{\text{oa}}$(o).
- checkAccess$_{\text{ABAC}}$(a:U, b:O, op:OP) $\equiv$ Rule$_{\text{op}}$(a:U, b:O) where Rule$_{\text{op}}$ is as stated in Definition 3. 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$_{\text{op}}$(a, b) is evaluated by substituting the values ua(a) for ua(u) and oa(b) for oa(o) in the Rule$_{\text{op}}$ expression. User a is permitted to do operation op on object b if and only if Rule$_{\text{op}}$(a, b) evaluates to True.

We also define a **partially defined ABAC system** to be a tuple, $\langle$U, O, UAValue, OAValue, POL$_{\text{ABAC–RuleSet}}\rangle$ where POL$_{\text{ABAC–RuleSet}}$ is a tuple $\langle$OP, UA, OA, RangeSet$\rangle$ where OP, UA, OA, and RangeSet are as in Definition 3 and RuleSet is undefined.

Table 1. Example data set 1

(a) UAValue		
User	ua1	ua2
u1	F	C
u2	F	B
u3	F	C
u4	G	D

(b) OAValue	
Object	oa1
o1	F
o2	G

(c) Range	
ua1	{F, G}
ua2	{B, C, D}
oa1	{F, G}

Definition 5 Equivalency
An EAS, $\langle$U, O, OP, AUTH, checkAccess$_{\text{AUTH}}\rangle$, and an ABAC system, $\langle$U, O, UAValue, OAValue, POL$_{\text{ABAC}}$, checkAccess$_{\text{ABAC}}\rangle$ with identical U, O, and OP are said to be equivalent iff, checkAccess$_{\text{AUTH}}$(u, o, op) $\iff$ checkAccess$_{\text{ABAC}}$(u, o, op) for all u $\in$ U, o $\in$ O, and op $\in$ OP.

Based on the foregoing, ABAC RuleSet Existence problem is defined as follows.

Definition 6 ABAC RuleSet Existence problem
Given, an EAS $\langle$U, O, OP, AUTH, checkAccess$_{\text{AUTH}}\rangle$ and a partially defined ABAC system $\langle$U, O, UAValue, OAValue, POL$_{\text{ABAC–RuleSet}}\rangle$ where U, O and OP are identical to the given EAS, does there exist a RuleSet so that the resulting ABAC system is equivalent to the given EAS? Such a RuleSet, if it exists, is said to be a suitable RuleSet.

To illustrate the ABAC RuleSet Existence problem consider the example data in Table 1, where U = {u1, u2, u3, u4}, O = {o1, o2}, UA = {ua1, ua2} and OA = {oa1}. Table 1(c) specifies the attribute ranges while Tables 1(a) and (b) gives attribute values for users and objects respectively. All of this specifies a partially defined ABAC system. Suppose we are given an EAS with AUTH={(u1, o1, op)}. A suitable RuleSet cannot exist since users u1 and u3 cannot be distinguished based on their attribute values. However, if AUTH={(u1, o1, op),(u3,o1,op)} then a suitable Rule$_{\text{op}}$ is ua1(u)=F $\wedge$ ua2(u)=C $\wedge$ oa1(o)=F.

3 ABAC RuleSet Existence Solution

The essential concept to solve the ABAC RuleSet Existence problem is that the attribute name, value combinations induce a partition on the set of user, object pairs. We formalize this intuition as follows.

Definition 7 Binary relation R
Given, a partially defined ABAC system, $\langle$U, O, UAValue, OAValue, POL$_{\text{ABAC–RuleSet}}\rangle$, the binary relation R on set UO = U $\times$ O is defined as
R $\equiv$ $\{((u1, o1), (u2, o2)) \,|\, (\forall ua \in UA.ua(u1) = ua(u2)) \wedge (\forall oa \in OA.oa(o1) = oa(o2))\}$

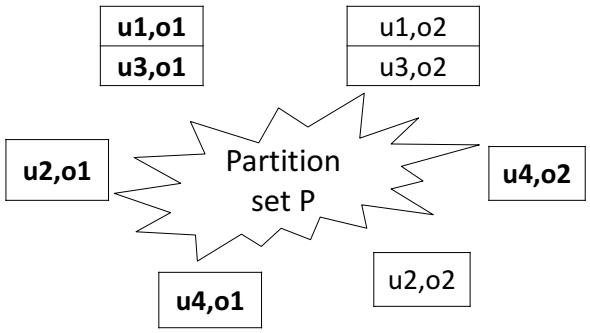

Figure 1. Partition set example.

