Abstract

Nowadays, NoSQL databases have been rapidly becoming the popular data platform for big data and real-time web applications. Simpler horizontal scaling, flexible schema designing, high performance data access have made NoSQL databases to be alternative approaches for traditional relational databases . However, there are some disadvantages in NoSQL, among which the lack of effective suppprt for access control and privacy protection is the most serious ones. The huger data we have, the more challenge in data protection we have to face. In this research, we address this issue by implemeting a comprehensive framework for enforcing attribute-based security policies stored in JSON document. We use Polish notation for modeling conditional expressions which are the combination form of subject, resource, and environment attributes so that the policies are flexible, dynamic and fine grained. Moreover, with the approach of attribute-based access control, we have proposed a flexible model struture for privacy protection called as Attribute-Based Privacy Protection so that it can be evaluated not only by access purpose but also by subject, resource, environment attributes. The experiment is carried out to illustrate the relationship between the processing time for access decision and the complexity of policies.

1. Introduction

Nowadays, the quanity of data is increasing exponentially by the development of social media appications, sensor for data acquisitions and smart phone utilization. NoSQL databases is the most popular approach to handle those semi and unstructured data for a scalable application. Like relational database, security is highly considered in NoSQL database, especially when working with huge volume data. For the last decade, Discretionary Access Control (DAC), Mandatory Access Control (MAC), Role Based Access Control (RBAC) have been used almostly to handle security. However, because of the rapid development of large scale dynamic systems, those traditional access controls have gradually reveal their disadvantages, for example, applied for only closed system, role explosion and inflexibility in specifying dynamic policies and contextual conditions. To overcome those problems, Attribute Based Access Control models have been recently investigated and according to Gartner‘s prediction: “By 2020, 70% of all businesses will use attribute based access control as the dominant mechanism to protect critical assets.”[]

Data privacy refers to the evolving relationship between technology and the legal right to, or public expectation of privacy in the collection and sharing of data about one’s self. Even when access control systems are successful in restricting unauthorized and unauthenticated users, they are ineffective as privacy protection for a large, decentralized system like the World Wide Web, where it is easy to copy or aggregate information. Private or sensitive information can be preserved by restricting the intended purpose of data access. According to “Data protection principles” [] based on Data Protection Act Organization , purpose is considered as the second principle: “Personal data shall be obtained only for one or more specified and lawful purposes, and shall not be further procecssed in any manner incompatible with that purpose or those purposes.” Extending those approaches, we have proposed an Attribute-Based Privacy Protection (ABPP) model which will use not only access purpose attribute but also user, resource, environment attributes to evaluate privacy policies so that we can specify privacy policies more dynamically and flexibly.

1. Related work

Our research is related to several topics in the area of access control and privacy protection for data management, namely policy specification, privacy-preserving data management systems. We now survey the most relevant approaches in these areas and point out the difference of our work with respect to these approaches.

Hua Wang, Lili Sun, and Vijay Varadharajan[] have proposed a purpose-based framework for supporting privacy preserving access control policies and mechanisms. They have also developed algorithms to help a system to detect and analyze the conflicts when adding new policies. However they don’t mention much about how to model the conditional expression and the algorithms they proposed just focused only on simple attributes lacked of evaluating conditional expression.

Prosunjit Biswas, Ravi Sandhu, and Ram Krishman[] have presented an attribute based protection model for JSON documents. Their approach is to add a new attibute called “security-label” to JSON elements and specify access control policies using these values. The advantage of the seperation of labeling and authorization policies is that they can be specified and administered independently possibly by different level of administrators. However, the number of label assignments can be very large because it is calculated by the exponential function. Therefore the space storage is considered to be a potential problem when the system is expanded.

