# Classification-Based Anomaly Prediction in XACML Policies

Maryam Davari(✉) and Mohammad Zulkernine

School of Computing, Queen's University, Kingston, Canada
{maryam.davari,mz}@queensu.ca

**Abstract.** XACML (eXtensible Access Control Markup Language) has gained significant interest as a standard to define Attribute-Based Access Control (ABAC) policies for different applications, especially web services. XACML policies have become more complex and difficult to administer in distributed systems, which increases the chance of anomalies (redundancy, inconsistency, irrelevancy, and incompleteness). Due to the lack of effective analysis mechanisms and tools, anomaly detection and resolution are challenging, particularly in large and complex policy sets. In this paper, we learn the characteristics of various types of anomalies to predict anomaly types of unseen policy rules with the help of data classification techniques. The effectiveness of our approach in predicting policy anomalies has been demonstrated through experimental evaluation. The discovered correlations between the anomaly types and the number of subject and resource attribute expressions can help system administrators improve the security and efficiency of XACML policies.

**Keywords:** Access control policies · XACML · ABAC · Policy anomalies · Classification-based anomaly prediction · Security

## 1 Introduction

Access control policies have been used to secure and control resource sharing in distinct applications such as web services (e.g., [26]), grid systems (e.g., [24]), and database federations (e.g., [4]). In recent years, Attribute-Based Access Control (ABAC) [7] policies have gained popularity in open distributed environments [21]. ABAC defines permissions based on attributes that can be any information describing subjects, resources, and environments, rather than their identities. ABAC access control policies are specified by XACML (eXtensible Access Control Markup Language) [22], which is a general-purpose access control policy language. XACML has been utilized in a variety of applications ranging from healthcare to transportation [1].

Due to the sophisticated expressiveness and increasing size of XACML policies, the consequences and effects of developed policies are not obvious to policy administrators. Some anomalies such as redundancy, inconsistency, irrelevancy, and incompleteness may arise in developing access control policies when there are


F. Li et al. (Eds.): SecureComm 2022, LNICST 462, pp. 3–19, 2023.
https://doi.org/10.1007/978-3-031-25538-0_1

not enough mechanisms to analyze a large number of policies. One of the issues with ABAC policies is redundancy detection and removal. When the number of policies to be parsed affects the response time of access requests, redundancy has to be addressed to avoid processing of unnecessary policies. In a policy set, multiple policies may conflict with each other which is referred to as inconsistency. Inconsistent policies overlap and yield different decisions. Inconsistency detection can mitigate conflict resolution activities. Irrelevancy occurs when access control policies are not suitable for any user's access requests. Irrelevancy detection can help policy administrators eliminate unused policies and make policy maintenance easier. Incompleteness refers to a situation when the current access control policies cannot cover an access request. By detecting this anomaly, some security issues (e.g., mistakenly allowing access to intruders) can be avoided. However, anomaly detection in XACML policies is complicated due to the fact that XACML policies may be aggregated by various parties and maintained by multiple administrators.

In this paper, we propose a data classification-based approach to predict anomaly types (redundancy, inconsistency, irrelevancy, and incompleteness) in XACML policies. To the authors' knowledge, there are no XACML policy rule sets that include these four types of anomalies. Therefore, the proposed approach begins by building XACML policy rules. The rules are then clustered based on their similarities and policy anomaly detection technique [3] is applied to each cluster. Various data classification techniques (e.g., Random Forest, Decision Tree) are then trained on these rules to discover the behavior of anomalies and predict anomaly types of unseen rules. The experimental results show that the classification-based approach is effective in predicting anomalies in large rule sets. Furthermore, some correlations among the number of subject and resource attribute expressions, rule sizes, and anomaly types are found. These insights can assist system administrators to make XACML policies more secure and efficient.

The major contributions of this paper can be summarized as follows:

- The design and implementation of XACML policy anomaly prediction using classification techniques.
- The anomaly (redundancy, inconsistency, irrelevancy, and incompleteness) formalization for XACML policies.
- The discovery of correlations among anomaly types, rule sizes, and rule attributes.

The rest of the paper is organized as follows: Sect. 2 presents background information on XACML. Section 3 describes the XACML policy anomaly prediction approach, which includes the XACML policy analysis and policy learning procedure for anomaly prediction. In Sect. 4, the experiments are discussed and the findings are analyzed. The related work, conclusion, and future work are presented in Sects. 5 and 6, respectively.

## 2 Overview of XACML

In this section, we provide background information about XACML policies [22]. Access control policy specification and formalization in the XACML language

have four parts: attributes and functions, rules, policies, and policy sets. XACML policies are centered around attributes and functions that represent the characteristics of subjects, resources, actions, and environments. A policy rule is made up of three parts: effect (indicating whether access will be permitted or denied), Boolean condition (specifying when the rule applies to an access request), and target (grouping subjects, resources, and actions).

A policy consists of a target, a set of rules, and a rule combining algorithm. A combining algorithm computes a decision when a policy has rules with conflicting effects such as *Deny-Overrides* (i.e., if any rule evaluates to "Deny", the final decision is "Deny"), *Permit-Overrides* (i.e., if any rule evaluates to "Permit", the final decision is "Permit"), and *First-Applicable* (i.e., the effect of the first rule that applies is the decision of the policy). A policy set is defined by a target, a set of policies, and a policy combining algorithm (which is the same as the rule combining algorithm). We provide formal definitions of XACML policies [27] in the following paragraphs.

