available at www.sciencedirect.com

ScienceDirect

journal homepage: www.elsevier.com/locate/cose

Computers & Security

Clustering subjects in a credential-based access control framework

*K. Stoupa**, *A. Vakali*

Aristotle University of Thessaloniki, Thessaloniki, Greece

ARTICLE INFO

Article history:
Received 5 November 2005
Revised 5 July 2006
Accepted 3 August 2006

Keywords:
Access control
Clustering users
Credentials
XML-based access control
Access request evaluation time

ABSTRACT

Currently, access control of distributed Internet resources (such as files, documents and web services) has become extremely demanding. Several new access control models have been introduced. Most of the proposed approaches increase the complexity of the access control procedure and at the same time expressing these models is becoming complicated. Improving the execution time of the access control procedures is a challenging task due to the increased number of resources (available over the Internet) and the size of the audience involved. In this paper, we introduce an approach for speeding up the access control procedure under an environment accessed by known subjects (i.e. subjects whose identity and attributes are known apriori through a subscription phase). This approach is based on some update functions (employed at the background during idle times) over files which are associated with subjects. The core task of the proposed update is its dynamic nature and its clustering of subjects according to their interests and credentials. Moreover, this work associates subjects with security policies that are most likely to be triggered according to (the subjects) interests. Credential-based access control is considered to properly protect frameworks distributing resources to known subjects and here emphasis is given to the complexity involved in order to decrease the access request evaluation time under a credential-based access control framework.

1. Introduction

Web-based environments offer a wide range of resources to a heterogeneous audience and the access control procedures involved should cope with a large number of policies expressing which clients (i.e. *subjects*) can access which protected resources (i.e. *objects*). In an effort to reduce the number of policies that need to be specified, modern access control models have been proposed (see Castano and Ferrari, 2003; Pallis et al., 2004; Stoupa and Vakali, in press for a discussion of the most widely-used access control models). Furthermore, centralized access control mechanisms cannot cope with distributed networked computing environments since they should cope with new security issues (such as separation of duties, and consistent control throughout the network; Ward and Smith, 2002). Moreover, the increased number of subjects leads to an exponential increase of the number of the needed security policies. Therefore, expression of policies for groups of subjects seemed mandatory. Credential-based access control model (Winslett et al., 1997) seems to be a solution for such frameworks, since each subject is associated with some attributes (forming its credentials) and authorizations are assigned to credentials (or credential expressions) and not subject identities.

Although, the goal of modern access control models was to reduce the number of policies that need to be specified, by

* *Corresponding author.*
E-mail addresses: kstoupa@acn.gr (K. Stoupa), avakali@csd.auth.gr (A. Vakali).
0167-4048/$ – see front matter © 2006 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cose.2006.08.006

grouping subjects and/or objects according to their characteristics or by organizing them into hierarchies where each set of user is assigned a security clearance (Yang and Li, 2004), the number of subjects and objects has been increased so much that the number of needed policies still remains huge. Additionally, a fine granularity level for access control is often a - requirement, in that web resources often contain information at different sensitivity levels. As a result, time delays required for evaluating the associated policies and granting or denying access may cause subject's inconvenience. Although many access control models have been proposed for the Web (Chakrabarti, 2003), few research efforts have addressed such a ''delay'' problem. For example, in an effort to propose effective access control services, Murata et al. (2003) have introduced the use of static analysis in improving access queries and thus decreasing time delays. Carminati and Ferrari (2005) propose the use of Access Control XML (AC-XML) documents containing for each object (either XML document or DTD) the associated policies. In this context, we focus on *speeding up* the access control procedure involved in a credential-based environment.

