Proactive Risk Assessment for Preventing Attribute-Forgery Attacks to ABAC Policies

Carlos E. Rubio-Medrano
Arizona State University
crubiome@asu.edu

Luis Claramunt
Arizona State University
lclaramu@asu.edu

Shaishavkumar Jogani
Arizona State University
sjogani@asu.edu

Gail-Joon Ahn*
Arizona State University
gahn@asu.edu

ABSTRACT

Recently, the use of well-defined, security-relevant pieces of runtime information, a.k.a., *attributes*, has emerged as a convenient paradigm for writing, enforcing, and maintaining authorization policies, allowing for extended flexibility and convenience. However, attackers may try to bypass such policies, along with their enforcement mechanisms, by maliciously *forging* the attributes listed on them, e.g., by compromising the attribute *sources*: operative systems, software modules, remote services, etc., thus gaining unintended access to protected resources as a result. In such a context, performing a proper risk assessment of authorization policies, taking into account their inner structure: rules, attributes, combining algorithms, etc., along with their corresponding sources, becomes highly convenient to overcome *zero-day* vulnerabilities, before they can be later exploited by attackers. With this in mind, we introduce *RiskPol*, an automated risk assessment framework for authorization policies, which, besides being inspired by well-established techniques for vulnerability analysis such as *symbolic execution*, also introduces the very first approach for proactively assessing risks in the context of a series of attacks based on unintended attribute manipulation via forgery. We validate our approach by resorting to a set of case studies we performed on both *real-life* policies originally written in the English language, as well as a set of policies obtained from the literature, which show not only the convenience of our approach for risk assessment, but also reveal that some of those policies are vulnerable to attribute-forgery attacks by just compromising one or two of their attributes.

CCS CONCEPTS

• **Security and privacy → Access control**; • **Software and its engineering → Risk management**;

KEYWORDS

Attribute-based Access Control; Risk Management, Attribute Forgery; Policy Bypassing; Zero-Day Vulnerabiities

ACM Reference Format:
Carlos E. Rubio-Medrano, Luis Claramunt, Shaishavkumar Jogani, and Gail-Joon Ahn. 2020. Proactive Risk Assessment for Preventing Attribute-Forgery Attacks to ABAC Policies. In *Proceedings of the 25th ACM Symposium on Access Control Models and Technologies (SACMAT '20), June 10–12, 2020, Barcelona, Spain.* ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3381991.3395615

1 INTRODUCTION

Contemporary software systems have increased in size and complexity, evolving from small, monolithic, closed, and proprietary infrastructures into a series of big, dynamic, distributed, heterogeneous, and highly-interconnected modules that, besides providing their intended functionality as efficiently as possible, also relieve developers from fully implementing code from scratch, allowing for them to focus instead on leveraging existing solutions to better meet their needs. As an example, there is nowadays a plethora of third-party *application programming interfaces* (APIs), web services, dynamic libraries, and so on that are provided by a considerable amount of independently-run *sources*, e.g., companies, institutions, government agencies, etc., thus depicting an emerging trend that is likely to stay in the foreseeable future.

In such a context, authorization policies may certainly benefit from leveraging security-related information that is provided by these sources to write rich and flexible policies that, besides meeting very specific needs, may also be evaluated and enforced in more efficient ways. With this in mind, *attribute-based access control* (ABAC) [17] has recently gained the attention of both academia and industry as a convenient way to specify, store, evaluate and enforce authorization policies by representing this security-related information as well-defined constructs known as *attributes*. However, despite the inherent benefits introduced by this emerging approach, some security concerns still exist, as modern software infrastructures are known to be the target of attacks that leverage existing and previously-unknown security vulnerabilities [12]. Moreover, *zero-day* attacks are now becoming more frequent, leaving security officers with little or no time to respond, thus having devastating consequences [5]. In the context of ABAC policies, attackers may try to leverage vulnerabilities in third-party software to deliberately

*Also with Samsung Research.

SACMAT '20, June 10–12, 2020, Barcelona, Spain

ACM ISBN 978-1-4503-7568-9/20/06…$15.00
https://doi.org/10.1145/3381991.3395615

modify attributes at will, thus allowing them to compromise sensitive protected resources by either gaining unintended access to them or by denying rightful users from accessing them at runtime.

In order to address these concerns, this paper proposes *RiskPol*, a trust-based, collaborative, and automated risk assessment framework for protecting ABAC policies, which is inspired in *symbolic execution* [19], a well-established technique for vulnerability discovery and analysis. In *RiskPol*, *trust* is modeled as a qualitative perception on the security state of a given software system, allowing for numerical scores representing *trust* to be assigned to attribute sources. Later, these scores are transfered to the attributes they provide, thus effectively *simulating* the runtime evaluation (execution) of the ABAC policy, and allowing for a consolidated score to be calculated, taking into account its inner *structure*, e.g., policy rules, combining algorithm, etc. This way, as the score assigned to the source of an attribute A changes, e.g., as a result of security vulnerabilities or attacks on the source itself, so does too the score assigned to A, thus ultimately affecting the overall risk scores of all the policies attribute A is listed on.

Overall, this paper makes the following contributions:

- First, we provide a definition for a series of attacks based on *forging* attributes by means of deliberate and unintended manipulation, which may be launched to bypass existing policy enforcement mechanisms, ultimately forcing the runtime evaluation of an ABAC policy to a result convenient to the attacker, e.g., allowing unintended access to protected resources.
- Second, we introduce the *policy-trusting* problem, which involves the assessment of the risks involved for a given policy or a set of policies in the presence of detected vulnerabilities in the attribute generation process, which may deviate in the aforementioned attribute-forgery attacks.
- Third, we describe an approach inspired in symbolically executing a given set of ABAC policies, taking as an input the trust scores of the attributes listed on them, obtaining an overall trust score for individual policies or policy sets, which is then used to accurately assess risks in the context of the attacks just mentioned.
- Finally, we present the results of different case studies involving both a set of policies collected from *real-life* ABAC enterprises, which allowed us to properly relate attributes, their corresponding sources, and their impact on assessing security risks for ABAC policies.

2 BACKGROUND

Defining Attributes. In ABAC, an authorization request is granted upon the satisfaction of constraints, a.k.a., rules, involving *attributes*: properties, characteristics, or traits of subjects, objects, and even environment conditions that are relevant under a given security context [17]. For the purposes of this paper, we abstractly define attributes as 3-tuples with the following composing elements: a *datatype*, which defines both the nature of the data and range held by the attribute, e.g., String, Integer, etc., a *name* or *identifier*, which uniquely identifies the attribute within a given ABAC implementation, and a set of *values*, all of them within the range defined for the corresponding *datatype*. As an example, <*String*, `OS.name`,

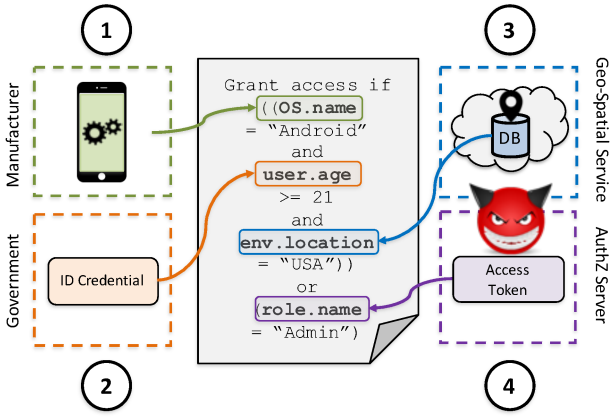

Figure 1: An ABAC policy depicting attributes from different sources. When evaluating a policy, attribute `OS.name` is provided by a device manufacturer, e.g., by means of a OS native call (1). Attribute `user.age` may be obtained from an ID credential issued by a local government (2). Also, the `env.location` attribute may be retrieved from a remote Geo-Spatial service (3). Finally, the `role.name` may be obtained from an access token released by an authorization server (4), which, if targeted by an attribute-forgery attack, may compromise the policy as a whole.

{"Android"}> denotes an attribute containing the name of the operative system running in a mobile device. Attributes are leveraged by *policy makers*, who are in charge of crafting policies by establishing relationships between attributes, *access entities*, e.g., end-users and protected resources, and access rights, a.k.a., *permissions*.

Attribute Sources. According to the U.S. National Institute of Standards and Technology (NIST) [17], dedicated infrastructures may be introduced in the foreseeable future allowing for attributes to be defined, created, and assigned to access entities. Such infrastructures, hereafter referred in this paper as *sources*, may be in turn deployed by different independently-run organizations such as companies, government agencies, non-profit corporations, etc., and may be implemented as operative system modules, dedicated application software, remote services, etc. This way, a given source may provide different attributes, and may be run by a single or a conglomerate of organizations in a collaborating scheme. In addition, a given organization may run different sources at once. This way, leveraging attributes from distinct sources may greatly increase the flexibility of ABAC, e.g., easier policy specification and enforcement: no need to manually assign attributes to entities, no need for entities to hold many different attributes at once.

