EAI Endorsed Transactions
on Security and Safety

Research Article EAI.EU

Formal Approach to Detect and Resolve Anomalies while Clustering ABAC Policies

Maryem Ait El Hadj[1,*], Ahmed Khoumsi[2], Yahya Benkaouz[3], Mohammed Erradi[1]

[1]Networking and Distributed Systems Research Group, ITM Team, ENSIAS, Mohammed V University in Rabat, Morocco

[2]Dept. Electrical & Comp. Eng., University of Sherbrooke, Canada
[3]Conception and Systems Laboratory, FSR, Mohammed V University in Rabat, Morocco

Abstract

In big data environments with big number of users and high volume of data, we need to manage the corresponding huge number of security policies. Using Attribute-Based Access Control (ABAC) model to ensure access control might become complex and hard to manage. Moreover, ABAC policies may be aggregated from multiple parties. Therefore, they may contain several anomalies such as conflicts and redundancies, resulting in safety and availability problems. Several policy analysis and design methods have been proposed. However, most of these methods do not preserve the original policy semantics. In this paper, we present an ABAC anomaly detection and resolution method based on the access domain concept, while preserving the policy semantics. To make the suggested method scalable for large policies, we decompose the policy into clusters of rules, then the method is applied to each cluster. We prove correctness of the method and evaluate its computational complexity. Experimental results are given and discussed.

Received on 11 October 2018; accepted on 16 November 2018; published on 03 December 2018

Keywords: ABAC Policies, Clustering, Access Domain, Conflict, Redundancy, Detection and Resolution, Permissive Resolution, Restrictive Resolution.

doi:10.4108/eai.13-7-2018.156003

1. Introduction

In the current big data environments, a huge amount of data can be generated from various sources, which require new forms of processing techniques in order to improve decision making. However, the rules regulating access to resources managed by such environments raise multiple security challenges. Hence, users need authorization systems to help them share their resources, data and applications with a large number of users without compromising security and privacy. Access control models represent a key component for providing security features.

Attribute-based access control (ABAC) has been suggested as a generic access control model [6, 17]. ABAC considers three categories of attributes: subject, resource, and environment. An attribute is assigned to a subject (e.g., user, application or process), resource (e.g., data structure, web service or system component) and environment (e.g., current time, location). These attributes may be considered as characteristics of anything that may be defined and to which a value may be assigned. ABAC representation is more expressive and fine-grained than existing access control models, because it might consider any combination of subject, resource and environment attributes. However, due to the huge number of rules and the policies distributed management, deploying and managing an ABAC model to ensure access control might become too complex and hard to manage. In fact, an ABAC policy in distributed applications may be aggregated from multiple parties and can be managed by more than one administrator [8]. Therefore, ABAC policies may contain several anomalies [7, 35], such as conflicts and redundancies, which may lead to both safety and

*Corresponding author. Email: maryem_aitelhadj@um5.ac.ma

availability problems. Hence, detecting and resolving automatically such anomalies in large complex policies is crucial.

In the present paper, we suggest a method to detect and resolve anomalies in ABAC policies. We introduce the notion of *Access Domain* of a rule, which models the set of values of the attributes considered in that rule. Based on this concept of access domain, we develop a method to detect and resolve rigorously anomalies in an ABAC policy, while retaining the policy semantics. In contrast, several existing methods resolve anomalies by simply removing one of the conflicting rules, which modifies the semantics of the policy. To make the suggested method scalable with great policies (i.e., policies with a huge number of rules), we decompose the policy into several clusters of rules, and then the method is applied to each cluster. A preliminary version of this work is given in [2] which presents succinctly a method to detect and resolve anomalies. Compared to [2], our contributions are as follows:

- We present with more details and explanations our method to detect and resolve anomalies.
- We prove formally correctness of our method, thus guaranteeing that the anomalies are detected and removed from the original policy, while preserving its semantics.
- We evaluate the computational complexity of the proposed approach of detection and resolution.
- We provide experimental results with a set of ABAC policies that demonstrate the time gained from clustering.

The rest of the paper is organized as follows: Section 2 presents the formal definitions of security rules and the considered anomalies. In section 3, we define formally our problem of anomaly detection and resolution, and then we present an outline of the method we have developed to solve our problem. Section 4 presents formally the method we have developed. In Section 5, we prove correctness of the method and evaluate its computational complexity. Section 6 reports and discusses experimental results. In Section 7, we present related work and recall our contributions. Finally, the conclusion and expected future work are presented in section 8.

2. Formal Definitions of Security Rules & Anomalies

In order to present a formal method to detect and resolve anomalies between the rules of a policy, we first need to define formally the rules and the anomalies.

2.1. Formal Definition of a Security Rule and its Access Domain

A policy P is a non-empty set of rules: $P = \{r_1, r_2, ..., r_n\}$. Each rule $r_i \in P$ is specified by a condition and an access decision. The condition of a rule is specified by one or several assignments $att \in V_{att}$, where att is a name that identifies an attribute and V_{att} is a set of possible values of att. There is at most one assignment for each attribute. An access decision of a rule is noted X_{act}, where X is the decision $Permit$ or $Deny$, and act is a set of (access) actions. $Permit_{read}$ and $Deny_{write}$ are two examples of action decisions. A rule $r_i \in P$ will be written as follows:

$$r_i : X_{act}(att_1 \in V_{att_1}, att_2 \in V_{att_2}, ..., att_m \in V_{att_m}) \quad (1)$$

Note that the absence of assignment for an attribute att means the implicit existence of the assignment $att \in ALL_{att}$, where ALL_{att} denotes the set of all possible values of att. We consider the three categories of attributes of ABAC: subjects, resources, environment. The assignments corresponding to the same category are separated by a comma ",", while a semicolon ";" means the passing to the next category. An access request is defined by attribute values (at most one value for each attribute) and one action. We say that a value v of an attribute att satisfies an assignment $att \in V_{att}$ of a rule r_i, if v is an element of V_{att}. We say that an access request R matches a rule r_i (we can also say r_i matches R) if every attribute value of R satisfies the corresponding assignment of r_i.

Example 2.1. $Permit_{read}$ (position ∈ {Doctor}, specialist ∈{Generalist}, department ∈ {Oncology}; type ∈ {PR/CAT} , formatType ∈ {AST} , degreeOfConfidentiality ∈ {Secret} ; organization ∈ {EMS} , time ∈ [8:00, 12:00]).

The attributes of the *Subject* category are: position, specialist, department. The attributes of the *Resource* category are: type, formatType, degreeOfConfidentiality. The attributes of the *Environment* category are: organization, time. Intuitively, this rule indicates that every generalist doctor belonging to oncology department is permitted to read a secret AST file from 8 am to 12 am in EMS.

Instead of formulating a rule r_i as in (1), we will use the equivalent formulation (2) which is more convenient to define anomalies and their detection and resolution. The idea is to specify a unique set of values (namely $V_{att_1} \times V_{att_2} \times ... \times V_{att_m}$) for the n-tuple $(att_1, att_2, ..., att_m)$, instead of specifying a set of values for each att_i. Such set $V_{att_1} \times V_{att_2} \times ... \times V_{att_m}$ is called the *access domain* of r_i and noted AD_{r_i}. Hence, a rule is expressed in the form $r_i = X_{act}((att_1, att_2..., att_m) \in AD_{r_i})$. We can also write $r_i = X_{act}(AD_{r_i})$ when the attributes are known from the context and can hence

be implicit.

$$r_i : X_{act}((att_1, att_2..., att_m) \in V_{att_1} \times V_{att_2} \times ... \times V_{att_m}) \quad (2)$$

2.2. Formal Definitions of the Considered Anomalies

Anomalies are defined as patterns in data that do not conform to a well-defined notion of normal behavior [7]. More specifically, in a security policy P, an anomaly may exist only if several rules of P match the same access request. We have considered two types of anomalies: *Redundancies* and *Conflicts*.