This paper proposes an access control technique that can be employed for protecting any web data source (whose structure will be given through XML files and XML Schemas), where known subjects can have access and the policies are specified according to the credential-based paradigm. The proposed technique aims at decreasing the access request evaluation time and we assume that the data, the credentials, and the policies are encoded in XML. Our paradigm can work with any XML-based security language but we have adopted X-Sec (Bertino et al., 2001) (an XML-based security language) to express the credentials and the policies. To simplify the process of evaluating subject credentials against access control policies, all the credentials a subject possesses are placed into an XML document, called *subject profile*. Subject profiles are maintained by the protected organization. More specifically, the main contribution of the proposed work is summarized in:

- using the so called ''*dynamic update*'' method on *associating subjects with policies* for speeding up the access request evaluation phase. Each subject, apart from other attributes, will be associated with policies which are more possible to be triggered in a future access. The list of the associated policies is modified dynamically as the subject accesses objects, and it is stored in a separate file associated with the subject. This method can be considered mostly in subscription-based systems, in that it requires the server to store the subject credentials.
- *grouping of subjects* according to their interests (Middleton et al., 2004; Wang et al., 2004; Xie and Phoha, 2001) and their credentials. This grouping is inspired from the ideas proposed in earlier research efforts related to Web clustering (Baldi et al., 2003; Chakrabarti, 2003; Jain et al., 1999; Jeng et al., 2002). Subjects are initially grouped into the (so called) *interest clusters* according to their interests such that subjects with common interests are organized into the same cluster. The interest clusters are refined by, also considering the credentials of the members, a filtering which may be applied by using two distinct practices:
  1. *Content-based filtering*: each subject is associated with a vector of objects he/she has accessed in the past and how many times. By using content-based filtering we can extract the interests of the subject, i.e. the object categories which he/she is most likely to request in the future.
  2. *Collaborative filtering*: identifying subjects who own similar interest profiles and measure the similarity between such profiles. It is assumed that subjects with similar interests and credentials are likely to trigger common access control policies. Therefore, we build a list of such policies.
- *presenting a complexity analysis* for the proposed access request evaluation procedure in order to prove improvement in the involved time. The complexity analysis estimates the complexity of the algorithms describing the access request evaluation procedure under the proposed approach (dynamic update approach) and the typical access request evaluation procedure. The results of the analysis are given in a separate table.

The rest of the paper is organized as follows. Section 2 gives a scenario that will be followed throughout the paper. Section 3 discusses the dynamic update approach. Section 4 gives the complexity analysis of the access request evaluation in the dynamic update approach and in a typical access control environment. Finally, Section 5 concludes the paper and discusses future work directions.

2. An access control scenario

Consider a network belonging to the organization ''X-Company'' which can be accessed by both internal users and external ones who request access through the Internet. In both cases requesting subjects should be known to the organization, thus they should subscribe first in order to be associated with some credentials. Some of the defined credential types in ''X-Company'' are: general manager, financial manager, accountant, and secretary. Moreover, we assume that this company stores its data under the XML standard so the protected (XML-oriented) documents are organized into categories according to their purpose (e.g. invoice, report) and they are stored in a separate database. Here, we use a particular XML document as a running example which refers to payroll information about employees of the ''X-Company''. As shown in Table 1, the DTD[1] specifies the payroll information about employees of a specific department.

A user requesting access to the ''X-Company'' should first subscribe in order to be associated with some credentials (*subscription phase*), i.e. each user is associated with both a *subject profile* (summarizing his/her credentials) and as many *credential files* as his credential types. In order to present the access request evaluation, we introduce a policy base, which has four distinct policies. According to X-Sec, we represent a policy p as a tuple:

$$p = (cred - expr, target, path, priv, type, prop),$$

[1] In this paper we use DTDs for brevity reasons. In the implementation XML Schemas are adopted.

Table 1 – An example of a resource DTD: the `payroll.dtd`

```
<!DOCTYPE payroll [
<!ELEMENT payroll (employee*)>
<!ELEMENT employee (name, hireDate, salary)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT hireDate (#PCDATA)>
<!ELEMENT salary (#PCDATA)>
<!ATTLIST payroll Dept CDATA #REQUIRED>
]>
```

to declare that subjects denoted by *cred-expr* (which is a specific credential name or a credential expression) are allowed or not allowed according to the *type* to exercise privilege *priv* on the portion *path* of the protection object *target*. The *prop* defines the propagation mode of the policy. Three options are provided for the propagation: *cascade*, meaning that the policy propagates to all the direct and indirect sub-elements of the elements on which the policy is specified; *first level*, meaning that the propagation is limited to the direct children; and *no-prop*, denoting no propagation option. Some examples of access control policies are presented in Table 2 which defines that all managers can read the salary and the hire date of every employee (policies 1, 2). Moreover, the general manager can modify every XML file following the `payroll.dtd` (policy 3) and the human resources manager can modify the hire date element of every file following the `payroll.dtd` (policy 4).

