A Novel Attribute-based Access Control System for Fine-Grained Privacy Protection

Ha Xuan Son
Department of Information Technology
FPT University
Can Tho city, Viet Nam
hxson@ctuet.edu.vn, sonhx4.fe.edu.vn

Nguyen Minh Hoang
Department of Information Engineering and Computer Science
University of Trento
Trento, Italy
minhhoang.nguyen@studenti.unitn.it

ABSTRACT

Public cloud have recently become popular platform for sharing data in real-time on web applications due to their simplicity in design and effectiveness in horizontal scaling and performance. However, no proper consideration has been fully investigated so far in privacy protection for public cloud. While many studies have acknowledged this issue, recent studies have not provided a fine-grained access control system for privacy protection in cloud system. As the data set becomes larger, we have to confront more privacy challenges. In this paper, we propose a comprehensive framework for enforcing attribute-based security protection in the public cloud together with the feature of data privacy protection by using privacy which define in XML format. With the proposed model, a request can be evaluated not only by access purpose but also by subject, action, resource, environment attributes. The experiment is carried out to illustrate the relationship between the processing time for access decision and the complexity of policies.

CCS Concepts

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

Keywords

ABAC, privacy, XML, document store, access control model

1. INTRODUCTION

The remarkable growth of Internet and social media applications over the past few decades leads to an exponential increase of data. By capturing and analyzing these data, enterprises obtain a better understanding about their customers, leading to better business decisions. However, with a vast amount of information available on the Web or Cloud, it is required a database system capable of storing and retriving of data in a well-structured way. Currently, cloud is the most popular platform to handle storage and sharing the data with the other. As in other relational databases, security must be highly considered as it has to process large volumes of data in could. The traditional approaches are effective in a small-scale system; however, in a large scale dynamic systems, they experience some serious problems such as role explosion, inflexibility in specifying dynamic policies and contexture conditions [2]. To overcome those issues, Attribute Based Access Control (ABAC) model has been investigated. The model grants access to a request only if it satisfies conditions on attributes of subject, resource and environment specified in policies [4]. With declarative mechanism to specify access permission, ABAC has proved its effectiveness on complex systems than RBAC with a fixed mechanism.

Although access control systems are successful in the prevention of unauthorized accesses and malicious users, they are ineffective in privacy protection for a large, decentralized system not only cloud flatform but also distributed network and Internet of Things. Our concentration in this work; therefore, aims at developing a system that is able to grant access control while providing effective privacy protection. As the purpose of the model is targeted both privacy and data governance issues, two challenges need to be solved. Firstly, access control should consider the case that the requesters wish to disclose limited information about them. Second, the system needs to develop its policy set which is able to process different levels of sensitive data to guarantee privacy protection.

To investigate the two issues addressed above, we have proposed a flexible model structure for privacy protection called *Attribute-based Access Control mechanism for privacy protection in Cloud System*. The model evaluates a request not only by its access purpose but also by subject, action, resource, environment attributes and function defined by users. According to ABAC mechanism requirement we describe complex policies containing information of user, action, resource, environment. We also build an implementation based on ABAC and store the data in MongoDB. Generally, the requests and policies are defined in XML format where administrators and users can easily define policies and requests.

This paper is organized as follow. In the second section, we briefly review related works. The following section illustrates our proposed architecture concept in detail and how it handles both access control and privacy protection. The fourth section presents the privacy-aware access control policies including the request structure and policy decision

ICCSP 2019, January 19–21, 2019, Kuala Lumpur, Malaysia

DOI: https://doi.org/10.1145/3309074.3309091

mechanism. Section 5 then describes our experimental designs and discusses the results. Finally, Section 6 presents our conclusions and future works.

2. RELATED WORKS

Most of the works in the literature focus on two directions: (i) constructing a whole new privacy-aware access control system based on ABAC model; and (ii) adding a level of privacy protection to a popular existence standard. Following the first trend, Hua Wang et al.[12] proposed a purpose-based framework for supporting privacy preserving access control policies and mechanisms. In this framework, the key component is a set of purpose-based access control policies that provide privacy protection by taking into account some important features (purposes and conditions). However, the way to model conditional expression is not clearly described; moreover, the conflicting algorithm only focused on simple attributes which are not properly evaluated with conditional expressions on them. Prosunjit Biswas et al. presented an attribute based protection model for JSON elements documents in [3]. To perform security protection, each JSON element is assigned a new attribute called "security label" which is used to define the access control policies. A benefit of this separation of labeling and authorization policies procedure is that each element can be specified and administered independently and possibly by different level of administrators. As a result, the privacy protection is done for each element of the database management systems (DBMSs). A drawback of this method might come from a huge number of labels needed to be assigned since the total number is growing exponentially.As a consequence, the process is time-consuming while requiring a large space storage when the system is expanded.