Definition 2.1. Redundancy occurs in a policy P, when P contains useless (or redundant) rules, i.e. rules whose removal does not modify the behavior of the P. Consider two rules $r_i = X_a(AD_{r_i})$ and $r_j = Y_b(AD_{r_j})$. r_i is redundant to r_j iff:

$$\begin{cases} AD_{r_i} \subseteq AD_{r_j} \\ X = Y, and \\ a \subseteq b \end{cases} \quad (3)$$

Intuitively, every decision taken by r_i on any request is also taken by r_j. Therefore, r_i is useless and hence can be removed from the policy. We consider redundancy as an anomaly, because it may affect the performance of a policy, since verifying if an access request respects a policy depends on the size of the policy.

Example 2.2. Consider the following rules r_1 and r_2:

- r_1: $Permit_{\{read,write\}}$((position; fileType; time) ∈ {Doctor, Nurse} × {Source, Documentation} × [8:00, 18:00]).
- r_2: $Permit_{\{read\}}$((position; fileType; time) ∈ {Nurse} × {Documentation} × [8:00, 18:00]).

r_2 is redundant to r_1, because ({Nurse} × {Documentation} × [8:00, 18:00]) ⊂ ({Doctor, Nurse}× {Source, Documentation} × [8:00, 18:00]) and {read} ⊂ {read, write}.

We define the following notions, given two rules $r_i = X_a(AD_{r_i})$ and $r_j = Y_b(AD_{r_j})$:

- Common access domain of r_i and r_j is the intersection of their access domains, i.e. $AD_{r_i} \cap AD_{r_j}$.
- Set of common actions of r_i and r_j is the intersection of their sets of actions, i.e. $a \cap b$.

Definition 2.2. A conflict occurs in a policy P, when P contains two or more rules that generate contradictory decisions on an access request. Consider two rules r_i=$X_a(AD_{r_i})$ and r_j=$Y_b(AD_{r_j})$. r_i and r_j present a conflict (or are conflicting) iff:

$$\begin{cases} AD_{r_i} \cap AD_{r_j} \neq \emptyset \\ X \neq Y, and \\ a \cap b \neq \emptyset \end{cases} \quad (4)$$

Intuitively, when an access request matches the common access domain of r_i and r_j ($AD_{r_i} \cap AD_{r_j}$), we have contradictory decisions (from $X \neq Y$) on common actions (from $a \cap b \neq 0$).

Example 2.3. Consider the following rules r_1 and r_2:

- r_1: $Deny_{\{read\}}$((position; fileType; time) ∈ {Doctor, Nurse} × {Source, Documentation} × [8:00, 18:00])
- r_2: $Permit_{\{read,write\}}$((position; fileType; time) ∈ {Nurse} × {Documentation} × [8:00, 18:00])

r_1 and r_2 are conflicting, because $AD_{r_1} \cap AD_{r_2} = Nurse \times Documentation \times [8:00, 18:00] \neq \emptyset$, while the action *read* is permitted by r_2 and forbidden by r_1. Intuitively r_1 forbids that nurses read the documentation, while r_2 permits it.

3. Problem Definition and Outline of its Resolution

3.1. Problem Definition

Access control models are concerned with determining the allowed activities of legitimate users, mediating attempt by a user to access a resource in a given system [24, 36]. In this paper, we consider Attribute-Based Access Control (ABAC) that is widely used as a generic access control model. Correctness of an ABAC policy is critical for the security of the system that uses it, because any error in ABAC definition may result in violations of security features (e.g., confidentiality, integrity). In large and distributed organizations with complex ABAC policies, deploying and managing an ABAC model to ensure authorization management might become too complex and hard to manage. An ABAC policy in distributed applications may be aggregated from multiple parties and can be managed by more than one administrator (distributed management). Therefore, it may contain several anomalies such as redundancies and conflicts (see definitions 2.1 and 2.2), which may lead to safety and availability problems. Moreover, manual inspection for correctness can be impractical, because of the huge number of rules and the policies distributed management [25]. Thus, detecting and resolving automatically such anomalies is essential to ensure that an ABAC policy conforms to desired correctness properties. The problem we aim to solve is formulated as follows:

Given a policy P defined by a set of rules formulated as shown in Expression (2) of section 2.1, the objective is to detect and remove from P: redundancies and conflicts, which have been formally expressed in definitions 2.1 and 2.2.

Anomaly detection and resolution is motivated by the fact that errors in the policy definition may compromise the system security. Conflicts, if not handled properly, may lead to inappropriate decisions. As a result,

conflicts may lead to safety problems by allowing unauthorized accesses, and availability problems by denying authorized accesses. As for redundancies, their detection and resolution is motivated by the fact that they affect the performance of the policy execution, since the response time of a policy to an access request depends on the number of rules to be parsed in the policy.

Let P be any security policy and Q be the policy obtained from P by our detection and resolution procedure. An understandable requirement is that Q must be generated from P in a bounded time, i.e. the procedure does not enter in an infinite loop. A second natural requirement is that P and Q must support the same set of access requests. A third obvious requirement is that Q must be anomaly-free. A fourth requirement which makes sense is that P and Q must take the same decision for every access request for which P is non-conflicting. The fourth requirement is necessary to avoid the generation of Q that permits requests that are not permitted by the original policy P. In the same way, it is necessary to avoid the generation of Q that denies requests that are not denied by P. From these requirements, we define correctness of our detection and resolution method as follows, where the access domain of a policy P is the union of the access domains of all the rules of P:

Definition 3.1. Our detection and resolution method (let us call it M) is said to be correct, if for every policy P given as input to M and the policy Q obtained by M from P, the following five conditions are satisfied:

- C_1. Q is obtained after a finite number of iterations.
- C_2. P and Q have the same access domain.
- C_3. Q is anomaly-free.
- C_4. For every access request rq: If P has no rule permitting rq, then Q has no rule permitting rq.
- C_5. For every access request rq: If P has no rule denying rq, then Q has no rule denying rq.

3.2. Outline of the Method to Detect and Resolve Anomalies

The suggested method to detect and resolve anomalies is preceded by two steps: rules extraction, and rules clustering:

- *Rules Extraction*: It consists in parsing the policy in order to recognize and extract its rules. The extracted rules are expressed with the formulation 2, section 2.1. Recall that we use the three attribute categories of ABAC: Subject, Resource and Environment.
- *Rule Clustering*: To make the detection and resolution method scalable for policies with a huge number of rules, we suggest to apply a clustering method to group similar rules in the same cluster, based on an adequate similarity score such that non-similar rules are unlikely to be redundant or conflicting. The similarity measure we adopt is presented in our previous work [1]. We recall that the similarity measure is a function that assigns a similarity score to any two given rules r_i and r_j. Such a score reflects the degree of similarity between r_i and r_j, with respect to their subject, resource and environment attributes values. We say that two rules r_i and r_j are similar if their similarity score is greater than a given $threshold$. Its worth noting that the resulted clusters satisfy two properties: (1) each cluster contains at least one rule and (2) every rule is contained in one or more clusters.

After the extraction and clustering of rules, we arrive at the actual phase of detection and resolution of anomalies, which is executed within each cluster. The method is presented in detail in section 4, and its correctness is formally proved in section 5.

4. Anomalies Detection and Resolution

After constructing clusters of rules, the proposed anomaly detection and resolution method attempts to detect and remove anomalies within each cluster. In this section, we first show how redundancies and conflicts are detected and resolved between two rules, then within a set (cluster) of rules.

4.1. Redundancy Detection and resolution between two rules

The response time of a policy to an access request depends on the number of rules to be parsed in the policy [26]. So redundancy (i.e. existence of useless rules) may affect the performance of a policy, and hence is treated as an anomaly. Thus, removing redundancies is considered as one of the effective solutions for optimizing ABAC policies and improving the performance in policy decision time.

Given two rules r_i=$X_a(AD_{r_i})$ and r_j=$Y_b(AD_{r_j})$, r_i is detected to be redundant to r_j if the three conditions (3) in definition 2.1 are satisfied. The resolution of that anomaly consists in removing r_i.