**Definition 1** Rule. A rule $Ru = (S, R, A, C, E)$ specifies a set of subjects $S$ (containing a set of subject attributes) can perform a set of actions $A$ $(a_1, a_2, \cdots, a_m)$ over a set of resources $R$ (consisting of a set of resource attributes) by effect $E$ under condition $C$. A policy $P = \{Ru_1, Ru_2, ..., Ru_m\}$ contains a set of rules $Ru_1, Ru_2, ..., Ru_m$.

**Definition 2** Subject Attribute. Attributes describe the characteristics of a subject. Let $S$ be a finite set of subjects and $Att_s$ is a finite set of subject attributes. The value of attribute $a \in Att_s$ for subject $s \in S$ is represented by the function $d_s(a, s)$. Some subject attributes have just one value, while others have multiple values. Single value attributes ($Att_{s,1}$) have a unique value for each subject (e.g., subject id), and multiple value attributes ($Att_{s,m}$) are a set of single values (e.g., courses).

**Definition 3** Resource Attribute. Attributes that describe the characteristics of resources. Let $R$ and $Att_r$ be finite sets of resources and resource attributes, respectively. The value of attribute $a \in Att_r$ for resource $r \in R$ is represented by function $d_r(a, r)$.

**Definition 4** Attribute Expression. An attribute expression contains a set of attributes, operators, and value tuples ($att\ op\ val$) (e.g., $security\_level > 10$). The operators we consider in this paper are $\{\leq, <, =, >, \geq\}$. A subject attribute expression ($e_S$) is a function $e$ that for each subject attribute $a \in Att_s$, $e_S(a)$ is either special value $\perp$ (that indicates there is no constraint on the value of attribute $a$) or a set of possible values. Similarly, resource attribute expression ($e_R$) is a function for resource attributes.

**Definition 5** Access Request. An access request is represented as a tuple $(S, R, A, C)$, with $S$ containing a finite set of subject attribute-value pairs $(att_{s1} = val1)$, $(att_{s2} = val2)$, $\cdots$, $(att_{sn} = valn)$. Similarly, $R$ is a finite set of resource attribute-value pairings $(att_{r1} = val1)$, $(att_{r2} = val2)$, $\cdots$, $(att_{rm} = valm)$. $A$ is the request action, and $C$ is a set of conditions.

**Definition 6** Any Value. If subject attributes or resource attributes of a rule are not specified, it signifies that they do not impose any constraints on attribute values, which we indicate with $att = *$.

In this paper, analysis and prediction are performed at the rule level, and the influence of combining algorithms is ignored. When the number of rules increases, the consequence of using the conflicting algorithms to override access control decisions can be unpredictable [21]. These algorithms can handle conflicts in incoming requests, but they were not developed to detect conflicts in policies.

**Table 1.** Sample policy rules.

| Rule | Subject | Resource | Action | Condition | Effect |
|---|---|---|---|---|---|
| $Ru_1$ | *adminRole= accountant* | *type=budget* | *update* | *s.project=r.project* | *Permit* |
| $Ru_2$ | *isEmployee=true* | *type=task; proprietary=false* | *request* | *s.expertise=r.expertise; s.project=r.project* | *Permit* |
| $Ru_3$ | *adminRole= accountant, planner* | *type=budget* | *read, update* | *s.project=r.project* | *Permit* |
| $Ru_4$ | *isEmployee=true* | *type=task, proprietary=false* | *request* | *s.project=r.project* | *Deny* |
| $Ru_5$ | *isEmployee=true* | *type=schedule* | *update* | *s.project=r.project* | *Deny* |
| $Ru_6$ | *adminRole=planner* | *type=budget* | *request* | *s.project=r.project* | *Permit* |

## 3 XACML Policy Anomaly Prediction

The classification-based anomaly prediction aims to learn the characteristics of rules with anomalies based on historical results and predict whether new rules are normal or anomalous. We regard the mapping between policy rules and anomaly types as a function of $f : x \rightarrow y$ in machine learning contexts, where $x$ is a rule and $y$ is a type of rule. $y$ indicates whether the rule is normal (i.e., has no anomaly) or anomalous (i.e., has an anomaly) in the binary classification. In the multi-class classification, $y$ can be normal or an anomaly type (redundancy, static and dynamic inconsistency, irrelevancy, and incompleteness). The main goal is to learn function $f$ to predict the type of an unseen rule. We provide XACML policy anomaly definitions in Sect. 3.1. Then, XACML policy rules are clustered and analyzed to identify the types of rules in Sect. 3.2. In the following section, we present a policy learning procedure (including data pre-processing and data-classification) that is required for anomaly prediction.

### 3.1 XACML Policy Anomaly Definitions

We begin by defining different types of anomalies for XACML policy rules. For further demonstration, we use some rules focusing on project management from [27] that are listed in Table 1.

**Definition 7** Redundancy (RED). Redundancy indicates similarities among rules. Attributes of two rules with the same identifiers have intersecting values. Detecting and removing redundancies can improve policy evaluation performance. Rule $Ru_j$ is redundant if and only if

- $\exists\, Ru_i \in ACP$.
- $\forall\, a \in Ru_j.e_S, \exists\, a' \in Ru_i.e_S, Att_s(a) = Att_s(a') \rightarrow a \cap a' \neq \varnothing \wedge$ $\forall\, b \in Ru_j.e_R, \exists\, b' \in Ru_i.e_R, Att_r(b) = Att_r(b') \rightarrow b \cap b' \neq \varnothing \wedge Ru_i.A \cap Ru_j.A \neq \varnothing \wedge Ru_i.E = Ru_j.E$.

For example, rule $Ru_3$ in Table 1 specifies that an accountant and a planner assigned to a project can read and update the budget. Rule $Ru_1$ indicates that an accountant assigned to a project can update the project budget. As a result, rule $Ru_1$ is redundant in comparison to rule $Ru_3$.