Defining Trust and Risk. We leverage the definition of trust provided by Gambetta [14]: *"Trust (or, symmetrically, distrust) is a particular level of the subjective probability with which an agent will perform a particular action, both before [we] can monitor such action (or independently of his capacity of ever to be able to monitor it) and in a context in which it affects [our] own action"*. In the context of ABAC policies, such definition may include a perception on the overall security state of the attribute creation and assignment processes (actions) as carried on by each source (agents). This includes any supporting software and hardware, as well as any business

logic that allows for the source to create and assign attributes to access entities. Such a perception may also include the way security guidelines and best practices are implemented within the organizational domains defined by the organizations running the sources. We also leverage the definition of risk as stated by Vaughn et al. [34]: *"Risk is the probability that a particular threat will exploit a particular vulnerability of the system."* As hinted in Section 1, such a definition in the context of ABAC policies may be extended to the probability of attackers (threats) exploiting security vulnerabilities in the attribute creation and assignment infrastructures depicted by the sources (systems). In addition, we also consider the probability that, once a given attribute source has been compromised, attackers may try to manipulate its attributes at will to compromise a set of ABAC policies.

Running Example. Fig. 1 presents a sample ABAC policy restricting access to a mobile application to end-users who are 21 years or older of age, are using a mobile phone running the Android OS, and are physically located in the United States. Also, access is granted to a development engineer identified by the role '*Admin*' for the purposes of testing, debugging, and control. In such a policy, attributes are obtained from different organizational sources, each of them implementing its own attribute creation and assignment infrastructure, which is in turn protected by an independently-run security domain. During policy evaluation time, attributes may be effectively retrieved from those sources and used for policy evaluation, e.g., a *policy information point* (PIP) [24].

3 PROBLEM STATEMENT

Assumptions. For the purposes of this paper, we make the following assumptions: first, we assume attackers perform an initial *reconnaissance* phase in which they can collect sufficient information about their targeted ABAC policies, e.g., attributes, rules, as well as the combining algorithms and the policy evaluation order. Second, we also assume the evaluation and enforcement infrastructures of ABAC policies stay out of reach and cannot be compromised by attackers, which can only resort to the *attribute-forgery* techniques to be discussed below. Finally, in this paper, we consider a subset of the well-known *eXtensible Access Control Markup Language* (XACML) [24], due to its maturity and deployment in practice for expressing ABAC policies. Listing 1 provides a sample XACML encoding featuring our running example as described in Fig. 1.

Attribute-Forgery Techniques. In the context of attribute-based policies, an attribute whose value can be deliberately modified without proper consent from its originating source may not provide strong security guarantees, as attackers may be allowed to modify the attribute's value at will to meet the requirements defined in a given policy, thus effectively bypassing it in unintended ways. Even in locally-run domains, attributes may be the subject of such attacks, e.g., changing file and system attributes such as names, current time, location, etc., as a result of the unintended actions carried out by dedicated malware agents. With this in mind, following the definition provided in Section 1, attackers may try to compromise attributes as follows: first, attackers may try to forge the attribute entirely, crafting the *datatype*, *name*, and *value* components before presenting it to a given policy evaluation engine. Second, attackers may try to manipulate the *value* component of an existing

Listing 1: Our Running Example Policy in XACML.

```
<?xml version="1.0">
<Policy PolicyId="SamplePolicy"
 RuleCombiningAlgId="deny-unless-permit">
  <Target/>
  <Rule Effect="Permit"><Target>
      <AnyOf><AllOf>
        <Match MatchId="string-equal">
        <AttributeDesignator AttributeId="OS.name"/>
        <AttributeValue>Android</AttributeValue>
        </Match>
      </AllOf></AnyOf>
      <AnyOf><AllOf>
        <Match MatchId="string-equal">
        <AttributeDesignator AttributeId="env.location"/>
        <AttributeValue>USA</AttributeValue>
        </Match>
      </AllOf></AnyOf>
      <AnyOf><AllOf>
        <Condition>
          <Apply
           FunctionId="integer-greater-than-or-equal">
          <AttributeDesignator AttributeId="user.age"/>
          <AttributeValue>21</AttributeValue>
          </Apply>
        </Condition>
      </AllOf></AnyOf>
  </Target></Rule>
  <Rule Effect="Permit"><Target>
      <AnyOf><AllOf>
        <Match MatchId="string-equal">
        <AttributeDesignator AttributeId="role.name"/>
        <AttributeValue>Admin</AttributeValue>
        </Match>
      </AllOf></AnyOf>
  </Target></Rule>
</Policy>
```

attribute at will, provided the modifications fall within the range defined by the *datatype* component. Third, attackers may try to manipulate attributes by compromising their creation and assignment infrastructures. As an example, referring back to Fig. 1, a dedicated malware may try to intercept native OS calls such that the value of the `OS.name` is changed. Moreover, attackers may also try to compromise the remote Geo-Spatial server providing the `env.location` attribute, such that it always returns a location within the United States despite the actual location of the end-user.

Attribute-Forgery Attacks. In the context of ABAC policies, we now introduce different attack models intended to subvert the access decision result rendered by a policy evaluation process.

- First, we introduce the A_{Permit} attack, which attempts to force the evaluation of an ABAC policy P to the *permit* result, thus granting unintended access to protected resources. This attack assumes there exists at least one rule within P that evaluates to *permit*, and also assumes the rule combining algorithm in place allows for such a decision to be ultimately delivered.
- Second, we introduce the A_{Deny} attack, which, conversely, is intended to force the evaluation of P to the *deny* result, thus allowing attackers to deny legit access to the resources mediated by P, thus effectively performing a *denial-of-service* (DoS) attack. We also assume there exists at least one rule within the target

policy that evaluates to *deny*, and a rule combining algorithm in place allows for such a result to be produced.

- Third, we also consider the A_{Indet} attack, which is intended to force the evaluation of P to the *indeterminate* result, in an attempt to remove P from the authorization decision process, following the XACML multi-policy evaluation scheme defined in [24], thus possibly forcing the evaluation of an alternative policy Q that happens to be more convenient for the attacker, e.g., a policy enlisting another set of attributes under the attacker's control. For this attack, we assume there exist at least two policies P and Q that are evaluated in sequential order as a response to an access request. Also, we assume the combining algorithm for P allows for the *indeterminate* decision to trigger the evaluation of policy Q as a consequence.

The Policy-Trusting Problem. Despite the inherent benefits of multiple-sourced attributes for attribute-based policies, previous work in the literature [17] [24] commonly assumes all existing attribute sources are fully trusted all the time. However, such an assumption may not be always feasible in practice, as modern ABAC infrastructures may rely on a heterogeneous and independently-run set of attribute sources as discussed in this paper, which may be the target of dedicated attacks tailored to disrupt their internal cyber-infrastructures. In such a context, the *policy-trusting* problem involves allowing policy makers and security officers to maintain a perception on the security state of the attribute sources they leverage for their ABAC policies. Later, such a perception should be then taken into account when crafting ABAC policies, such that only attributes from *good-standing*, trusted, sources are used, allowing for policy makers to properly assess the risks involved when the security state of a given attribute source is perceived to have deteriorated at a given moment of time.

Assessing Risks for ABAC Policies. With all this in mind, risk assessors may need to properly assess the risks involved for a given set of ABAC policies when an attribute, or a set of attributes, has been compromised by means of attribute-forgery techniques. Referring back to Section 2, risk can be understood as the probability that a particular policy-based attack will be carried out against an ABAC policy given a particular vulnerability, e.g., a forged set of attributes, has been detected. In such a context, risk assessors may be concerned with following questions:

Q-1 **Given a set of *compromised* attributes *A* listed in an ABAC policy *P*, what is the risk that a successful policy-based attack can be carried out against *P*?.** Referring to our running example, risks assessors may want to know if the A_{Permit} attack can be carried out against our sample policy if the `user.age` attribute has been compromised, i.e., by forging the credential containing it to depict a value greater or equals to 21. As depicted in Fig. 1 and Listing 1, compromising such an attribute in isolation may not allow for the A_{Permit} to take place, as the structure of our sample policy, by means of an AND operator, requires two other attributes, namely the `OS.name` and the `env.location` to be compromised as well in order to gain access to the mobile application.

Q-2 **Given a set of *uncompromised* attributes *U* listed in *P*, what is the attribute set *U'* ⊆ *U* that will increase the risk of *P* failing for any of the policy-based attacks?** Following our running example, risks assessors may want to know what attributes may make the A_{Permit} attack more likely. In such a case, compromising the `role.name` attribute, i.e., by forging the access token containing it to depict the '*Admin*' value, may allow for such an attack to take place.

Q-3 **Given that *P* is likely to fall for a policy-based attack *X*, are there any *structural* modifications to *P*, e.g., adding/removing attributes and / or rules, such that the risk of *X* successfully targeting *P* is diminished?** Following our previous example, compromising the `role.name` attribute may make the A_{Permit} attack more likely to succeed. Possible modifications of our sample policy may include removing such attribute completely, thus making the OR operator unnecessary, and simplifying the policy as a result.

In the next section, we present an approach focused on question Q-1 , helping risk assessors to accurately estimate an answer to it. In addition, in Section 7, we discuss how our approach may help in the development of solutions for questions Q-2 and Q-3, which are left for future work.

4 *RISKPOL*: TRUST-BASED RISK ASSESSMENT FOR ABAC POLICIES

In order to provide support for solving the problems just discussed, we now present *RiskPol*, an automated framework that allows for both policy makers and security officers to become *risk assessors* for the policies under their control, such that security incidents can be properly addressed and mitigated. In the rest of this section, we elaborate on the intrinsics of our proposed approach, which is graphically depicted in Fig. 2.