3. The dynamic update approach

The proposed dynamic update approach is tailored for a subscription-based environment (such as a confined network of an organization), where subjects should first subscribe in order to request access. Such environments operate as the "X-Company" network which may be accessed over the web. Each subject is associated with some credentials during an initial so called subscription phase (e.g. such credentials can be manager, financial manager, accountant, and secretary).

Fig. 1 illustrates the modules underlying the dynamic update approach and the phases involved which are described as follows:

1. *Subscription phase*: the subject interacts with the subscription module by sending his/her credentials and his/her interests (optionally). Upon the receipt of such information the module creates a number of credential files associated with the subject and a *subject profile*, summarizing the credential types of the newly subscribed subject. Moreover, it creates an *interest profile*, containing the explicitly defined interests of the subject.
2. *Access control phase*: this phase is divided into two subphases:
   a. *Dynamic Update*: in order to understand the functionality of this task some extra files are required, which are encoded in XML, created and dynamically updated in the background during idle times. Those are (a) the *access log file*, containing the objects that have been accessed by the subject and the *frequency* of accesses (for how many times there have been accessed objects of this category), and (b) the *subject policy file*, containing the list of policies that are more likely to be triggered by the associated subject, and (c) the *object category file* associating each XML Schema with a category. The category of each `.xml` file is defined by the category of the XML Schema it follows. Fig. 1 depicts the process of generating and updating these files, which is summarized in the following:
      i. The *update access log file* module periodically scans the systems' log file and updates the subjects' access log files. Initially, this file is empty and it is updated after the associated subject has been granted at least one access.

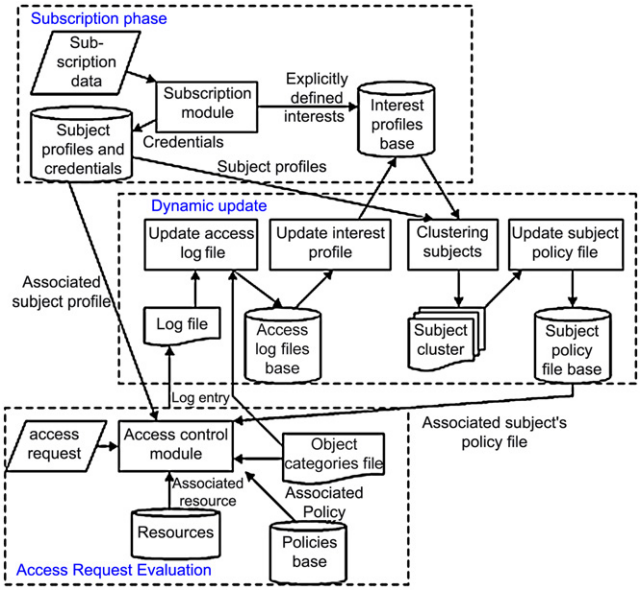

Fig. 1 – Dynamic update approach.

Table 2 – An example of access control policies

SN	Cred-expr	Target	Path	Type	Priv	Prop
1	/manager	payroll.dtd	/payroll//salary	+	Read	No-prop
2	/manager	payroll.dtd	/payroll//hireDate	+	Read	No-prop
3	/manager[@type = "general"]	payroll.dtd	–	+	Write	–
4	/manager[@type = "humanResources"]	payroll.dtd	/payroll//hireDate	+	Write	No-prop

ii. The *update interest profile* module takes as input the access log file and according to the access frequencies of each object type updates implicitly defined interest part of the interest profile of the associated subject.
iii. The *clustering subjects* module takes as input the interest profiles and the subject profiles of all subjects and clusters them according to their interests and credentials using collaborative filtering techniques (a more detailed description is given in Section 3.2).
iv. Finally, the *update subject policy file* module updates the policy files of all the subjects belonging in the same cluster with the policies that have been triggered by the other members of the cluster.

