Mutated Policies: Towards Proactive Attribute-based Defenses for Access Control

Carlos E. Rubio-Medrano, Josephine Lamp, Adam Doupé, Ziming Zhao and Gail-Joon Ahn
The Center for Cybersecurity and Digital Forensics
Arizona State University
[crubiome,jalamp,doupe,zzhao30,gahn]@asu.edu

ABSTRACT

Recently, both academia and industry have recognized the need for leveraging real-time information for the purposes of specifying, enforcing and maintaining rich and flexible authorization policies. In such a context, security-related properties, a.k.a., *attributes*, have been recognized as a convenient abstraction for providing a well-defined representation of such information, allowing for them to be created and exchanged by different independently-run organizational domains for authorization purposes. However, attackers may attempt to compromise the way attributes are generated and communicated by recurring to hacking techniques, e.g., forgery, in an effort to bypass authorization policies and their corresponding enforcement mechanisms and gain unintended access to sensitive resources as a result.

In this paper, we propose a novel technique that allows for enterprises to *pro-actively* collect attributes from the different entities involved in the access request process, e.g., users, subjects, protected resources, and running environments. After the collection, we aim to carefully select the attributes that uniquely identify the aforementioned entities, and randomly *mutate* the original access policies over time by adding additional policy rules constructed from the newly-identified attributes. This way, even when attackers are able to compromise the original attributes, our *mutated* policies may offer an additional layer of protection to deter ongoing and future attacks. We present the rationale and experimental results supporting our proposal, which provide evidence of its suitability for being deployed in practice.

1 INTRODUCTION

Recently, emerging authorization paradigms have been introduced to leverage information collected in real-time for the purposes of access mediation, in an effort to provide enhanced flexibility for crafting, enforcing and maintaining authorization policies. Within those proposals, *attribute-based access control* (ABAC) [5] has attracted the interest of both academia and industry due to its support for representing such real-time information by means of *attributes*: security-relevant properties and characteristics that may be originated within different security domains, which in turn may be implemented by independently-run organizations. In such a context, collaborating organizations may leverage attribute-based policies for mediating access to a set of shared resources, allowing for entities, e.g., users, within the context of one organization to access the resources provided within the context of another one. However, while highly-flexible and convenient, such a scheme may depend on each participating organization being fully-capable of *provisioning*: locating, producing and communicating attributes in a secure and efficient manner. When such an attribute provisioning process is compromised, undesired consequences may be triggered as a result. As an example, a successful hacking incident in one participating organization may result in the security of the overall access mediation framework being affected, as controlling the creation and distribution of attributes may allow attackers to fully bypass the authorization enforcement mechanisms deployed by both organizations, thus ultimately resulting in unwanted access to sensitive resources.

To address this problem, we present an approach inspired by *moving target defense* (MTD) [7], a promising defensive paradigm based on the idea of proactively changing, e.g., *moving*, system configurations in an effort to deter potential attacks. The key idea of MTD is to increase the difficulty and cost to an attacker to perform a successful attack. In our approach, we aim to analyze attributes that are collected from both runtime traces of mission-critical applications as well as from diverse available sources, and are ultimately combined into a dataset to be referred to as the *attribute bag*. Taking an original authorization policy as an input, our approach first obtains the *original* attributes listed in the policy and inspects the attribute bag to locate attributes that are *correlated* to the original ones by leveraging well-established machine learning techniques such as *association analysis* [15]. Later, these *newly-extracted* attributes are used to *enhance* the original policy, by adding additional constraints to existing policy rules, thus producing a new *mutated* policy that is then forwarded to the access mediation infrastructure for enforcement.

The intuition behind our approach is that the entities involved in a given access request, e.g., end-users and protected resources, typically exhibit additional attributes besides the ones in the original policy. In our system, if the original attributes are compromised, the newly-extracted ones, which we assume stay uncompromised, may still deter the unintended exploitation of the original policy. Furthermore, the attacker does not know which of the additional attributes will be checked, which will increase the cost and time burden on the attacker to achieve an unauthorized access. In addition, we aim to mitigate the harm to usability, such as end-users no longer able to access previously-available resources, by striving to obtain a high degree of correlation between the original attributes

MTD'17, October 30, 2017, Dallas, TX, USA

ACM ISBN 978-1-4503-5176-8/17/10…$15.00
https://doi.org/10.1145/3140549.3140553

and the newly-extracted ones. With this in mind, this paper provides the following contributions:

(1) We introduce a novel moving target defense-based approach to provide extended security protection based on access control policies by identifying the entities involved in access requests via the attributes they depict, resulting in proactively *expanding* an original policy;
(2) We implement our approach and evaluate its effectiveness and efficiency on various sample policies. Our experimental results show that correlated attributes do exist in real-life policies and that correlated attributes can be efficiently discovered and used to create mutated policies for enhanced security protection.

This paper is organized as follows: we start by briefly reviewing some important background topics, along with a running example and some other key considerations for our approach in Section 2. We then elaborate on the problem being addressed in this paper in Section 3. Our proposal is later described in Section 4, and we present the results of our experimental evaluation in Section 5. We compare our approach with related work in Section 6 and present some discussion as well as our plans for future work in Section 7. Finally, we present our concluding remarks in Section 8.

2 BACKGROUND

2.1 Attribute-based Access Control

In *attribute-based access control* (ABAC) [5] a request is granted upon the satisfaction of constraints involving properties, characteristics, or traits of subjects, objects, and even environment conditions that are relevant under a given security context, also known as *attributes*. Such attributes are managed and provisioned by organizations/ institutions a.k.a., *attribute authorities* in [5], who are in charge of defining, creating and assigning them to access entities. Attributes may be in turn leveraged by *policy makers*, who are in charge of crafting policies by establishing relationships between access entities, attributes and permissions. Following the description provided by the U.S. National Institute of Standards and Technology (NIST) [5], dedicated infrastructures may be introduced in the foreseeable future allowing for attributes originating from different authorities to become relevant under the organizational domain the policy makers belong to, depicting an *enterprise* scenario. This scenario is based on the concept of collaboration between different organizations for *provisioning* (e.g., locating and communicating) attributes at runtime, thus potentially introducing enhanced flexibility, convenience and efficiency for policy specification and enforcement.