4.1 Determining Scores for Attributes

Relating Trust and Risk. Following the definitions introduced in Section 2, within our proposed *RiskPol* approach, trust is modeled as a perception on the security state of the attribute creation and assignment processes as carried out by sources. In addition, risk is conceived as a perception on the likehood of attackers exploiting security vulnerabilities found in such attribute creation and assignment processes, such that the attribute-forgery attacks also discussed in Section 3 can eventually take place. Therefore, within *RiskPol*, a high degree of trust on either attribute sources, the attributes themselves, as well as on ABAC policies, has a direct correspondence with a low level of perceived risk, meaning that there is a low likelihood that an attack on an ABAC policy will take place. Conversely, a low level of trust is perceived as a high level of risk, meaning that an attack is highly likely and a risk assessment procedure, by the means to be discussed in this section, is recommended.

Modeling Trust. Following the definition just described, in *RiskPol*, trust is modeled as a numerical value to be defined in the context of a given implementation, which may maintain a consistent numerical scale, e.g., a mathematical total order over a set, to allow for calibrating and comparing different values of trust between distinct sources. As an example, a sample trust scale may include values in the set {0, 1}, being 1 the score indicating *complete* trust and being 0 the one denoting no trust at all (distrust). In such a setting, a value of 0 denotes a lesser value of trust with respect

to 1, e.g., 0 <1. As just mentioned, implementations of our *RiskPol* approach may also rely on their own trust scales that better fit their specific needs, e.g., introducing intermediate values for different degrees of trust.

Attribute-level Scores. With this in mind, trust in attribute sources can be represented by a *source-level* score, which can be initially assigned subjectively by individual risk assessors, as shown in Fig. 2, which happen to have an educated view of the current security state of such a source, e.g., the existence and/or absence of vulnerabilities. This way, attributes in a given ABAC policy can be trusted as much as the trust score assigned to their corresponding sources. As shown in Definition 1, an attribute scores can be directly obtained from the score assigned to its source infrastructure.

Definition 1.
Score(Attribute) = Score(SOURCE(Attribute))

Referring back to our running example shown in Fig. 1, the device manufacturer, which provides the `OS.name` attribute, should be trusted to properly retrieve the value of the OS running on the device by means of a dedicated kernel-level service that can be queried through a native OS call. Initially, such a source may receive a trust value of 1. However, if such a source is eventually found to be affected by a newly-discovered vulnerability, e.g., it is possible to change the name of the OS as retrieved by the native call, risk assessors may change their perception of trust with respect to such a source as a result, e.g., reducing it to 0. Later, in Section 7, we discuss an alternative approach for initially assigning the initial source-level scores, which involves a set of collaborating third-parties, potentially allowing for extended reliability and convenience.

4.2 Calculating Scores for Policy Rules.

Rule-level Scores. As mentioned in Section 1, within *RiskPol*, trust scores for ABAC policies are calculated by leveraging their *inner* structure, e.g.,their rule-combinatorial algorithm, e.g., *deny-overrides*, the logical operators on each rule, as well as the number of attribute-based rules they contain, to intelligently combine the scores obtained for each attribute, as just described before, into a consolidated *policy-level* one. We start our discussion detailing how the scores for each policy rule can be obtained, we then continue describing how to combine such rule-level scores into a consolidates policy-level one, and then finalize this section by showing how the scores from different ABAC policies can be combined into a single *multi-policy-level* score.

Processing Policy Rules. Policy rules encoded in XACML policies can be expressed in *disjunctive normal form* (DNF), which clearly separates each rule/condition independently and provides a convenient abstract representation of the internal structure of each policy, allowing for each attribute listed in an original policy rule to appear only once in its corresponding DNF formula, thus providing a convenient representation for accurately calculating risk for the purposes of our approach. Initially, we parse XACML policy rules into an intermediate representation in the form of *conjuctive normal form* (CNF) by leveraging the `<AnyOf>`and `<AllOf>`predicates on attributes. For instance, in the XACML representation of our running example shown in Listing 1, each attribute located inside such constructs is added inside an independent formula component, subsequently known as *DNF-Terms*, which are then linked together

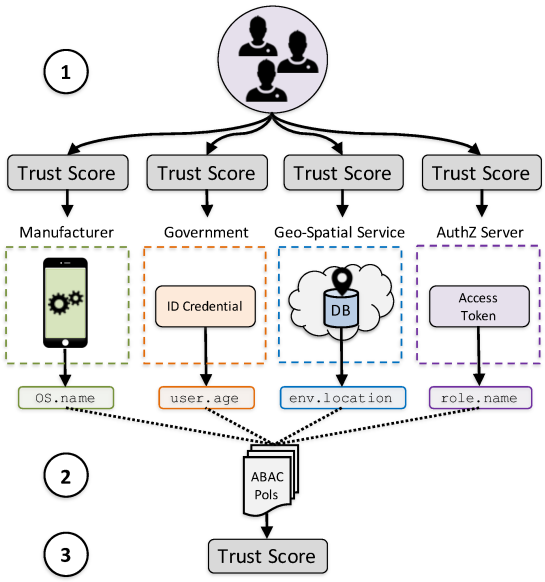

Figure 2: In *RiskPol*, initial trust scores for attribute sources are determined by risk assessors (1). Later, such scores are forwarded to the attributes the sources provide (2), and are combined using a mathematical model, ultimately creating a trust score to estimate risk (3).

to form a CNF formula by means of the ∧ operator. Later, our implementation converts CNF representations into DNF by leveraging the well-known De Morgan's laws [26]. For instance, our running example can be then listed as a DNF formula of the form ((`OS.name`) ∧ (`user.age`) ∧ (`env.location`)) ∨ (`role.name`).

Definition 2.
Score(DNF-Term) = max(Scores(ATTRS(DNF-Term)))

Once each rule policy has been converted into a DNF representation, an intermediate trust score is obtained for each DNF-Term following Definition 2: the auxiliary functions ATTRS and *Scores* (not shown) return a set containing the trust scores of each attribute listed in a given DNF-Term, following the approach depicted in Definition 1. Once such a set of scores has been calculated, the maximum value among them is returned as a result.

Definition 3.
Score(Rule) = min(Scores(DNF-TERMS(Rule)))

Subsequently, the trust score for a policy rule is obtained by calculating the minimum score among the trust scores obtained for each of its composing DNF-Terms, as shown in Definition 3. With that in mind, the auxiliary functions DNF-TERMS and *Scores* retrieve both the DNF-Terms themselves as well as their corresponding scores following Definition 2.

Assessing Risks for Policy Rules. In *RiskPol*, the use of the minimum and maximum functions in Definitions 2 and 3 follows an approach in which a policy/rule is as trusted (i.e., as *risky*) as its less trusted (i.e., its *riskier*) rule/attribute. As mentioned earlier, attributes inside DNF-Terms are joined together using AND operators. Therefore, in order to compromise a DNF-Term, e.g., forcing it

to be evaluated to *true*, all of the attributes inside of the DNF-Term must be compromised. Following such a reasoning, the maximum function in Definition 2 selects the score of most trusted attribute, i.e., the one that may be the most difficult to compromise. In a similar fashion, DNF-Terms inside policy rules are joined together by means of OR operators. Therefore, Definition 3 selects the minimum trust score among all the DNF-Terms inside a policy rule, as compromising a single one of them may be enough to compromise the enclosing policy rule, and, the ABAC policy as a whole.

4.3 Calculating Scores for Policies.

Policy-level and Multi-Policy-level Scores. Once the scores for each of the rules contained within an ABAC policy have been calculated, the overall *policy-level* score can be obtained by taking such *rule-level* scores along with the rule combining algorithm implemented by the policy, following the description of the XACML rule combining algorithms as presented in [24]. In a similar fashion, the scores obtained for a set of ABAC policies can be combined into a single *multi-policy-level* score by combining their corresponding *policy-level* scores along with their policy combining algorithm, as shown in Definitions 4 and 5. In both procedures, the auxiliary function ALG retrieves the combining algorithm of each policy or set of policies.

Definition 4.
Score(Policy) = combine(Scores(RULES(Policy)), ALG(Policy))

Definition 5.
Score(Policies) = combine(Scores(Policies)), ALG(Policies))

Policy-level Score Calculation Algorithm. The process of trust score calculation for XACML policies is shown in Algorithm 1. As an initial step, the input policy is parsed by an auxiliary routine (not shown), retrieving the policy's rule combining algorithm (if any)[1], as well as the sets of policy rules whose decision result is either *deny* or *permit* (line 1). The retrieval of a trust score for each rule proceeds in line 4 as follows: initially, both sets are then merged into a single set (line 4) for convenience purposes. Then, each rule r is converted to an abstract DNF representation as described before, which is made up of different *DNF-terms*, e.g., each sub-formula that is not glued together by means of an OR operator.

As an example, the DNF representation of our running example ((`OS.name`) ∧ (`user.age`) ∧ (`env.location`)) ∨ (`role.name`) can be further divided into the DNF-terms ((`OS.name`) ∧ (`user.age`) ∧ (`env.location`)) and (`role.name`). The trust score for an DNF-term is calculated by obtaining the maximum value among the trust scores defined for the attributes listed on it (lines 8-15). Later, the minimum trust score obtained among all the DNF-terms is used as the final trust score for policy rule r (lines 16-21), separating between the rules whose final decision result is *deny* and the ones whose result is *permit*. Finally, the auxiliary algorithm `combine_scores()` combines the scores of all policy rules and policy sets in a process we describe next.