The proposed dynamic update functions are designed for being executed in the background and on idle times (e.g. late at night) such that the system idle times are exploited towards improving the access request evaluation process.

b. *Access Request Evaluation*: after the dynamic update functions have taken place, the access control module responses more quickly to the access request of the subjects. Indeed, upon a new access request arrives the module scans the policy file of the requesting subject and if it finds a matching policy it triggers it otherwise, it scans the policy base to find a policy granting the access. If a policy is found a reply is sent to the subject and the systems log file is updated.

3.1. Structure of the files involved

The proposed dynamic update function is based on some more files such as the *access log file* (its DTD is given in Fig. 2(a)) which is used to organize the accesses of the associated subject (i.e. such a file contains a list of entries organized according to object categories). Each entry associates an object category with the number of accesses to it, the type of access mode and the policy used. In order to generate or update this file (*update access log file* module), the system's log file should be scanned. Every time the system finds an entry referring to a subject, it activates the associated access log file. The system finds the category of the accessed object (by scanning the *object categories file*) and the access mode and finds the appropriate `access` element in order to increase the frequency by one and add the identifier of the policy triggered by that access. To find the new interests of the subject, the `frequency` element should become zero periodically in order to discover if interests change or not.

The interest profile contains both explicitly defined interests (i.e. object categories) and implicitly defined, and its DTD is given in Fig. 2(b). The part of the file that is automatically updated is the implicitlyDefined element. The term "interests" refers to object categories (i.e. similar objects, or objects belonging to the same category, e.g. invoice) that the subject has explicitly declared to be of her/his interest. Certainly, such interests may change and the system can implicitly realize such by analyzing the access log file of the subject. Thus, the interest profile update module may frequently scan the access log file in order to find out whether the subject frequently accesses specific object categories (i.e. the frequency attribute of the specific object category element is greater than a threshold) which do not belong to his interests list. In such a case another object category is added to the implicitlyDefined element.

The *subject policy file* (its DTD is given in Fig. 3) contains the list of policies that are more likely to be triggered by the associated subject. This file is organized according to the object categories for which one the accesses frequency is defined. Moreover, for each object category, it contains a list of policies referring to this category or to a specific object belonging to this category.

Each `policy` element has the same elements as those defined in Section 2 (i.e. *cred-expr*, *path*, *target*, *priv*, *type*, *prop*). When the file is initialized it includes the object categories which are the explicitly defined subject's interests (according to the interest profile) and later on, the access log file is scanned in order to find the policies referring to the implicit interests. Those functions are executed by the *update subject policy file* module.

3.2. Clustering of subjects

We propose the organization of subjects into clusters in order to improve the execution time of the overall access request evaluation process. The clusters are defined based on twofold information:

1. *Interests*: by scanning the interest profiles of the subjects, where each object category belonging in the interests of a subject is associated with a frequency and based on this information we can adopt an approach originating from the Latent Semantic Indexing (Jiang, 1997) idea to represent important associative relationships between subjects and interests.

```
<?xml version="1.0"?>
<!DOCTYPE accessLogFile [
<!ELEMENT accessLogFile (access+)>
<!ELEMENT access (objectCategory, accessMode,

frequency, policyID)>
<!ELEMENT objectCategory (#PCDATA)>
<!ELEMENT accessMode empty>
<!ATTLIST accessMode
type=(read|write|execute)>
<!ELEMENT frequency (#PCDATA)>
<!ELEMENT policyID (#PCDATA)>
]>
```

(a)

```
<?xml version= "1.0" ?>
<!DOCTYPE interestProfile [
<!ELEMENT interestProfile (explicitlyDefined,

implicitlyDefined)>
<!ELEMENT explicitlyDefined (objectCategory?)>
<!ELEMENT implicitlyDefined (objectCategory?)>
<!ELEMENT objectCategory (#PCDATA)>
]>
```

(b)