Lemma 1 R is an equivalence relation.
Proof: Trivial by inspection.

The resulting partitions induced by R on UO are formally referred to as follows.

Definition 8 Partition set P
Let P = $\{P_1, P_2, ..., P_n\}$ be the equivalence classes of R. Each $P_i \in P$ is called a partition element (or simply partition) and P is called the partition set. 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) $\equiv$ For any (u1, o1) $\in P_i$, (UV(u1) $\cup$ OV(o1))
UV(u:U) $\equiv$ $\{(ua, value) | ua \in UA \wedge value = ua(u)\}$
OV(o:O) $\equiv$ $\{(oa, value) | oa \in OA \wedge value = oa(o)\}$

For instance, using the example data in Table 1, UO = {(u1, o1), (u1, o2), (u2, o1), (u2, o2), (u3, o1), (u3, o2), (u4, o1), (u4, o2)}. The resulting partition set is shown in Fig. 1. The PV set for the partition containing (u1, o1) and (u3, o1) is $\{(ua1, F), (ua2, C), (oa1, F)\}$.

Finally, we introduce the following notion.

Definition 9 Conflict-free partition
Given an EAS, $\langle U, O, OP, AUTH, checkAccess_{AUTH}\rangle$ and partition set P, a $P_i \in P$ is conflict-free with respect to a specific op $\in$ OP iff the following statement is true,
$(\forall (u, o) \in P_i.(u, o, op) \in AUTH) \vee (\forall (u, o) \in P_i.(u, o, op) \notin AUTH)$
P_i has conflict with respect to op $\in$ OP otherwise. Partition set P is conflict-free iff for all op $\in$ OP, every $P_i \in P$ is conflict-free with respect to op. P is called a conflicted partition set, otherwise.

An example of conflict-free partition set is presented in Fig. 1 where, user-object pairs in bold black belong to AUTH with respect to OP = {op}, while others are not. By inspection, all partitions in Fig. 1 are conflict-free with respect to given AUTH and op $\in$ OP. Hence, resulting partition set is conflict-free. The concept of conflict-free partitions is used to solve ABAC RuleSet Existence problem as follows.

Theorem 1 Given an ABAC RuleSet Existence problem instance, a suitable RuleSet exists iff P is conflict-free.

Proof:

Only if part is proved by contraposition. If P is not conflict-free, by definition, a conflict partition $P_i \in P$ with respect to a specific op $\in$ OP, contains some $(u, o) \in P_i$ which are permitted in AUTH, while others are not. Since all $(u, o) \in P_i$ are represented by same $PV(P_i)$, these two logical parts of P_i cannot be separated using UAValue and OAValue. Thereby, $Rule_{op}$ generation is not possible. Note that this only if proof is independent of the actual policy language for $Rule_{op}$.

If part is proved by constructing a suitable RuleSet and showing that, if P is conflict-free then resulting ABAC system with generated RuleSet is equivalent to EAS. By definition, RuleSet contains a $Rule_{op}$, for each op $\in$ OP. For a op $\in$ OP, $Rule_{op}$ is given by:

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

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

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

To prove equivalency between the resulting ABAC system with RuleSet and EAS, it is necessary and sufficient to show that, for a specific op in OP, $(a, b, op) \in AUTH \iff Rule_{op}(a, b)$, for all $a \in U$, $b \in O$.