Combining Policy Rule Scores. Table 1 provides an abstract representation of the the `combine_scores()` algorithm, which implements the logic for combining *rule-level* and *policy-level* scores,

[1]In case no combining algorithm is specified for a given XACML policy, our approach assumes the *Deny-Overrides* algorithm by default.

Data: An XACML Policy *POL*.
Result: A Set of Trust Scores for *POL* depicting the Policy Attacks described in Section 3.

```
1  (Comb − Alg, R_Deny, R_Permit) =
     parse_XACML_Policy(POL);
2  S_Deny = {};
3  S_Permit = {};
4  R = R_Deny ∪ R_Permit;
5  foreach rule r in R do
6      Term_Scores = {};
7      DNF_TERMS = convert_to_DNF(r);
8      foreach term in DNF_TERMS do
9          ATTR_Scores = {};
10         ATTRS = get_Attribute_IDs(term);
11         foreach attr in ATTRS do
12             ATTR_Scores = ATTR_Scores ∪
                 get_Attribute_Score(attr);
13         end
14         Term_Scores ∪ max(ATTR_Scores);
15     end
16     if r in R_Deny then
17         S_Deny ∪ min(Term_Scores);
18     end
19     if r in R_Permit then
20         S_Permit ∪ min(Term_Scores);
21     end
22 end
23 return combine_scores(S_Deny, S_Permit, Comb − Alg);
```

Algorithm 1: Trust Score Calculation for XACML Policies.

which is in turn based in the description of XACML combining algorithms as shown in [24]. Taking as an input the sets S_{Permit} and S_{Deny} of trust scores defined for the rules depicting the *Permit* and *Deny* results respectively, and the rule combining algorithm defined for the policy under processing, the procedure retrieves a consolidated score for each of the attacks defined in Section 3, namely, A_{Permit}, A_{Deny}, and A_{Indet}. As an example, the two rules contained within of our running example policy shown in Listing 1 will be included within the policy's S_{Permit} set, as such an evaluation decision has been set for each of them, whereas the policy's S_{Deny} set would be empty. Next, the `combine_scores()` algorithm obtains the trust score of the policy rule that would allow for each attack to be carried out successfully, thus following the approach implemented by *RiskPol* in which a policy is trusted as much as its *less-trusted*, i.e., *riskier*, rule. For instance, in order for an attacker to carry on the A_{Deny} attack on an XACML policy implementing the *Deny Overrides* combining algorithm, thus ultimately forcing the *Deny* result, at least one of the policy rules depicting the *Deny* decision must be compromised. Therefore, the `combine_scores()` algorithm retrieves the score of the less-trusted rule depicting such a decision, which can be in turn found as the minimum trust score value contained within the S_{Deny} set. A graphical depiction of this strategy is shown in Fig. 3.

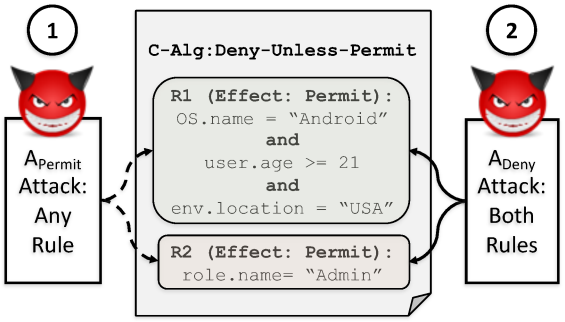

Figure 3: Attacking our running example policy: given the *Deny-Unless-Permit* combining algorithm, attackers must compromise *any* of the listed rules to successfully perform the A_{Permit} attack (1). Subsequently, an attacker must compromise *both* rules to perform the A_{Deny} attack (2).

4.4 Assessing the Risks of Attacks.

Following the reasoning just explained, we now describe how to combine the scores of policy rules and sets into a consolidated one for risk assessment. For readability purposes, we hereafter refer to either a policy rule, or a policy as a whole (in the case of policy sets), as *targets*. Also, we refer to the successful runtime evaluation of a target T, which has been obtained by means of any of the attribute-forgery attacks discussed in this paper, as the *firing* of T. Conversely, in the opposite direction, the prevention of a successful evaluation of a target T is referred as *neutralizing* T. In practice, a target may be neutralized by forcing the policy evaluation engine to deliver the *Not Applicable* effect by means of an attribute-forgery attack. Also, a target T is also said to be *indeterminate* if the policy evaluation engine under attack will retrieve the *Indeterminate* effect as a result of a runtime error induced by means of an attribute-forgery attack. As an example, some XACML *policy decision point* (PDP) implementations [1] will retrieve *Indeterminate* as a result of a parsing error when processing attributes at runtime. Such an error can be in turn induced by means of attribute manipulations as it is described in this paper.

- A_{Permit} **for D-O or F-A Targets.** For the A_{Permit} attack to succeed when the *Deny-Overrides* (D-O) algorithm is used, an attacker must neutralize all targets depicting the *Deny* effect, and then make sure at least one target depicting the *Permit* effect is fired. Therefore, `combine_scores` algorithm first selects the minimum value contained within the S_{Permit} set, which corresponds to the *less-trusted*, a.k.a., the *riskier*, target depicting the *Permit* result, i.e., the easiest for an attacker to fire. Later, the algorithm selects the maximum value of contained within the S_{Deny} set, which corresponds to the *most-trusted* target depicting the *Deny* result, i.e., the harder for an attacker to neutralize. The same procedure is followed for the *First-Applicable* (F-A) combining algorithm, as the `combine_ scores` procedure assumes an scenario in which targets that are combined using the F-A algorithm are evaluated in a random order.
- A_{Permit} **for P-O or D-U-P Targets.** For the A_{Permit} attack and targets featuring the *Permit-Overrides* (P-O) algorithm, an attacker must fire at least target depicting the *Permit* effect, regardless of the evaluation result of all other ones. Therefore, the `combine_scores` algorithm selects the lowest score within the S_{Permit} set, which corresponds to the *less-trusted* target being considered. The same strategy is applied for the *Deny-Unless-Permit* (D-U-P) algorithm.
- A_{Permit} **for P-U-D Targets.** For the A_{Permit} attack and targets depicting the *Permit-Unless-Deny* (P-U-D) algorithm, the attacker must make sure all targets depicting the *Deny* effect are neutralized. In that regard, the effort to effectively neutralize all *Deny* targets may be as much as neutralizing the less-riskier one. Thus, the `combine_scores` algorithm returns the largest score contained within the S_{Deny} set.
- A_{Deny} **for D-O or P-U-D Targets.** For targets depicting the *Deny-Overrides* (D-O) or the *Permit-Unless-Deny* (P-U-D) combining algorithms, an attacker must fire at least one target depicting the *Deny* effect, regardless of any other target under consideration. Therefore, the effort required may be as hard as defeating the less-trusted target. Thus, the `combine_scores` algorithm selects the minimum score within the S_{Deny} set.
- A_{Deny} **for P-O or F-A Targets.** In order for the A_{Deny} attack to succeed in targets depicting the *Permit-Overrides* (P-O) or the *First-Applicable* (F-A) algorithms, all targets depicting the *Permit* effect must be neutralized, and at least one target depicting the *Deny* effect must be fired. With that in mind, the `combine_scores` algorithm selects the maximum score between all the targets depicting the *Permit* effect, which corresponds to the one that is the hardest to neutralize. Also, the minimum score within the S_{Deny} set, which corresponds to the easiest target featuring *Deny* to fire, is also selected. From these two intermediate results, the algorithm selects the maximum score, which, once again, represents the action that is likely harder for the attacker to achieve.
- A_{Deny} **for D-U-P Targets.** For the A_{Deny} attack to succeed in targets featuring the *Deny-Unless-Permit* (D-U-P), all targets depicting the *Permit* effect must be neutralized, regardless of any other ones under consideration. Therefore, the `combine_scores` algorithm retrieves the maximum value in the S_{Permit}

Table 1: The *combine_scores* Auxiliary Algorithm. D-O stands for *Deny-Overrides*, P-O for *Permit-Overrides*, D-U-P for *Deny-Unless-Permit*, P-U-D for *Permit-Unless-Deny*, and F-A for *First-Applicable*.

Alg.	A_{Permit}	A_{Deny}	A_{Indet}
D-O	max(min(S_{Permit}), S_{Deny})	min(S_{Deny})	max($S_{Deny} \cup S_{Permit}$)
P-O	min(S_{Permit})	max(min(S_{Deny}), S_{Permit})	max($S_{Deny} \cup S_{Permit}$)
D-U-P	min(S_{Permit})	max(S_{Permit})	N/A
P-U-D	max(S_{Deny})	min(S_{Deny})	N/A
F-A	max(min(S_{Permit}), S_{Deny})	max(min(S_{Deny}), S_{Permit})	max($S_{Deny} \cup S_{Permit}$)

set, which in turn corresponds to the hardest target featuring *Permit* that must be neutralized by the attacker.