Running Example. Let us assume a sample ABAC policy that is defined in the context of *electronic health records* (EHRs) [8], a technology intended to increase the efficiency of healthcare organizations, thus leading to improved continuity and coordination of care. Such a policy grants access to the EHRs of a given patient to all staff members whose attribute labeled as *Staff ID* matches one of the attributes values listed in a predefined set. Fig. 1 shows a sample dataset, hereafter to be referred to as an *attribute bag*, which is constructed by combining data obtained from access logs derived from enforcing our sample policy at runtime as well as attributes collected from different authorities. As an example, the data record

	Staff ID	Match IDs	Location	Role	Credential	Decision
1	A11235	FALSE	Surgery	Surgeon	MD	FALSE
2	A43452	TRUE	ER	Nurse	RA	TRUE
3	A83420	TRUE	ER	Physician	MD	TRUE
4	A83425	TRUE	ER	Physician	MD	TRUE
5	A89736	TRUE	ER	Nurse	RA	TRUE
6	A11236	TRUE	ER	Surgeon	MD	TRUE
7	A83421	TRUE	ER	Physician	MD	TRUE
8	A83422	TRUE	Pediatrics	Physician	MD	TRUE
9	A56112	FALSE	ER	Lab Tech	CLIA	FALSE
10	A77777	FALSE	Radiology	Staff	RN	FALSE

Figure 1: A sample attribute bag based on EHRs depicting attributes in the original policy (green, straight border line), correlated attributes (orange, dotted border line), as well as non-correlated ones (yellow, vertical lines).

depicted in the first row shows an access request issued by a staff member with the attribute <*Staff ID*, String, A11235>. Assuming such value was not listed in any rule included in our sample policy, this request was therefore denied, as shown by the value of the *Decision* attribute set to *FALSE* in the last column. For illustrative purposes, our attribute bag contains an additional artificially-created attribute *Match IDs*, which is set to the value of *TRUE* when a rule in our sample policy contained a value matching the one defined for the aforementioned *Staff ID* attribute. Otherwise, the value for the *Match IDs* attribute is set to *FALSE*. Other attributes shown in Fig. 1 will be further discussed, along with their corresponding coloring scheme, in Section 4.

2.2 Association Analysis

Association analysis [15] is a method for finding relationships between data items in large sets, commonly referred as *association rules* or simply as *associations* in the rest of this paper. Associations of the form $A \rightarrow B$ denote a relationship between the data items labeled as A and B, e.g., data records containing A are likely to contain B as well. It is important to note that associations do not imply causality; rather, they suggest a statistical relationship between the items. The strength of a given association can be measured using the metrics known as *support* and *confidence*. Support measures the number of data records that contain the items listed in the association and can be used to eliminate uninteresting associations, as well as to enable the efficient discovery of associations within large data sets. Support is defined as $s(A \rightarrow B) = \sigma(A \cup B)/ N$, where $\sigma(A \cup B)$ stands for the count of the data records where A and B appear together, and N stands for the count of all records in the dataset. As an example, following Fig. 1, s(`Match IDs` = *TRUE* $\rightarrow$ `Decision` = *TRUE*) can be calculated as σ(`Match IDs` = *TRUE* $\cup$ `Decision` = *TRUE*)/10 = 7/10 = 0.70. Confidence determines how often items in one subset of records appear in another subset and enables the measurement of reliability of inference implied by a given association. It is defined as $c(A \rightarrow B) = \sigma(A \cup B)/\sigma(A)$, where $\sigma(A \cup B)$

stands as described before and σ(A) stands for the count of records that contain A. Using Fig. 1 as an example again, c(`Match IDs` = *TRUE* $\rightarrow$ `Decision` = *TRUE*) = σ(`Match IDs` = *TRUE* $\cup$ `Decision` = *TRUE*)/σ(`Match IDs` = *TRUE*) = 7/7 = 1.0.

3 ATTACK MODEL AND ASSUMPTIONS

As described in Section 2, ABAC relies on specifying, collecting and processing attributes belonging to access entities involved in a given request, e.g., users, protected resources and the environment. In such a context, the unintended assignment of such attributes to entities may completely compromise the security of the access mediation process. Based on this, we assume an attack model where the attributes listed in a given policy become *compromised* by an attacker, for example, by creating an unintended attribute-access entity assignment, or by deliberately manipulating the value or set of values depicted by a correctly-assigned attribute. The way an attacker may be able to compromise a given attribute may include, but may not be limited to the following: an unintended software error, forgery or a hacking incident compromising the infrastructure where attributes are created and assigned to entities, e.g., an attribute authority.

For the purposes of this paper, we assume the following: first, we devise an attribute collection model in which attributes originated from different authorities are leveraged for the purposes of policy crafting, and in which data containing attribute-based information is available for analysis. In the context of the application domain depicted in our running example, data collection may include a preprocessing step in which data from the access logs is combined with information extracted from EHRs themselves and any supporting platforms such as clouds or operative systems. As an example, Fig. 1 shows data collected for each access request made in the context of an EHR. For each request, the following items are shown: the result of evaluating the request, and a description of the collected attributes and their corresponding values, a.k.a., the *attribute bag*. Second, we assume that the software framework handling the specification and runtime evaluation of authorization policies, as well as the software modules implementing our approach (including the collection procedure described above), are out of reach for an attacker. As an example, even when a given authorization policy, along with its listed attributes, may be known to the attacker, he/she has no way to deliberately change its contents, either by removing the policy as a whole or by adding or removing rules at will.

4 OUR APPROACH: PROACTIVE ATTRIBUTE-BASED DEFENSES

As introduced in Section 1, our approach, shown in Fig. 2, is intended to dynamically *expand* access control policies by detecting and incorporating the attributes that are strongly *correlated* to the entities involved in access requests. By mutating policies, an access control system is expected to cope with attacks such as the ones mentioned in Section 3, including *zero-day* ones which have not been previously reported or detected by security officials. In this section, we first describe our approach by extending the description of ABAC introduced in Section 2.1. Then, we provide an informal description of our ideas by leveraging our running example in Section 4.2, and move on to provide a more detailed description in Section 4.3 by leveraging the *association analysis* techniques discussed in Section 2.2.

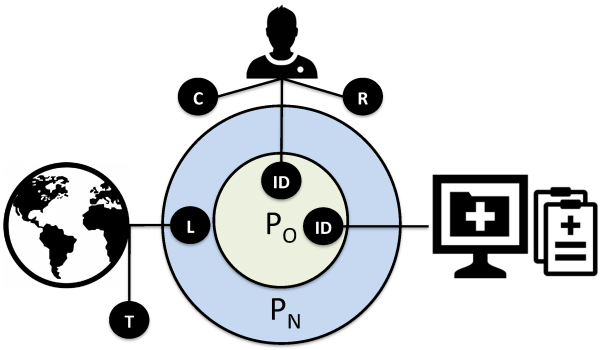

Figure 2: A graphical depiction of our approach based on the *attribute bag* shown in Fig. 1: the original policy, labeled as P_O, is defined over a set of *original* attributes obtained from both the requesting staff member (*ID*) and the EHRs themselves (*ID*). Later, an additional attribute *L*, obtained from an environmental source, is found to be correlated to the *original* ones, and is used to craft a *mutated* policy P_N.

4.1 Correlation-based Policy Mutation

Describing ABAC. We start the description of our approach by first extending the discussion on ABAC introduced in Section 2.1. In such a model, an ABAC policy P_O can be represented by means of the *permission assignment* (PA) and *attribute assignment* (AA) relations. As an example, an attribute a that is required for granting a given permission p can be modeled as an entry of the form (a,p) in the PA relation. In addition, an access entity u, e.g., an end-user, is assigned an attribute a if there exists an entry of the form (u,a) in the AA relation. In such a context, the auxiliary function *assigned(u,a)* returns *true* if and only if an access entity u is assigned an attribute labeled as a, e.g., there is an entry of the form (u,a) in AA, and returns *false* otherwise. For a given access entity u, the set of assigned attributes can be obtained by inspecting the attribute bag collected for it at runtime, e.g., *assigned(u,a)* may return *true* if a is contained within u's bag. Conversely, the auxiliary function *related(a,p)* returns *true* if and only if the aforementioned attribute a is related to the requested permission p, e.g., there exists an entry of the form (a,p) in PA, and returns *false* otherwise. With this in mind, the authorization process for a user entity u requesting a permission p can be modeled by means of a function called *granted(u,p)*, which returns *true* if there exists an attribute a such that both *assigned(u,a)* and *related(a,p)* return *true*, and returns *false* otherwise.