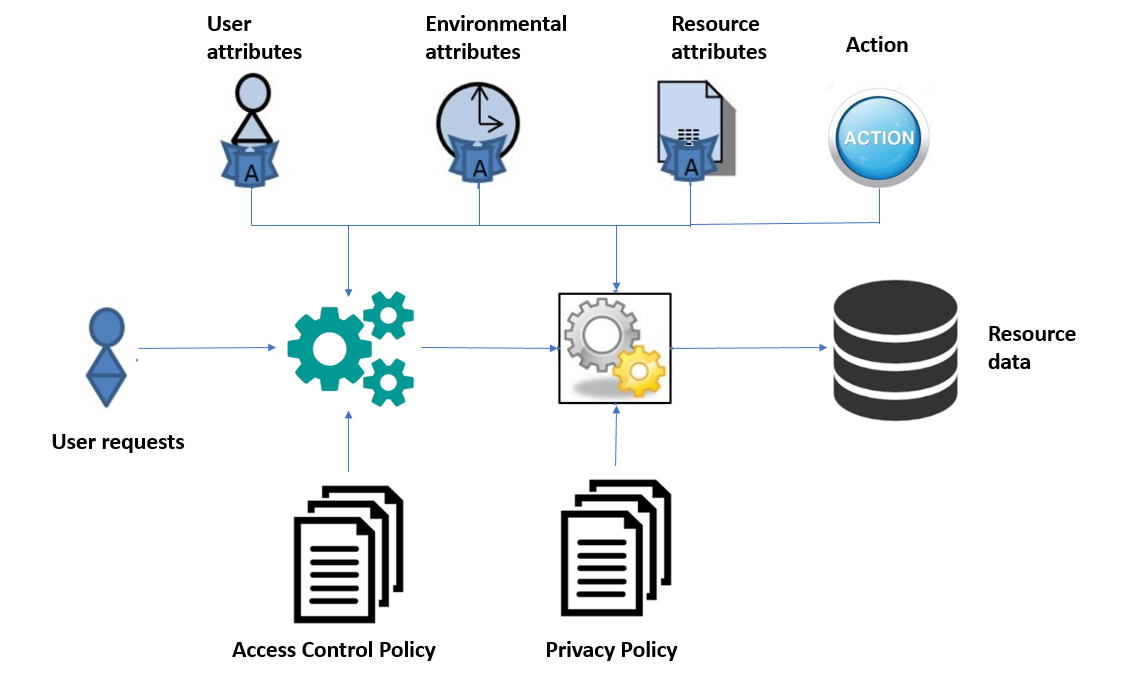
Ji-Won Byun, Ninghui Li[] have presented a comprehensive approach for privacy preserving access control based on the notion of purpose. A key feature of their approach is that it allows multiple purposes to be associated with each data element and also supports explicit prohibitions, thus allowing privacy officers to specify that some data should not be used for certain purposes. However, there are two disadvantages in their approach. First, because they assign intended purpose to each data element, it result to the size of data storage will be double. Second, their approach allows a field to have only a privacy value, the value of the field should be related to purpose in practice. For example, suppose we have a SSN field, their approach can not solve the following rule: " If purpose is Marketing then SSN field will show only Area Numbers. If purpose is Third-party then SSN field will show Group Numbers and Serial Numbers".

Haibo Shen[] has proposed a semantic-aware attribute based access control model (SABAC) by combining the Semantic Web technologies with the attribute based access control. SABAC use the Web Ontology Language standard to represent the ontology of the resources and users and uses eXtensible Access Control Markup Language as the policy language.

1. Attribute-Based Access Control and Privacy Protection Models

In this section, we describe the base theory of this research. When a subject access an object, the authorization process is carried out though two stages called as 2-stage authorization:

* First stage: access control policy authorization verifies that the request is legitimate with rights for the subject to access data.
* Second stage: request is transfer to this stage for checking privacy compliance based on privacy policies.