**Definition 8** Inconsistency (INCON). Inconsistency can be divided into two categories: static and dynamic.

**Definition 8-1** Static inconsistency (SINCON). Static inconsistency refers to a situation when there are at least two similar rules (i.e., two rules with the same attribute identifiers have intersecting values) in the policy set that conflict with each other. Consider rules $Ru_i$ and $Ru_j$. These two rules are statically inconsistent if and only if

- $\forall\, a \in Ru_i.e_S, \exists\, a' \in Ru_j.e_S, Att_s(a) = Att_s(a') \rightarrow a \cap a' \neq \varnothing \wedge \forall\, b \in Ru_i.e_R, \exists\, b' \in Ru_j.e_R, Att_r(b) = Att_r(b') \rightarrow b \cap b' \neq \varnothing \wedge Ru_i.A \cap Ru_j.A \neq \varnothing \wedge Ru_i.E \neq Ru_j.E$.

For example, rule $Ru_2$ specifies that an employee working on a project can request to work on a non-proprietary task whose required areas of expertise are among the employee's areas of expertise, while rule $Ru_4$ specifies that an employee working on a project cannot request to work on the non-proprietary task. Rules $Ru_2$ and $Ru_4$ are statically inconsistent.

**Definition 8-2** Dynamic inconsistency (DINCON). Dynamic inconsistency refers to a situation when an incoming access request triggers at least two rules with conflicting decisions. The dynamic inconsistency relies on access requests and occurs at runtime. Consider rules $Ru_m$ and $Ru_n$. These two rules are dynamically inconsistent with respect to request $req$ if and only if

- $\exists\, req = (S', R', A')$.

– $\exists\, Ru_m \in ACP \wedge Ru_n \in ACP \,|\, (\exists\, a \in S', \exists\, a' \in Ru_m.e_S,\, Att_s(a) = Att_s(a') \rightarrow a \cap a' \neq \varnothing) \wedge (\exists\, a''' \in S' \wedge \exists\, a'' \in Ru_n.e_S,\, Att_s(a''') = Att_s(a'') \rightarrow a''' \cap a'' \neq \varnothing) \wedge (\exists\, b \in R', \exists\, b' \in Ru_m.e_R,\, Att_r(b) = Att_r(b') \rightarrow b \cap b' \neq \varnothing) \wedge (\exists\, b''' \in R', \exists\, b'' \in Ru_n.e_R,\, Att_r(b''') = Att_r(b'') \rightarrow b''' \cap b'' \neq \varnothing) \wedge A' \cap Ru_m.A \neq \varnothing \wedge A' \cap Ru_n.A \neq \varnothing \wedge Ru_m.E \neq Ru_n.E.$

For example, when an incoming request is

*<s.adminRole=accountant, s.isEmployee=true; r.type={budget, schedule}; s.project=r.project; action =update>*

dynamic inconsistency occurs. Both rules $Ru_1$ and $Ru_5$ satisfy the conditions, while they have conflicting effects. Therefore, rules $Ru_1$ and $Ru_5$ are dynamically inconsistent.

**Definition 9** Irrelevancy (IRR). Irrelevancy refers to a scenario where a rule is never triggered for any kind of access request. A rule is irrelevant if and only if

– $\exists\, Ru \in ACP.$
– $\nexists\, req = (S', R', A') \,|\, \forall\, a \in Ru.e_S, \exists\, a' \in S',\, Att_s(a) = Att_s(a') \rightarrow a \cap a' = a \wedge \forall\, b \in Ru.e_R, \exists\, b' \in Ru',\, Att_r(b) = Att_r(b') \rightarrow b \cap b' = b \wedge A' \cap Ru.A \neq \varnothing.$

As an example, rule $Ru_6$ specifies that a planner assigned to a project can request to get information about the project budget. However, according to rule $Ru_3$, an accountant and a planner assigned to a project can read and update the budget without sending the request. As a result, rule $Ru_6$ is irrelevant with respect to rule $Ru_3$.

**Definition 10** Incompleteness (INCOM). Rules are incomplete when existing rules are unable to cover an access request. A rule is incomplete if and only if

– $\exists\, req = (S', R', A').$
– $\nexists\, Ru \in ACP \,|\, \forall\, a \in Ru.e_S, \exists\, a' \in S,\, Att_s(a) = Att_s(a') \rightarrow a \cap a' = a \wedge \forall\, b \in Ru.e_R, \exists\, b' \in Ru',\, Att_r(b) = Att_r(b') \rightarrow b \cap b' = b \wedge A' \cap Ru.A \neq \varnothing.$

A contractor working on a project, for example, requests information regarding the project schedule. However, there is no policy to handle the request.