In the second research direction, most of the studies focus on improving the privacy protection for the popular ABAC standards, `eXtensible Access Control Markup Language` (XACML). Claudio A.Ardagna et al.in [1] proposed a system that extend the traditional XACML architecture with a combination with PRIME, a solution supporting privacy-aware access control. As a result, the system provides a flexible access control functionality of XACML with the data governance and privacy features of PRIME. In detail, the system has two main blocks: (i) PrimeLife XACML Engine is responsible for granting access control and (ii) Data Handling Decision Function (DHDF) is in charged of privacy and data handling functionalities. When an access request is needed to be considered, the request is forwarded to both blocks. The final decision is taken by combining the access control process and the DHDF data handling evaluation process. Only if a request comes from an authorized users that satisfied both evaluation procedure, it will be granted access to the required data.

Another study based on XACML is presented in [6, 10, 9, 11]. In this work, a system which insert privacy policies in access control solution to NoSQL database is developed. The main component responsible for privacy protection is called `Access control as a service solution` (ACCAAS).Administrators can store access control policies in ACCAAS solution for each element. When a request is forwarded to the ACCAAS system, it decides whether or not that user is authorized. If yes, then it sends a request to the data system asking for the required data. If not, it would return a "Deny" to the requester.

The biggest advantages of these solutions is that they support privacy protection on each element of the DBMSs. Moreover, they are easily integrated with XACML policies which is a widely-used standard in real world and considered many parameters for granted access at the same time (e.g.purpose, obligation, user information, etc.). However, for each request, since it is processed parallelly with the access control procedure and privacy-aware procedure, the combined results can be only "Permit" or "Deny". In this paper, we would like to extend the ability of the system of evaluating the request and granting permissions according to the level of authorized users. In detail, while processing a request, based on the policies and credential restrictions defined before, the system replies with three statuses: (i) Permit; (ii) Deny; and (iii) Partially Permit. The level of permissions depends on the level of privacy protection that the administrator sets up at the beginning. By this way, we can ensure the privacy protection for fine grained element of the database storing in cloud. In the next section, our architecture is described in detailed.

3. SUPPORTING PRIVACY PROTECTION MECHANISM IN CLOUD

As our proposed model based on ABAC, the model controls access by 4 main attribute types: (i) user attributes; (ii) attributes associated with the resource to be accessed; (iii) request attribute including execute, read, write; and (iv) current environment context. Figure 1 depicts the proposed model. The architecture contains the following main components.

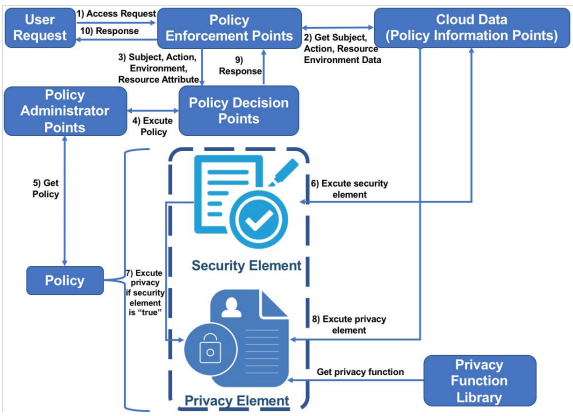

Figure 1: Proposed privacy-aware access control model

- **Policy Enforcement Points (PEP):** responsible for receiving requests from users. Moreover, it performs access control by making decision requests and enforcing authorization decisions.
- **Policy Information Points (PIP):** serves as the source of attribute values, or the data required for policy evaluation. Other components can send request to *PIP* whenever they need more information or data.
- **Policy Decision Points (PDP):** responsible for receiving and examining requests. It retrieves and evaluates applicable policies. After the evaluation process,

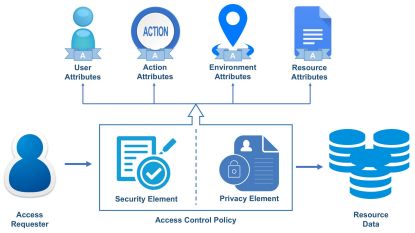

Figure 2: Two levels of protection in the attribute-based access control model

it returns the authorization decision to PEP. It is the core component of the model.