Example 4.1. Consider the previous example 2.2, where we have shown that r_2 is redundant to r_1:

- r_1: $Permit_{\{read,write\}}$((position; fileType; time) ∈ {Doctor, Nurse} × {Source, Documentation} × [8:00, 18:00])
- r_2: $Permit_{\{read\}}$((position; fileType; time) ∈ {Nurse} × {Documentation} × [8:00, 18:00])

As already explained, redundancy of r_2 w.r.t. r_1 means that the effect of r_2 is included in the effect of

r_1, which implies that r_2 is useless in the presence of r_1. That is why, this redundancy is resolved by simply removing r_2 (and keeping r_1).

4.2. Conflict Detection and resolution between two rules

Given two rules r_i=$X_a(AD_{r_i})$ and r_j=$Y_b(AD_{r_j})$, r_i and r_j are detected to be conflicting if the three conditions (4) in definition 2.2 are satisfied. Since $X \neq Y$, let us take X=$Permit$ and Y=$Deny$. Recall that the intuition of a conflict between two rules is the existence of access requests for which the two rules do not agree whether to permit or deny them. We consider the following two resolution strategies:

- Permissive resolution: to permit the access requests for which the two rules disagree. This is realized by not modifying r_i and replacing r_j by the following two rules:
  - $r'_j = Deny_b(AD_{r_j} \setminus AD_{r_i})$
  - $r''_j = Deny_{b\setminus a}(AD_{r_i} \cap AD_{r_j})$

Intuitively, the unique modification that has been done is not denying the common actions of r_i and r_j for requests matching both r_i and r_j. It is easy to check that r_i, r'_j and r''_j are conflict-free with each other.

- Restrictive resolution: to deny the access requests for which the two rules disagree. This is realized by not modifying r_j and replacing r_i by the following two rules:
  - $r'_i = Permit_a(AD_{r_i} \setminus AD_{r_j})$
  - $r''_i = Permit_{a\setminus b}(AD_{r_i} \cap AD_{r_j})$

Intuitively, the unique modification that has been done is not permitting the common actions of r_i and r_j for requests matching both r_i and r_j. As in the permissive resolution, it is easy to check that r'_i, r''_i and r_j are conflict-free with each other.

Example 4.2. Consider the following two rules r_1 and r_2:

- r_1: $Permit_{\{read,write\}}$((position; fileType; time)$\in$ {Doctor, Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00])
- r_2: $Deny_{\{read,create\}}$((position; fileType; time)$\in$ {Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00])

The access domains of r_1 and r_2 are respectively, AD_{r_1} = {Doctor, Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00], AD_{r_2} = {Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00]. Therefore, the common access domains is $AD_{r_1} \cap AD_{r_2}$=AD_{r_2}, and the set of common actions is $\{read, write\} \cap \{read, create\} = \{read\}$. r_1 and r_2 are conflicting because the three conditions (4) given in definition 2.2 hold, i.e.: their common access domain and set of common actions are not empty, while their decisions are opposite.

The resolution of such conflict is as follows: if we consider the *permissive resolution*, r_1 is not modified (because the decision of r_1 is $Permit$) and r_2 is replaced by the following two rules:

- r'_2=$Deny_{\{read,create\}}$((position; fileType; time)$\in \emptyset$), so this rule is not considered since its access domain is empty.
- r''_2=$Deny_{\{create\}}$((position; fileType; time)$\in$ {Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00])

If we consider the *restrictive resolution*, r_2 is not modified (because the decision of r_2 is $Deny$) and r_1 is replaced by the following two rules:

- r'_1 =$Permit_{\{read,write\}}$((position;fileType;time)$\in$ {Doctor} $\times$ {Documentation} $\times$ [8:00, 18:00])
- r''_1=$Permit_{\{write\}}$((position;fileType;time)$\in$ {Nurse} $\times$ {Documentation} $\times$ [8:00, 18:00])

4.3. Anomaly detection and resolution in a cluster of rules

Anomaly detection and resolution in a cluster is an iterative process that consists in verifying the existence of anomalies and, if any, in modifying the rules of the cluster until the set of rules is anomaly-free. The approach consists in first constructing a graph (N, L), where N is a set of nodes, and L is a set of edges, where each edge is defined by a pair of nodes, i.e. $L \subseteq N \times N$. Each node represents a rule r_i, and each edge (r_i, r_j) means that we have to verify if there is an anomaly between r_i and r_j, and resolve it, if any. Initially, all nodes are connected, i.e. L consists of all pairs $(r_i, r_j) \in N \times N$ such that $i \neq j$. This is the input of Algorithm 1 which will verify and modify iteratively the graph until we obtain a graph without edges, which means that we have obtained an anomaly-free set of rules. At each iteration of Algorithm 1, the anomaly detection and resolution is applied to every edge (r_i, r_j) of L as explained below:

- If we detect that one of the two rules is redundant to the other one, then the resolution consists in removing the redundant rule from N (lines 4-9).
- If a conflict between r_i and r_j is detected (line 10), we have seen in subsection 4.2 that there are two strategies.
- In the **permissive resolution** (lines 15-22), r_i is not modified and r_j is replaced by r'_j and r''_j. Therefore, the graph is updated as follows:
  1. In the node r_j, replace the AD and act of r_j by the AD and act of r'_j.

Algorithm 1 Anomaly Detection and Resolution

```
Require: Graph (N, L): N is a set of nodes (corresponding to rules), L is a set of
    edges ⊆ N × N
Ensure: Set of nodes N (i.e. graph without edge)
 1: procedure AnomalyResolution(N, L)
 2:     while L is not empty do
 3:         Consider an edge (n1, n2) of L
 4:         if n1 and n2 have the same decision (Permit or Deny) then
 5:             if all actions of n1 are also actions of n2 then
 6:                 Remove n1 from N
 7:             else if all actions of n2 are also actions of n1 then
 8:                 Remove n2 from N
 9:             end if
10:         else if n1 and n2 have different decisions and common actions then
11:             Let np be the node among n1 and n2 whose decision is Permit,
                    and let Ap be its set of actions
12:             Let nd be the node among n1 and n2 whose decision is Deny,
                    and let Ad be its set of actions
13:             Let CD be the common access domain of np and nd
14:             Let CA be the set of common actions of np and nd
15:             if the resolution strategy is permissive then
16:                 Subtracts CD from the access domain of nd
17:                 for every node n of N other than np and nd do
18:                     if L does not contain the edge (nd, n) then
19:                         Insert the edge (nd, n) in L
20:                     end if
21:                 end for
22:                 Insert in N a new deny node nn whose access domain is CD
                        and set of actions is Ad \ CA
23:             else if the resolution strategy is restrictive then
24:                 Subtracts CD from the access domain of np
25:                 for every node n of N other than np and nd do
26:                     if L does not contain the edge (np, n) then
27:                         Insert the edge (np, n) in L
28:                     end if
29:                 end for
30:                 Insert in N a new permit node nn whose access domain is
                        CD and set of actions is Ad \ CA
31:             end if
32:             for every node n ∈ {np, nd, nn} do
33:                 if the access domain of n or its set of actions is empty then
34:                     Remove n from N
35:                 end if
36:             end for
37:             if nn has not been removed from N then
38:                 for every node n of N other than nn do
39:                     Insert the edge (n, nn) in L
40:                 end for
41:             else if np and nd have not been removed from N then
42:                 Remove the edge (np, nd ) from L
43:             end if
44:         end if
45:     end while
46:     return N
47: end procedure
```

2. Insert a new node in N that contains the AD and act of r_j''.
3. Update L by linking r_j' and r_j'' to all the nodes of the graph, except r_i (no link is created between r_j' and r_j'').
4. Remove the edge (r_i, r_j) from L, because there is no anomaly between them (after the modification of r_j in Point 1).