Fig. 2 – The DTD of (a) access log file and (b) interest profile.

```
<!xml version="1.0">
<!DOCTYPE subjectPolicyFile [
<!ELEMENT subjectPolicyFile (objectCategory+)>
<!ATTLIST subjectPolicyFile id ID #REQUIRED>
<!ELEMENT objectCategory (policy+)>
<!ATTLIST objectCategory name #CDATA #REQUIRED>
<!ATTLIST objectCategory frequency #CDATA>
<!ELEMENT policy (cred-expr, target, path?, priv, type, prop)>
<!ATTLIST policy id ID #REQUIRED>
<!ELEMENT cred-expr (#PCDATA)>
<!ELEMENT target (#PCDATA)>
<!ELEMENT path(#PCDATA)>
<!ELEMENT priv empty>
<!ATTLIST priv value (read|write|execute)>
<!ELEMENT type empty>
<!ATTLIST type value (+|-)>
<!ELEMENT prop empty>
<!ATTLIST prop type (no-prop|first-level|cascade)>
]>
```

Fig. 3 – DTD of the subject policy file.

2. *Credentials*: consider the credentials of the subjects and prior to considering a new policy triggered by a subject for clustering we evaluate both the subject's interests and the credentials.

Clustering is executed by the *clustering subjects* module (Fig. 1) in order to continuously update subject policy files with the policies triggered by other subjects belonging to the same cluster. Moreover, the list of policies will be updated automatically and it will be modified every time the interest of the associated subject changes. We also consider a threshold defining the maximum number of policies that can be added in the policy list concerning an object category, and every time this threshold is exceeded, garbage collection takes place (i.e. update of the policy list to contain only frequently triggered policies).

3.3. *The access request evaluation process*

Each access request can be modeled by the tuple (*cred-expr*, *target*, *path*, *priv*), which involves the elements similar to those defined previously in the policy tuple (in Section 2). The access request evaluation procedure proposed here under the dynamic update approach involves the following tasks:

- *Task 1*: find the subject policy file associated with the requesting subject.
- *Task 2 (implemented by scanSubjectPolicyFile( ))*: scan the list of policies in the subject policy file related to the category of the requested object until finding a match, i.e. evaluate the policies until you find a policy that can be triggered. If a match is found, return the policy to the access control module else execute task 3.
- *Task 3 (implemented by scanPolicyBase( ))*: find in the policy base the list containing policies related to the *obj-cat* of the requested object and scan it until you find a policy that can be triggered. If a match is found, return the policy to the access control module else execute task 4.
- *Task 4 (optional)*: return the policy id value to the access control module.

The overall access request process is sketched in Fig. 4 where the above tasks are highlighted in an algorithmic fashion. The functionality of this algorithm is dependent on the following characteristics:

- The objects are organized into categories according to their XML Schema. For instance, with reference to the example given in Fig. 1, since `payroll.xml` is an instance of `payroll.dtd`, then such XML file belongs to the object category *payroll* (object categories are identified by the term *obj-cat*).
- Credential types are not organized into hierarchies for simplicity reasons.
- The policy base is also organized according to the object category of the object that a policy protects.

```
Algorithm 1 : Access Request Process
INPUT: A request r=(cred-expr, target, path, priv)
OUTPUT: A policy id that can be triggered or a null value if no policy can be triggered.

PolicyId pid=null;
find the subjectPolicyFile associated with the requesting subject  //Task 1
pid=scanSubjectPolicyFile(r, subjectPolicyFile);  //Task 2
if pid!=null: return pid;
else: goto policy base;
      pid=scanPolicyBase(r);    //Task 3
      return pid;  //Task 4
```

Fig. 4 – The access request processing algorithm.

- Only the no-prop propagation option is considered for simplicity reasons.