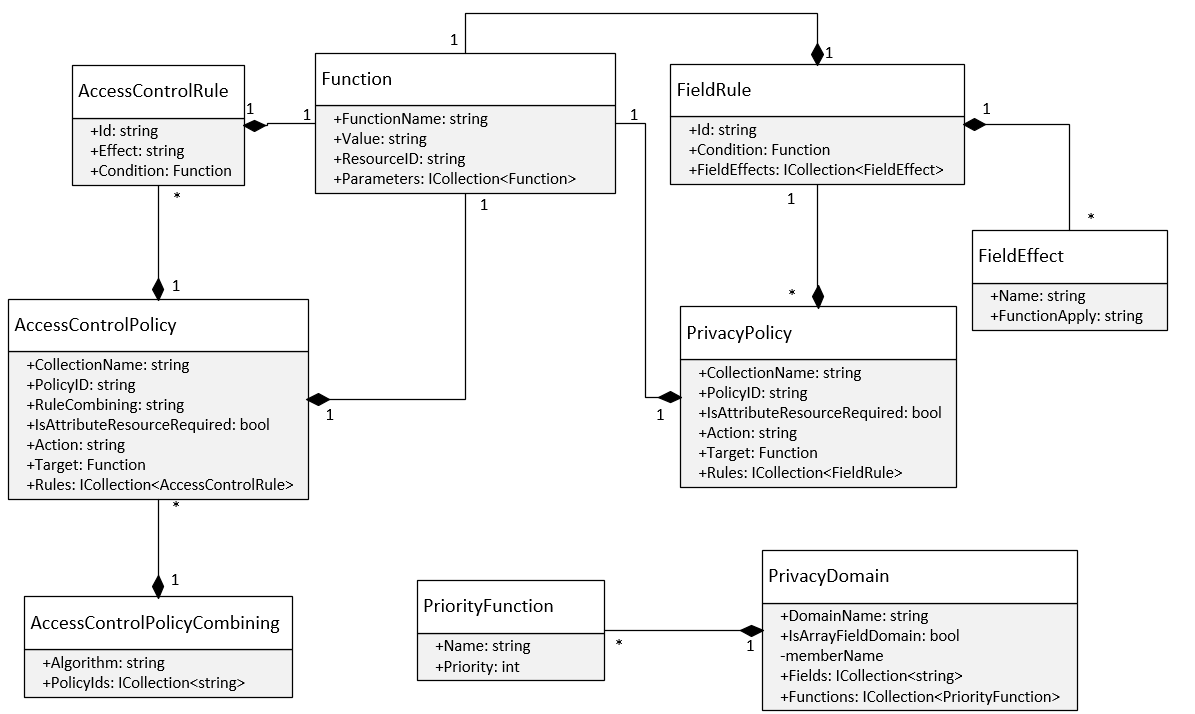


Figure[]

**Access Control Policies**: contain policies which are used to determine whether a subject can access resources. The decision is made based on rules inside policies which are the boolean expressions evaluated by user’s defined function, subject, resource, environment attribute. Those policies are specified and managed by administrators.

**Privacy Policies**: contain policies which are used to determine whether some fields of a record in resources should be shown, hidden or blurred when a user access to it. The privacy protection is made based on rules inside policies which are the boolean expressions evaluated by user’s defined function, subject, resource, environment attribute. Those policies are specified and managed by administrators.

1. Policy Structure:
   1. General Structure:



Figure[] Privacy and Access Control Policy class diagram

We will explain more about how we model the conditional expression of policies in *Function* class. It is a recursive structure. If the value of “Value” field is defined so it means that this function represents for a constant value or the JSON path to get data from resource (in this case "ResourceID" field is also defined) . If the value of "Value" field is null it means that this is the function and the value of "FunctionName" field must be defined. For example we have a conditional expression: "Equal (Subject.role, intern) And GreaterThan (Subject.age, 18)". Then the result will be:

{

"function\_name" : "And",

"parameters" : [

{

"function\_name" : "Equal",

"parameters" : [

{ "value" : "role", "resource\_id" : "Subject" },

{ "value" : "intern", "resource\_id" : null }

]

