Poster: Constrained Policy Mining in Attribute Based Access Control

Mayank Gautam
IIT Kharagpur, India
gautamayank@gmail.com

Sadhana Jha
IIT Kharagpur, India
sadhanajha@iitkgp.ac.in

Shamik Sural
IIT Kharagpur, India
shamik@cse.iitkgp.ernet.in

Jaideep Vaidya
Rutgers University, USA
jsvaidya@business.rutgers.edu

Vijayalakshmi Atluri
Rutgers University, USA
atluri@rutgers.edu

ABSTRACT

In practical access control systems, it is important to enforce an upper bound on the time taken to respond to an access request. This response time is directly influenced by the size (often called the *weight*) of each of the underlying access control rules. We present a constrained policy mining algorithm which takes an access control matrix as input and generates a set of attribute based access control (ABAC) rules, such that the weight of each rule is not more than a specified value and the sum of weights of all the rules is minimized. Our initial experiments show encouraging results.

CCS CONCEPTS

• **Security and privacy** → **Access control;** *Authorization;*

KEYWORDS

ABAC, Policy mining, Constraint, Weight

ACM Reference format:
Mayank Gautam, Sadhana Jha, Shamik Sural, Jaideep Vaidya, and Vijayalakshmi Atluri. 2017. Poster: Constrained Policy Mining in Attribute
Based Access Control. In *Proceedings of SACMAT'17, June 21–23, 2017, Indianapolis, IN, USA, ,* 3 pages.
DOI: http://dx.doi.org/10.1145/3078861.3084163

1 INTRODUCTION

Over the past few years, attribute based access control (ABAC) [5] has gained increasing importance due to its ability to unify most of the other access control models and its promise to facilitate ad-hoc and dynamic collaboration among multiple domains. Further, traditional access control models such as discretionary access control (DAC), which uses an access control matrix (ACM), and role-based access control (RBAC) [4], which uses the notion of roles, have been shown to be ineffective in handling fine grained access control, thus generating an urgent need for existing organizations to migrate to ABAC.

The first step towards this is to create an appropriate set of ABAC rules (collectively called the ABAC policy of the organization). Each rule comprises a set of user, object and environmental attribute name-value pairs along with the name of an operation the rule permits. Granularity of access decision can be controlled by varying the number of attributes considered in such rules. However, adding more number of attributes affects the time needed to evaluate each rule when an access request is made. It also renders the ABAC policy unsuitable for automated security analysis. The organization, therefore, would like to limit the *weight* of each rule, which is a function of its size.

Keeping the above mentioned requirement in mind, we propose a constrained policy mining algorithm that takes an ACM as input and generates a minimal set of ABAC rules with a constraint on the maximum weight for each individual rule. Here, minimality is in terms of the total weight of all the rules. While there is some work reported in the literature on an unconstrained version of the problem [1], [6], and also on constrained versions of role mining in the context of RBAC [3], to the best of our knowledge, this is the first ever attempt towards formulating and solving ABAC policy mining under constraints.

2 ABAC POLICY SPECIFICATION

An ABAC system consists of a pre-defined set of user attributes ($\mathcal{U}_a$) and object attributes ($\mathcal{O}_a$). We leave aside environmental attributes for the sake of brevity. The proposed approach can be easily extended to include those as well. The possible set of values an attribute $a \in \{\mathcal{U}_a \cup \mathcal{O}_a\}$ can acquire is denoted as $V(a)$. $UAV(u, a) \in V(a)$ denotes the value of the attribute a for a given user u. OAV is defined similarly for objects.

The set of attribute name-value pairs associated with a user u and an object o are represented as $\mathcal{A}_u$ and $\mathcal{A}_o$, respectively. For instance, consider a user named *Tom* who is an adult and has premium membership in an online movie viewing application. This is represented as $\mathcal{A}_{Tom} = \{\text{ageGroup} = \textit{adult}, \text{memberType} = \textit{premium}\}$. An ABAC rule is of the form $\langle UC, OC, OP\rangle$, where UC, OC and OP respectively represent a set of user attribute name-value pairs, object attribute name-value pairs and the name of an operation. For instance, a rule of the form $\langle\{memberType = premium\}, \{release = recent\}, view\rangle$ conveys that any *premium* member can *view* movies released *recently*. . Each rule r is assigned a weight W, which can be defined in various ways. In this paper, we consider it to be the count of the total number of elements in r. For instance, for the rule given above, $W(r) = 3$. For a complete ABAC policy P comprised of a set of ABAC rules, the total weight, $TW(P)$ is the sum of the weights of the individual rules in P, i.e., $TW(P) = \sum_{r \in P} W(r)$.