### 3.2 Rule Clustering and Analysis

Rule clustering makes the policy analysis scalable. A number of clustering algorithms exist such as K-means [10] and hierarchical clustering [12]. However, they face various challenges (e.g., determining the number of clusters, cluster initialization) and are not effective for XACML rules. In this paper, we present

a clustering algorithm that groups rules sharing similarities into a cluster as the likelihood of anomalies (especially redundancies and static inconsistencies) among similar rules is high. Similarities between rules in terms of subjects ($S_s$), resources ($S_r$), actions ($S_{act}$), and conditions ($S_{con}$) are calculated for rules $Ru_i$ and $Ru_j$ as follows:

- $S(Ru_i, Ru_j) = w_s\, S_s(Ru_i, Ru_j) + w_r\, S_r(Ru_i, Ru_j) + w_a\, S_{act}(Ru_i, Ru_j) + w_c\, S_{con}(Ru_i, Ru_j)$
- $S_s(Ru_i, Ru_j) = \sum_{att_k \in (Ru_i.Att_s \cap Ru_j.Att_s)} [\, (d_s(att_k, Ru_i.s) \cap d_s(att_k,\ Ru_j.s)) \,/\, (d_s(att_k\ ,\ Ru_i.s) \cup d_s(att_k\ ,\ Ru_j.s)) \,]$
- $S_r(Ru_i, Ru_j) = \sum_{att_k \in (Ru_i.Att_r \cap Ru_j.Att_r)} [(d_r(att_k, Ru_i.r) \cap d_r(att_k,\ Ru_j.r)) \,/\, (d_r(att_k\ ,\ Ru_i.r) \cup d_r(att_k\ ,\ Ru_j.r)) \,]$
- $S_{act}(Ru_i, Ru_j) = [\, Ru_i.act \cap Ru_j.act \,] \,/\, [\, Ru_i.act \cup Ru_j.act \,]$
- $S_{con}(Ru_i, Ru_j) = [\, Ru_i.con \cap Ru_j.con \,] \,/\, [\, Ru_i.con \cup Ru_j.con \,]$

where $w_s + w_r + w_a + w_c = 1$. The weight assignment may depend on application needs. We consider weights of subject ($w_s$), resource ($w_r$), action ($w_a$), and condition ($w_c$) equal (all weights are assigned to 1/4). When the similarity score of two rules exceeds a threshold, they are grouped into a cluster. This score may fluctuate depending on the rule set. In our work, we use the threshold of 0.8 which was suggested by Lin et al. [14], and it works fine. Furthermore, each cluster has at least one rule, and each rule is grouped into one or more clusters. We consider *AND* operator in the rule definition. A rule containing Boolean expressions (e.g., *OR*, *NOT*) is split into several rules.

When the rules are clustered, we apply the formal tree-based policy modeling technique [3] for each cluster to analyze policy rules. To keep the tree as slim as possible, we define the data structure of the tree as follows. The first level of the tree is made up of action nodes, which show the actions of systems. Each action node in the tree points to the resource nodes on the second level. Each resource node is connected to the third level of the tree which contains subject nodes. Each subject node points to the condition nodes at the fourth level of the tree. Each condition node can then have one or two leaf nodes that represent the effects of the rule. Leaf nodes of each rule store *Rule ID* and a *Counter_ref* variable. The *Counter_ref* indicates whether the rule was triggered by any access request or not. Anomalies within each cluster are detected by traversing the policy trees from root to leaf node. Redundancies and static inconsistencies are detected in each cluster[1]. Dynamic inconsistencies, irrelevancies, and incompleteness can be detected by evaluating incoming access requests. When an incoming access request is issued, a cluster with the highest similarity to the request is identified. Then, rules in the corresponding cluster are evaluated with respect to the incoming access request according to Definitions 7-10 mentioned in Sect. 3.1.

### 3.3 Policy Learning Procedure for Anomaly Prediction

To discover the characteristics of rules with anomalies, machine learning techniques are applied to the rules generated by the XACML policy analysis (pre-

[1] For more details, please refer to [3].

sented in Sect. 3.2). Before applying the techniques, the rules need to be pre-processed as follows:

1) *Rules are parsed to organize attribute orders.* The rule components are divided into non-category and category attributes. Subject attributes, resource attributes, actions, conditions, and effects are non-category attributes. The type of rules that can be normal or anomalous is a category attribute.
2) *Missing attribute values are handled.* Missing attribute values can arise in any application. An approach for dealing with missing values is included in some classification algorithms. For example, a missing value of a numerical attribute is substituted with an average value of its attributes. They do not, however, take into account the semantics of data. We address missing values for subject and resource attributes using Definition 6. If values of action and effect components are missing, the effect of the rule becomes "not applicable".
3) *Continuous attributes are treated as some non-overlapping ranges to be efficient in data mining.* For example, the age attribute that has continuous numerical values is converted into a range (e.g., *infant*, *child*, *young adult*).
4) *Conditions in rules are addressed.* For example, a subject can access a resource during a specific time slot (*8am-5pm*) in a particular location (e.g., *office*). Permission is granted only if all the conditions are satisfied. Conditions, subject attributes, and resource attributes can be expressed as Boolean expressions; for example, *subject.security_level* $>$ *10*. In policy sets, rules are not uniformly structured necessarily. Rules may have complex Boolean expressions with variable lengths. To apply data classification techniques, we normalize these Boolean expressions. Boolean expression is converted to Disjunctive Normal Form (DNF) ($C_1 \vee C_2 \vee ... \vee C_i$). Then, the rule is divided into $i$ rules with distinct conditions.
5) *Intervals between rule components are managed.* We find all potential unique intervals among rule components when they overlap. For example, Rule 1 allows accountants and planners to change the project budget between *8am* and *5pm*. Rule 2 denies accountants the right to change the project budget between *12pm* and *1pm*. To convert the rules into non-overlapping rules, our algorithm identifies all the boundaries: *8am*, *12pm*, *1pm*, and *5pm*. The algorithm rewrites the rules as follows: Rule 1 allows accountants and planners to change the project budget between *8am* and *12pm*, Rule 2 denies accountants the right to change the project budget between *12pm* and *1pm*, Rule 3 allows planners to change the project budget between *12pm* and *1pm*, and Rule 4 permits accountants and planners to change the project budget between *1pm* and *5pm*.
6) *Imbalanced categories of anomalies in rule sets are addressed.* Usually, the number of instances with anomalies is far less than the number of instances without anomalies. Applying the classification techniques to an imbalanced rule set has a high likelihood of over-fitting [16] (i.e., the category with the dominant instance biases the classifier toward itself). We over-sample the

minority classes by generating synthetic instances using SMOTE (Synthetic Minority Oversampling Technique) [2].