To prove the only if part: by inspection, each $(u, o) \in U \times O$ belongs to only one partition in P. Let $(a, b) \in P_i$. Since P is conflict-free, corresponding $P_i \times \{op\}$ must be a subset of AUTH. By construction, $Rule_{op}$ includes a conjunctive clause for every $P_i \times \{op\} \subseteq$ AUTH, which evaluates to True for any user-object pair in P_i. Since $Rule_{op}$ is a disjunction of such conjunctive clauses, thereby, $Rule_{op}(a, b)$ evaluates to True. Hence, only if part is proved.

To prove if part: by inspection, if $Rule_{op}(a, b)$ evaluates to True, then there must be a conjunctive clause of $Rule_{op}$, which is evaluated to True. By construction, each such conjunctive clause in $Rule_{op}$ is representing a specific $P_i \in P$ where, $P_i \times \{op\} \subseteq$ AUTH. Since P is conflict-free, every user-object pair in corresponding P_i is permitted with respect to op, thus belongs to AUTH. Thereby, $(a, b, op) \in$ AUTH, which proves if part.

Hence, given P is conflict-free, generated suitable RuleSet completes the ABAC system equivalent to given EAS.

Based on this result, a formal algorithm for ABAC RuleSet existence problem is presented in Algorithm 1.

Corollary 1 Complexity of Algorithm 1 is $O(|OP| \times |U| \times |O|)$.

Algorithm 1 ABAC RuleSet Existence Algorithm

Require: EAS, Partially defined ABAC system where U, O, and OP are identical to EAS

Ensure: Partition set P and SUCCESS or FAILURE

```
1: Partition P := ∅
2: while ∃(u, o) ∈ U × O do
3:    if ∃P_i ∈ P.PV(P_i) = (UV(u) ∪ OV(o)) then
4:       P_i := P_i ∪ {⟨u, o⟩}
5:    else
6:       Create new P_i = {⟨u, o⟩}
7:       P := P ∪ P_i
8:    U × O := U × O – {⟨u, o⟩}
9: while ∃op ∈ OP.∃P_i ∈ P.!((P_i × {op} ⊆ AUTH) ∨
   (P_i × {op} ⊆ ¬AUTH)) do
10:   return FAILURE
11: return SUCCESS
```

Proof:

Given an equivalence relation R as in definition 7, the complexity of partition set P generation is $O(|U| \times |O|)$, considering partition creation, search, and insertion operations take constant time. For each op $\in$ OP, checking whether P is conflict-free or not, requires $O(|OP| \times |U| \times |O|)$ as the maximum number of possible partition is $(|U| \times |O|)$. Hence, the overall complexity is $O(|OP| \times |U| \times |O|)$. Note that the size of any attribute range does not impact this complexity.

Simple rule generation will be illustrated with an example presented in Fig. 1. Since P is conflict-free with respect to given AUTH, using the rule construction procedure listed in Theorem 1, corresponding conjunctive clauses for each $P_i \in P$ where $P_i \times \{op\} \subseteq$ AUTH in Fig. 1 are $<$ ua1(u) = F$\wedge$ua2(u) = C$\wedge$oa1(o) = F $>$, $<$ ua1(u) = F$\wedge$ua2(u) = B$\wedge$ oa1(o) = F $>$, $<$ ua1(u) = G $\wedge$ ua2(u) = D $\wedge$ oa1(o) = F $>$, and $<$ ua1(u) = G $\wedge$ ua2(u) = D $\wedge$ oa1(o) = G $>$. By construction, $Rule_{op}$ consists of disjunction of all the conjunctive clauses listed here. Now, any rule simplification approach can be used for further minimization.

4 Unrepresented partitions

Given range of attributes and a specific set of attribute value assignment to users and objects, it is quite possible that some attribute value combinations will not show up while generating partition set. We call these partitions as "unrepresented", since the range of attributes clearly allows presence of those, but due to the peculiarity in the given user and object attribute value assignment, these partitions remain empty. For instance, all possible attribute value combinations (each one represents a possible partition) for Table 1 data is given by {FCF, FBF, GDF, GDG, FCG, FBG,

FDF, FDG, GBF, GCF, GBG, GCG}, considering an order of $< ua1, ua2, oa1 >$. Only first six combinations of this set are present, while the remaining six are unrepresented according to the given data. The ABAC policy mining approaches in [9, 12] ignore unrepresented partitions, whereby the generated rules may or may not authorize these attribute value combinations to have access. If these unrepresented partitions get populated in future, this may lead to unexpected checkAccess decisions.