- In the **restrictive resolution** (lines 23-30), r_j is not modified and r_i is replaced by r_i' and r_j''. Therefore, the graph is updated as follows:

1. In the node of r_i, replace the AD and act of r_i by the AD and act of r_i'.
2. Create a new node that contains the AD and act of r_i''.
3. Update L by linking r_i' and r_i'' to all the nodes of the graph, except r_j (no link is created between r_i' and r_i'').
4. Remove the link between the nodes of r_i and r_j, because there is no anomaly between them (after the modification of r_i in Point 1).

- If no anomaly is detected between a pair of linked nodes r_i and r_j, the resolution algorithm simply removes the link between r_i and r_j (lines 41-43). Also, remove any node whose rule has an empty access domain or empty set of actions (lines 32-36).

Example 4.3. Consider a cluster consisting of the following four rules:

- $r_1 = Permit_{\{read,write\}}$ ((position; fileType; time) ∈ {Doctor, Nurse} × {Documentation} × [8:00, 18:00])
- $r_2 = Permit_{\{read\}}$ ((position; fileType; time) ∈ {Nurse} × {Documentation} × [8:00, 18:00])
- $r_3 = Deny_{\{read,delete\}}$ ((position; fileType; time) ∈ {Nurse} × {Source, Documentation} × [8:00, 18:00])
- $r_4 = Deny_{\{write,create\}}$ ((position; fileType; time) ∈ {Nurse} × {Source, Documentation} × [8:00, 18:00])

Since we consider a cluster of four rules, we get the 4-node and 6-edge as shown in Graph 1 of figure 1. Where each node represents a rule, and each edge represents the possibility of the existence of an anomaly.

- **First iteration**: Let us consider the pair (r_1, r_2) of Graph 1. Since $AD_{r_2} \subseteq AD_{r_1}$ and {read} ⊆ {read, write} thus, r_2 is redundant to r_1. The resolution procedure will remove r_2 and its edges from Graph 1. We obtain Graph 2 of figure 1.
- **Second iteration**: Let us consider the pair (r_1, r_3) of Graph 2. Since $AD_{r_1} \cap AD_{r_3}$= {Nurse} × {Documentation}× [8:00, 18:00]≠ ∅, a conflict is detected between r_1 and r_3. If we use the permissive resolution, we keep r_1 and r_3 is replaced by the following two rules, we then obtain Graph 3 of figure 1:

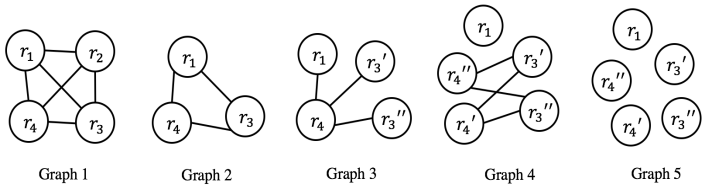

Figure 1. Constructed graphs from the cluster presented in example 4.3

  - r'_3=$Deny_{\{read,delete\}}$((position; fileType; time)∈{Nurse}×{Source}× [8:00, 18:00])
  - r''_3 = $Deny_{\{delete\}}$((position; fileType; time) ∈ {Nurse} × {Documentation} × [8:00, 18:00])

- **Third iteration**: Let us consider the pair (r_1, r_4) of Graph 3. Since $AD_{r_1} \cap AD_{r_4}$ = {Nurse} × {Documentation}× [8:00, 18:00]) ≠ ∅, a conflict is detected between r_1 and r_4. If we use the permissive resolution, we keep r_1 and r_4 is replaced by the following two rules, we then obtain Graph 4 of figure 1:

  - r'_4 =$Deny_{\{write,create\}}$((position; fileType; time)∈{Nurse}× {Source}× [8:00, 18:00])
  - $r4''$ = $Deny_{\{create\}}$((position; fileType; time)∈{Nurse} × {Documentation} × [8:00, 18:00])

- **Iterations 4 to 7**:The connected pairs of Graph 4 are (r'_3, r''_4), (r''_3, r'_4), (r'_3, r'_4) and (r''_3, r''_4). For each of these pairs, the intersection of access domains or the intersection of sets of actions is empty. Therefore, their four links are removed through the four iterations 4 to 7. We obtain Graph 5 of figure 1, that has no link. Therefore, the algorithm terminates.

5. Correctness and Complexity

5.1. Local Correctness

Our detection and resolution procedure generates a policy Q from P. In step 2 (rule clustering, section 3.2), P is decomposed into several clusters $P_1, P_2, \cdots, P_k$, where for each P_i we apply algorithm 1 to obtain Q_i. Then, all Q_i are aggregated to obtain Q. In this section, we prove correctness of the detection and resolution method, which is stated by the following theorem.

Theorem 1. Given a policy $P = P_1, P_2, \cdots$, for each cluster P_i ($i = 1, 2, \cdots$), algorithm 1 is correct w.r.t P_i and Q_i, that is the following conditions C_1-C_5 are satisfied w.r.t. each pair (P_i, Q_i). More precisely:

- C_1. Q_i is obtained after a finite number of iterations.
- C_2. P_i and Q_i have the same access domain.
- C_3. Q_i is anomaly-free.
- C_4. For every access request rq: If P_i has no rule permitting rq, then Q_i has no rule permitting rq.
- C_5. For every access request rq: If P_i has no rule denying rq, then Q_i has no rule denying rq.

In the following, we prove theorem 1.

For each cluster i (i=1, 2, ...), Algorithm 1 proceeds iteratively, where at each iteration $k+1$ ($k \geq 0$), a graph $G_i(k+1)$ is computed from a graph $G_i(k)$, where each graph $G_i(k)$ represents a policy noted $P_i(k)$. In particular, the original graph $G_i(0)$ corresponds to the original policy P_i, and the final graph $G_i(q)$ corresponds to the resulting policy Q_i obtained after a finite number q of iterations (finiteness of q comes from C_1 which is proved in section 5.1). Each iteration of Algorithm 1 consists in processing a pair of rules (r_1, r_2) in one of the following three cases:

- Case a: there is no anomaly between r_1 and r_2.
- Case b: there is a redundancy between r_1 and r_2.
- Case c: there is a conflict between r_1 and r_2.

We will consider cases a, b and c in the following proofs.

Proof of condition C_1. In an iteration of the algorithm:

- In case a: the link between the two rules is removed.
- In case b: the redundant rule is removed.
- In case c: one of the two rules (let AD denote its access domain) is split into two rules (let AD_1 and AD_2 denote their respective access domains) such that: $AD = AD_1 \cup AD_2$, $AD_1 \cap AD_2 = \emptyset$, and $AD_1 \neq \emptyset$ or $AD2 \neq \emptyset$.

We have the following:

1. Case a (resp. b) decreases the number of links (resp. nodes) of the graph.
2. Case c increments by 1 the number of nodes of the graph.
3. From 1 and 2 and the fact that the size of $G_i(0)$ is finite, the size of every $G_i(k)$ is finite.
4. From 1 and 3, we cannot have an infinite number of consecutive iterations executing cases a and b.
5. The size of the access domain of P_i is finite, because the domain of each attribute is finite.
6. Case c splits an access domain AD in two disjoint access domains AD_1 and AD_2, such that at least one of the two access domains is nonempty.
7. From 1, 5 and 6, the total number of iterations executing case c is finite.
8. From 4 and 7, the algorithm executes a sequence of iterations containing a finite number of cases c, such that two cases c are separated by a finite number (possibly 0) of cases a and b. Hence, the sequence is finite.

In the following, q denotes the finite number of iterations, i.e. $G_i(q)$ is the graph of the resulting policy Q_i.

Proof of condition C_2. In iteration $k+1$ (for $0 \leq k < q$) of the algorithm:

- In case *a*: the algorithm removes a link, without modifying any rule of the policy. Therefore, $P_i(k+1)$ and $P_i(k)$ have the same access domain.
- In case *b*: the algorithm removes a rule whose access domain is included in another rule of the policy. Hence, $P_i(k+1)$ and $P_i(k)$ have the same access domain.
- In case *c*: The algorithm replaces a rule whose access domain is AD by two rules whose access domains AD_1 and AD_2 are such that $AD = AD_1 \cup AD_2$. Hence, $P_i(k+1)$ and $P_i(k)$ have the same access domain.