The two above mentioned functions, namely the `scanSubjectPolicyFile()` (in task 2) and the `scanPolicyBase()` (in task 3) are highlighted in Fig. 6. Fig. 6(a) summarizes the *scanSubjectPolicyFile* employed by the access control module to verify whether the access request can be granted only by inspecting the subject policy file. This function makes use of the *policyEvaluation()* function (given in Fig. 5), that compares the relating fields of an input policy p and an access request *r* to discover if they match (in case of no match true is returned). More specifically, in comparing policy p and request *r* the following checking actions take place:

1. Is the *priv* of the request equal to that of the policy?
2. Does the requesting subject own the credential type given in the *cred-expr* of the request?
3. Is the *path* of the request equal to that of the policy and is any of the following cases also addressed?
   a. The *target* of both policy and request is similar.
   b. The *target* of the request is an XML file and the evaluated policy's target is the DTD that is followed by the request's target.

In case that there is no matching policy found in the subject policy file, we have to scan the part of the policy base containing policies relating to the *obj-cat* of the target in the input access request. Such a task is implemented through function *scanPolicyBase()* which takes as argument the request *r*, and returns the *policy id* if an appropriate policy can be found (summarized in Fig. 6(b)). The following section presents the complexity analysis of the access request evaluation process by estimating the complexity of each task described previously.

4. Complexity analysis of access request evaluation procedure

In order to evaluate the proposed access request process functions we identify the main parameters involved which are summarized in Table 4. More specifically we consider that the number of object categories is *n* and that for the subject policy file of subject i, in the *j*-th object category we have $(soc_i)_j$ policies. Since we consider to adopt garbage collection, each $(soc_i)_j$ will be smaller than a threshold msoc which is the maximum number of policies that can be found in the policy list of an object category. Moreover, the policies in the policy base are also organized according to the object category of the resource they refer to and if the number of policies for a category *i* is poc_i the maximum number of policies (under the *n* object categories) is $mpoc = \max(poc_1, poc_2, \ldots, poc_n)$.

According to the dynamic update approach the response time of the access request evaluation process will be calculated by adding the completion times of the following tasks (discussed in Section 3.3):

1. *Task 1*: find the subject policy file associated with the requesting subject.
2. *Task 2*: scanning subject policy file in order to retrieve an appropriate policy (implemented by scanSubjectPolicyFile()).

```
boolean policyEvaluation (Policy p, Request r)
{
        Boolean found=false;
        goto the associated subject file

        /*The loop evaluates the cred-expr field of the policy
       against the subject types found in the subject profile of the subject subj-id. */

        for(i=0;i<c_i;i++)
        {   if (priv of policy p is equal to that of request r)
           {   if((the p credential type of cred-expr == credential type i of subj-id)
               {found=true;
                if (cred-expr of r is a credential expression referring to
                                        credential type i)
                { open credential file referring to type i;
                  if (attribute condition is addressed)
                            found=true;
                  else found=false;
                }
               }
               if (found=true)
                    if ((path of policy p is null)|| (
                (( path of policy p is equal to that of request r)&&(target of
                            request r is equal to that of policy p))||
                ((target of request is an XML file)&&(target of policy is the DTD that
                      follows the target of request)))
                {  return true;}
            }
        }
    }
    return false;
}
```

Fig. 5 – The policy evaluation function.

Table 3 – Times associated with the access request evaluation process

Parameter	Description
RT_{ARE}	Time for evaluating an access request
RT_{FSPF}	Time for finding the appropriate subject policy file (completion time of task 1)
RT_{SSPF}	Time for scanning subject policy file and retrieving an appropriate policy (completion time of task 2)
RT_{SPB}	Time for scanning policy base and retrieving an appropriate policy (completion time of task 3)
T_{Send}	Time for sending the policy id to the access control module (completion time of task 4)
RT_{PE}	Completion time of the policy evaluation function.
T_{SCF}	Time for opening a credentials file
T_{FMP}	Time for scanning the policy list until we find a matching policy.
RPL_s	Time for retrieving a policy list from a specific subject policy file.
RPL_p	Time for retrieving a policy list from the policy base.

3. *Task 3* (*this task is only executed in the worst case*, i.e. no associated policy is found in the subject policy file): scanning policy base for finding a policy to be triggered (implemented by scanPolicyBase( )).
4. *Task 4*: send the policy identity to the access control module.