},

{

"function\_name" : "GreaterThan",

"parameters" : [

{ "value" : "age", "resource\_id" : "Subject" },

{ "value" : "18", "resource\_id" : null }

]

}

]

}

We also proposed an algorithm which converts the conditional expression in text format to the *Function* structure:

**Algorithm :** Algorithm for parsing conditional expression  
**Input**: *rawExpression*: string

**Output**: *function*: *Function* class

**1** Let *listToken*: List<string>**←** *getTokens*(*rawExpression*); /\* gets tokens from *rawExpression*  
**2** Let *stackToken:* Stack<string> /\* A stack stores names of functions \*/

**3** Let *queueToken:* Queue<string> /\* A queue stores tokens in Reverse Polish Notation form \*/

**4** Let *queueFunction:* Queue<Function> /\* A queue stores functions \*/

**5 begin**

**6 foreach** *token* **in** *listToken* **do**

**7 if** *IsFunctionName***(** *token )* **Or** *IsLogicalOperator***(** *token )* **Or** *token ==* "("**then**

**8** *stackToken*.**push** (*token*)

**9 else if** *token ==* ")"**then**

**10 while** *stackToken*.length > 0 **do**

**11 var** *temp* :*= stackToken*.pop()

**12 if** *temp ==* "("**then**

**13** *queueToken***.**enqueue**(***stackToken*.pop())

**14 *break***

**15 else** *queueToken***.**enqueue**(***temp*)

**16 end while**

**17 else** *queueToken.*enqueue (*token*)

**18 end foreach**

**19 while** *stackToken.*length > 0 **do** *queueToken.*enqueue (*stackToken.*pop())

**20 while** *queueToken.*length > 0 **do**

**21** *token* := *queueToken*.dequeue()

**22 if** *IsFunctionName***(** *token )* **do**

**23 var** *function* := Function.CreateFunction (token)

**24 for** i := 1 to *GetNumberParameters* (*function*) **do**

**25** *function.*Parameters.Add (*queueFunction.*dequeue())

**26 end for**

**27** *queueFunction*.enqueue (function)

**28 else** *queueFunction*.enqueue (Function.CreateConstantValue(*token*))

**29 end while**

**30 return** *queueFunction.*enqueue()

**31 end**

* 1. Access Control Policy Structure:

We specify access control policy structure as follows:

{

"policy\_id" : "Policy 1",

"collection\_name" : "Department",

"action" : "read",

"rule\_combining" : "permit-overrides",

"is\_attribute\_resource\_required" : true,

"target" : { //Equal (Subject.active, True) },

"rules" : [

{

"id" : "rule 1",

"effect" : "Permit",

"condition" : { // Equal (Resource.dept\_name, Subject.department) }

}

]

}

* 1. Privacy Policy Structure:

We specify privacy policy structure as follows:

{

"collection\_name" : "Department",

"policy\_id" : "policy 2",

"action" : "read",

"is\_attribute\_resource\_required" : true,

"target" : { // Equal (Subject.role, intern) },

"rules" : [

{

"rule\_id" : "rule 1",

"condition" : { //},

"field\_effects" : [

{ "name" : "dept\_id", "effect\_function" : "DefaultDomainPrivacy.Hide" },

{ "name" : "dept\_no", "effect\_function" : "DefaultDomainPrivacy.Show" },

]

}

]

}

* 1. Conflict resolving approach:

Because an access control policy may contain multiple rules and we have many policies. Each rule, access control policy may evaluate to different decisions (Permit, Deny). Therefore our approach is to use combining rule algorithm inherited from XACML[].

In privacy policies, the conflict situation occurred when there are multiple rules in a privacy policy which are satisfied the condition. It results to that we have many privacy functions to be applied to one field of object. Therefore we have added a new structure named “PrivacyDomain” to solve conflict. We also specify a constraint that a field in resource can only belong to at most two domains, one is default domain which contains two basic privacy functions: show and hide, another domain is configured by administrator.