To have an insight using the same data set for ABAC policy mining where authorizations are presented in Fig. 1, [12] derives two rules without user and object id, <true, true, {op}, {ua1=oa1}> and <ua1={G}, true, {op}, ∅>. Since [12] works fine for all possible user-object pair, first four of the attribute value combinations are allowed, while FCG and FBG are denied. Amongst unrepresented attribute value combinations, {GCG, GBG, FDF} satisfies the first clause of the rule, and {GBF, GCF, GBG, GCG} are accepted by the second clause. Only {FDG} gets rejection according to the generated rules! The question of unrepresented attribute combinations is treated as a rule simplification concern, rather than a security concern in [12]. Another approach in [9] generates two rules, < (oa1 = F) > and < (ua2 = D) > and works fine for given authorizations. In case of unrepresented partition, {FDF, FDG, GBF, GCF} gets acceptance, while {GBG, GCG} is denied. This mixed response shows "don't care" for unrepresented attribute value combinations again. The security architect should be at least aware of these unrepresented combinations. He might decide to don't care, or take suitable action based on his expertise. In our approach of the previous section unrepresented partitions are assumed to have deny access which is a conservative and safe security posture.

5 ABAC RuleSet Infeasibility Correction

We know from Section 3 that if the partition set is conflicted a suitable RuleSet cannot exist. There are at least two approaches to dealing with this situation in practice. One approach is to construct RuleSets that are only approximately equivalent to the given AUTH relation. Various notions of approximation can be defined and their security implications analyzed. The second approach, which we study in this paper, is to introduce additional attributes to reconcile the conflicted partitions. This leads us to introduce the following notion.

Definition 10 ABAC RuleSet Infeasibility Correction problem
Given, EAS $\langle U, O, OP, AUTH, checkAccess_{AUTH}\rangle$ and a partially defined ABAC system with unspecified RuleSet where, U, O, and OP are identical to the given EAS such that the partition set P is conflicted, the ABAC Ruleset Infeasibility Correction problem is to 1) add new attributes UA and/or OA, and 2) assign appropriate values to the added new attributes, so that it is possible to generate a suitable RuleSet.

Ideally, the newly added attributes should have semantic significance grounded in the underlying application domain, and should be assigned meaningful values appropriate to different users and objects. This will presumably require expert input from security architects, perhaps aided by artificial intelligence, machine learning and other automated techniques. Study of such approaches is beyond the scope of this paper. Here we investigate a purely automated approach which introduces new "artifical" attributes with "artificial" values.

We note that conflict-free partitions that are authorized for access with respect to an operation op can have rules generated as explained in the proof of Theorem 1, ignoring consideration of any new attributes. However, the conflicted partitions need to be further refined by means of these new attributes to remove the conflict. There can be many ways to do this. Clearly the minimum possible split of a conflicted partition is into two refined partitions, one with all user-object pair permitted for op while the second contains the denied ones. The maximum possible split is to put each tuple in the conflicted partition into its own refined partition. One possible approach to constructing appropriate refinements is described below.

Given ABAC RuleSet Infeasibility Correction instance, consider a $P_i \in P$ which is conflicted with respect to some op ∈ OP. Define the binary relation R_{Pi} on P_i as:
$R_{Pi} \equiv \{((u1,o1), (u2,o2))|\forall o \in O.\forall op \in OP.((u1, o, op)\in AUTH\Leftrightarrow(u2, o, op)\in AUTH) \wedge \forall u \in U.\forall op \in OP.((u, o1, op)\in AUTH\Leftrightarrow(u, o2, op)\in AUTH)\}$

Lemma 2 R_{Pi} is an equivalence relation.
Proof: Trivial by inspection.