Attribute-based Populations. In the model just described, a policy rule $r \in P_O$ may allow for a constraint-based specification of the set of attributes $S_r \in A$ that are related to a given permission p, e.g., of the set of tuples in PA relating S_r with p. Similarly, a set of access entities E can be further constrained into different subsets, also known as *populations*, following the attribute assignments defined in the AA relation. As an example, $U_a \subseteq E$ identifies the population of users that are assigned the attribute a, e.g., all

Algorithm 1 Correlated-based Policy Mutation

Require: An original policy P_O, an attribute bag A
Ensure: A mutated policy P_N
1: $P_N \leftarrow \emptyset$
2: $R \leftarrow$ getRulesFromPolicy(P_O)
3: **for all** $r_i \in R$ **do**
4: $S_r \leftarrow$ getAttributesFromRule(r_i)
5: $C_r \leftarrow$ findCorrelatedAttributes(S_r, A)
6: $r'_i \leftarrow$ getRandomMutatedRule(r_i, C_r)
7: $P_N \leftarrow P_N \cup r_i'$
8: **end for**
9: **return** P_N

users u for which the evaluation of *assigned(u,a)* returns *true*. In our approach, we extend this notion by taking as an input an original rule r and retrieving the set of attributes $S_r \in A$ that are listed in it to identify a population $U_{S_r} \in E$.

Policy Mutation. With this in mind, we aim to identify a set of attributes C_r that are assigned to the entity or entities holding the ones contained S_r, that is, for each attribute c in C_r, there exists an access entity u such that both (u,c) and (u,a) exist in AA, for some attribute a in U_r. This way, we aim to identify a population $U_{C_r} \subseteq E$ such that $U_{S_r} = U_{C_r}$ for the sets of attributes labeled as S_r and C_r. Later on, following the model for ABAC described before in this section, we also aim to update the PA relation by inserting a new rule r' such that for a given $c \in C_r$, a new tuple (c, p) is added to PA, thus creating a mutated policy as a consequence.

Algorithm Description. Algorithm 1 contains a description of the approach just described: initially, we model an original access control policy P_O as a set of constraint-based *rules* $R = \{r_1, r_2, r_3, ..., r_n\}$ for some $n > 0$ (Lines 1-2). Then, for each rule $r \in R$, we introduce the set S_r of attributes that are listed in it (Line 4). In addition, we also model the *attribute bag* as described before as a set of attributes A such that $A \cap S_r \neq \emptyset$ for all $r \in R$. Given the original policy P_O, our approach then aims to produce a *mutated* policy P_N as follows: for each rule $r \in R$, we locate the set of attributes $C_r = \{c_1, c_2, ... c_p\} \subseteq A$, $C_r \neq S_r$, that are *correlated* to the set of attributes S_r (Line 5). Then, r may become a new rule r' by randomly choosing a subset $c \in C_r$, and adding it to the set of original attributes S_r, such that $S_{r'} = S_r \cup c$ [1] (Line 6). Later, the set of modified rules $R' = \{r'_1, r'_2, ... r'_n\}$ is put together to create the new mutated policy P_N (Lines 7 and 9), which is then forwarded to a policy evaluation module for further enforcement. This way, P_N is said to *expand* the original P_O policy by including rules that depict additional correlated attributes besides the original ones. We repeat the above procedure in an effort to produce many different policy mutations. For such a purpose, an interactive approach may randomly produce modified rules by selecting only a subset of the set C_r of correlated attributes each time, in such a way that the resulting rules may vary from time to time. Recall that correlated attributes appearing in previous mutations may be selected to appear once again in a new mutation in case they are randomly selected by the auxiliary procedure shown in Line 6. In addition, new correlated attributes may be collected in the attribute bag, thus possibly producing different mutated policies as a result. In general, if n correlated attributes are present in an attribute bag, then 2^n possible combinations of them may be used to produce mutated policies.

Addressing MTD Goals. Returning to the discussion presented before in this section, in an ideal case for our policy mutation approach, populations U_{S_r} and U_{C_r}, which define the set of original and correlated attributes respectively, should be equivalent to each other. However, such a $U_{S_r} = U_{C_r}$ correspondence is not always guaranteed to appear in practice. In such a context, the following cases may arise:

(1) In the $U_{S_r} \subset U_{C_r}$ case, as the size of the population defined for the correlated attributes exceeds the one of the original ones, a potential *safety* problem arises, as the resulting rule r' may be more *relaxed* than the original, thus potentially resulting in members of a population not originally included in r being granted access to a protected resource by means of r';
(2) In the $U_{C_r} \subset U_{S_r}$ case, the size of the population defined by the original attributes is larger than the size of the population defined by the correlated ones. In such a case, the newly introduced rule r' is likely to be more *restrictive* than its original counterpart r, e.g., access entities included in the population defined by U_{S_r} may not be covered by the population defined by means of U_{C_r}, thus resulting in a *usability* problem, which represents a potential harm for the adoption of MTD-based techniques;
(3) For completeness, in the $U_{C_r} \not\subset U_{S_r}$, $U_{S_r} \not\subset U_{C_r}$, and $U_{C_r} \cap U_{S_r} \neq \emptyset$ cases, both safety and usability problems arise, following the descriptions mentioned above.

Hence, our approach is therefore intended to minimize both the harm to usability as well as the safety concerns of our proposed mutated policies. Strictly speaking, when producing a mutated rule r', we aim to identify the two populations U_{S_r} and U_{C_r} in such a way that the following function properties are set to minimal values:

Definition 4.1. f-usability = $| U_{S_r} | - | U_{C_r} |$

Definition 4.2. f-safety = $| U_{C_r} | - | U_{S_r} |$

Having said this, our approach allows to properly balance such safety and usability functions by introducing numerical thresholds depicting acceptable minimal values for the two, thus allowing for policy makers to further customize our approach depending on the context of their own security and application domains, as it will be later discussed in Section 4.2. In the next section, we describe our approach to get an estimate of the aforementioned populations by inspecting our proposed *attribute bag* using the *association analysis* techniques introduced in Section 2.2.

4.2 Finding Correlated Attributes

Green Attributes. Following the description for policy mutation described before, finding the set C_r of correlated attributes is core to our approach. For such a purpose, we aim to find patterns relating the attributes in the attribute bag with the ones contained in the set S_r of original attributes. For illustrative purposes, assume the sample original policy and the attribute bag data shown

[1] In case $C_r = \emptyset$, then $r' = r$.

in Fig. 1, which were introduced in Section 2 and depict a single rule granting access to the EHRs of a given patient if the value of attribute *Staff ID* as shown by the requesting end-user is equal to a value contained in a predefined access control list. Such a relationship has been captured in the *Match IDs* attribute that is included in Fig. 1. Our attribute correlation approach can be then described as follows: we start by first finding the relationship between the attributes in the original set S_r, e.g., *Staff ID* and *Match IDs*, and the access decision with a value of *true*, in an effort to identify within the data records depicting the attribute bag, the ones that belong to the requested access being granted according to our original policy. In Fig. 1, such relation is represented by records with cells colored in *green* with black border line and represent the population U_{S_r} described in Section 4.1.