3 CONSTRAINED ABAC POLICY MINING

The input for policy mining is an ACM, whose rows and columns represent users and objects, respectively. A cell (u, o) of the ACM stores the permission p that a user u has on an object o, and is denoted as (u, o, p).

3.1 Problem Definition

Definition 3.1. Constrained ABAC Policy Mining Problem (CAPM): Given an access control matrix UP, user attributes $\mathcal{U}_a$, object attributes $\mathcal{O}_a$, user attribute assignments UAV, object attribute assignments OAV and an integer value c, generate a set P of ABAC rules such that the following conditions are satisfied: (i) each entry in UP is covered by some rule $r \in P$ (ii) no user gets any extra permission, which is not present in UP through some rule $r \in P$ (iii) for each rule $r \in P$, $W(r) \leq c$, and (iv) $TW(P)$ is minimum.

3.2 Complexity Analysis of CAPM

To determine the complexity class of CAPM, we first define D_CAPM, the decision version of CAPM.

Definition 3.2. D_CAPM: Given $\mathcal{U}_a$, $\mathcal{O}_a$, UAV, OAV, UP and two integer values c and k, does there exist a set P of ABAC rules such that the rules in P together cover all (u, o, p) $\in UP$, and for each $r \in P$, $W(r) \leq c$ and TW(P)$\leq k$.

THEOREM 3.3. *D_CAPM is NP-complete*

PROOF. To prove D_CAPM is NP-complete, we show that D_CAPM is in NP and it is NP-hard.

D_CAPM is in NP: Suppose a certificate comprising $\mathcal{U}_a$, $\mathcal{O}_a$, UAV, OAV, UP along with a set P of ABAC rules and two integer values c and k are given and it is claimed that the rules in P together cover all (u, o, p) $\in UP$, for each rule $r \in P$, $W(r) \leq c$ and $TW(P) \leq k$. This claim can be easily verified by computing the weight of each rule in P, the total weight $TW(P)$ and searching a rule corresponding to each entry in the given UP. The above verification can be done in polynomial time. Thus, D_CAPM is in NP.

D_CAPM is NP-hard: We show that D_CAPM is NP-hard by reducing the decision version of unconstrained ABAC policy mining problem (D_APM), which is known to be NP-complete [6], to D_CAPM. D_APM can be stated as: Given a set of user attributes $\mathcal{U}'_a$, object attributes $\mathcal{O}'_a$, user attribute assignments UAV', object attribute assignments OAV' and an access control matrix UP', does there exist a set P' of ABAC rules such that the rules in P' together cover all (u, o, p) $\in UP'$. Given an instance $\langle$ $\mathcal{U}'_a$, $\mathcal{O}'_a$, UAV', OAV', $UP'\rangle$ of D_APM, the steps for reducing it to a D_CAPM instance $\langle$ $\mathcal{U}_a$, $\mathcal{O}_a$, UAV, OAV, UP, $c\rangle$ are as follows:

- set $\mathcal{U}_a = \mathcal{U}'_a$, $\mathcal{O}_a = \mathcal{O}'_a$, $UAV = UAV'$, $OAV = OAV'$
- set $UP = UP'$ and set c as a very large number

The above reduction can be done in polynomial time. Also, a *yes* (respectively *no*) answer to the constructed *D_CAPM* instance gives a *yes* (respectively *no*) answer to the *D_APM* instance. Hence, D_CAPM is NP-hard. Since, D_CAPM is NP-hard and is in NP, it is NP-complete. □

3.3 Solving CAPM using Constrained Weighted Set Cover Heuristic

Showing D_CAPM to be NP-complete in the last sub-section implies that one is not likely to find an efficient solution for CAPM. We, therefore, develop a heuristic solution for CAPM by mapping it to a variant of the well-known set cover problem. A constrained weighted set cover problem (CWSC) [2] can be defined as follows: Given a universal set U, a collection S of subsets of U, a function f that assigns a positive integer weight to each set $s \in S$, and an integer k, find a minimum weight sub collection M of S, such that for each set $m \in M$, $f(m) \leq k$ and $\cup_{m \in M} = U$.