Hence, R_{Pi} induces a partition on P_i.

Definition 11 Partition set S_i
Let S_i be the partition on P_i induced by R_{Pi} and denoted by $S_i = \{S_{i1}, S_{i2}, ..., S_{im}\}$, 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 of R_{Pi}, each $S_{ik} \in S_i$ is represented by a collection of (attribute name, value) pairs, $PV(S_{ik})$ where,
$PV(S_{ik}) \equiv$ For any $(u1, o1) \in S_i, (UV(u1) \cup OV(o1))$

The concept of conflict-free partition from Section 3 is extended to S_i.

Definition 12 Conflict-free partition set S_i
Given an EAS, $\langle U, O, OP, AUTH, checkAccess_{AUTH}\rangle$ and partition set S_i stated in definition 11, a $S_{ik} \in S_i$ is conflict-free with respect to a specific op ∈ OP and given AUTH in

Table 2. Example data set 2

(a) UAValue	
User	uat1
u1	F
u2	F
u3	F
u4	G
u5	G

(b) OAValue	
Object	oat1
o1	F
o2	F
o3	F
o4	G

(c) Range of attributes	
uat1	{F, G}
oat1	{F, G}

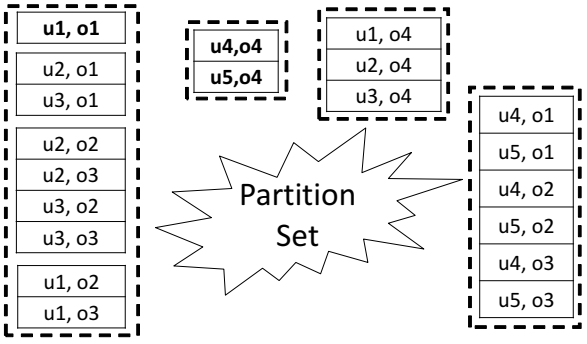

Figure 2. Refined partition set example.

EAS, iff the following statement is true:
($\forall$(u,o) $\in$ S_{ik}.(u,o,op) $\in$ AUTH) $\vee$ ($\forall$(u,o)$\in$ S_{ik}.(u,o,op) $\notin$ AUTH)
S_{ik} has conflict with respect to op $\in$ OP otherwise. Partition set S_i is conflict-free with respect to given AUTH in EAS iff for all op $\in$ OP, every $S_{ik} \in S_i$ is conflict-free with respect to op.

Fig. 2 shows the resulting partitions for Table 2 where, bold black user-object pairs belong to AUTH with respect to OP = {op} and rest are not. Here, U = {u1, u2, u3, u4, u5}, O = {o1, o2, o3, o4}, UA = {uat1}, OA = {oat1}, and Table 2 shows user attribute value assignment (UAValue), object attribute value assignment (OAValue), and range of attributes in (a), (b), and (c), respectively. To make visual comparison, the dotted rectangles in Fig. 2 shows partition set P for Table 2 as defined in Section 3. The leftmost conflicted parition is refined into four sub-partitions.

Lemma 3 Given a conflict partition $P_i \in P$ with respect to op $\in$ OP, the following holds:

a. S_i is conflict-free

b. S_i refines P_i

c. For all $S_{ik} \in S_i$, PV(S_{ik}) is the same

Proof:
By inspection of definition S_i, it is conflict-free. S_i refines P_i because for each $S_{ik} \in S_i$, $S_{ik} \subseteq P_i$. Since each $P_i \in P$ is identified by an unique PV(P_i) and S_i does the refinement only, therefore, for all $S_{ik} \in S_i$, PV(S_{ik}) is same and (c) follows.

Definition 13 Given a partition $P_i \in P$, let $uList_i$ and $oList_i$ denote the sets of users and objects present in P_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 4 $S_i = \{ul_{i1}, ..., ul_{iq}\} \times \{ol_{i1}, ..., ol_{ir}\}$.
Proof: Trivial by inspection of definitions.