Orange Attributes. Next, we strive to find relationships between the original set S_r (*green*), as identified by the previous step, and some other attributes in the attribute bag. As an example, in Fig. 1, the value of attribute *Location* is the same when the value of the attribute *Match IDs* is *true* and the access decision depicts the value of *true*. Such a relationship is displayed in Fig. 1 in the *orange* color and dotted border line. As mentioned before, in order for this step to be meaningful for the purposes of our approach, this relationship should be as strong as possible, that is, the vast majority of the records depicting the original attributes should also depict the newly-correlated ones. Referring back to Fig. 1, the number of records containing cells colored in *green* and the ones containing cells in *orange* should be the same or stay within a close margin with respect to the total of data records in the attribute bag.

Yellow Attributes. Next, we strive to identify the relationship between the candidate *orange* attributes and the *false* access decision value, in an effort to make sure these newly-discovered correlated attributes are not shared by entities getting the *false* access decision in the attribute bag data. The intuition behind this is that the *orange* attributes should only be assigned to the entities with legitimate access according to our original policy. Such a relationship is represented by cells depicting the *yellow* coloring with vertical lines in Fig. 1. With all this in mind, our approach should identify the candidate *orange* attributes in such a way that their relation to the original ones (*green*) is strong, whereas the relation with the *yellow* ones is kept to a minimum for safety and usability purposes, respectively. Ideally, the number of records with cells in *yellow* should be minimal in respect to the number of records depicting the *green* and *orange* colorings, e.g., close to zero, as a large number of such *yellow* records would imply a potential security vulnerability. If such conditions are met, the combination of the records depicting the *orange* and *yellow* records represent the U_{C_r} population described in Section 4.1 and the *orange* attributes are said to depict the set C_r, which can be then used to create mutations of the original policy. Following our running example, the newly-discovered orange attribute *Location* would be then used to create a new rule that, besides comparing the value *Staff ID* attribute with the ones contained in a predefined access list, also requires the location of the device initiating the request to be equal to the value *ER*.

Algorithm Description. Algorithm 2 shows a pseudo-code description for finding *orange* attributes by leveraging *association analysis* [15]. Such an algorithm takes as an input the set S_r of attributes listed in a given policy rule *r*, the attribute bag identified as *A*, as well as a pair of numeric thresholds labeled as *usability* (*u_f*) and *safety* (*s_t*) respectively. The *usability* threshold is intended to provide an indication of the degree of similarity desired between the number of records depicting *green* and *orange* attributes. As mentioned before, an ideal case would consider the two populations represented by the *green* and *orange* attributes being the same, thus implying a threshold value of zero. However, in practice, such a requirement can be relaxed up to a certain predefined value greater than zero that may be dependent on the application domain within a certain level of tolerance. In a similar fashion, the *safety* threshold provides an indication of the desired relationship between the number of records depicting potential *orange* attributes and the records depicting *yellow* ones. Once again, an ideal situation would consider the number of records containing *yellow* attributes to be zero. However, such a requirement may be relaxed up to a certain positive threshold depending on the application domain, assuming the security officials implementing our approach are knowledgeably willing to do so. Therefore, these thresholds then provide a way for estimating the usability and safety functions described in Definitions 4.1 and 4.2 as mentioned before.

Algorithm 2 then proceeds as follows: we start by leveraging association mining techniques to identify the ones relating both the original (*green*) set of attributes and the positive access decision

Algorithm 2 Finding Correlated Attributes

Require: A set S_r of original attributes, an attribute bag A, a *usability* threshold u_t and a *safety* threshold s_t
Ensure: A set C_r of correlated attributes

```
1:  C_r ← ∅
2:  G ← getGreenAssociations(S_r, A)
3:  for all g ∈ G do
4:    if c(g) = 1.0 then
5:      O ← getOrangeAssociations(g, A)
6:      for all o ∈ O do
7:        usability ← (s(g) - s(o)) / s(g)
8:        if c(o) = 1.0 and (usability ≤ u_t) then
9:          Y ← getYellowAssociations(g, o, A)
10:         if size(Y) ≠ 0 then
11:           for all y ∈ Y do
12:             if c(y) = 1.0 then
13:               safety ← (s(y) - s(o)) / s(y)
14:               if safety ≤ s_t then
15:                 C_r ← C_r ∪ getAttributes(o)
16:               end if
17:             end if
18:           end for
19:         else
20:           C_r ← C_r ∪ getAttributes(o)
21:         end if
22:       end if
23:     end for
24:   end if
25: end for
26: return C_r
```

case (Line 2). Following the naming convention introduced earlier in this section, we refer to those associations as *green* ones. As an example, following the notation introduced in Section 2, our running example would depict a green association of the form *Match IDs = True* $\rightarrow$ *Decision=True*. Details on the generation of all color-named associations mentioned in Algorithm 2 are to be discussed in Section 4.3. Next, we inspect each of the produced *green* associations in order to locate potential *orange* attributes that are depicted by the namesake associations (Line 5). Following our running example, an orange association of the form *Match IDs=True* $\wedge$ *Location=ER* $\rightarrow$ *Decision=True* will be produced.

In the next step, we explore each of the *orange* associations in order to locate attributes that have a strong correlation with the *green* ones (Lines 6-23). In such a context, two verifications are performed in lines 7-8: first, we check that the confidence level of the two inspected associations, either green and orange, are equal to 1.0. As it will be detailed in Section 4.3, having a confidence level of such a value is important for our approach. Second, we calculate a *usability* index based on the two current associations being inspected, and we compare it against the predefined threshold provided as an argument to our algorithm. If both verifications pass, that is, there is a strong correlation between both associations and their involved attributes, we continue our analysis in lines 9-22.

Recall from a previous description that our approach also aims to detect our so-called *yellow* attributes, which may potentially deviate in a security vulnerability as they may imply our newly-discovered orange attributes are also shared by entities not getting access granted to the protected resources. With this in mind, we then find so-called *yellow* associations depicting our current orange ones being processed (Line 9). Continuing with our running example, we obtain a yellow association of the form *Location=ER* $\rightarrow$ *Decision=False*. For each obtained yellow association we calculate the *safety* index and compare it against the predefined safety threshold that was also provided as an input to our algorithm. In case there is a positive verification, the attributes in the current orange association under inspection are added to the resulting set by means of the utility function *getAttributes* (Lines 12-17). Finally, in case no yellow associations were initially obtained, we move on to directly add the attributes in the current orange association to our resulting set in line 20.

4.3 Leveraging Attribute-based Associations

In this section, we detail the rationale behind the *getGreenAssociations*, *getOrangeAssociations* and *getYellowAssociations*, as well as an insight on the calculation of confidence levels as depicted in Algorithm 2.