- A_{Indet} **for D-O, P-O or F-A Targets.** In order for the A_{Indet} attack to succeed on targets featuring the *Deny-Overrides* (D-O), the *Permit-Overrides* (P-O), as well as the *First-Applicable* (F-A) algorithms, combining algorithm, at least one target must be forced to the *Indeterminate* effect, and all targets featuring the *Deny* (D-O) and the *Permit* (P-O) effects must be neutralized. That being said, the `combine_scores` retrieves the maximum value within the S_{Deny} and S_{Permit} sets, as the effort invested by the attacker to force the *indeterminate* corresponds to the hardest target among all of the ones under consideration. Finally, it must be noticed that according to the description of the policy combining algorithms featured in [24], the A_{Indet} attack is not possible when the *Deny-Unless-Permit* (D-U-P) and *Permit-Unless-Deny* (P-U-D) algorithms are in use.

Alternatives for the F-A Algorithm. An alternative scenario for the *First-Applicable* combining algorithm may consider a rule evaluation order, other than random, that is known to the attacker. With that in mind, alternative calculation formulas may be listed as:

- A_{Permit}: max(max(First(S_{Deny})), S_{First_Permit}),
- A_{Deny}: max(max(First(S_{Permit})), S_{First_Deny}),
- A_{Indet}: max($S_{Deny} \cup S_{Permit}$),

where First(S) obtains the very first score within the set labeled as S, and S_{First_Permit} and S_{First_Deny} denote the trust score assigned to the first rule known to the attack that depicts either the *Permit* or the *Deny* result, respectively. Finally, our `combine_scores()` algorithm assumes the input XACML policies are *well-formed*, that is, policies depicting a given rule combining algorithm have at least one rule listing the algorithm's decision result. As an example, well-formed policies depicting the *Deny-Overrides* algorithm have at least one rule depicting the *Deny* decision result. Handling of policies failing to exhibit this feature is left as an decision for implementers of our *RiskPol* approach.

5 EXPERIMENTAL EVALUATION

In order to provide evidence of the suitability of *RiskPol* for accurately assessing risks on ABAC policies of varying nature, structure, and attribute sources, we first present a case study depicting ABAC policies written in XACML that were collected from a series of previous work in the literature. Later, we present a second case study involving policies written in the English language that were also collected from online sources, which were then manually translated into an XACML representation for further analysis. For each case study, we present descriptions of the experimental datasets, the methodologies used, as well as the results obtained, which focused mostly on addressing Question Q-1 as depicted in Section 3. We implemented all supporting code for *RiskPol* in Java, leveraging a Dell Laptop running Windows 10, with 16 GB of RAM and 1128 GB of HD storage. Such implementation is available for distribution upon request to the authors.

Moreover,in the rest of the experiments discussed in this paper, we assumed a trust scale in the set defined by {0,1}, where 0 denotes the absence of trust and 1 denotes the complete presence of it.

Table 2: Results for the Natural 2 Policy.

Attribute ID	A_{Permit}	A_{Deny}	A_{Indet}
Role	1	1	-1
Action	1	1	-1
Report	0	0	-1
Project	0	0	-1
Portfolio	0	0	-1

Conversely, a score of 0 for any of the A_{Permit}, A_{Deny} and A_{Indet} attacks denotes that carrying on such an attack is possible under the attribute trust configuration scenario, whereas a score of 1 denotes the attack is highly unlikely.

5.1 Real-Life/ Literature XACML Policies.

Dataset Description. Tables 8 and 9, which for spacing reasons are shown in Appendix A, present a summary of the XACML policies that we collected from different online and literature sources for the purposes of our case study. As an example, policies HGABAC [3], NGAC [13], Natural 1 and 2 [33], Example 7, 11, 13 and 15[30], and PolTree [21] were extracted from previous work in the literature focused on ABAC, whereas policies such as KMarket [32] and the Continue [20] where obtained from public repositories of production software. On average, they contained from 2 to 6 policy rules each, and each policy rule contained from 2 to 6 different attributes.

Methodology. Initially, we inspected each of the original policy files for compliance with the subset of the XACML language we are considering as a part of this work, as mentioned in Section 3. For some policies whose syntactic structure would slightly differ, e.g., using constructs not considered in our approach, a semantics-preserving transformation was attempted. As an example, policies containing all attribute-based constraints within the `<Target>` construct only, were transformed into a new semantically-equivalent policy containing the same constraints within a `<Rule>` construct, which supported by *RiskPol*, similar to the exemplary syntax shown in Listing 1. Later, we focused mostly on simulating a situation as the one defined by Question Q-1, we started by simulating one compromised attribute, and continue with sets of two and three attributes from the ones listed on each policy. Finally, we simulated the A_{Permit}, A_{Deny} and A_{Indet} attacks previously discussed in Section 3, by setting different trust scores for each source of the aforementioned attributes and ultimately leveraging our implementation of *RiskPol* to calculate Policy-level scores for each policy in our sample set, following the procedures described in Algorithm 1 and Table 1.

Results. As a result of our experiments, we noticed that several of the surveyed policies are vulnerable to attribute-forgery attacks in case one, two or three attributes listed on them are compromised. As an example, Table 2 shows an excerpt of the experimental results for policy Natural 2 [33]. When a single attribute, namely `Report`, `Project`, or `Portfolio` was compromised, the A_{Permit} and A_{Deny} attacks were possible. In addition, as shown in Table 3, different attribute sets of size 2, e.g., {Role, Action} and {Action, Report}, also allow for the A_{Permit} and A_{Deny} attacks to become feasible. That

Table 3: Results for the Natural 2 Policy.

Attr 1	Attr 2	A_{Permit}	A_{Deny}	A_{Indet}
Role	Action	0	0	-1
	Report	0	0	-1
	Project	0	0	-1
	Portfolio	0	0	-1
Action	Report	0	0	-1
	Project	0	0	-1
	Portfolio	0	0	-1
Report	Project	0	0	-1
	Portfolio	0	0	-1
Project	Portfolio	0	0	-1

Table 4: Results for the PPS_pcMember_rc Policy Set.

Policy	Attr 1	Attr 2	A_{Permit}	A_{Deny}	A_{Indet}
1	role	action-type	0	1	0
2	role	action-type	0	1	0
3	role	UserId	1	0	0
4	role	action-type	0	1	0

may be due to the presence of several OR (|) operators in the original policy rule R1:

```
R1: ((((Role, "CPM Advisor") & (Action, "access")) |
(Report, "CP&E Reports") | (Project, "Views") |
(Portfolio, "Portfolio Reporting Views")))
```

In an additional example, Table 4 shows an experiment in which attribute sets of size 2 were taken as compromised for the PPS_pcMember_rc policy set. In a similar fashion, the combination of attribute `role` with several other attributes in such a policy ultimately resulted in the three attribute-forgery attacks to become possible. Similarly, Table 5 shows the results of an experiment featuring policy HGABAC in which attribute sets of size 3 were simulated compromised. This time, the A_{Permit} attack was possible in several attribute combinations, whereas the A_{Deny} attack was only possible for the `Role`, `Action`, and `Type` attributes.

Overall, we observed the following patterns in our surveyed policies: first, we noticed that the presence of many OR (|) operators in the CNF representation of such policies may allow for attribute-based attacks to become possible. Second, we noticed the absence of such operators, and the presence of policy rules having attributes joined together by means of the AND (&) operator has the opposite effect, as the aforementioned attacks become more difficult to perform. Such a case can be observed in policy PolTree, which is shown in Table 8 in Appendix A. With those insights, we believe a plausible mitigation strategy may include reducing the number of OR operators and increasing the number of attributes within a policy that are joined together using the AND operator. We further elaborate on how to effectively implement such an idea in practice as a part of our discussion for addressing Question Q-3 in Section 7, which details our plans for future work.

Table 5: Results for the HGABAC Policy.

Attr 1	Attr 2	Attr 3	A_{Permit}	A_{Deny}	A_{Indet}
Role	Action	Type	0	0	-1
Department	Action	Type	0	1	-1
Java	Action	Type	0	1	-1
C	Action	Type	0	1	-1
C++	Action	Type	0	1	-1

Table 6: Results for the NC-2 English-to-XACML Policy.

Name	A_{Permit}	A_{Deny}	A_{Indet}
(network, "state")	1	1	1
(network, "agency")	1	1	1
(location, "local")	1	1	1
(location, "remote")	1	1	1
approved	1	1	1
(security, "policies")	1	0	1
(work, "request")	1	0	1
negative-impact	1	0	1
(apparatus, "computer")	1	1	1
(network, "access")	1	1	1

5.2 English-Language Policies.

Dataset Description. In this case study, we resorted to a series of *real-life* authorization policies contained in documents written in the English language, most of them in PDF format, which were obtained from different online sources. Overall such documents depict a series of authorization policies and constraints over a variety of application domains: network, web-based, and system-level administration, as well as physical security of enterprises. Among their sources, we can cite the National Health Service (NHS) of the United Kingdom [22], the State Government of North Carolina [31], among others. Table 7 shows convenient representation of such policies, their source documents, along with the subsets of attributes that, if compromised, allow for attribute-forgery attacks to succeed, as we will discuss next.