Definition 14 Introducing new user and object attributes
If P_i is a conflict partition, the following steps are proposed where, UND specifies "Unknown" status of an attribute value assignment.

1. UA = UA $\cup$ exU and OA = OA $\cup$ exO where, exU and exO are new user and object attributes, respectively. Initially, for all u $\in$ U, exU(u) := UND and for all o $\in$ O, exO(o) := UND.

2. To ensure clarity, $PV_{new}(S_{ik} \in S_i)$ is introduced.
   $PV_{new}(S_{ik}) \equiv$ For any (u1, o1) $\in$ S_{ik}, (UV_{new}(u1) $\cup$ OV_{new}(o1))
   $UV_{new}(u{:}U) \equiv \{(ua, value) | ua \in (UA \cup exU) \wedge value = ua(u)\}$
   $OV_{new}(o{:}O) \equiv \{(oa, value) | oa \in (OA \cup exO) \wedge value = oa(o)\}$
   Here, Range(exU) and Range(exO) are sets of unique random values where Range(exU) $\cap$ Range(exO) = $\emptyset$. The sets of random values are chosen so that new attribute and corresponding range can be added in an automated manner without human provided input.

3. Given a $P_i \in P$, algorithm 2 is used to assign appropriate values to the newly added attributes. Inside partitionCorrection, each of the q partitions$\in$ $\{ul_{i1}, ..., ul_{iq}\}$ is assigned an unique random value from Range(exU). Hence, every user in the same partition gets the same exU value. Similarly, each of the r partitions$\in$ $\{ol_{i1}, ..., ol_{ir}\}$ is assigned an unique random value from Range(exO). Hence, every object in the same partition gets the same exO value.
   Note: By Definition 13, $uList_i$ and $oList_i$ is further partitioned using universal quantifications on the sets U,