Green Associations. As it was hinted earlier, the supporting function *getGreenAssociations* obtains our proposed *green* associations by initially processing the original attribute bag and selecting the ones that contain both the attributes as listed in the original policy as well as the access decision attribute depicting the value of *True*. For readability purposes, we denote green associations as GREEN $\rightarrow$ DECISION=*True*. In our approach, we combine the different attributes as listed in an original policy rule into the aforementioned GREEN set. For instance, assuming attributes labeled as g1, g2 and g3, we would inspect the original set of associations derived from the attribute bag in order to select the one listed as {g1 $\wedge$ g2 $\wedge$ g3} $\rightarrow$ DECISION=*True*. As mentioned before, in our running example a green association would take the form *Staff ID=A11235* $\wedge$ *Match IDs = True* $\rightarrow$ *Decision=True*. In addition, associations of the form GREEN $\rightarrow$ DECISION=*True* may be also present in the attribute bag due to the existence of *redundant* rules in the original policy, e.g., a user holding the GREEN and getting the DECISION=*True* attribute value because of another rule different to the one being processed at a given moment of time. For the purposes of our approach, we assume redundant rules have been identified and removed beforehand, as discussed in [4].

Orange Associations. In a similar fashion, our supporting function *getOrangeAssociations* selects the ones that depict the form GREEN $\wedge$ ORANGE $\rightarrow$ DECISION=*True*, where antecedent GREEN and consequent DECISION=*True* stand as before and ORANGE stands for an additional set of attributes not included in GREEN, e.g., GREEN $\cap$ ORANGE = $\emptyset$. Following the example shown above, we aim to detect all associations of the form {g1 $\wedge$ g2 $\wedge$ g3 $\wedge$ o1 $\wedge$ ... $\wedge$ oN} $\rightarrow$ DECISION=*True* for some green attributes g1, g2 and g3 and some orange attributes o1 to oN for some N > 0. Referring back to our running example, an orange association would then take the form *Staff ID=A11235* $\wedge$ *Match IDs=True* $\wedge$ *Location=ER* $\rightarrow$ *Decision=True*. As shown in Algorithm 2 (Lines 3-5), we aim to obtain the orange associations that are related to a given green one. With this in mind, we populate the GREEN set with the attributes depicted in the current green association being processed so we can locate the orange ones as just explained. In our current implementation, to be further discussed in Section 5, we leverage our supporting tools to enlist all possible associations from the input dataset, e.g., the attribute bag labeled as *A* in Algorithm 2, and parse them sequentially looking for the ones depicting the pattern described above.

Yellow Associations. In addition, our auxiliary function *getYellowAssociations* computes the set of *yellow* associations by taking the aforementioned ORANGE as an input and inspecting the original set of associations in order to locate the ones meeting the form $\neg$GREEN $\wedge$ ORANGE $\rightarrow$ DECISION=*False*, where ORANGE stands as explained before and $\neg$GREEN denotes the *inverse* of the set of green attributes introduced before, that is, all other attributes not in the GREEN nor in the ORANGE sets. Also, DECISION=*False* the inverse of DECISION=*True*, that is, the attribute holding the result of the access request depicting the value of *False*. In our running example, a yellow association would take the form *Location=ER* $\rightarrow$ *Decision=False*. Moreover, associations of the form $\neg$GREEN $\wedge$ ORANGE $\rightarrow$ DECISION=*True* may need to be considered as well in order to take into account policy rules granting the access request based on the attributes contained in the current ORANGE set. Even when they may not necessarily represent a security vulnerability (assuming they are accessing the same protected resource), they may add little value to our approach as the ORANGE attributes may be already shared by some other entities depicted within our attribute bag, thus defeating our initial purpose of *expanding* an original authorization policy by taking into account previously-unconsidered attributes, as it is graphically depicted in Fig. 2.

Leveraging Support. Our approach also relies heavily on the concepts of support and confidence, which, as mentioned in Section 2, are core for properly selecting *interesting* associations within

a certain application domain. As shown in Algorithm 2, we leverage support in order to measure the degree of correlation between the green, orange and yellow attributes, thus estimating the size of the populations U_{S_r} and U_{C_r} described in Section 4.1. From the theory of association analysis [15], given two associations of the form A $\rightarrow$ C and A $\wedge$ B $\rightarrow$ C, support for A $\wedge$ B $\rightarrow$ C is at most as the support for A $\rightarrow$ C. Therefore, the support for an orange association of the form GREEN $\wedge$ ORANGE $\rightarrow$ DECISION=*True* is as most as the support of a green association of the form GREEN $\rightarrow$ DECISION=*True*, assuming the attributes listed in GREEN are the same for both. With this in mind, we want the support of a green and its respective orange association to be as close as possible, ideally, the same. In addition, we also want the support of our detected yellow associations to be as close to zero with respect to the support of the orange ones, in an effort to provide a quantitative measurement for minimizing potentially security vulnerabilities, as discussed in Section 4.1.

Leveraging Confidence. As it is also shown in Algorithm 2, we leverage confidence as a metric for the quality of all detected color-named associations. Referring back to Section 2, confidence of our green associations can be defined as c(GREEN $\rightarrow$ DECISION=*True*) = s(GREEN $\cup$ DECISION=*True*)/ s(GREEN). Consequently, in order to get a confidence level of 1.0, the values of s(GREEN $\cup$ DECISION=*True*) and s(GREEN) must be the same, that is, the number of records containing the green attributes is the same as the number of records containing both the green attributes as well as the positive access decision, which would then imply that the authorization policy from where the attribute bag was collected only grants access when the green attributes are present. In such a context, any value less than 1.0 would imply that s(GREEN) > s(GREEN $\cup$ DECISION=*True*), which may occur when there exists another rule in the policy granting access to the same resource by means of the same green attributes, e.g., redundant rules or when the policy enforcement mechanism exhibits some faulty behavior, such as denying an access request even when the green attributes, as stated by the original policy, are shown at request time. A similar situation may happen with the orange and yellow rules: any confidence value less than 1.0 may indicate an anomalous situation that may compromise the effectiveness of our approach.

5 EXPERIMENTAL EVALUATION

As mentioned in Section 1, we aim to provide experimental evidence that can support the implementation of our proposed approach in practice. With this in mind, we have developed a series of experiments with two main objectives:

First, we aim to prove the feasibility of our approach by showing that such correlated attributes, which are the key to our proposed policy mutation approach, do exist in datasets obtained from both real-world policies and accompanying attribute-based data. However, as stated in Section 2, the implementation of attribute-collecting infrastructures is still in its infancy. Therefore, to the extent of our knowledge, no publicly available datasets depicting a rich collection of attributes exists yet. To alleviate this problem, we perform an initial case study in which we leveraged a sample dataset containing both access logs and attribute-based information from Amazon, a major software company. Such dataset was

```
ACTION,TARGET,DATE,TIME,COMPANY,DEPT,JOB,LOC,DEC
rem,11044,Mar_2005,DURING_OH,3290,3302,3431,6,true
add,11044,Mar_2005,DURING_OH,3290,9910,4576,6,true
add,11044,Apr_2005,DURING_OH,3290,3302,3431,6,true
add,11044,Apr_2005,DURING_OH,3290,3861,3431,6,true
add,11044,Apr_2005,DURING_OH,3290,4956,3431,6,true
add,11044,Apr_2005,DURING_OH,3290,9910,4576,6,true
add,11044,Apr_2005,DURING_OH,3290,5374,2322,6,true
add,11044,Jul_2005,OUT_OF_OH,3290,3861,3431,6,true
add,11044,Jul_2005,OUT_OF_OH,3290,4956,3431,6,true
add,11044,Jul_2005,OUT_OF_OH,3290,3302,3431,6,true
add,11044,Jul_2005,OUT_OF_OH,3290,9910,4576,6,true
```

Figure 3: Sample records contained in the Amazon dataset.

obtained from enforcing an authorization policy over a large set of users and protected resources, and provides a good experimental ground as it contains data from access requests as well as data collected from user profiles, which effectively depict the real-time collection of attributes as intended for our approach.