Hence, in each iteration $k+1$ $(k \geq 0)$, $P_i(k+1)$ and $P_i(k)$ have the same access domain. By applying this result to all the iterations 1 to q, we obtain that Q_i and P_i have the same access domain.

Proof of condition C_3. Consider the following condition A:

A: *For every pair of unlinked rules r_1 and r_2, there is no anomaly between r_1 and r_2.*

In iteration $k+1$ (for $0 \leq k < q$) of the algorithm:

- In case *a*: the algorithm removes a link.
- In case *b*: the redundant rule is removed.
- In case *c*: let us consider the two resolution strategies.
  - *Permissive resolution strategy:* r_2 is replaced by r_2' and r_2'' which are then linked to all rules in the graph, except that no link is added between r_1, r_2' and r_2'' because there is no anomaly between them.
  - *Restrictive resolution strategy:* r_1 is replaced by r_1' and r_1'' which are then linked to all rules in the graph, except that no link is added between r_1', r_1'' and r_2 because there is no anomaly between them.

We have the following:

1. In all cases *a*, *b* and *c*, we have not created any pair of unlinked anomalous rules.
2. From 1, we deduce that if $G_i(k)$ satisfies A, then $G_i(k+1)$ satisfies A.
3. $G_i(0)$ satisfies condition A, because all its rules are linked.
4. From 2 and 3, we deduce that the graph $G_i(q)$ of Q_i satisfies A.
5. From 4 and the fact that $G_i(q)$ has no link, we deduce that Q_i is anomaly-free.

Proof of condition C_4. Let us first prove that for any access request rq : If $G_i(k)$ has no rule permitting rq, then $G_i(k+1)$ has no rule permitting rq. Consider an access request rq and assume that $G_i(k)$ has no rule permitting rq. In iteration $k+1$, for $0 \leq k < q$:

- In case *a*: Since the resolution consists in removing a link, no new rule permitting rq is created.
- In case *b*: Since the resolution consists in removing a rule, no new rule permitting rq is created in another rule of the policy. Hence, $P_i(k+1)$ and $P_i(k)$ have the same access domain.
- In case *c*:
  - *Permissive resolution strategy:* Since the resolution consists in replacing a deny-rule by two deny-rules, no new rule permitting rq is created.
  - *Restrictive resolution strategy:* in iteration $k+1$: A rule $r = Permit_a(AD)$ is replaced by two rules $r_u = Permit_a(AD_1)$ and $r_v = Permit_c(AD_2)$, such that $AD = AD_1 \cup AD_2$, $AD_1 \cap AD_2 = \emptyset$, and $c \subseteq a$. It is easy to see that: r permits rq, if and only if r_u or r_v permits rq, hence no new rule permitting rq is created.

From the fact that $G_i(k)$ has no rule permitting rq and the fact that no new rule permitting rq is created in iteration $k+1$, we deduce that $G_i(k+1)$ has no rule permitting rq. By applying this result to all the iterations 1 to q, we obtain that if P_i has no rule permitting rq, then Q_i has no rule permitting rq.

Proof of condition C_5. The proof of C_5 is obtained from the proof of C_4, by just replacing a few words as follows:

- Permit(ting) is replaced by Deny(ing), and vice versa.
- Permissive is replaced by Restrictive, and vice versa.

5.2. Correctness

We have proved in section 5.1 that for each $i = 1 \cdots k$: algorithm 1 satisfies C_1-C_5 w.r.t P_i and Q_i. In this section, we prove that C_1, C_2, C_4 and C_5 are satisfied w.r.t. (P, Q). Regarding C_3, we will prove that it is satisfied with a high probability. Indeed, C_3 is not satisfied in rare cases due to clustering. The latter is motivated by the fact that it improves the performance (as shown by experiment results in section 6).

- **Proof of C_1 and C_2**
  1. $P = \{P_1, P_2, \cdots, P_k\}$, where the access domain of P is the union of the access domains of all P_i $(i = 1 \cdots k)$.

2. $Q = \{Q_1, Q_2, \cdots, Q_k\}$, where the access domain of Q is the union of the access domains of all Q_i $(i = 1 \cdots k)$.
3. For each $i = 1 \cdots k$: Q_i satisfies C_1 and C_2 (section 5.1).
4. From 1,2 and 3, Q is executed in a finite number of iterations, and have the same access domain as P. Hence, Q satisfies C_1 and C_2.

- **Proof that C_3 is satisfied in most cases, i.e. Q is anomaly-free with high probability**

1. For each $i = 1 \cdots k$: Q_i is anomaly-free (proved in section 5.1)
2. Given the similarity measures used in the clustering process [1], the probabilities of existence of inter-clusters anomalies (i.e. anomalies between rules of different clusters) are much lower than the probabilities of existence of intra-cluster anomalies (i.e. anomalies between rules in the same cluster). In other words, the probability of not detecting anomalies due to clustering is very small.
3. Consider the case of a rule R of P that is element of two clusters P_i and P_j. If R is not modified or removed in the computations of Q_i and Q_j, R will be element of Q_i and Q_j. However, the grouping process of all the Q_i to obtain Q inserts R only once in Q, and hence avoids to create a new redundancy in Q.
4. From 1, 2 and 3, Q satisfies C_3 with high probability.

- **Proof of C_4**

We have proved that for any request rq: if P_i has no rule permitting rq, then Q_i has no rule permitting rq. Consider an access request rq, and assume that P has no rule permitting rq.

1. $P = \{P_1, P_2, \cdots, P_k\}$, from the fact that P has no rule permitting rq, we deduce that for each $i = 1 \cdots k$: P_i has no rule permitting rq.
2. For each $i = 1 \cdots k$: if P_i has no rule permitting rq, then Q_i has no rule permitting rq.
3. From 1 and 2, we deduce that for each $i = 1 \cdots k$: Q_i has no rule permitting rq.

From the fact that $Q = \{Q_1, Q_2, \cdots, Q_k\}$ and the fact that for each $i = 1 \cdots k$: Q_i has no rule permitting rq, we deduce that Q has no rule permitting rq.

- **Proof of C_5**

The proof of C_5 is obtained from the proof of C_4, by just replacing the word "permitting" by "denying".

5.3. Complexity of Algorithm 1

We use the following notation:

- n is the size of the policy P, i.e. its number of rules,
- m is the number of attributes used to define the rules of P,
- d is the size of the access domain of P; it is the product of the domain sizes of all attributes, i.e. $d = \prod_{i=1}^{m} |V_{att_i}|$.

We have seen in section 5.1 that in each of its iterations, the algorithm is in one case among three: case a, case b, or case c. The best situation in terms of execution time occurs when we first have successively uniquely case b (redundancy), because it removes one node and one or more edges, while case a removes one edge and no node. The worst situation in terms of execution time occurs when we first have successively uniquely case c, because it increases the number of nodes and edges. Hence, the worst case scenario that leads to a graph without edges should be to have two phases: phase 1 consists of successive iterations of case c, and phase 2 consists of successive iterations of case a.

Complexity of phase 1 (successive iterations of case c):

1. Splitting a set of x elements $x - 1$ times results in x singletons.
2. The maximum size of the access domain of any rule of P is d.
3. From 1 and 2, the number of times where a given rule of P is split due to case c, is upper-bounded by $d - 1$.
4. From 3, if we consider the n rules of P, we obtain that the number of iterations of case c is upper bounded by $n \times (d - 1)$.
5. From 4 and the fact that case c increments the number of rules by at most 1, after a series of $n \times (d - 1)$ cases c, we obtain a policy whose number of rules is upper-bounded by $n \times d$.
6. From 5, after a series of $n \times d$ cases c, we obtain a policy whose number of edges is upper-bounded by $\frac{(n \times d) \times (n \times d - 1)}{2}$ which is in $O(n^2 \times d^2)$.
7. The treatment of each iteration of case c consists mainly in:

   - Computing the intersection of the access domains of two rules, which is in $O(d^2)$ from 2.