Each rule is a feature vector containing subject attributes, resource attributes, actions, conditions, effects, and anomaly types. A subset of rules with anomalies are then utilized to learn anomaly characteristics. The accuracy of classification algorithms may vary depending on applications. There is no single algorithm that can outperform other algorithms in all feasible applications. Therefore, we apply five classification techniques from diverse categories (e.g., tree-based classifiers, distance-based classifiers, probabilistic classifiers) to the rules: Random Forest (RF), Decision Tree (DT), Naive Bayes (NB), Support Vector Machine (SVM), and K-Nearest Neighbors (KNN). The classifiers may have inherent classification inaccuracies, and various classifiers may provide different outcomes. As a result, we integrate the findings of multiple classifiers using the majority voting (MV) [18]. The majority voting technique ensures that the decisions of the classifiers are in agreement. Its decision, in particular, is a class that the majority of classifiers predict. When the classifiers' decisions are not in agreement, it chooses a class at random.

## 4 Experimental Evaluation

The primary goal of this section is to evaluate the effectiveness and efficiency of the proposed policy anomaly prediction approach. To achieve this, we perform various experimental evaluations as described below.

### 4.1 Rule Sets and Settings

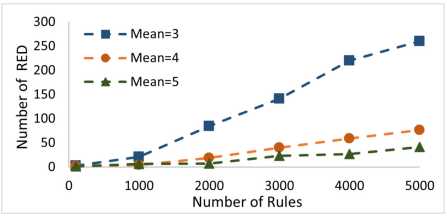
We are unable to get large real-world rule sets to evaluate the proposed approach. Therefore, we create 18 synthetic rule sets. The number of subject attribute expressions and resource attribute expressions are selected based on a normal distribution with distinct means and variances. In the experiments, three means of attribute expression for both subjects and resources are set to 3, 4, and 5, and variances are set to 1. Six rule sets with size of 100, 1000, 2000, 3000, 4000, and 5000 are built for each mean. Rule sets $\{RS_1, \cdots, RS_6\}$, $\{RS_7, \cdots, RS_{12}\}$, and $\{RS_{13}, \cdots, RS_{18}\}$ are constructed for the means of attribute expressions 3, 4, and 5, respectively. Attribute values can have different domains in practice. We define attribute values as an integer type with values ranging from 1 to 100. Subject attributes and resource attributes each have a lower and upper threshold. The total number of subject and resource attributes is 20. Each rule can have [1, 10] actions and [0, 10] conditions that are uniformly and randomly selected. The effect of each policy is randomly picked as either "Permit" or "Deny".

We build a set of access requests to evaluate the effectiveness of the proposed approach in predicting dynamic inconsistencies, irrelevancies, and incompleteness. The mean of attribute expressions for the requests is set to 10 and

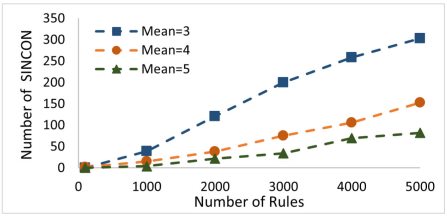
the variance is set to 2. The request set contains more than 10,000 requests. We consider that requests only have one action. Requests with more than one action are rewritten as multiple requests with one single action. The proposed approach is implemented in Java 11. The experiments are conducted using an Intel Core i7 1.99 GHz processor with 16 GB of RAM.

### 4.2 Policy Analysis

Rule sets $\{RS_1, \cdots, RS_{18}\}$ are analyzed based on the approach described in Sect. 3.2. The analyzed rule sets are considered training rule sets for Sect. 4.3.

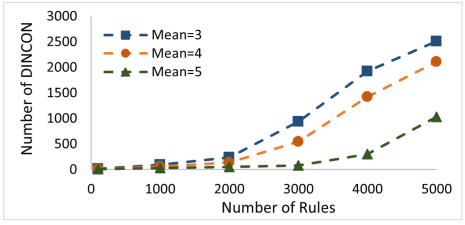



(a) Average Redundancy Rules.

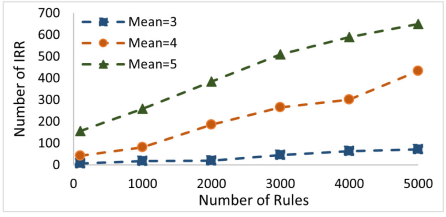



(b) Average Static Inconsistency Rules.

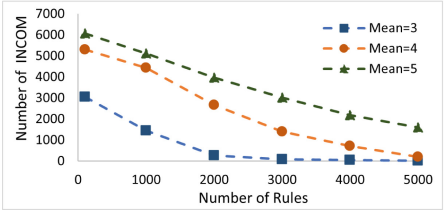
**Fig. 1.** Average redundancy and static inconsistency rules for 18 rule sets.




(a) Average Dynamic Inconsistency Rules.




(b) Average Irrelevancy Rules.




(c) Average Incompleteness Rules.

**Fig. 2.** Average dynamic inconsistency, irrelevancy, and incompleteness rules.

**Result Analysis.** Figures 1a and 1b show the average number of redundancies and static inconsistencies for 18 rule sets, respectively. On average, there is a significant number of redundancies and static inconsistencies. It is also observed that these two anomalies behave similarly with respect to the number of rules and means of attribute expressions. The average number of redundancies and static inconsistencies increases with the number of rules. However, as the means of attribute expressions increase, the growth rates of redundancies and static inconsistencies decrease.