To solve CAPM using the heuristics available for the CWSC problem, we map the CAPM problem to the CWSC problem as follows. Given an instance $\langle$ $\mathcal{U}_a$, $\mathcal{O}_a$, UAV, OAV, UP, $c\rangle$ of CAPM, an instance $\langle U, S, f, k\rangle$ of CWSC can be generated as follows:

- Form U by performing the union of $\langle$ $\mathcal{A}_u$, $\mathcal{A}_o$, $p\rangle$ for each (u, o, p) $\in$ UP.
- Set k equal to c, f: $S \rightarrow \{1...c\}$
- Form S by generating all possible combinations of the set ($\mathcal{U}_a \cup \mathcal{O}_a \cup OP$), such that each generated set is constrained to have weight computed using the function f to be less than or equal to c.

If the total weight of the set of subsets obtained after solving the CWSC instance is t, then the total weight TW of the generated ABAC policy P would also be t. Since the range of the weight function is $\{1...c\}$, it is ensured that for each rule $r \in P$, $W(r) \leq c$. A minimum value of t ensures a minimum value of $TW(P)$.

An algorithm for solving CAPM (*Mine-ABAC-Policy*) is given in Algorithm 1. The set *UniversalUPOSet* consists of tuples of the form $\langle$ $\mathcal{A}_u$, $\mathcal{A}_o$, $p\rangle$, where each tuple represents a (u, o, p) $\in UP$. For each $\langle$ $\mathcal{A}_u$, $\mathcal{A}_o$, $p\rangle \in$ *UniversalUPOSet*, *TotalAttrSet* stores the set of all attribute value pairs in $\mathcal{A}_u$, $\mathcal{A}_o$ and p. All possible ABAC rules having cardinality less than c are generated using the set of attributes present in the set *TotalAttrSet* and are stored in the set *AllPossibleRules*. From the generated set, rules that represent at least one entry in UP are selected and stored in *ValidRules*[i], where i represents the i^{th} tuple in *UniversalUPOSet*. After retrieving the content of the *ValidRules* set for each $\langle$ $\mathcal{A}_u$, $\mathcal{A}_o$, $p\rangle$ $\in$ *UniversalUPOSet*, a set of *EffectiveRules* is formed by combining the content of all the available *ValidRules*. For each rule $r_j \in$ *EffectiveRules*, *TuplesCoveredinUP*[r] stores the set of (u, o, p) $\in UP$ covered by r. The generated set

Algorithm 1: Mine-ABAC-Policy

```
Input: $UP$, $\mathcal{U}_a$, $\mathcal{O}_a$, $UAV$, $OAV$, $c$
Output: $Policy$
$i$, $EffectiveRules \leftarrow 0$, $ValidRules[i]$, $RulesCoveredinUP$,
$UniversalUPOSet$, $UOP \leftarrow \phi$
for each $(u, o, p) \in UP$ do
    $UniversalUPOSet \leftarrow UniversalUPOSet \cup \{\langle \mathcal{A}_u, \mathcal{A}_o, p\rangle\}$
end
for each $\langle \mathcal{A}_u, \mathcal{A}_o, p\rangle \in UniversalUPOSet$ do
    $TotalAttrSet \leftarrow \mathcal{A}_u \cup \mathcal{A}_o \cup p$
    $AllPossibleRules \leftarrow r| \ r \in 2^{\mathcal{A}_u \cup \mathcal{A}_o \cup p}$ and $|r| \leq c$
    $ValidRules$[i] $\leftarrow$ r$'$ | r$'$ $\in$ $AllPossibleRules$ and $r'$ covers at
    least one $(u, o, op) \in UP$
    i++
end
for $j \leftarrow 0$ to $i$ do
    $EffectiveRules = EffectiveRules \cup ValidRules[j]$
end
for $j \leftarrow 0$ to $|EffectiveRules|$ do
    $TuplesCoveredinUP[j] \leftarrow$ set of $(u, o, p)$ covered by the
    j$^{th}$ rule in $EffectiveRules$
end
$UOP \leftarrow$ store each $(u, o, p) \in UP$
$TempRules \leftarrow GenMinSetCover(UOP, TuplesCoveredinUP)$
$Policy \leftarrow Merge\text{-}Rules(TempRules)$ return $Policy$
```