- **Policy Administrator Points (PAP):** responsible for creating policies and storing them in the system.

Access control and privacy protection works as follows. Firstly, when a user sends an access request to the PEP module, PEP generates a request to the *Repository Interface*, asking for user information (step 2). This information contains user's data, what action he could perform with the requested data and the resource environment data. After getting the data from *PIP*, after that PEP sends the request and user's information to PDP (step 3). Next step, PDP gets the applicable policies from the PAP and retrieves required attribute for evaluation. Each policy contains two elements: (i) security element and (ii) privacy element. Firstly, the request is checked with the list of security elements. Apparently, if a single condition is not satisfied, the returned is "Deny" and user's request is not granted access. If all security conditions are satisfied, the security element is executed and the requested data is loaded from the databas (step 6). Meanwhile, privacy checking procedure is enabled (step 7) and privacy function is loaded from the Privacy Function Library. The request is then evaluate through each level of privacy condition. The more conditions it satisfies, the more data is revealed to response to the user request. In this way, if a user possesses an administration permission to the requested data, he can perform his action (execute, read, modify or delete) on the whole data. On the other hand, if his permission is limited to a certain, he can only perform actions to a part of the data while the remain of the data is hided from him.

4. PRIVACY-AWARE ACCESS CONTROL POLICIES

The key to ensuring privacy protection access control is identifying how policies can be defined. As we mentioned before, a fundamental requirement of privacy policies is policies have to support fine-grained level. Figure 2 illustrates the structure of our policies: when a request is forwarded, the authorization process is carried out through two filter stages called as 2-stage authorization (i) security filter; and (ii) privacy filter. In security stage, the authorization verifies that the request is legitimate with rights for the access requester to access data based on security elements. In privacy stage, the request is transferred to this stage for checking privacy compliance based on privacy elements.

4.1 Policy Structure

As in ABAC model, a policy includes one or multiple rules and can be created from several elements. Among the elements, the `privacy` element is responsible for determining whether access data should be shown, hidden or generalized. A typical policy can be specified as follows:

- **policy_id:** identifier of policy
- **data_name:** name of collection or table containing resource data
- **rule_combining:** responsible for solving the conflict of rules
- **is_attribute_resource_required:** a derived field used to determine whether the policy needs attribute resource to evaluate conditions of target or rules.
- **target:** conditional expression specifies when the policy should be applied to.
- **security:** an array field with each element in it is a rule which contains `id` field, `effect` field (value of this field can either `Permit` or `Deny`) and condition.
- **privacy:** the privacy protection engine is also based on rules which are the Boolean expressions evaluated by user's defined function, subject, resource, environment attribute.

Each rule of security element defines a conditional expression that is modeled as a function tree structure. They return a value specified in the element `Effect` if the condition is `true`. For the privacy element, each rule is an array field with each element is similar to an obligation (in XACML) containing `id` field, `field_effect` field and condition. It is worth to note that the `field_effect` field which has an array type describes the list of data disclosure levels for each field of XML data constrained by these rules.

As an example, consider the following `Policy 1`.

```
<policy policy_id="policy_1"
          data_name="dataTable"
          rule_combining="first-applicable"
          is_attribute_resource_required="false">
   <Target>
       <Subjects><AnySubject/></Subjects>
       <Resources><AnyResource/></Resources>
       <Actions><AnyAction/></Actions>
    </Target>
    <security>
       <sec_id>sec_1</sec_id>
       <effect>Permit</effect>
       <resource "urn:oasis:names:tc:xacml:1.0:
         function:string-equal">
         <AttributeValue>dept_name</AttributeValue>
       </resource>
   </security>
   <privacy/>
</policy>
```

Privacy element is created from privacy rules and defined in `<privacy>...</privacy>`. These elements are Boolean expressions evaluating by user's defined function, subject, resource, and environment attribute. Here, as a constraint, a field of the resource can only belong to at most two areas.