Figure 2 shows the average number of dynamic inconsistencies, irrelevancies, and incompleteness for 18 rule sets. Figure 2a indicates that the behavior of dynamic inconsistencies is similar to the behavior of redundancies and static inconsistencies in terms of rule numbers and the means of attribute expressions. Similar behavior for static and dynamic inconsistencies was found by Liu et al. [15]. The average number of static inconsistencies is lower than the average number of dynamic inconsistencies. The reason is that policy administrators pay more attention to rules with similar attributes when creating policies. This can help decrease the number of static inconsistencies. It is difficult for administrators to determine whether rules with different attribute identifiers are inconsistent. Therefore, dynamic inconsistencies are ignored. As the majority of inconsistencies are dynamic inconsistencies, some of the static inconsistency detection approaches [11,13,21,25] may not successfully satisfy all actual system requirements.

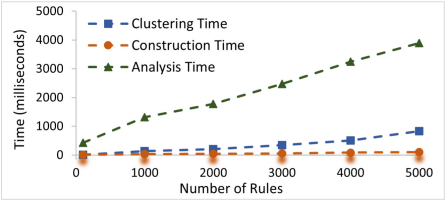



**Fig. 3.** Clustering, construction, and analysis time for 10,000 access requests for rule sets $RS_1, \cdots, RS_6$.

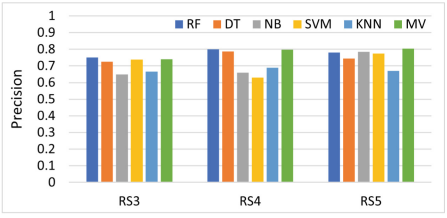
Despite the results shown in Figs. 1a, 1b, and 2a, the growth rates of irrelevancies increase when the means of attribute expressions increase (shown in Fig. 2b). Similar to redundancies, static inconsistencies, and dynamic inconsistencies, the average number of irrelevancies rises with the number of rules. On the other hand, the average number of incompleteness decreases with the number of rules. It can be observed from Fig. 2c that the average number of incompleteness increases as the number of attribute expressions grows. However, larger means of of attribute expressions have slower decreasing rates.

It can be observed from the above analysis that merely advising system administrators to employ a large number of attributes to generate inconsistency-free rule sets [15] is not effective. Although a high mean of attribute expressions

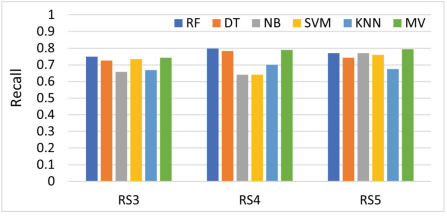
can result in rule sets with fewer redundancies and (static and dynamic) inconsistencies, it can also raise irrelevancies and incompleteness. A low mean of attribute expressions, on the other hand, results in rule sets with fewer irrelevancies and incompleteness, while it can also result in more redundancies and (static and dynamic) inconsistencies. Redundancies and static inconsistencies rely on rules and are independent of access requests, while dynamic inconsistencies, irrelevancies, and incompleteness depend on access requests. As the number of access requests is far greater than the number of rules, rule sets with a low mean of attribute expressions can cause fewer anomalies.

As the anomaly detection technique needs to search all trees to find anomalies in the rule set, we collect time for building clusters, constructing policy trees, and analyzing access requests. These time-based metrics rely on the number of rules. In Fig. 3, we show the time for rule sets $RS_1, \cdots, RS_6$. As this figure indicates, rule clustering takes longer than tree construction. In addition, the policy analysis takes a longer time than clustering and tree construction, which is reasonable as we consider 10,000 access requests.

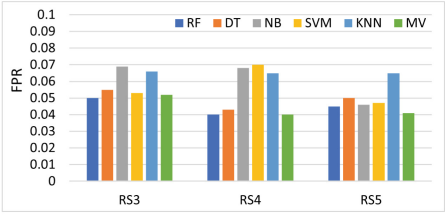

(a) Accuracy of the Rule Sets.



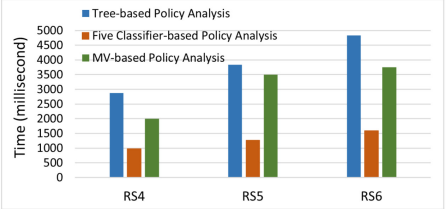

(b) Precision of the Rule Sets.




(c) Recall of the Rule Sets.




(d) False Positive Rate of the Rule Sets.




(e) Average Analysis Time.

**Fig. 4.** Efficiency of Anomaly Prediction for Rule Sets $RS_4$, $RS_5$, $RS_6$.

### 4.3 Anomaly Classification and Prediction

For the classification-based analysis, we use rule sets $RS_4$, $RS_5$, and $RS_6$. As the mean of the attribute expression is 3, the rule sets can balance the number of various anomalies (presented in the previous section). Various classifiers are constructed for the rule sets based on the approach presented in Sect. 3.3. We consider both binary and multi-class classifications. The classifiers are trained on 70% of the data and use 10-fold cross-validation and the Weka library [6], a set of machine learning algorithms for data mining tasks.