Methodology. Initially, documents were obtained through a series of online searches depicting the keywords *authorization*, *access policy* and *access control* on a popular search engine. Next, for each obtained document, a preliminary analysis was conducted to determine if it contained text describing authorization policies. As an example, sentences of the form: "*Access to restricted and/or highly restricted data shall be restricted to authorized individuals who require access to the information as part of their job responsibilities*" [31] were categorized as potential authorization policies. Next, a list of attributes, along with their hypothetical attribute sources were identified for each candidate document, and a translation from the English language to XACML format was then carried out. A simplified version of our translation process is show in Table 7. Finally, we proceeded with our experimental analysis by resorting to a technique similar to the one discussed previously for our first case study: we simulated the A_{Permit}, A_{Deny} and A_{Indet} attacks by

Table 7: A Set of English-Language Policies Collected for Experimental Purposes.

Policy Info	CNF - Rule Effect	A_{Permit}	A_{Deny}	A_{Indet}
NC-1 (P-U-D) [31]	R_1: ((ID, "root") | (ID, "admin")) & ((end, "disable") | (change, "credentials")) - Deny R_2: (unemployed) | (needless) - Deny R_3: ((end, "system accounts") | (remove | transfer) | ((change, "info") | (change, "permission"))) & (notify) - Deny	{ID, change unemployed, notify}	{unemployed} {needless}	N/A
NC-2 (D-O) [31]	R_1: ((network, "state") & (network, "agency")) & ((location, "local") | (location, "remote")) & (approved) - Permit R_2: ((security, "policies") | (work, "request") | (neg-impact)) - Deny R_3: ((apparatus, "computer") | (network, "access control system") & (!approved) - Deny	N/A	{security} {work} {neg-impact}	{security, apparatus, approved}
GPRC (D-O) [16]	R_1:(software, "mechanism") & (ID & (security, "password") & unique) - Deny R_2: ((software, "system") & (software, "application")) & ((security, "2-factor authentication") & lock) - Permit	N/A	{software, ID, security, unique}	{software, ID, security, unique}
SANS (D-O) [29]	R_1: (encrypted & ((security, "pass-phrase") & (secure)) - Permit R_2: (ID & (security, "password")) & (secure) - Permit R_3: ((ID, "InfoSec") | (ID, "manager")) & (!approved) - Deny	N/A	{ID, approved}	{ID, approved}
NHS (D-U-P) [22]	R_1: (ID & approved) & ((ID & unique) & private) & ((change, "password") & approved) - Permit	N/A	N/A	N/A

setting different trust scores for each attribute source and ultimately calculating Policy-level scores for each policy in our set.

Results. Overall, the results obtained confirm the previous ones observed for our first case study: policy structure has considerable influence on Policy-level *RiskPol* scores. In addition, we observed that policies containing a varied number of DNF-terms, which are mix together by means of AND operators, obtain better Policy-level scores, ultimately results in less-risky policies. Table 6 contains the sample results for the NC-2 policy shown in Table 7. As with the results featured for our previous case study, each row in Table 6 shows the result of manually setting the *RiskPol* source-level score of each attribute to 0 whereas the rest of the attributes in the policy stay with a value of 1. In most cases, the structure of such a policy prevents the occurrence of our discussed policy attacks when a single attribute gets a low source-level score. Only the A_{Deny} attack was possible when modifying the `security`, `work`, and `negative-impact` attributes, due to the structure of policy NC-2, e.g., the use of OR (|) operators in rule R_2 as well as the *Deny-Overrides* combining algorithm.

6 RELATED WORK

Risk Models for Authorization. Previous work has focused on developing assessment models for dynamic permission assignment based on the current risk state of the system. This way, before assigning a permission to an end-user, the system evaluates the risk of doing so based on recent (possibly real-time) information [11] [9]. Following this paradigm, Ni et al. [23] introduced an approach leveraging a fuzzy engine for estimating risk before releasing a permission to a given user. In the context of role-based access control (RBAC) [28], Bijon et al. [4] proposed an extension to the core RBAC model that includes a so-called *risk-threshold* as a part of RBAC user *sessions*, allowing for a given session to be dropped in case the calculated risk value goes beyond a predefined threshold, thus potentially preventing user-based abuse of already-authorized permissions. Similarly, Chen et al. [7] presented an approach leveraging XACML as the policy language for expressing RBAC policies, extending the language with specific construct to model risk as well. In the context of attribute-based authorization, Kandala et al. [18] presented an approach combining the concept of risk and ABAC, developing a model based on UCON [25] extensions. Alternatively, Choi et al. [10] presented a risk assessment framework for ABAC in the context of medical information systems. Our *RiskPol* approach is also intended to incorporate a perception on the security state of a given system before an authorization policy can be evaluated and enforced, such that the system becomes aware of recent events, e.g., vulnerabilities or incidents, which may have an impact on the overall authorization process. However, our approach is intended to protect policies themselves from attacks that may originate from compromised attributes, instead of evaluating risk for each user in isolation before or during the time the authorization process takes place. In our approach, a risk assessment performed over a whole policy may affect all users being served by it, e.g., by applying one of the proactive actions discussed in Section 7. Therefore, our policy-level approach differs from previous approaches on risk assessment that calculate risk at the user-level only.

Credential-based Risk Analysis. Risk assessment approaches have been also proposed for credential-based access control. As an example, Chapin et al. [9] introduced a trust management logic that provides formal risk assessment by associating risk levels with authorization elements, allowing for tolerable levels of risk to be rigorously enforced. In addition, Goodrich et al. [15] provided a solution for an authenticated dictionary for attribute-based credentials,

allowing for attribute sources to collectively publish information to a common repository, which can be later queried by other parties through the network. While these approaches have influenced our *RiskPol* approach, ABAC comprises a wider model that may include credentials as an implementation subset. As an example, credentials may be used to securely communicate attributes between sources and policy evaluation engines. Also, they may provide proof of a correct attribute-user assignment as stated by the corresponding attribute source. Moreover, ABAC provides a wider theoretical model in which attributes may be also retrieved by other implementation strategies. As an example, while certain attributes may benefit from a cryptography-based protection while in transit, some implementations, e.g., an XACML PIP module, may simply retrieve the attribute directly from sources by implicitly trusting both the communication channel, e.g., a native OS call, as well as its originating source (the OS), as it is depicted in the `OS.name` attribute included in our running example shown in Fig. 1.

Distributed Risk Assessment and Attribute Dictionaries. Finally, Aven [2], introduced a risk assessment framework modeling both security and safety in the concept of information technology infrastructures. Information about security incidents is generated by a set of trusted partners and distributed actively to remote enterprise, thus allowing for the fast and efficient dissemination of security-related issues. Our *RiskPol* approach is inspired by such a concept as it also relies on strong collaborative settings for sharing information that can be valuable for risk assessment. However, our approach goes a step further by providing means for each risk assessor to automatically calculate policy-level scores by leveraging the information previously-shared by the risk valuators, thus providing the foundations for an automated approach for efficient and expedited risk assessment.

7 DISCUSSION AND FUTURE WORK

Collaborative Risk Assessment. Following the approach presented in Section 4, we expect risk assessors to work together to determine proper scales for modeling trust in the context of different domains or collaborative communities. As an example, security officers handling risk within a Medical context may want to have their own perception of trust based in the attributes and the corresponding attribute sources they consume within their own domain. For such a purpose, we have envisioned a scheme in which collaborators may rely on each other for determining changes in trust scores as well as for distributing those updated scores to other members of the community. In such a scheme, risk assessors may delegate the initial assessment of source-level scores to a series of third-party, well-deputed organizations known as *risk valuators*, which, besides having updated knowledge on the attribute creation and assignment infrastructures implemented by sources, may also be able to promptly assess when a recently-discovered vulnerability, or a security incident, may have an impact on the overall trust perception of a given source. In practice, risk assessors may be allowed to choose n different valuators so they can implement a *k-out-of-n* strategy for score updates, e.g., allowing for k valuators to suggest a change in a given score before such a change is actually implemented. This way, risk assessors and valuators may engage in a *stock market* model [8], in which information on updated source-level scores is distributed in proactive, expedite, and continuous ways, thus possibly improving the overall risk assessment process as a result.

Limitations. Despite our promising results, the inherent benefits of our approach for risk assessment, and the potential for handling the attribute-forgery attacks we have presented as a part of this paper, we have also identified the following limitations to our approach. As shown in Section 5, the current version of our approach handles only a relevant subset of the vast XACML syntax, as it is described in [24]. However, as shown in Section 5.1, our approach can cover the syntax constructs depicted by the policies we have collected for our first case study, which showcases the applicability of our approach for handling real-life authorization scenarios. Future work may focus on providing support for extended XACML syntax as well as other policy combining algorithms described in [24] which were not supported as a part of this work. In addition, we plan to continue our work on the generation of synthetic policies which can be generated in different sizes and with varying structural characteristics, at the same time the effectiveness of our proposed approach is not affected.

Addressing Question Q-2. As mentioned in Section 3, Question Q-2 is concerned with determining what attributes within the set A of attributes listed in a target policy set may be more suitable for successfully performing an attribute-forgery attack. A naive approach would include considering using *RiskPol* on all subsets of attributes contained within the powerset 2^A. However, such an approach may be quite inefficient in the presence of a large set of target policies and / or in the case of A being of a large size. Moreover, considering all subsets within such powerset may not be feasible in practice. For instance, compromising the entire subset $A' = A$ may be quite difficult if multiple *independently-run* attribute sources exist for all attributes in A. Based on the experimental results shown in Section 5, future work may focus on detecting subsets within 2^A of size 1 to 3, as compromising such a number of attributes may be enough to successfully launch an attack.