The first one is default data area containing two basic privacy functions to represent the status of the data, i.e.show or hide. It is worth to note that the `field_effect` field which has an array type describes the list of data disclosure levels for each field. Each element in `field_effects` has two components: (i) component `name` storing the path to the field; and (ii) component `effect_function` containing `DomainPrivacy` and `Function` value where `DomainPrivacy` denotes the privacy domain and `Function` denotes the name of privacy functions on that domain including `Hide` or `Show`. The other one is configured by the administrator. Below, an example of privacy structure for `Policy 1` is illustrated:

```
<policy policy_id="policy_1"
             data_name="dataTable"
             rule_combining="first-applicable"
             is_attribute_resource_required="false">
<Target>
  <Subjects><AnySubject/></Subjects>
  <Resources><AnyResource/></Resources>
  <Actions><AnyAction/></Actions>
</Target>
  <security/>
  <privacy>
     <pra_id>pra_1</pra_id>
     <condition>
       <resource "urn:oasis:names:tc:xacml:1.0:
         function:string-equal">
         <AttributeValue>dept_name</AttributeValue>
       </resource>
     </condition>
     <field_effects>
      <DomainPrivacy>
        <name>dept_id</name>
      </DomainPrivacy>
      <Function>
        <function_name>Show</function_name>
      </Function>
     </field_effects>
     <field_effects>
      <DomainPrivacy>
        <name>dept_name</name>
      </DomainPrivacy>
      <Function>
        <function_name>Hide</function_name>
      </Function>
     </field_effects>
     </condition>
   </privacy>
</policy>
```

As shown in the code, we assumed that `DefaultDomainPrivacy` is the protected area. When a request is made, depending on the evaluation of the model, the data has two statuses `Hide` or `Show`.

4.2 Algorithm

The **Input** of this algorithm is the list of policy stored in PAP and the request sent from an access requester. We assume that *listPolicy* is a list storing the policies in PAP and *request* is a variable stores the value of subject, action, environment, resource. First, we find the best policy which allow the request to access the data resource. If the resource value and the action value between *policy* and *request* does not equal, the system will consider the next policy (line 2). Next, the value of *is_attribute_resource_required* is checked. The attribute of the required resource is returned if the condition is **true** (line 3 and 4). After that, we compare the value of *request* to the `Target` element. If the *request* can fulfill all target constraints, the `Security` and `Privacy` elements is evaluated. Otherwise, we move to the next policy (line 5). *listSecRule* and *listPriRule* are the variable storing the `rule` of `Security` and textttPrivacy respectively. Apparently, if a single condition is not satisfied, the returned value is **false** and user's request is not granted access (**while loop** from line 9 to line 13). We only execute the `Privacy` element if and only if the access request in `Sercurity` element is `Permit` (line 15). According to the `name` of `field_effect`, the `effect_function` is executed. Finally, the algorithm continues with the remaining policy (between line 22 and line 26) and returns the value of *response* in line 30.

Algorithm 1 Algorithm for evaluating policy and request

Input: List<Policy>, Request
Output: *respone* Respone class
Let *listPolicy*: List<Policy>
Let *request*: Request ← $getValue(Request)$

```
 1: for policy in listPolicy do
 2:   if GetCollectionName (policy, request) and GetAction (policy, request) then
 3:     if is_attribute_resource_required then
 4:       //Get the attribute the required resource
 5:     else if Target(policy, request) then
 6:       listSecRule = policy.GetSecurityRule(policy)
 7:       listPriRule = policy.GetPrivacyRule(policy)
 8:       flag = true
 9:       while secRule in listSecRule and flag do
10:         if !Condition(secRule,request) then
11:           flag = false
12:         end if
13:       end while
14:       response = RuleCombining(secRule.GetEffect())
15:       while priRule in listPriRule and flag do
16:         if Condition(priRule,request) then
17:           //Choose field_effects by name
18:           //Execute effect_function
19:         end if
20:       end while
21:     else
22:       response = Response(policy,Request)
23:     end if
24:   else
25:     //Continue with next policy in listPolicy
26:     response = Response(policy,Request)
27:   end if
28: end for
29: return response
```

5. EXPERIMENT

The proposed architecture was implemented for two cases: (i) with simple data structure; and (ii) with complex data structure. For the first scenario, structure of each resource consists of ten fields (key – value) and one document.On the other hand, the second one contains and array of embedded

documents. Each record has an array of embedded documents field containing at least five elements inside. In general, all experiments are included in total five policies. Moreover, to observe the difference between the performances of policy with single security element (traditional solution) and policy with security and privacy elements (our solution), the processing time of each case is recorded. The system configuration for the experiments is a 64-bit machine with 8GB of RAM and 2.8 GHz Intel Core i5 CPU running macOS High Sierra. The prototype is implemented by C#, .NET Core and MongoDB v4.0 for storing data.