**Evaluation Metrics.** To assess the efficiency of our classification-based anomaly detection approach, we report accuracy, precision, recall, and false positive rate (FPR) defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \tag{1}$$

$$Precision = \frac{TP}{TP + FP} \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \tag{3}$$

$$FPR = \frac{FP}{FP + TN} \tag{4}$$

True Positive (TP) is the number of rules that are predicted as anomalies, and are truly anomalies. False Positive (FP) is the number of rules that are predicted as anomalies, while they are truly normal. False Negative (FN) is the number of rules that are predicted as normal, while they are truly anomalies. True Negative (TN) is the number of rules predicted as normal, and they are truly normal.

**Analysis Results.** The effectiveness of multi-class classifiers is displayed in Fig. 4. The accuracy, precision, and recall of all five classifiers (excluding K-Nearest Neighbors) are above 70% using $RS_6$ as shown in Figs. 4a, 4b, and 4c, respectively. Figure 4d shows that the false positive rate ranges from 4% to 7%, which is critical for security. The low false positive rate shows that the classification-based detection methods are capable of detecting all anomalies. However, the effectiveness of a classification-based approach varies based on the employed classifiers. Some of the classifiers do not behave consistently. For example, K-Nearest Neighbors has about 70% accuracy, precision, and recall in $RS_4$. It improves slightly in $RS_5$, but it becomes worse in $RS_6$. The majority voting, on the other hand, achieves above 75% accuracy, precision, and recall for all rule sets. The binary class classifiers indicate that the average accuracy, precision, and recall for all six classification algorithms (including majority voting) are 99% and the average false positive rate is 0.8%.

Figure 4e depicts the average analysis time for discovering anomalies using the tree model, five classifiers, and majority voting technique. In general, classifiers

outperform tree-based and majority voting policy analysis. The majority voting is slower than five classifiers as it integrates the results of all classifiers employed in classification-based policy analysis. However, it performs better than tree-based policy analysis. Therefore, it can be concluded that majority voting is effective in predicting policy anomalies.

## 5 Related Work

Policy anomaly assessment has attracted the attention of many researchers, and different approaches have been proposed in this area. There are some efforts to detect redundancies [11], inconsistencies [11,13,15,21,25], irrelevancies [17], and incompleteness [20] in XACML and ABAC policies. Jebbaoui et al. [11] proposed a semantic-based approach to detect redundancies and conflicts in XACML policies. The proposed approach was developed for a new set-based language named SBA-XACML. Shu et al. [21] focused on statically-conflicting rules and presented a method to detect conflicts in ABAC policies by using rule reduction and binary-search. They decomposed ABAC rules to identify redundancies and reduce the number of intersection operations. To enhance the efficiency of the proposed approach, they used binary search simultaneously. Liu et al. [15] presented a conflict detection method for ABAC policies based on the proposed definition of rule conflict. They transferred implicit conflicting rules to explicit conflicting rules and evaluated their approach by the proposed metrics.

The Satisfiability Modulo Theories (SMT) solver was another method for detecting conflict in XACML policies. The Boolean portion of satisfiability checking was separated from algorithms that employed property checking using SMT logic solvers [25]. Rami et al. [17] detected conflicts, unreachable policies (i.e., irrelevancies), and incompleteness by using Answer Set Programming (ASP). Expressing attributes that do not exist in AnsProlog [23] (e.g., strings) is difficult to be modeled. Also, there are no experimental results to show the applicability of their work. Fisler et al. [5] presented a tool called Margrave to analyze policies. It translates policies into Multi-Terminal Binary Decision Diagrams (MTBDDs) to answer user queries. Khoumsi et al. [13] proposed an automaton-based approach for modeling, developing, and analyzing policies. They divided firewall security policies into conflicting and non-conflicting anomalies.

There are a few works that use data mining-based and data classification techniques to detect anomalies in access control policies. Shaikh et al. [20] adopted data classification techniques to detect incompleteness in access control policies. This approach consisted of three steps. Attributes were ordered and Boolean expressions were normalized, a decision tree (which was a modification of C4.5) was generated, and the proposed anomaly detection algorithm was executed on the decision tree. In another study by Shaikh et al. [19], a modified C4.5 algorithm was proposed to detect inconsistencies and incompleteness in access control policies. When the number of XACML policies increases, the complexity and computation of some proposed algorithms grow, which makes the approach inapplicable. Abu Jabal et al. [9] proposed a framework based on the provenance

technique for policy analysis. They proposed structure-based and classification-based approaches that focus on the RBAC domain.

Various approaches have been investigated to address the problem of anomalies in access control rules [8]. Most of the existing approaches were developed in the network management and RBAC domains. These studies are not effective for predicting all four types of anomalies in XACML policies. Our classification-based policy anomaly prediction approach is time efficient comparing tree-based anomaly detection. Furthermore, past policy analysis research has concentrated on only a few types of anomalies (the inconsistency being the most studied), whereas our approach attempts to discover all types of anomalies.

## 6 Conclusion and Future Work

A large number of XACML policies may raise the risk of anomalies (redundancy, inconsistency, irrelevancy, and incompleteness) in distributed systems. Anomaly detection, on the other hand, is a difficult and expensive operation. In this paper, we have presented an anomaly prediction approach for XACML policies based on the data classification techniques. The proposed approach extracts XACML policy rules and clusters them based on their similarities. Anomalies in each cluster are then detected using the policy anomaly detection technique. Various data classification techniques are trained on the rules to identify the behavior of anomalies and predict the anomaly types of new rules. The experimental results have shown that the majority voting technique can obtain accuracy, recall, and precision of 80% and a false positive rate of 4%. Therefore, the proposed approach has the capability to predict anomalies. Furthermore, the experiments have shown that anomaly types can be correlated to the number of rules and attribute expressions. It is notable that the limitation of the proposed approach is the lack of real-world policies. As part of the future work, we plan to find an appropriate number of attribute expressions to minimize the number of anomalies.