Second, we aim to show that the detection of correlated attributes from a given attribute bag can be efficiently achieved. To this end, we conducted another case study in which we performed experiments with a set of synthetic policies derived from real-world environments, which depict a varying level of correlation and a varying number of records and attributes contained in the attribute bag. We also simulated a series of attack scenarios based on the model described in Section 3: we inspected the policies under study to *compromise* the attributes *granting* access to the most sensitive resources. Later, we artificially added a series of correlated attributes which were then used to create a series of mutated policies.

For our experiments, we utilized WEKA 3.7 [3], a tool set implementing several data mining techniques such as the association analysis used in our approach, and used Java and Python as our programming languages. With respect to association analysis, we utilized the *Apriori* algorithm as described in [12]. We performed our experiments in a MacBook Pro 2012 2.9 GHZ Intel Core i7 with 8 GB RAM and 1 TB HD. Our self-developed tools, including the ones providing support for data generation, correlation detection, and data preprocessing, are available upon request to the authors.

5.1 Case Study 1: A Large-scale Enterprise

Dataset Overview. For our first case study, we aim to show the feasibility of our approach by implementing it on top of existing access control mechanisms deployed in practice. To this end, we leveraged a dataset which depicts both access logs as well as user-based attribute information compiled by Amazon and other subsidiary companies [13]. This dataset was made available through the UC Irvine Machine Learning repository [11]. For the purposes of our experiment, we leveraged the user-based attribute information along with the access logs to produce an attribute bag as depicted in our approach. In addition, we assumed an authorization model depicting access control lists, granting access to resources based on a list of identifiers depicted by end-users. The dataset depicts 30,000 different users, 6,454 protected resources and 716,063 access log entries. Fig. 3 shows an excerpt of an attribute bag depicting our experiment: information from access logs, depicted by the first four attributes listed, was combined with attributes assigned

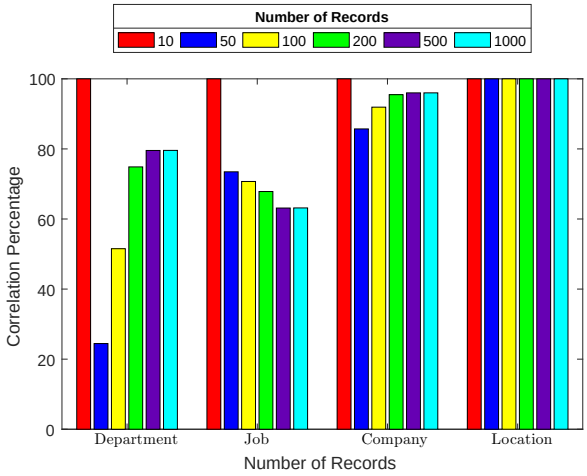

Figure 4: Experimental results for the Amazon dataset.

to end-users as depicted by the last six, e.g., attribute *TARGET* represents the resource being accessed and attribute *JOB* depicts the organizational job code (role) as held by an end-user. The result of the access request is depicted by the *DEC* (Decision) attribute. We also carried out an extensive preprocessing step tailored to *normalize* the attribute values in an effort to make them more suitable for analysis. As an example, the values depicted by attributes *DATE* and *TIME*, as depicted in Fig. 3, were normalized to literal values in such a way that they can be processed by our supporting framework WEKA.

Experiment Results. Our experiments included the *TARGET* and *DEC* attributes as our green ones. Following the approach described in Section 4.2, we were able to detect orange rules depicting a set of attributes correlated to those. As an example, Fig. 4 shows our experimental results for an attribute bag similar to the one depicted in Fig. 3. We were able to identify a strong correlation with the attributes *COMPANY* and *LOC* (Location), which may be due to the fact that resources may be accessed by end-users belonging to the same organizational units, which may also happen to be physically located within the same premises. Also, we detected a moderate correlation with respect to the *DEPT* (Department) and *JOB* attributes, which may indicate that several organizational positions have been granted access to the same resources.

During our experimental process, we were also able to obtain the following insights: First, we observed that our supporting framework, WEKA, originally detects a significant number of original associations that are then later parsed to obtain the green and orange ones as described in Section 4.3. This may have an influence on the overall runtime of our implementation, thus possibly affecting its deployment for real-time systems. Second, as expected, we also observed that varying levels of our proposed usability threshold would eventually result in different attributes being detected as correlated, e.g., the *JOB* attribute mentioned before. However, varying the safety threshold has no tangible effects due to the fact that the Amazon dataset depicts only *positive* access requests, e.g., requests that were always granted, therefore, no negative access

Table 1: Statistical composition of a set of sample policies.

Name	A's	Rules	A's/Rule	A's/Entity
Healthcare	10	9	3	3
Online Video	4	6	3	3
Project Mgmt	12	12	4	5
University	10	10	2	3

requests exist, thus turning the calculated safety index to zero in all experiments.

We also conducted an additional series of experiments by partitioning the original attribute bag as it was originally constructed into a series of *sliced* files from different sizes, each of them depicting only one protected resource, as it is shown in Fig. 3, which contains entries belonging to the same value for the *TARGET* attribute. In such experiments, we observed that the smaller the file in size, the more attributes it can detect as correlated, despite high coverage and safety thresholds, as it is shown in Fig. 4. That may be due to the fact that a smaller number of requests in the original dataset may either indicate a small homogeneous user population or, more likely, such a small number of requests may not accurately depict a representation of all the users that ultimately may have runtime access to the resources being analyzed.

Limitations. Even though the Amazon dataset provides a good experimental ground as it contains data from access requests as well as data collected from user profiles, still some limitations persist: first, as mentioned before, the original access logs depict only positive cases, thus turning the number of yellow associations detected to zero. Second, the produced attribute bag depicts no variations in the degree of correlation between attributes and access requests, as we have explored in the synthetic policies depicted in the next section. Finally, as mentioned before, despite the fact our sample dataset does not contain an extensive attribute bag as it may be optimal for the purposes of our approach, we believe our discovery of correlated attributes in real-life data provides significant evidence of the suitability of our approach to be implemented on top of future attribute-collection infrastructures such as the ones devised by NIST and described in Section 2.