Addressing Question Q-3. As mentioned in Section 3, Question Q-3 refers to the situation in which a policy set is deemed as risky. A straight-forward approach would include shutting down access to all the resources being guarded by a *now-risky* policies. However, such a conservative scheme may have negative consequences when it comes to user experience, as commonly-accessed resources may be unexpectedly denied as a result. In addition, such a strategy gets complicated by the fact continuous monitoring by security officials may be needed to restore the shutdown resources back to service. An alternative approach would include augmenting it with additional attributes and rules, originated from other more trusted sources, such that the overall trust score goes back to a value above the risk threshold. As an example, previous work has explored the possibility of enhancing ABAC policies with additional attributes that are automatically retrieved from users [27].

Advanced Risk Assessment Models. In addition, as stated in Section 2, a risk assessment model must consider the probability that, once a given attribute source has been compromised by attackers, they will try to specifically target the set of ABAC policies protecting the resources of a given organizational entity. In our

RiskPol approach, as described in Section 4, we have taken a rather conservative approach by assuming that every single policy whose enlisted attributes are compromised may automatically become at risk, that is, the probability of attackers attacking any of those policies is the same. With this in mind, future risk assessment models may consider adding extra parameters to allow for assessors to introduce the probability that attackers may try to specifically target their organizational policies, assuming a previous security vulnerability or incident within their corresponding attribute sources has been found, thus producing a customized assessment as a result.

8 CONCLUSIONS

In this paper, we have discussed the problem of assessing risks for ABAC policies in the presence of potential attacks based on unintended attribute manipulation and forgery. In order to address this problem, we have presented *RiskPol*, a novel and intuitive solution for properly assessing the risks involved in trusting the attribute creation and assignment infrastructures as provided by heterogeneous sources, such that the risks of potential attribute-forgery attacks can be effectively assessed and mitigated. Finally, we believe that *RiskPol* can serve as a strong foundation for implementing a proactive framework involving the distribution, collection, and processing of source-based trust scores for the purposes of automated risk assessment, allowing for the proper reaction and handling of zero-day vulnerabilities, as well as the subsequent mitigation of potential security incidents, among a series of collaborative partners.

ACKNOWLEDGMENTS

This work was partially supported by grants from the National Science Foundation (NSF-ACI-1642031, NSF-IIS-1527268) and by a grant from the Center for Cybersecurity and Digital Forensics at Arizona State University.

REFERENCES

[1] AT&T. 2020. XACML 3.0. https://github.com/att/XACML. (2020).
[2] T. Aven. 2007. A unified framework for risk and vulnerability analysis covering both safety and security. *Reliability Eng. and Sys. Safety* 92, 6 (2007), 745–754.
[3] S. Bhatt, F. Patwa, and R. Sandhu. 2017. ABAC with Group Attributes and Attribute Hierarchies Utilizing the Policy Machine. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC '17)*. ACM, 17–28.
[4] K. Z. Bijon, R. Krishnan, and R. Sandhu. 2012. *Risk-Aware RBAC Sessions*. Springer Berlin Heidelberg.
[5] L. Bilge and T. Dumitras. 2012. Before We Knew It: An Empirical Study of Zero-day Attacks in the Real World. In *Proc. of the 2012 ACM Conf. on Computer and Communications Security (CCS '12)*. ACM, 833–844.
[6] D. Brossard, G. Gebel, and M. Berg. 2017. A Systematic Approach to Implementing ABAC. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC'17)*. ACM, 53–59.
[7] Liang C., Luca G., and Timothy J. N. 2013. XACML and Risk-Aware Access Control. In *Proc. of the Int. Workshop on Security in Info. Sys. (ICEIS 2013)*. 66–75.
[8] K. Campbell, L. A. Gordon, M. P. Loeb, and L. Zhou. 2003. The Economic Cost of Publicly Announced Information Security Breaches: Empirical Evidence from the Stock Market. *Journal of Computer Security* 11, 3 (April 2003), 431–448.
[9] P. Chapin, C. Skalka, and X. S. Wang. 2005. Risk Assessment in Distributed Authorization. In *Proc. of the 2005 ACM Workshop on Formal Methods in Sec. Eng. (FMSE '05)*. ACM, 33–42.
[10] D. Choi, K. Dohoon, and Seog P. 2015. A Framework for Context Sensitive Risk-Based Access Control in Medical Information Systems. *Computational and Mathematical Methods in Medicine 2015* 265132 (2015).
[11] N. Dimmock, A. Belokosztolszki, D. Eyers, J. Bacon, and K. Moody. 2004. Using Trust and Risk in Role-based Access Control Policies. In *Proc. of the 9th ACM Symp. on Access Control Models and Technologies (SACMAT '04)*. ACM, 156–162.
[12] K. A. Farris, S. R. McNamara, A. Goldstein, and G. Cybenko. 2016. A preliminary analysis of quantifying computer security vulnerability data in the wild. (2016), 9825 - 9842 pages.
[13] D. Ferraiolo, R. Chandramouli, R. Kuhn, and V. Hu. 2016. Extensible Access Control Markup Language (XACML) and Next Generation Access Control (NGAC). In *Proc. of the 2016 ACM International Workshop on Attribute Based Access Control (ABAC '16)*. ACM, 13–24.
[14] D. Gambetta. 1988. Can We Trust Trust?. In *Trust: Making and Breaking Cooperative Relations*. Basil Blackwell, 213–237.
[15] M. T. Goodrich, M. Shin, R. Tamassia, and W. H. Winsborough. 2003. *Authenticated Dictionaries for Fresh Attribute Credentials*. Springer Berlin Heidelberg, Berlin, Heidelberg, 332–347.
[16] GRPC. 2019. IT Access Control and User Access Management Policy. https://www.gprc.ab.ca/about/administration/policies/fetch.php?ID=320. (2019). [Online; accessed Sep-23-2019].
[17] V. C Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone. 2014. Guide to attribute based access control (ABAC) definition and considerations. *NIST Special Publication* 800 (2014), 162.
[18] S. Kandala, R. Sandhu, and V. Bhamidipati. 2011. An Attribute Based Framework for Risk-Adaptive Access Control Models. In *2011 Sixth International Conference on Availability, Reliability and Security*. 236–241.
[19] J. C. King. 1976. Symbolic Execution and Program Testing. *Comm. ACM* 19, 7 (July 1976), 385–394.
[20] Margrave, An API for XACML Policy Verification and Change Analysis. 2020. Margrave Continue Example. http://www.margrave-tool.org/v1+v2/margrave/versions/01-01/examples/continue/. (2020).
[21] R. Nath, S. Das, S. Sural, J. Vaidya, and V. Atluri. 2019. PolTree: A Data Structure for Making Efficient Access Decisions in ABAC. In *Proc. of the 24th ACM Symposium on Access Control Models and Technologies (SACMAT '19)*. ACM, 25–35.
[22] NHS Digital. 2019. Access Control Sample Policy. https://webarchive.nationalarchives.gov.uk/20180307183605/https://digital.nhs.uk/cyber-security/policy-and-good-practice-in-health-care/access-control/example-policy. (2019).
[23] Q. Ni, E. Bertino, and J. Lobo. 2010. Risk-based Access Control Systems Built on Fuzzy Inferences. In *Proc. of the 5th ACM Symp. on Information, Computer and Comm. Security (ASIACCS '10)*. ACM, 250–260.
[24] OASIS Standard. 2013. eXtensible Access Control Markup Language (XACML) Version 3.0. (2013, January 22). http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.html. (2013). [Online; accessed September-23-2019].
[25] J. Park and R. Sandhu. 2004. The UCONABC Usage Control Model. *ACM Trans. Inf. Syst. Secur.* 7, 1 (Feb. 2004), 128–174.
[26] W. Rautenberg. 2009. *A Concise Introduction to Mathematical Logic* (3rd ed.). Springer Publishing Company, Incorporated.
[27] C. E. Rubio-Medrano, J. Lamp, A. Doupé, Z. Zhao, and G-J. Ahn. 2017. Mutated Policies: Towards Proactive Attribute-based Defenses for Access Control. In *Proc. of the 2017 Workshop on Moving Target Defense (MTD '17)*. ACM, 39–49.
[28] R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman. 1996. Role-Based Access Control Models. *Computer* 29, 2 (Feb. 1996), 38–47.
[29] SANS. 2019. SANS Consensus Policy Resource Community. https://www.sans.org/security-resources/policies/network-security/pdf/remote-access-policy. (2019). [Online; accessed Sep-23-2019].
[30] D. Servos and S. L. Osborn. 2017. Current Research and Open Problems in Attribute-Based Access Control. *ACM Comput. Surv.* 49, 4, Article 65 (Jan. 2017).
[31] State of North Carolina. 2019. Access Control Policy SCIO-SEC-301-00. https://files.nc.gov/ncdit/documents/Statewide_Policies/SCIO_Access_Control.pdf. (2019). [Online; accessed Sep-23-2019].
[32] The Asela Project. 2020. XACML Examples. https://svn.wso2.org/repos/wso2/people/asela/xacml/sample/kmarket/resources/lib/. (2020).
[33] R. C. Turner. 2017. Proposed Model for Natural Language ABAC Authoring. In *Proc. of the 2nd ACM Workshop on Attribute-Based Access Control (ABAC'17)*. ACM, 61–72.
[34] R. B. Vaughn, R. Henning, and A. Siraj. 2003. Information assurance measures and metrics - state of practice and proposed taxonomy. In *Proc. of the 36th Annual Hawaii Int. Conf. on System Sciences, 2003*. 10.