- Adding edges to other nodes, which is in $O(n \times d)$ from 5.

8. From 4 and 7, the complexity of phase 1 (i.e. the treatment for all iterations of case c) is in $O(n \times d(d^2 + n \times d)) = O(n \times d^3) + O(n^2 \times d^2) = O(n \times d^2(d + n)) = O(n \times d^2 max(d, n))$.

Example 5.1. Consider a policy P with 3 rules. Each rule has two attributes att_1 and att_2, such that $att_1 \in \{v_1, v_2, v_3\}$ and $att_2 \in \{v_4, v_5, v_6, v_7\}$. The access domain size of P is $d = \prod_{i=1}^{2} |V_{att_i}| = 3 \times 4 = 12$. The maximum number of rules that can be obtained after a series of case c is $n \times d = 3 \times 12 = 36$ rules (rules with singleton access domains).

Complexity of phase 2 (successive iterations of case a):

1. We have seen that after a series of case c of length $n \times (d - 1)$, we have a graph whose number of edges is in $O(n^2 \times d^2)$.
2. The effect of case a is to remove an edge.
3. From 1 and 2, the number of iterations of case a is in $O(n^2 \times d^2)$.
4. The treatment of each iteration consists mainly in computing the intersection of the access domains of two rules, which is in $O(1)$, because after the series of case c in phase 1, the ADs of the rules are singletons.
5. From 3 and 4, the complexity of phase 2 (i.e. the treatment for all iterations of case a) is in $O(n^2 \times d^2)$.

Total complexity (phase 1 + phase 2):

Since the complexity of phase 1 is greater than the complexity of phase 2, the former is the order of the total complexity : $O(n \times d^2(max(d, n))$. We assume that attributes are not constant parameters, i.e. $|V_{att_i}| \geq 2$, for every $i = 1 \cdots m$, which implies $d \geq 2^m$. Therefore, the number of attributes has an exponential effect on the complexity.

6. Experimental Results

To evaluate the suggested approach, we have implemented our method of anomaly detection and resolution in Algorithm 1 with Java programming language in the experimental environment indicated in Table 1. We have applied our method to several examples of important sizes. We construct a set of ABAC policies (synthetic datasets), composed of the combination of eight subject attributes, four resource attributes and two environment attributes. An evaluation on real dataset would be preferable, however no benchmark has been published in this area, and real medical data are hard to obtain because of confidentiality constraints. We have generated ABAC policies of up to 15000 rules.

Table 1. Experimental Environment

OS	bits	Memory	CPU
macOS High Sierra version 10.13.3	64	8G	Intel(R) Core(TM) i5-5257U CPU @ 2.70GHz

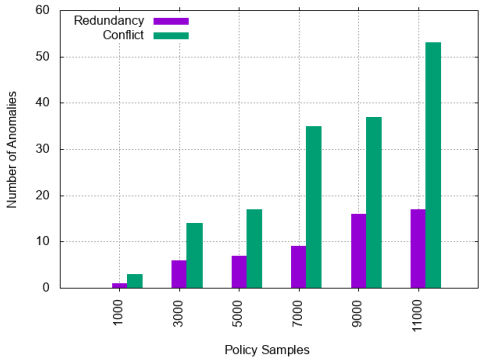

Figure 2. The number of anomalies in the generated policies

Figure 2 illustrates how the numbers of redundancies and conflicts in the generated policies increase proportionally with the policy size.

To analyze our results, we have considered the criterion of total execution time. More precisely, we have analyzed how the running time is influenced by the following parameters:

- n: the size of the considered ABAC policy
- The threshold value (see Rule clustering in section 3.2)
- d: the size of access domain

As indicated in section 3.2, our method consists of three steps: 1) rule extraction, 2) rule clustering, and 3) anomalies detection and resolution. The latter step consists in executing algorithm 1 in each cluster. Figure 3 shows the sum of running times of algorithm 1 for all the clusters (for $d = 16$). Figure 4 shows the total running time, i.e. the time of figure 3 plus the times of steps 1 and 2 (for $threshold = 0.8$ and $d = 16$). The curves explicitly show that the running time increases with the number of policy rules. As depicted in figures 3 and 4, the running time of algorithm 1 represents less than 7% of the total running time. This percentage is due to the fact that the time required for the first two steps (i.e., extraction and clustering) represents more than 90% of the total running time (higher than the time required of the last step). The time complexity of rule extraction is in $O(n \times d)$. The time complexity of rule clustering is in $O(n^2 \times d^2)$, n^2 is due to the fact that in a set of n rules, we have to verify $\frac{n \times (n-1)}{2}$ pairs of rules. d^2 is due to the fact that when verifying a pair

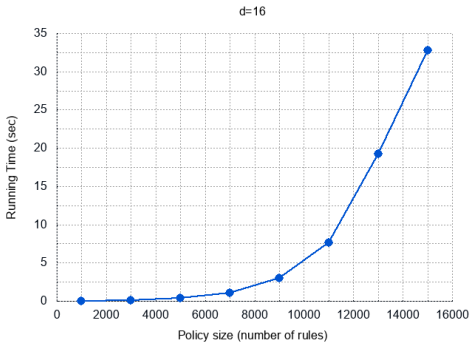

Figure 3. Running time of Algorithm 1

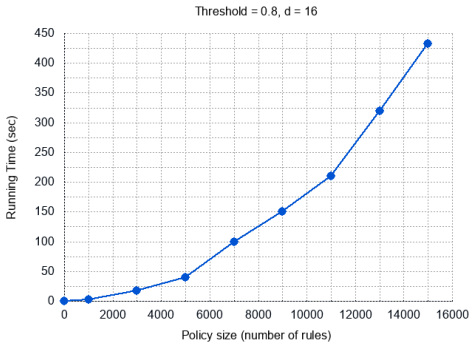

Figure 4. Total running time

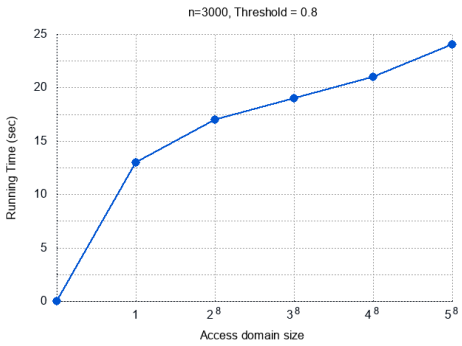

Figure 5. Running time vs. Access Domain

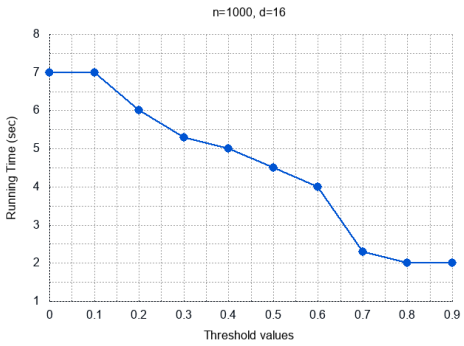

Figure 6. Running time vs. threshold

of rules r_i and r_j in order to classify them, we compute their similarity score, this latter is mainly based on comparing access domain of r_i and r_j. Therefore, the time complexity of steps 1 and 2 is in $O(n^2 \times d^2)$. Anomalies detection and resolution is executed in each cluster, where the number of rules in each cluster is less than n (the time complexity of step 3 is given in subsection 5.3). Moreover, a cluster may contain only one rule, thus the time required for anomalies detection and resolution in that cluster is 0. Therefore, the time required for step 3 is less than the time required for steps 1 and 2.

Figure 5 shows the total running time as a function of the size of access domain $d = \prod_{i=1}^{m} |V_{att_i}|$ (for n = 3000 and $threshold$ = 0.8). Each rule is composed of eight attributes (i.e., m = 8), and for each attribute we considered $|V_{att_i}|$= 1, 2, 3, 4 and 5. The obtained curve demonstrates the impact of the access domain sizes (i.e., 1, 2^8, 3^8, 4^8 and 5^8) on the performance (running time). The obtained results are justified in subsection 5.3 by an evaluation of the time complexity.