5.2 Case Study 2: A Set of Synthetic Policies

Dataset Overview. For our second case study, we leveraged a sample access control policy developed using our running example detailed in Section 2 and Fig. 1, which is based on allowing access to the EHRs of a given patient to all staff members whose *Staff ID* attribute depicts a value that is contained within a predefined authorization list. Our first experiment was designed to test different levels of correlation between a single pair of green and orange attributes, e.g., the *Match IDs* and the *Location* attributes depicted in our running example. For such a purpose, we developed data generation tool that is able to produce datasets depicting customized attribute bags with a variable number of attributes and data records, as well as a variable level of correlation between two or more attributes. Using this tool, we produced different datasets varying the number of records and the level of correlation between the green and orange attributes, while keeping the usability and safety thresholds fixed to a value of 0.005 and 0.0, respectively. We also used the tool to produce attribute bags with a fixed number

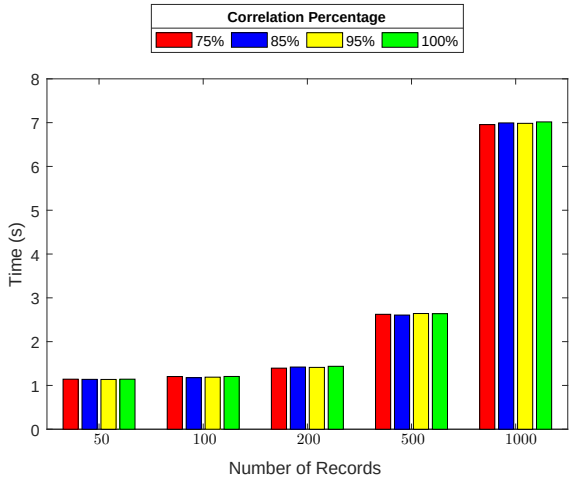

Figure 5: Experimental results for our running example.

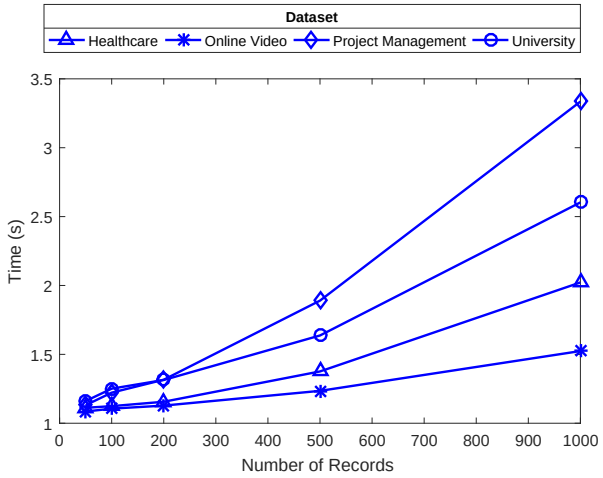

Figure 6: Results when varying the number of records.

of records, a fixed number of correlated attributes, a fixed value for the usability and safety thresholds as explained before, and a varying level of correlation as well as a changing number of non-correlated attributes. For the second dataset, For our experimental dataset, we leveraged a set of realistic policies developed in the context of the related work performed by Xu and Stoller [18], who strove to provide attribute-based authorization policies in the context of policy mining. The survey set contains the following policies: a *Healthcare* policy, which depicts a different setting in the context of EHRs as compared to our running example, an *Online Video* policy tailored for restricting the distribution of video contents to users based on their age and membership status, e.g., regular and premium users, a *Project Management* policy that provides access to project planning, scheduling and budgeting for a set of users based both on their organizational jobs as well as some other environmental constraints, and finally, a *University* policy, which allows for teaching-related duties, e.g., grade preparation and updating, to be governed by a set of rules based on institutional positions, such as students, staff and faculty members. Table 1 provides some statistics as depicted by such policies. For each policy, we obtained the number of attributes being defined, the number of rules, the approximate number of attributes used within each rule, as well as the approximate number of attributes being assigned to each entity within the application domain of the policy.

Simulated Attacks. We simulated a series of attacks depicting our model as described in Section 3: for each policy, we selected the rules that grant access to the most sensitive resources and operations, and simulated an attack on the attributes included in them. For instance, leveraging our running example as depicted in Fig. 1, we simulated an attack compromising the *Staff ID* attribute, allowing for attackers to deliberately manipulate such an attribute to meet the values depicted by EHR records, thus also altering the value of the *Match IDs* attribute and bypassing the authorization policy being enforced as a result. In such a scenario, adding a correlated attribute such as the *Access Location* one, as proposed in our approach, may allow for deterring an ongoing attack. Our experimental process also leveraged our data generation tool mentioned before to populate attribute bags depicting realistic access patterns and attribute-based data as required by the application domain of each surveyed policy. As an example, an attribute bag produced for the attack scenario just described may include a set of entries depicting a varying distribution for different values of the *Access Location* attribute, in an effort to evaluate the efficiency of our approach for detecting different levels of correlation between attributes.

Experiment Results. Our results, shown in Fig. 5, indicate that our implementation is able to detect the correlated attributes within a comparable response time independently of the level of correlation, only exhibiting an expected linear dependency with the number of records. Fig. 6 shows an instance of our experimental results when a series of attack scenarios such as the one just described have been carried out over our set of policies. As with the previous experiment, we varied the number of records included in the attribute bag while keeping the other parameters fixed, including a level of correlation of 95%. As with our sample running policy, our implementation can detect correlated attributes in a linear correspondence with the number of records being inspected. We observed a similar behavior in other instances of the same experiment.

6 RELATED WORK

Self-adaptive Systems. Venkatasubramanian et al. [17] and Bailey et al. [1] developed adaptive authorization models intended to dynamically modify policies. The former used an authorization model to adapt user privileges in response to system actions within emergency situations in Smart architectures whereas the latter developed a reactive system that analyzes abnormal user behavior and tightens or loosens the authorization policy accordingly, intended to prevent the risk of attacks originated by *insiders* within an organization. Our approach is similar in the ideology of an adaptive, preventive and proactive system, but is tailored to defeat attacks originated by *external* agents by means of further identifying

the entities that have authorized access rather than any traces of undesired or abnormal behavior.

Detecting Policy Violations. Association mining has been used in the context of access control in order to identify irregularities and violations in policy permissions or user accesses in authorization policies, as well as in the creation of policies based on logs of user behavior within the system. For instance, Parkinson et al. [14] leveraged such a technique in order to identify irregular file permissions in Microsoft's New Technology File System (NTFS) [6]. Moreover, Lee et al. [9] developed a framework that analyzes patterns of user behavior to identify anomalies for intrusion detection. In the same context, Treinen and Thurimella [16] provided so-called alarm streams based on associations to detect new attack rules that are consequently added to the intrusion detection rule list. We leverage these approaches as an inspiration for the one presented in this paper. However, our approach differs in that we strive to identify attributes that can lead to additional rules related to authorized accesses, e.g., the ones that can further identify entities having authorized access to resources, in order to successfully create *expanded* mutated policies, as compared to identifying user violations or permission anomalies, as depicted by such related work.

Policy Mining. In addition, Xu and Stoller [18] used an access control list to mine access rules in the automated development of an ABAC policy. In this method, permission tuples were broadened and merged to create comprehensive and generalizable rules that cover a larger number of permission relationships in the efforts of creating an established, fixed ABAC policy. However, opposed to this approach, our model focuses on finding more specialized, new attribute permission relationships to create unique rules that can be exchanged over time, expanding our original authorization policy. Furthermore, Bauer et al. [2] used association mining on access logs that display the history of user accesses in order to prevent access control mis-configurations by identifying and predicting valid user accesses policy changes to integrate into an authorization policy. Implementation of the policy changes relies on a system administrator, and part of the approach includes determining who to contact in the system in order to make the identified policy changes. Similarly, LeMay et al. [10] developed a constraint based approach to improve existing access control policies. Also relying heavily on a system administrator to implement the policy improvements, the model detects policy violations and formulates suggestions on policy improvements. Instead of a retroactive system that determines policy improvements based on past permission mis-configurations or violations, our approach strives to identify new rules in an automated fashion, reducing the amount of work expected from an administrator to carry out policy changes.