Therefore, the response time of the access request evaluation process in the *worst case*, when no policy is found in the subject policy file, is given by the mathematical formula (1):

$$RT_{ARE} = RT_{FSPF} + RT_{SSPF} + RT_{SPB} + T_{Send} \quad (1)$$

while in the *mean case* (which is estimated to take place in the 80–90% of cases) the total response time is given by formula (2).

$$RT_{ARE} = RT_{FSPF} + RT_{SSPF} + T_{Send} \quad (2)$$

The definition of the times involved in formula (1) are given in Table 3.

In order to evaluate the overall complexity of the proposed access control approach we should estimate the complexities involved in each of the four tasks and thus, the total complexity of the dynamic update approach in the *worst case* will be:

$$O_{ARE} = O_{FSPF} + O_{SSPF} + O_{SPB} + O_{Send} \quad (3)$$

while in the *mean case* it will be:

$$O_{ARE} = O_{FSPF} + O_{SSPF} + O_{Send} \quad (4)$$

where

- O_{ARE}: complexity of access request evaluation process.
- O_{FSPF}: complexity of finding the associated subject policy file (task 1).
- O_{SSPF}: complexity of scanning subject policy file to find a policy to trigger (task 2).
- O_{SPB}: complexity of scanning policy base to find a policy to trigger (task 3).
- O_{Send}: complexity of returning an identity of the policy to be triggered (task 4).

Table 4 – Parameters characterizing the access control procedure

Parameter	Description
m	The number of subjects.
n	The number of object categories.
$(soc_i)_j$	The number of policies in the policy list concerning the j-th object category in the subject policy file of the subject i.
c_i	The number of credential types of subject i
msoc	Maximum number of policies in a policy list referring to an object category in a subject policy file.
poc_i	The number of policies in the policy base concerning objects belonging in the i-th object category.
mpoc	Maximum number of policies in a policy list referring to an object category in the policy base.
mc	Maximum number of subject types that a subject can have.

4.1. *Calculating complexities for each task*

In order to calculate the complexity of the access request evaluation process we should first evaluate the complexity of each of the four involved tasks discussed in Section 3.3. Since tasks 2 and 3 involve the policy evaluation function it would be better if we begin our analysis by computing the complexity of this function.

4.1.1. *Policy evaluation function (Fig. 5)*

Since the *cred-expr* field is a credential type or a credential expression (referring to a credential type), in order to decide if there is a match we have to compare it with the credentials in the subject profile of the requesting subject. In case it is a credential expression, the associating credential file should be opened in order to check if the condition is addressed. Such a function will be executed at most once and we define this as a time by the constant T_{SCF}. Therefore, if we consider that c_i is the number of credential types of subject i, for each policy evaluation we may have c_i checking actions. If $mc = \max(c_1, c_2, \ldots, c_m)$, for the m subjects, then the completion time of the policy evaluation function will be:

$$RT_{PE} = T_{SCF} + mc \times T_{check}$$

Since T_{check} is considered to be constant, the complexity of the policy evaluation function is (Table 4)

$$O_{PE} = O(T_{SCF} + mc \times T_{check} +) \approx O(mc). \quad (5)$$

4.1.2. *Task 1: find the subject policy file associated with the requesting subject*

In the dynamic update approach, we first scan the subject policy file of the requested subject in order to discover a policy that can be triggered. The time for the retrieval of the appropriate subject policy file is a constant (defined by RT_{FSPF}) since indexes are used on finding the appropriate subject policy file. The time for accessing a specific entry in the file using the index is proportional to the mean seek time,

rotational latency and the page transfer time which are specified for every storage system. Thus, the complexity of this task is:

$$O_{FSPF} = O(RT_{FSPF}) \approx \mathbf{cs}_1$$

where cs_1 is a constant.