Algorithm 2: Merge-Rules

```
Input: $TempRules$
Output: $TempRules$ after merging
$Policy \leftarrow \phi$
for each $(r_1, r_2) \in TempRules$ do
    if $(|r_1 + r_2| \leq c$ and does not uncover any $(u, o, p) \in$
    $UP)$ then
        $r' \leftarrow uc(r_1) \cup uc(r_2) \cup oc(r_1) \cup oc(r_2) \cup op(r_1) \cup$
        $op(r_2)$
        $TempRules \leftarrow TempRules \setminus \{r_1, r_2\} \cup \{r'\}$
    end
end
return $TempRules$
```

of subsets along with the set *UOP* containing all (*u*, *o*, *p*) $\in$ *UP* is then passed to a function called *GenMinSetCover*, which returns the set *TempRules* representing the minimal weighted set of rules covering all entries in *UP*.

For further reduction of the weight of the generated policy, we perform an additional merging step on the set of rules generated by *GenMinSetCover*. Merging of rules is done using the *Merge-Rules* function given in Algorithm 2. It compares each pair of rules r_1 and r_2 in *TempRules*, checks for redundancy, and if $|W(r_1) + W(r_2)| \leq c$, merges r_1 and r_2 if merging does not introduce any new (*u*, *o*, *p*) which is not in *UP*. The set of rules generated after the merging operation forms the final ABAC policy.

4 EXPERIMENTAL RESULTS

We implemented our approach described in the last section and compared it with the policy mining algorithm presented in [6] using the data sets made available by them (http://www3.cs.stonybrook. edu/stoller). The experiments were run on an Intel Xeon E5-2697 v2 processor @ 2.70 GHz with 256 GB RAM.

Data set	W	TW ($\|P\|$)	Time (*in secs*)
Online-Video	4	21(6)	1.13
	4	19 (6)	0.09
Healthcare	4	117(33)	4.80
	4	88 (26)	2.50
University	8	243 (53)	9.10
	5	223 (49)	5.40
Project-Management	7	185 (40)	9.10
	5	182 (38)	19.90

Table 1: Comparative performance of the proposed approach with [6]

Table 1 shows the maximum value of the weights W of the individual rules, total weight TW of the policy along with the number of rules $|P|$ within parenthesis, as well as the execution time for each approach. It may be noted that, each row of the table has two sub-rows, the first corresponds to [6] and the second is the value obtained by our approach. From the table, it can be seen that for most of the data sets, our approach outperforms the existing approach in terms of the total weight of the policies, number of rules as well as the execution time. Note that, while the existing approach does not explicitly try to impose any constraint on the weight of individual rules, the rules it generates are seen to have higher values of weights and hence, would not be valid if constraints are imposed. For most of the data sets, the proposed algorithm generates lightweight policies in lesser time, thus making it effective for mining ABAC policies under constraint.

5 CONCLUSION

This work presents a constrained policy mining algorithm for attribute based access control systems. Minimizing the total weight of all the rules in the mined policy while ensuring that no individual rule can have weight greater than a pre-specified constraint, helps to develop ABAC systems that are easy to enforce and are also amenable to efficient security analysis. Future work would involve enforcing other types of constraints and improving the efficiency of the approach.

REFERENCES

[1] Y. Benkaouz, M. Erradi, and B. Freisleben. Work in Progress: K-Nearest Neighbors Techniques for ABAC Policies Clustering. *ACM Intl. Workshop on ABAC*, 72–75.

[2] M. Cygan, L. Kowalik, and M. Wykurz. Exponential-time Approximation of Weighted Set Cover. *Inf. Proc. Let.* (2009), 957–961.

[3] P. Harika, M. Nagajyothi, J. C. John, S. Sural, J. Vaidya, and V. Atluri. Meeting cardinality Constraints in Role Mining. *IEEE Trans. Dep. and Sec. Com.* (2015), 71–84.

[4] R. S. Sandhu, J. E. Coyne, H. L. Feinstein, and C. E. Youman. Role-based Access Control Models. *IEEE Computer* (1996), 38–47.

[5] D. Servos and S. L. Osborn. Current Research and Open Problems in Attribute-Based Access Control. *ACM Comp. Sur.* (2017), 1–45.

[6] Z. Xu and S. D. Stoller. Mining attribute-based access control policies. *IEEE Trans. Dep. and Sec. Com.* 12, 5 (2015), 533–545.