The threshold used in the clustering algorithm influences the result of clustering [21, 22] (i.e. the number of clusters) which in turn influences the results of the execution time of algorithm 1 (in all the clusters). Figure 6 shows the total running time based on different thresholds for the same policy (for n = 1000 and d = 16). The obtained curve demonstrates the impact of the threshold values (i.e., 0, 0.5, 0.6, 0.7, 0.8 and 0.9.) on the performance (running time). The obtained results can be explained by the fact that when the *threshold* decreases, the sizes of the obtained clusters increase, and hence the running time also increases. In the extreme case where *threshold* = 0 (i.e., similar to applying our method on the whole set of rules without clustering), we obtain the worst running time. These results demonstrate the time gained from using clustering. The threshold impact becomes negligible from the value 0.7. On the average, the best running time is obtained from the threshold 0.8. Thus, the default value of the selected threshold for our experiments is set to 0.8.

7. Related Work and Contributions

Attribute-based access control (ABAC) policies support fine-grained access control. Therefore, they are more flexible in governing the access to information and resources in a variety of applications including web services [20, 33], Cloud Computing [10, 28, 31] collaborative environment [23, 32], Internet Of Things [39–41] and so on. Attribute-based policies regulate users requests based on set of conditions related to the requestors and the demanded resources. However, ABAC policies are often complex and conflict prone.

In an ABAC policy, multiple rules may overlap, which means one access request may match several rules with the same effect. Moreover, multiple rules, with conflicting access decisions, may match the same request. These kinds of anomalies may lead to both: safety problems (allowing unauthorized accesses) and availability problems (denying an access in emergencies). Therefore, detecting and resolving anomalies is an important aspect of dealing with ABAC policies.

Khoumsi et al. [11] categorize the anomalies into two categories: a *conflicting anomaly* and an *non-conflicting anomaly*. The first category occurs when a request matches several rules that have different actions (conflicts). Whereas, the second occurs when the same request matches several rules that have the same action (redundancies). On the other hand, Moffett et al. [13] have defined *conflict* by three synonyms: *difference*, *disagreement* and *opposition*. Where they have categorized a conflict into: conflict of modality, conflict between imperative and authority policy, conflict of duties and conflict of priorities.

eXtensible Access Control Markup Language (XACML) [3] is the most convenient way to express ABAC policies. XACML defines an XML schema that supports the ABAC model. In fact, an XACML policy in distributed applications may be aggregated from multiple parties and can be managed by more than one administrator [8] which may arise anomalies between rules. Various research efforts have been devoted to anomaly detection of XACML policies using verification techniques [8, 15, 16, 18, 29, 34]. The most important policy analysis techniques and formal approaches are presented by [37]. For instance, Mourad et al. [15] use the Unified Modelling Language (UML) to detect conflicting and redundant rules prior to their enforcement in the system. Ramli et al. [16] uses Answer Set Programming (ASP) to detect incompleteness, conflicting and unreachability XACML Policies. Although, this approach has some limitation in modeling XACML dealing with types of attributes which do not belong to Ansprolog, such as strings. Martin et al. [18] encode the policy rules in Coq [5] with two fields, The first field is the rule effect and the second field combines the four elements of XACML: "subject-resource-action-condition" referred as *srac*. If two rules have identical *srac* with different effects, a conflict is detected. Otherwise, if the effects are similar, then a redundancy in detected. On the other hand, [8, 37] consider representing XACML policies as decision trees to detect and resolve conflicts and redundancies. Another representation of XACML policies was proposed by [19]. it represents the XACML using Prolog which uses constraint logic programming techniques (CLP), which are well-adapted to hierarchical XACML policy logic and avoid pair-wise comparisons altogether by taking advantage of Prolog's built-in powerful indexing system. In addition, the authors in [34] consider SAT modulo theories (SMT) [27] as the underlying reasoning method for the analysis of XACML policies.

The resolution of anomalies was already handled by XACML itself. in fact, XACML offers a set of *Rule Combining Algorithms* (RCA) to overcome the issue of conflicting rules: Deny-Overrides, Permit-Overrides, First Applicable and Only-One-Applicable. For instance, deny-overrides returns deny if one of the conflicting policies evaluates to deny. Otherwise, the result is permit [30]. However, The RCAs need to be defined manually and at a priory stages by the policy administration. Moreover, only one RCA can be applied to all kinds of detected anomalies. Therefore, this technique remains static and can not be applied to distributed and dynamic systems. Therefore, several research efforts [9, 12, 14] have been addressed for dynamic anomaly resolution strategies. For instance, Kagal et al. [9] have considered the low priority technique to resolve the conflict, i.e. negative authorizations are allowed. When Matteucci et al. [12] have proposed a strategy for policy conflict resolution based on multi-criteria decision. Where the decision is taken based on some calculations of multiple criteria retrieved from the policies' attributes and represented in a matrix. In addition, Bauer et al. [4] adopted a data-mining technique to remove inconsistencies occurring between access control policies and user's intentions. In contrast, our proposed method detects and resolves anomalies within ABAC policies caused by overlapping relations (i.e., the intersection of access domains).

With respect to the solutions proposed in the papers mentioned above, our approach aims at defining a generic strategy for anomaly detection and resolution. First of all, our approach to detect and resolve anomalies within ABAC policies takes into account a large set of rules and attributes. To make the suggested method scalable with the huge number of rules, it proposes decomposing the policy into clusters of rules. Where the anomaly detection and resolution method is performed in each cluster of rules, instead of the whole policy set, which implies less processing time. The proposed approach is mainly based on the concept of rule access domain. The main advantage of the suggested approach is guaranteeing that the semantics of the original policy is preserved (the proof is presented in section 5). This is done by decomposing a given rule into access domains, based on this technique, we identify accurately the domain of conflict. Therefore, apply the resolution only to that domain by rewriting the conflicting rules. Furthermore, we consider two types of resolution strategies: restrictive resolution, where we permit less actions and deny more actions;

and permissive resolution, where we permit more actions and deny less actions.

8. Conclusion

We have presented a formal method that detects and resolves anomalies in ABAC policies. The suggested method uses a concept called access domain, which is used to accurately identify and resolve effectively policy anomalies. To make the suggested method flexible with scalable policies, the proposed approach is preceded by rules extraction and rules clustering. Where the policy is decomposed into several clusters of rules, and then the method is applied to each cluster. Besides, we consider two types of resolutions, permissive resolution and restrictive resolution. An important advantage of the suggested approach is guaranteeing that the semantics of the original policy is preserved. We have proved the correctness of the method and evaluate its computational complexity. Furthermore, we have proposed the method while providing an algorithm that was implemented and experimented.

As future work, we already started implementing a parallel version of the proposed approach using *MapReduce* technique, in order to improve the running time. We also aim to conduct a real case study and automate the procedure through lessons learned.

References

[1] M. Ait El Hadj, M. Ayache, Y. Benkaouz, A. Khoumsi, and M. Erradi (ICETE 2017) Clustering-based Approach for Anomaly Detection in XACML Policies. the 14th International Joint Conference on e-Business and Telecommunication. Volume 4: SECRYPT. pp.548-553.

[2] El Hadj, M.A., Benkaouz, Y., Khoumsi, A. and Erradi, M., 2017, December. Access Domain-Based Approach for Anomaly Detection and Resolution in XACML Policies. In International Conference on Innovations in Bio-Inspired Computing and Applications (pp. 298-308). Springer, Cham.

[3] Anderson, A., Nadalin, A., Parducci, B., Engovatov, D., Lockhart, H., Kudo, M., Humenn, P., Godik, S., Anderson, S., Crocker, S. and Moses, T., 2003. Extensible Access Control Markup Language (XACML) version 1.0. OASIS.

[4] Bauer, L., Garriss, S. and Reiter, M.K., 2011. Detecting and resolving policy misconfigurations in access-control systems. ACM Transactions on Information and System Security (TISSEC), 14(1), p.2.