Algorithm 2 PartitionCorrection

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} //Def. 13
oL := {ol_i1, ..., ol_ir} //Def. 13
if ∃u ∈ uList_i.exU(u) = UND then
    while ∃partu ∈ uL do
        uRandom := v ∈ Range(exU) such that ∀u ∈ U \ partu.exU(u) ≠ v
        For all u1 ∈ partu, exU(u1) := uRandom
        uL := uL \ partu
if ∃o ∈ oList_i.exO(o) = UND then
    while ∃parto ∈ oL do
        oRandom := v ∈ Range(exO) such that ∀o ∈ O \ parto.exO(o) ≠ v
        For all o1 ∈ parto, exO(o1) := oRandom
        oL := oL \ parto
        return S_i //{ul_i1, ..., ul_iq} × {ol_i1, ..., ol_ir}
```

O, and OP. Thereby, regardless of conflict partitions, once the exU and exO values are assigned by algorithm 2, they remain unchanged throughout the entire RuleSet generation process.

Lemma 5 Based on Definition 14, for each $S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique.

Proof: Follows trivially from Lemma 4 and Definition 14.

For instance, given a conflict partition, P_i in Fig. 2 where only (u1, o1) belongs to AUTH with respect to op, it is refined into four new partitions. Initially, $uList_i$ is $\{u1, u2, u3\}$ and $oList_i$ is $\{o1, o2, o3\}$. According to Algorithm 2, $uList_i$ is further partitioned into $\{\{u1\}, \{u2, u3\}\}$. Similarly, $oList_i$ is further partitioned into $\{\{o1\}, \{o2, o3\}\}$. The resulting refined partitions has same PV, given by $\{(uat1, F), (oat1, F)\}$. According to Definition 14 and Algorithm 2, let exU value for $\{u1\}$ and $\{u2, u3\}$ be 1 and 2, respectively. Similarly, $\{o1\}$ and $\{o2, o3\}$ are assigned 3 and 4 for exO, respectively. Thereby, resulting unique PV_{new} value for the refined partitions are {(uat1, F), (oat1, F), (exU, 1), (exO, 3)}, {(uat1, F), (oat1, F), (exU, 1), (exO, 4)}, {(uat1, F), (oat1, F), (exU, 2), (exO, 3)}, and {(uat1, F), (oat1, F), (exU, 2), (exO, 4)}, respectively.

Theorem 2 Given, an ABAC RuleSet Infeasibility Correction problem instance, it is always possible to find a suitable RuleSet such that the resulting ABAC system is equivalent to given EAS.

Proof:

The theorem will be proved by construction. For a specific op ∈ OP, $Rule_{op}$ construction steps are as follows. It is assumed that, partition set P construction does not depend on exU and exO.

1. Each conflict-free partition $P_i \in P$ is included in $Rule_{op}$ as conjunctive clause (same as Theorem 1) where, $P_i \times \{op\} \subseteq$ AUTH.

2. Each conflict partition $P_i \in P$ is refined by Definitions 13 and 14, which generate conflict-free partitions only and ensure that for each such $S_{ik} \in S_i$, $PV_{new}(S_{ik})$ is unique. For each of the resulting partition $S_{ik} \in S_i$, a conjunctive clause is included in $Rule_{op}$ only if $S_{ik} \times \{op\} \subseteq$ AUTH. For conflict partitions only, $Rule_{op}$ is given by:

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

where $cp(P) = \{P_i \in P \text{ is a conflict partition}\}$, $S_{ik} \in partitionCorrection(P_i)$, and $S_{ik} \times \{op\} \subseteq$ AUTH.

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

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

Here, $Rule_{op}$ is disjunction of all the conjunctive clauses generated in step 1 and 2. By definition, RuleSet contains a $Rule_{op}$, for each op ∈ OP. Hence, RuleSet can be generated. To prove equivalency between the resulting ABAC system with RuleSet and EAS, it is necessary and sufficient to show that, for a op in OP, $(a, b, op) \in AUTH \iff Rule_{op}(a, b)$ where $a \in U$, $b \in O$.

To prove the only if part: by inspection, $(a, b) \in U \times O$ belongs to only one partition, a $P_i \in P$. If P_i is conflict-free with respect to op then $P_i \times \{op\}$ must be a subset of AUTH. Hence, step 1 works. If P_i is a conflict partition, step 2 works. Let, (a,b)$\in S_{ik}$ where, $S_{ik} \in S_i$. Hence, $S_{ik} \times \{op\}$ must be a subset of AUTH. Since $Rule_{op}$ is disjunction of all the conjunctive clauses generated in step 1 and 2, thereby, $Rule_{op}(a, b)$ evaluates to true. Hence, only if part is proved.

To prove if part: by inspection, if $Rule_{op}(a, b)$ evaluates to true, then there must be a conjunctive clause of $Rule_{op}$, which is evaluated to true. By construction, each such conjunctive clause in $Rule_{op}$ is representing a specific partition where partition $\times \{op\} \subseteq$ AUTH. Since each such partition is conflict-free, every user-object pair in corresponding partition is permitted with respect to op, thus belongs to AUTH. Thereby, $(a, b, op) \in AUTH$, which proves if part. Hence, it can be concluded that generated suitable RuleSet proposed by the steps above, completes the ABAC system, and equivalent to given EAS.