APPENDIX A

Finally, we present a summary of the policies we collected as a part of our experimental procedures described in Section 5. For each policy, we include its CNF representation, its main policy combining algorithm, as well as a reference to its source. In addition, we list the subsets of attributes listed on each policy that, if compromised, allow for each attribute-forgery attack to succeed.

Table 8: A Set of XACML Policies Collected from the Literature and Online Sources.

Policy Info	CNF - Rule Effect	A_{Permit}	A_{Deny}	A_{Indet}
HGABAC (D-U-P) [3]	R1: ((Role, "IT_Manager") | (Department, "IT")) & ((Action, "Read") & (Type, "Network")) -Permit R2: ((Role, "DevOps_Manager") | (Java, "True")) & ((Action, "Read") & (Type, "Dev")) - Permit R3: ((Role, "DevOps_Manager") | (Java, "True") | (C, "True") | (C++, True)) & ((Action, "Read") & (Type, "Deploy")) - Permit R4: (((Role, "CTO")) & ((Action, "Read") & (Type, "General"))) - Permit	{Role, Action, Type} {Department, Action, Type}	{Role, Action, Type}	N/A
NGAC (P-O) [13]	R1: (((Action, "read") | (Action, "write")) & ((Role, "doctor") | (Role, "intern")) & ((Diff_Locations, "True"))) - Deny R2: ((Role, "intern") & (Action, "write")) - Deny R3: ((Action, "read") | (Action, "write")) & ((Role, "doctor") & (PatientStatus, "crititcal")) - Permit	{Action, Role, PatientStatus}	{Action, Role, PatientStatus}	{Action, Role, PatientStatus}
Metamodel (D-U-P) [6]	R1: ((role, "manager") & (actionId, "view") & (objType, "rec")) - Permit R2: (((role, "employee")) & ((actionId, "view") | (actionId, "edit")) & ((objectType, "record")) & ((department, "own "))) - Permit R3: ((((role, "employee") & (actionId, "edit"))) & ((owner, "employee")) & ((status, "draft"))) - Permit R4: ((((role, "manager") & (actionId, "publish"))) & ((status, "final")) & (((owner, "employee") & (subordinate, ""employee")))) - Permit	{role, actionId, objType}, {role, actionId, owner, status}	N/A	N/A
Natural 2 (D-U-P) [33]	R1: ((((Role, "CPM Advisor") & (Action, "access")) | (Report, "CP&E Reports") | (Project, "Project Views") | (Portfolio, "Portfolio Reporting Views"))) - Permit	{Report}, {Project}, {Portfolio} {Role, Action}	{Report} {Project}, {Portfolio} {Role, Action}	N/A
Example 7 (D-U-P) [30]	R1: (((ID, "manager")) & ((Time, "09:00:00") & (Time, "17:00:00")) & ((Location, "office"))) - Permit R2: (((ID, "clerk")) & ((Sys_load, "low") | (ID, "manager"))) - Permit	{ID}	{ID, Time, Location}	N/A
Example 11 (D-U-P) [30]	R1: (((Role, "Guest")) & ((Time, "09:00:00") & (Time, "17:00:00"))) - Permit R2: (((Role, "Adult")) & ((Age, "20")) & ((Country, "Japan") | (Country, "New Zealand"))) - Permit R3: (((Role, "Adult") & (Role, "American")) & ((Age, "18")) & ((Country, "Canada") | (Country, "USA"))) - Permit	{Role, Time}, {Role, Age, Country}	N/A	N/A
Example 13 (D-U-P) [30]	R1: (((Object, "Profile")) & ((Occupation, "student")) & ((Common Friends,"5"))) - Permit R2: (((Object, "Profile")) & ((Bob's Friend, "True"))) - Permit R3: (((Object, "photo")) & ((Trust, "0.25"))) - Permit	{Object, Bob's Friend}, {Object, Trust}	N/A	N/A
Example 15 (D-U-P)	R1: (((SecureLevel, "5")) & ((JobTitle, "junior-manager") | (JobTitle, "senior-manager")) & ((Location, "False"))) - Permit	N/A	N/A	N/A
Natural 1 (D-U-P) [33]	R1: ((((Role, "Pharma Scientist") & (Action, "Scan"))) & ((Info, "Trial team details") | (Info, "Trial Team CV")) & ((Merit_Committee, "True")) & (((Certificate, "Board") & (Certificate, "Surger"))) & ((Date, "2017-03-01"))) - Permit	N/A	N/A	N/A
PolTree (D-U-P) [21]	R1: (Des, "Professor") & (Dept, "CSE") & (Type, "Assignment") & (Conf, "High") & (Day, "Weekday") & (op, "Modify") - Permit R2: (Des, "Professor") & (Dept, "CSE") & (Type, "Paper") & (Conf, "High") & (Day, "Weekday") & (op, "Modify") - Permit R3: (Des, "Student") & (Dept, "CSE") & (Type, "Assignment") & (Conf, "High") & (Day, "Weekend") & (op, "Read") - Permit R4: (Des, "Professor") & (Dept, "ECE") & (Type, "Assignment") & (Conf, "Low") & (Day, "Weekend") & (op, "Modify") - Permit	N/A	N/A	N/A

Table 9: A Set of XACML Policies Collected from the Literature and Online Sources.

Policy Info	CNF - Rule Effect	A_{Permit}	A_{Deny}	A_{Indet}
KMarket-Blue (D-O) [32]	R1: ((totalAmount, "100")) - Deny R2: (((resource-id, "Liquor") | (resource-id, "Medicine"))) - Deny R3: (((resource-id, "Drink"))) & ((amount, "10")) - Deny	N/A	{totalAmount} {resource-id}	N/A
KMarket-Sliver (D-O) [32]	R1: ((totalAmount, "500")) - Deny R2: (((resource-id, "Liquor"))) - Deny R3: (((resource-id, "Drink"))) & ((amount, "50")) - Deny R4: (((resource-id, "Medicine"))) & ((amount, "5")) - Deny	N/A	{totalAmount}, {resource-id} {resource-id, amount}	N/A
KMarket-Gold (D-O) [32]	R1: ((totalAmount, "1000")) - Deny R2: (((resource-id, "Liquor"))) & ((amount, "10")) - Deny	N/A	{totalAmount}, {resource-id, amount}	N/A
PPS_ isMeetingFlag_rc (F-A) [20]	P1-R1: (((role, "pc-chair")) & ((action-type, "read")) & ((action-type, "write"))) - Permit P2-R1: (((role, "pc-member")) & ((action-type, "read"))) - Permit	N/A	N/A	N/A
PPS_paper_rc (F-A) [20]	P1-R1: (((role, "pc-chair")) & ((action-type, "delete"))) - Permit P2-R1: (((role, "pc-member")) & ((action-type, "read")) & ((isEq-meetingPaper-resId, "true"))) - Permit P3-R1: (((role, "pc-member")) & ((action-type, "create"))) - Permit	{role, action-type}	N/A	{role, action-type}
PPS_paper-conflicts_rc (F-A) [20]	P1-R1: (((role, "pc-chair")) & ((role, "admin")) & ((action-type, "read")) & ((action-type, "write"))) - Permit P2-R1: (((role, "pc-member")) & ((isConflicted, "true")) & ((action-type, "read"))) - Permit P3-R1: (((role, "pc-member")) & ((isMeeting, "true")) & ((action-type, "read"))) - Permit P4-R1: (((isConflicted, "true"))) - Deny	{role, action-type}, {role, isMeeting, action-type}	{isConflicted}	{isConflicted}, {role, action-type} {role, isMeeting, action-type}
PPS_paper-review_rc (F-A) [20]	P1-R1: (((role, "pc-chair")) & ((isConflicted, "false"))) - Permit P2-R1: (((role, "pc-chair") & ((isSubjectsMeeting, "true")) & & ((action-type, "read"))) - Permit P3-R1: (((role, "pc-chair")) & ((action-type, "create")) & ((action-type, "delete"))) - Permit P4-R1: (((isConflicted, "true"))) - Deny P5-R1: (((role, "pc-member")) & ((isConflicted, "false")) & ((action-type, "read"))) - Permit P5-R2: (((role, "pc-member")) & ((isConflicted, "false")) & ((action-type, "read")) & ((phase, "discussion"))) - Permit	{role, isConflicted}, {role, action-type}	{isConflicted}	{role, isConflicted}, {role, action-type}
PPS_paper-submission_rc (F-A) [20]	P1-R1: (((role, "pc-chair")) & ((role, "pc-member")) & ((action-type, "chair"))) - Permit P2-R1: (((role, "subreviewer")) & (action-type, "read"))) - Permit	{role, action-type}	N/A	{role, action-type}
PPS_ pcMember_rc (F-A) [20]	P1-R1: (((role, "pc-member")) & ((action-type, "read"))) - Permit P2-R1: (((role, "admin")) & ((action-type, "write")) & ((action-type, "create"))) - Permit P3-R1: (((role, "pc-member")) - Deny P4-R1: (((role, "admin")) & ((action-type, "delete"))) - Permit	{role, action-type}	{role, action-type}	{role, action-type}