[5] Blanqui, F., 2013. Introduction to the coq proof assistant. Lecture notes available on https://who. rocq. inria. fr/Frederic. Blanqui.

[6] Bonatti, P.A. and Samarati, P., 2002. A uniform framework for regulating service access and information release on the web. Journal of Computer Security, 10(3), pp.241-271.

[7] Chandola, V., Banerjee, A. and Kumar, V., 2009. Anomaly detection: A survey. ACM computing surveys (CSUR), 41(3), p.15.

[8] Hu, H., Ahn, G.J. and Kulkarni, K., 2013. Discovery and resolution of anomalies in web access control policies. IEEE transactions on dependable and secure computing, p.1.

[9] Kagal, L., Finin, T. and Joshi, A., 2003, October. A policy based approach to security for the semantic web. In International semantic web conference (pp. 402-418). Springer, Berlin, Heidelberg.

[10] Khan, A.R., 2012. Access control in cloud computing environment. ARPN Journal of Engineering and Applied Sciences, 7(5), pp.613-615.

[11] Khoumsi, A., Erradi, M. and Krombi, W., 2016. A formal basis for the design and analysis of firewall security policies. Journal of King Saud University-Computer and Information Sciences.

[12] Matteucci, I., Mori, P. and Petrocchi, M., 2013. Prioritized execution of privacy policies. In Data Privacy Management and Autonomous Spontaneous Security (pp. 133-145). Springer, Berlin, Heidelberg.

[13] Moffett, J.D. and Sloman, M.S., 1994. Policy conflict analysis in distributed system management. Journal of Organizational Computing and Electronic Commerce, 4(1), pp.1-22.

[14] Mohan, A. and Blough, D.M., 2010, April. An attribute-based authorization policy framework with dynamic conflict resolution. In Proceedings of the 9th Symposium on Identity and Trust on the Internet (pp. 37-50). ACM.

[15] Mourad, A., Tout, H., Talhi, C., Otrok, H. and Yahyaoui, H., 2016. From model-driven specification to design-level set-based analysis of XACML policies. Computers & Electrical Engineering, 52, pp.65-79.

[16] Ramli, C.D.P.K., 2015. Detecting incompleteness, conflicting and unreachability xacml policies using answer set programming. arXiv preprint arXiv:1503.02732.

[17] Shu, C.C., Yang, E.Y. and Arenas, A.E., 2009, July. Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In IEEE International Symposium on Policies for Distributed Systems and Networks (pp. 182-185). IEEE.

[18] St-Martin, M. and Felty, A.P., 2016, January. A verified algorithm for detecting conflicts in XACML access control rules. In Proceedings of the 5th ACM SIGPLAN Conference on Certified Programs and Proofs (pp. 166-175). ACM.

[19] Stepien, B. and Felty, A., 2016, August. Using Expert Systems to Statically Detect" Dynamic" Conflicts in XACML. In 2016 11th International Conference on Availability, Reliability and Security (ARES) (pp. 127-136). IEEE.

[20] Yuan, E. and Tong, J., 2005, July. Attributed based access control (ABAC) for web services. In Web Services, 2005. ICWS 2005. Proceedings. 2005 IEEE International Conference on. IEEE.

[21] Lin, D., Rao, P., Ferrini, R., Bertino, E. and Lobo, J., 2013. A similarity measure for comparing XACML policies. IEEE Transactions on Knowledge and Data Engineering, 25(9), pp.1946-1959.

[22] Guo, S., 2014. Analysis and evaluation of similarity metrics in collaborative filtering recommender system.

[23] Jha, S., Sural, S., Atluri, V. and Vaidya, J., 2016, November. An administrative model for collaborative

management of abac systems and its security analysis. In Collaboration and Internet Computing (CIC), 2016 IEEE 2nd International Conference on (pp. 64-73). IEEE.

[24] Samarati, P. and de Vimercati, S.C., 2000, September. Access control: Policies, models, and mechanisms. In International School on Foundations of Security Analysis and Design (pp. 137-196). Springer, Berlin, Heidelberg.

[25] Yang, P., Gofman, M.I., Stoller, S.D. and Yang, Z., 2015. Policy analysis for administrative role based access control without separate administration. Journal of Computer Security, 23(1), pp.1-29.

[26] Hu, H., Ahn, G.J. and Kulkarni, K., 2011, June. Anomaly discovery and resolution in web access control policies. In Proceedings of the 16th ACM symposium on Access control models and technologies (pp. 165-174). ACM.

[27] Barrett, C. and Tinelli, C., 2018. Satisfiability modulo theories. In Handbook of Model Checking (pp. 305-343). Springer, Cham.

[28] Younis, Y.A., Kifayat, K. and Merabti, M., 2014. An access control model for cloud computing. Journal of Information Security and Applications, 19(1), pp.45-60.

[29] Zhang, A., Ji, C., Bao, Y. and Li, X., 2017. Conflict analysis and detection based on model checking for spatial access control policy. Tsinghua Science and Technology, 22(5), pp.478-488.

[30] Moses, T., 2005. Extensible access control markup language (xacml) version 2.0. Oasis Standard.

[31] Riad, K., Yan, Z., Hu, H. and Ahn, G.J., 2015, October. AR-ABAC: a new attribute based access control model supporting attribute-rules for cloud computing. In 2015 IEEE Conference on Collaboration and Internet Computing (CIC) (pp. 28-35). IEEE.

[32] John, J.C., Sural, S. and Gupta, A., 2016, July. Authorization Management in Multi-cloud Collaboration Using Attribute-Based Access Control. In Parallel and Distributed Computing (ISPDC), 2016 15th International Symposium on (pp. 190-195). IEEE.

[33] Hemdi, M. and Deters, R., 2016, October. Using REST based protocol to enable ABAC within IoT systems. In Information Technology, Electronics and Mobile Communication Conference (IEMCON), 2016 IEEE 7th Annual (pp. 1-7). IEEE.

[34] Turkmen, F., den Hartog, J., Ranise, S. and Zannone, N., 2015, April. Analysis of XACML policies with SMT. In International Conference on Principles of Security and Trust (pp. 115-134). Springer, Berlin, Heidelberg.

[35] Chandola, V., Banerjee, A. and Kumar, V., 2012. Anomaly detection for discrete sequences: A survey. IEEE Transactions on Knowledge and Data Engineering, 24(5), pp.823-839.

[36] Karp, A.H., Haury, H. and Davis, M.H., 2009. From ABAC to ZBAC: the evolution of access control models. Hewlett-Packard Development Company, LP, 21.

[37] Shaikh, R.A., Adi, K. and Logrippo, L., 2017. A data classification method for inconsistency and incompleteness detection in access control policy sets. International Journal of Information Security, 16(1), pp.91-113.

[38] Karafili, E., Pipes, S. and Lupu, E.C., 2017, August. Verification techniques for policy based systems. In 2017 IEEE SmartWorld, Ubiquitous Intelligence & Computing, Advanced & Trusted Computed, Scalable Computing & Communications, Cloud & Big Data Computing, Internet of People and Smart City Innovation (SmartWorld/SCALCOM/UIC/ATC/CBDCom/IOP/SCI) (pp. 1-6). IEEE.

[39] Qiu, J., Du, C., Su, S., Zuo, Q. and Tian, Z., 2018. A Survey on Access Control in the Age of IoT.

[40] Neto, A.L.M., Pereira, Y.L., Souza, A.L., Cunha, I. and Oliveira, L.B., 2018, April. Attributed-based authentication and access control for IoT home devices: demo abstract. In Proceedings of the 17th ACM/IEEE International Conference on Information Processing in Sensor Networks (pp. 112-113). IEEE Press.

[41] Bezawada, B., Haefner, K. and Ray, I., 2018, March. Securing Home IoT Environments with Attribute-Based Access Control. In Proceedings of the Third ACM Workshop on Attribute-Based Access Control (pp. 43-53). ACM.