7 DISCUSSION AND FUTURE WORK

Meeting MTD Goals. Following the discussion on MTD presented in Section 2, our approach strives to reduce the probability of carrying out a successful attack such as the ones described in Section 3, by limiting the amount of time available for an attacker to exploit a compromised attribute. For such a purpose, we continuously mutate policies that leverage correlated attributes, such as the orange ones discussed above. This way, even when a small number of correlated attributes is detected, our approach helps produce several different mutated policies from those attributes, thus providing a framework in which such policies are interchangeably enforced at a random pace over time, as discussed at the end of Section 4.1

Trust Model. Furthermore, even when an attacker may be aware of our proposed approach, we believe the random selection of a subset of orange attributes may complicate the prediction of the next policy mutation. In addition, we also assume the subset of orange attributes cannot be compromised by an attacker, at least until the next policy mutation. We base such assumption on the fact that in case an attacker can potentially modify any attribute at will at any time (including both the green and orange ones), not only our approach can be circumvented, but also the original attribute-based policy (and any other policies) that may be in place for access mediation purposes. Further work may focus on incorporating a trust model for attributes, in such a way only the ones that are deemed as *highly* trusted within a certain context can be selected for policy mutation.

Calculating Thresholds. As mentioned in Section 4, the choice for our proposed usability and safety thresholds is left for security administrators to make. Based on the experiences collected during our experimental process, we believe administrators may benefit from statistical sampling techniques such as the ones depicted in Fig. 4, which show that the levels for strongly-correlated attributes stay within a comparable range no matter the number of entries contained in attribute bags. In addition, the choice of a safety threshold may be influenced by the attribute trust model described before: if highly-trusted attributes are used within a bag, a larger safety threshold can be in place.

Correlation Time. Following the discussion presented above, a periodical renewal of our proposed mutated policies may also allow the identification of changes in the way attributes are correlated over time, in such a way that attributes that may no longer be correlated with each other can be removed from a mutated policy. In a similar fashion, newly correlated ones may be introduced into our mutated rules/policies as well. This is due to the fact that attribute assignment, as done by the authorities introduced in Section 2.1, may change continuously over time, therefore affecting the way some attributes may appear as correlated to others for the purposes of our approach. Future work may focus on detecting such attribute-correlation periods in such a way that the next mutation cycle can be effectively determined to avoid existing mutated policies rejecting previously-granted requests because of unexpected changes in attribute correlations and assignments.

8 CONCLUSIONS

In this paper, we have presented a novel approach inspired by MTD that allows for authorization policies to be dynamically *expanded* to include additional *correlated* attributes that better identify the entities involved in access requests, potentially decreasing the chances of a successful attack against the original policies as a consequence. As shown by the provided experimental evidence, our approach can be successfully implemented on top of real-life scenarios. As of today, we are exploring lines of future work to enhance the suitability of our approach for deployment in practice.

ACKNOWLEDGMENTS

The authors would like to thank Marthony Taguinod for his valuable contributions towards this work, which was partially supported by a grant from the National Science Foundation (NSF-SFS-1129561), a grant from the Department of Energy (DE-SC0004308) and by a grant from the Center for Cybersecurity and Digital Forensics at Arizona State University.

REFERENCES

[1] Christopher Bailey, David W Chadwick, and Rogério De Lemos. 2011. Self-adaptive authorization framework for policy based RBAC/ABAC models. In *2011 IEEE Dependable, Autonomic and Secure Computing (DASC)*. IEEE, 37–44.

[2] Lujo Bauer, Scott Garriss, and Michael K Reiter. 2011. Detecting and resolving policy misconfigurations in access-control systems. *ACM Transactions on Information and System Security (TISSEC)* 14, 1 (2011), 2.

[3] Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explor. Newsl.* 11, 1 (Nov. 2009), 10–18.

[4] Hongxin Hu, Gail-Joon Ahn, and Ketan Kulkarni. 2012. Detecting and resolving firewall policy anomalies. *IEEE Dependable and Secure Computing* 9, 3 (2012), 318–331.

[5] Vincent C Hu, David Ferraiolo, Rick Kuhn, Adam Schnitzer, Kenneth Sandlin, Robert Miller, and Karen Scarfone. 2014. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication* 800 (2014), 162.

[6] Ewa Huebner, Derek Bem, and Cheong Kai Wee. 2006. Data hiding in the NTFS file system. *digital investigation* 3, 4 (2006), 211–226.

[7] Sushil Jajodia, Anup K Ghosh, Vipin Swarup, Cliff Wang, and X Sean Wang. 2011. *Moving target defense: creating asymmetric uncertainty for cyber threats*. Vol. 54. Springer Science & Business Media.

[8] Jing Jin, Gail-Joon Ahn, Hongxin Hu, Michael J Covington, and Xinwen Zhang. 2011. Patient-centric authorization framework for electronic healthcare services. *Computers & Security* 30, 2 (2011), 116–127.

[9] Wenke Lee, Salvatore J Stolfo, and Kui W Mok. 1999. A data mining framework for building intrusion detection models. In *Security and Privacy, 1999. Proceedings of the 1999 IEEE Symposium on*. IEEE, 120–132.

[10] Michael LeMay, Omid Fatemieh, and Carl A Gunter. 2007. Policymorph: interactive policy transformations for a logical attribute-based access control framework. In *Proc. of the 12th ACM Symposium on Access Control Models and Technologies*. ACM, 205–214.

[11] M. Lichman. 2013. UCI Machine Learning Repository. (2013). http://archive.ics.uci.edu/ml.

[12] Bing Liu, Wynne Hsu, and Yiming Ma. 1998. Integrating Classification and Association Rule Mining. In *Fourth International Conference on Knowledge Discovery and Data Mining*. AAAI Press, 80–86.

[13] Ken Montanez. 2016. Amazon Access Samples Data Set. (2016). http://archive.ics.uci.edu/ml/datasets/Amazon+Access+Samples.

[14] Simon Parkinson, Vassiliki Somaraki, and Rupert Ward. 2016. Auditing file system permissions using association rule mining. *Expert Systems with Applications* 55 (2016), 274–283.

[15] Ramakrishnan Srikant and Rakesh Agrawal. 1996. Mining quantitative association rules in large relational tables. In *ACM SIGMOD Record*, Vol. 25. ACM, 1–12.

[16] James J Treinen and Ramakrishna Thurimella. 2006. A framework for the application of association rule mining in large intrusion detection infrastructures. In *Recent Advances in Intrusion Detection*. Springer, 1–18.

[17] Krishna K Venkatasubramanian, Tridib Mukherjee, and Sandeep KS Gupta. 2014. CAAC–An Adaptive and Proactive Access Control Approach for Emergencies in Smart Infrastructures. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)* 8, 4 (2014), 20.

[18] Zhongyuan Xu and Scott D Stoller. 2015. Mining attribute-based access control policies. *IEEE Dependable and Secure Computing* 12, 5 (2015), 533–545.