Based on last example, two partitions {(u1, o1)}, and {(u4, o4), (u5, o4)} are included in $Rule_{op}$. Hence, $Rule_{op}$

is (uat1(u) = F $\wedge$ oat1(o) = F $\wedge$ exU(u) = 1 $\wedge$ exO(o) = 3) $\vee$ (uat1(u) = G $\wedge$ oat1(o) = G) and the RuleSet is $\{\text{Rule}_{\text{op}}\}$. In this example, both exU and exO are used for RuleSet Infeasibility Correction. If every user in U is represented by distinct user attribute value combination, exU is not required. The same condition holds for objects and exO.

Asymptotic complexity of ABAC RuleSet Infeasibility Correction is $O(|OP| \times (|U| \times |O|)^3)$. Given a partition set P with conflict, checking whether each $P_i \in P$ is conflict-free or not takes $O(|OP| \times (|U| \times |O|))$. If a $P_i \in P$ is in conflict with respect to a op $\in$ OP, Algorithm 2 is called to refine P_i only. Inside Algorithm 2, corresponding list of users in P_i is futher partitioned by comparing each user-user pair, hence takes $O(|U|^2)$ complexity. Similarly, list of objects in P_i is partitioned; hence takes $O(|O|^2)$ complexity. Since a partition cannot have more than |U| users and |O| objects, while loops inside PartitionCorrection have upper bound O(|U|) and O(|O|), respectively. Hence, overall asymptotic complexity of PartitionCorrection algorithm is $O((|U| \times |O|)^2)$. Thereby, overall complexity is given by $O(|OP| \times (|U| \times |O|)^3)$.

6 Related works

To the best of our knowledge, feasibility notion of Attribute-Based Access Control policy mining problem has been formally studied for the first time in this paper. Hence, there are no previous works directly related to this problem. However, there exist notable previous works on the field policy mining, such as rule mining, role mining [6, 8], ABAC policy mining, Relationship-Based Access Control (ReBAC) mining [1], and so on. ABAC policy mining problem was first introduced formally in [12]. Given an access control list policy as input, [12] finds out equivalent ABAC policy. Here, ABAC rule is a tuple specifying sets of users, objects, operations, and constraints involving user and objects attributes. Although constraints give more generalized rule, but only a few forms of constraints are allowed in this paper. Another work with authorization data as input is [9]. Despite having the same asymptotic complexity, [9] shows better performance with respect to total execution time. In [9], two algorithms for ABAC policy mining, ABAC-FDM, and ABAC-SRM have been proposed. ABAC-FDM is accurate but due to its exponential complexity, more efficient ABAC-SRM is proposed. An out of the box approach is given in [3], which deals with positive as well as negative ABAC rules. This work basically depends on PRISM, an existing rule mining algorithm.

Based on variety of the input, some other notable ABAC policy mining works are: RBAC [10], log data [11] and sparse log [2]. A deep learning approach towards ABAC policy mining from logs using Restricted Boltzmann Machine (RBM) has been presented in [7]. Apart from these works, an evolutionary computation approach for ABAC policy mining is presented in [5], based on incremental learning of single rules and search-optimizing features.

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] T. Bui, S. Stoller, and J. Li. Mining relationship-based access control policies. In *SACMAT*, pages 239–246, 2017.

[2] C. Cotrini, T. Weghorn, and D. Basin. Mining ABAC rules from sparse logs. In *EuroSP*, pages 31–46. IEEE, 2018.

[3] P. Iyer and A. Masoumzadeh. Mining positive and negative attribute-based access control policy rules. In *SACMAT*, pages 161–172, 2018.

[4] X. Jin, R. Krishnan, and R. Sandhu. A unified attribute-based access control model covering DAC, MAC and RBAC. *DBSec*, 12:41–55, 2012.

[5] E. Medvet et al. Evolutionary inference of attribute-based access control policies. In *Evolutionary Multi-Criterion Optimization*, pages 351–365. Springer, 2015.

[6] B. Mitra et al. A survey of role mining. *ACM CSurv. 2016*.

[7] D. Mocanu, F. Turkmen, and A. Liotta. Towards ABAC policy mining from logs with deep learning. In *IS 2015*.

[8] I. Molloy et al. Evaluating role mining algorithms. In *SACMAT*, pages 95–104, 2009.

[9] T. Talukdar et al. Efficient bottom-up mining of attribute based access control policies. In *IEEE CIC 2017, 339-348*.

[10] Z. Xu and S. Stoller. Mining attribute-based access control policies from RBAC policies. In *IEEE CEWIT*, 2013, 1-6.

[11] Z. Xu and S. Stoller. Mining attribute-based access control policies from logs. In *DBSec*, 2014, 276–291.

[12] Z. Xu and S. Stoller. Mining attribute-based access control policies. *IEEE TDSC*, 12(5):533–545, 2015.