## References

1. https://www.oasis-open.org/ XACML references and products, version 1.85. https://www.oasis-open.org/committees/download.php/42588/xacmlRefs-V1-85.html#Products. (Accessed 28 Dec 2021)
2. Chawla, N.V., Bowyer, K.W., Hall, L.W., Kegelmeyer, W.P.: Synthetic minority over-sampling technique: SMOTE. J. Artifi. Intell. Res. **16**, 321–357 (2002)
3. Davari, M., Zulkernine, M.: Policy modeling and anomaly detection in ABAC policies. In: Luo, B., Mosbah, M., Cuppens, F., Ben Othmane, L., Cuppens, N., Kallel, S. (eds.) CRiSIS 2021. LNCS, vol. 13204, pp. 137–152. Springer, Cham (2022). https://doi.org/10.1007/978-3-031-02067-4_9
4. Dawson, S., Qian, S., Samarati, P.: Providing security and interoperation of heterogeneous systems. In: Security of Data and Transaction Processing, pp. 119–145. Springer (2000). https://doi.org/10.1007/978-1-4615-4461-6_5

5. Fisler, K., Krishnamurthi, S., Meyerovich, L.A., Tschantz, M.C.: Verification and change-impact analysis of access-control policies. In: Proceedings of the 27th International Conference on Software Engineering, pp. 196–205 (2005)
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten. I.H.: The WEKA data mining software: an update. ACM SIGKDD Explorations Newsletter **11**(1), 10–18 (2009)
7. Hu, V.C., et al.: Guide to attribute based access control (ABAC) definition and considerations (draft). NIST Special Public. **800**(162), 1–54 (2013)
8. Jabal, A.A., et al.: Methods and tools for policy analysis. ACM Comput. Surv. (CSUR) **51**(6), 1–35 (2019)
9. Jabal, A.A., et al.: Profact: A provenance-based analytics framework for access control policies. IEEE Trans. Serv. Comput. (2019)
10. Jain, A.K., Dubes, R.C.: Algorithms for clustering data. Prentice-Hall Inc. (1988)
11. Jebbaoui, H., Mourad, A., Otrok, H., Haraty, R.: Semantics-based approach for detecting flaws, conflicts and redundancies in XACML policies. Comput. Elect. Eng. **44**, 91–103 (2015)
12. Johnson, S.C.: Hierarchical clustering schemes. Psychometrika **32**(3), 241–254 (1967)
13. Khoumsi, A., Erradi, M., Krombi, W.: A formal basis for the design and analysis of firewall security policies. J. King Saud Univ. Comput. Inf. Sci. **30**(1), 51–66 (2018)
14. Lin, D., Rao, P., Ferrini, R., Bertino, E., Lobo, J.: A similarity measure for comparing xacml policies. IEEE Trans. Knowl. Data Eng. **25**(9), 1946–1959 (2012)
15. Liu, G., Pei, W., Tian, Y., Liu, C., Li, S.: A novel conflict detection method for ABAC security policies. J. Ind. Inf. Integr. **22**, 100200 (2021)
16. Ostrand, T.J., Weyuker, E.J.: How to measure success of fault prediction models. In: 4th International Workshop on Software Quality Assurance
17. Ramli, C.D.P.K.: Detecting incompleteness, conflicting and unreachability XACML policies using answer set programming. arXiv preprint arXiv:1503.02732 (2015)
18. Ruta, D., Gabrys, B.: Classifier selection for majority voting. Inf. Fusion **6**(1), 63–81 (2005)
19. Shaikh, R.A., Adi, K., Logrippo, L.: A data classification method for inconsistency and incompleteness detection in access control policy sets. Int. J. Inf. Sec. **16**(1), 91–113 (2017)
20. Shaikh, R.A., Adi, K., Logrippo, L., Mankovski, S.: Detecting incompleteness in access control policies using data classification schemes. In: 5th International Conference on Digital Information Management (ICDIM), pp 417–422. IEEE (2010)
21. Shu, C.-c., Yang, E.Y., Arenas, A.E.: Detecting conflicts in ABAC policies with rule-reduction and binary-search techniques. In: International Symposium on Policies for Distributed Systems and Networks, pp. 182–185. IEEE (2009)
22. OASIS Standard. Extensible access control markup language (XACML) version 3.0. 2008. http://docs.oasis---open.or/xacmmL2.0/access_control-xacml-2.0core.spec---OS.pa1 (2013)
23. Sureshkumar, A., De Vos, M., Brain, M., Fitch, J.: Ape: An ansprolog* environment. See De Vos and Schaub **2007**, 101–115 (2007)
24. Thompson, M.R., Essiari, A., Mudumbai, S.: Certificate-based authorization policy in a PKI environment. ACM Trans. Inf. Syst. Sec. (TISSEC) **6**(4), 566–588 (2003)
25. Turkmen, F., den Hartog, J., Ranise, S., Zannone, N.: Analysis of XACML Policies with SMT. In: Focardi, R., Myers, A. (eds.) POST 2015. LNCS, vol. 9036, pp. 115–134. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-46666-7_7

26. Wimmer, M., Kemper, A., Rits, M., Lotz, V.: Consolidating the access control of composite applications and workflows. In: Damiani, E., Liu, P. (eds.) DBSec 2006. LNCS, vol. 4127, pp. 44–59. Springer, Heidelberg (2006). https://doi.org/10.1007/11805588_4
27. Xu, Z., Stoller, S.D.: Mining attribute-based access control policies. IEEE Trans. Depend. Sec. Comput. **12**(5), 533–545 (2014)