Table 1 compares the performances of both policies on the two cases simple and complex data structure. On analyzing the table, it can be observed that as the number of records increases, the gap difference between processing time of both policies expands sharply. Considering the case of simple structure, when the number of records is 2000, this gap is only 0.253 second; however, it increases to 1.938 second as the number of records reaches 12000. A similar situation happens in the case of complex structure as this difference rises from 0.643 second to 2.553 second. For a database of up to 12000 records, the difference of approximately 2.5 seconds is acceptable. It is worth to note that as the complexity of the data structure increases, the time needed to process a record increases.

In order to analyze in detail the performance of each case, Table 1 also presents the average processing time for each record. While the traditional solution needs around 1 millisecond to process a record, the proposed model requires 1.15 and 1.38 millisecond depends on the complexity of the data structure. Again, with the development of computer system nowadays, this difference is acceptable.

Table 1: Processing time (measure in millisecond) for the model with and without privacy policy on different data structure

Num. records	Privacy element		No privacy element	
	Simp. structure	Comp. structure	Simp. structure	Comp. structure
2000	2282.5	2870.9	2029.9	2228.4
4000	4455.6	5508.4	4071.3	4313.1
6000	6743.6	8204.1	5556.8	6829.4
8000	8897.9	10978.4	6741.9	79659
10000	11841.1	13622.5	8989.9	9990.5
12000	14066.7	16636.1	12127.9	14082.7
Aver. each record	**1.1497**	**1.3767**	**0.9409**	**1.0811**

6. CONCLUSIONS

In this paper, we have proposed an Attribute-based Access Control for fine-grained privacy protection in cloud systems. The model defines two levels of protection as a filter on the policy, i.e.security filter and privacy filter. The privacy element allows the system to show or hide the requested data based on credential restrictions defined before and reply to the requester with three statuses: (i) Permit; (ii) Deny; and (iii) Partially Permit. From the analysis of the experimental results obtained on several data structure, we can state that the proposed model is implemented successfully and the difference of processing time between our solution and the traditional one is acceptable. In future work, we aim to apply the model to healthcare system in which the requirements for privacy protection is at the highest level while supporting dynamic policy is needed. Moreover, we also plan to apply a new approach [7, 8] to our scheme whereby the system will be greater flexibility, availability while ensuring security and privacy for system. Finally, an additional issue to be investigated is the problem of conflict resolved in privacy data [5]. This problem could be alleviated by considering different levels of priority for each policy.

7. REFERENCES

[1] C. A. Ardagna et al. An xacml-based privacy-centered access control system. In *Proceedings of the first ACM workshop on Information security governance*, pages 49–58. ACM, 2009.

[2] E. Bertino et al. Access control for databases: concepts and systems. *Foundations and Trends® in Databases*, 3(1–2):1–148, 2011.

[3] P. Biswas, R. Sandhu, and R. Krishnan. An attribute-based protection model for json documents. In *International Conference on Network and System Security*, pages 303–317. Springer, 2016.

[4] V. C. Hu et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, 800(162), 2013.

[5] M. H. Nguyen et al. A dynamic solution for fine-grained policy conflict resolution. In *International Conference on Cryptography, Security and Privacy*. ACM, 2019.

[6] H. X. Son and E. Chen. Towards a fine-grained access control mechanism for privacy protection and policy conflict resolution. *International Journal of Advanced Computer Science and Applications*, 10(2), 2019.

[7] H. X. Son, T. K. Dang, and F. Massacci. Rew-smt: A new approach for rewriting xacml request with dynamic big data security policies. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*, pages 501–515. Springer, 2017.

[8] H. X. Son, T. K. Dang, and L. K. Tran. Xacs–dypol: Towards an xacml–based access control model for dynamic security policy.

[9] H. X. Son et al. Rew-xac: an approach to rewriting request for elastic abac enforcement with dynamic policies. In *Advanced Computing and Applications (ACOMP), 2016 International Conference on*, pages 25–31. IEEE, 2016.

[10] H. X. Son et al. Toward an privacy protection based on access control model in hybrid cloud for healthcare systems. In *The 12th International Conference on Computational Intelligence in Security for Information Systems*. Springer, 2019.

[11] Q. N. T. Thi et al. Using json to specify privacy preserving-enabled attribute-based access control policies. In *International Conference on Security, Privacy and Anonymity in Computation, Communication and Storage*. Springer, 2017.

[12] H. Wang, L. Sun, and V. Varadharajan. Purpose-based access control policies and conflicting analysis. In *IFIP International Information Security Conference*, pages 217–228. Springer, 2010.