Example:

{

"domain\_name" : "DateTimeDomain",

"fields" : ["Employee.personal\_info.birth\_date"],

"is\_sub\_policy" : false,

"hierarchy" : [

{ "name" : "ShowYear", "priority" : 1 },

{ "name" : "ShowMonthAndYear", "priority" : 2 }

]

}

First we will explain more detail about the “field\_effects” field in privacy policy structure. It is an array field and the number of elements in this field is equal to the number of single value field in resource. Each element has the following structure:

"name": is the path to the single value field.

"effect\_function": This field has only 2 value patterns. First is "Optional" value, second is "X.Y" value where X is privacy domain, and Y is name of privacy function in that domain.

For example, we have the conflict privacy table.

|  |  |
| --- | --- |
| **Fields** | **Conflict Privacy Functions** |
| name | Optional, DefaultDomain.Show |
| personal\_info.birth\_date | DateTimeDomain.ShowMonthAndYear,  DateTimeDomain.ShowYear,  DefaultDomainPrivacy.Show |
| personal\_info.ssn | SsnDomain.AreaNumber, SsnDomain.SerialNumber, Optional |

The privacy function will be chosen using the following rule:

P(“Optional”) < P(“DefaultDomain.Show”) < P(X.Y1) < … < P(X.Yn) < P (“DefaultDomain.Hide”)

where P(X.Y) stands for priority of privacy function Y in domain X. The priority is configured by admininstrator in “PrivacyDomain” structure.

Applying this rule to above conflict table, we will have the following result:

|  |  |
| --- | --- |
| **Fields** | **Privacy Function Chosen** |
| name | DefaultDomain.Show |
| personal\_info.birth\_date | DateTimeDomain.ShowYear |
| personal\_info.ssn | SsnDomain.AreaNumber. |

* 1. Advanced privacy policy support for array of embedded subdocuments:

JSON object can contain not only single-value field (string, number, boolean value) but also contain an array of objects field (it’s also known as array of embedded subdocuments in mongoDB) which makes the JSON object more complex. To support privacy for array of objects field, we extend the value of "effect\_function" field in Privacy Policy structure so that its value can be an identifer of another privacy policy which target to this array field.

For example:

A complex JSON Object:

"Project Bonus" Resource:

{

"project\_name" : "E-learning system",

"bonuses": [ //this is array of embedded subdocuments

{ "name": "John", "bonus": 600 },

{ "name": "Bob", "bonus": 900

] }

We have a privacy rule: "Employee can only view his/her bonus for each project. "

The equivalent sub-privacy policy will be specified as followings:

{

"policy\_id": "Policy 3",

**"collection\_name" : "ProjectBonus.bonuses",**

….

"rules" : [

{

"rule\_id" : "rule 1",

"field\_effects" : [

{ "name" : "name", "effect\_function" : "DefaultDomain.Show" },

{ "name" : "bonus", "effect\_function" : " DefaultDomain.Hide" },

],

"condition" : { //Equal (Subject.Name, Bob)}

},

]

}

Then the main privacy policy for "ProjectBonus" collection will use the value of "policy\_id" of this sub-privacy policy for the "effect\_function" field:

{

"policy\_id": "Policy 1",

**"collection\_name" : "ProjectBonus",**

….

"rules" : [

{

"rule\_id" : "rule 1",

"field\_effects" : [

{ "name" : " project\_name", "effect\_function" : "DefaultDomain.Show" },

{ **"name" : "bonuses",** **"effect\_function" : " Policy 3"** }

],

"condition" : { //assume that this condition is satisfied } }, ] }

Now, assuming that John want to view "ProjectBonus" collection, the result will be:

{

"project\_name" : "E-learning system",

"bonuses": [

{ "name": "John", "bonus": 600 },

{ "name": "Bob", "bonus": "" }

]

}

Bonus for Bob has been hidden due to sub-privacy policy.

1. Experiment
2. Conclusion

References