4.1.3. Task 2: (scanSubjectPolicyFile – Fig. 6(a))

This task's complexity will be evaluated by the actions of (a) finding the appropriate policy list, and (b) scanning the policy list until we find a matching policy, i.e:

$$RT_{SSPF} = RPL_s + T_{FMP}$$

The first action is executed in a constant time RPL_s since we also use indices for finding policy lists in the subject policy file, whereas for the second action we need to use the function *getPolicy()* which involves the retrieval of a policy. Since msoc is the maximum number of policies in a list the loop should be executed msoc times and every step of the loop requires a call to the *policyEvaluation()* function which according to Eq. (5) has a complexity of $O(mc)$. Thus, the complexity of the second action is $O(msoc \times mc)$. Thus, the complexity of the whole task is:

$$O_{SSPF} = O(RPL_s + msoc \times mc) \approx c + O(msoc \times mc) \approx \mathbf{O}(\mathbf{msoc} \times \boldsymbol{mc}).$$

4.1.4. Task 3: (scanPolicyBase – Fig. 6(b))

This task's complexity is characterized by the complexity of (a) finding the appropriate policy list in the policy base, and (b) scan the policy list until we find a matching policy which is send to the access control module. The completion time of this task is given by the following formula:

$$RT_{SPB} = RPL_p + T_{FMP}$$

Again, the first action is executed in a constant time RPL_p and since mpoc is the maximum number of policies in a list, the loop should be executed msoc times. Every step of the loop involves the execution of function `policyEvaluation` which has a complexity of $O(mc)$, i.e. the complexity of the second action is $O(mpoc \times mc)$ and the complexity of the whole task is:

$$O_{SPB} = O(RPL_p + mpoc \times mc) \approx c + O(mpoc \times mc) \approx \mathbf{O}(\mathbf{mpoc} \times \boldsymbol{mc}).$$

where c is a constant.

4.1.5. Task 4: sending reply to the access control module

If no policy is found either in the subject policy file or in the policy base a failure message is sent to the access control module otherwise the policy identity is sent. The time required for this task is a constant and thus:

$$O_{Send} = O(T_{Send}) \approx \mathbf{cs}_2$$

4.2. Overall complexity of the access request evaluation process

In order to evaluate the complexity of the proposed approach, we will consider an indicative example: suppose that a person A.V. is the general manager of the "X-Company" and another person K.S. is the manager of the human resources department. After conducting clustering both of the above subjects belong to the same cluster since they have similar interests and credentials. If one of their common interests is objects belonging to the "payroll" category, then the subject policy files of both of them will contain policies referring to this category. Moreover, consider the following actions:

- A.V. requests to read the salary element of `payroll.xml` following the `payroll.dtd`. In such a case the policy 1 from Table 2 is triggered and access is allowed. Since this access refers to an object belonging in the interests of the subject, an entry is added in the policy list of the associated object category of her subject policy file.
- K.S. has already read and modified the `hireDate` element of `payroll.xml` by triggering the second and the fourth policy, respectively (Table 2). Since, the above object belongs to her interests, the two policies are added to her subject policy file.
- In idle times the *update subject policy file* module will enrich the subject policy file of A.V. with those policies triggered by K.S. (and the other members of the cluster) which are included in the payroll category policy list of her file and

```
/* retrieve a policy from the Source which is
/*             given as an argument. */

Policy getPolicy(Source from);

PolicyId scanSubjectPolicyFile (Request r,
                                Source s)
{
    find the policy list associated with the
             obj-cat of target of request r;
    do
    {   policy=getPolicy(s);
        if (policyEvaluation(policy, r)):
            return policy.id;
    } while (the obj-cat of the policy target
           is equal to that or request target)
    return null;
}
```

(a)

```
PolicyId scanPolicyBase(Request r)
{
    find list associated with obj-cat of
                                    r.target;
    do
    {
        policy=getPolicy(PolicyBase);
        if (policyEvaluation(policy, r):
            return policy.id;
    } while (obj-cat of the policy target
           is equal to that or request target)
    return null;
}
```

(b)

Fig. 6 – Find a matching policy in the (a) subject policy file and (b) policy base.

VS. Therefore, the list of K.S. will be enriched with the first policy and that of A.V. with the second policy.

Since subjects tend to request access to the same objects most of the time, after some accesses their subject policy files will contain the policies that are most likely to be triggered. Thus, *the worst case*, when both the subject policy file and the policy base should be scanned, *will occur very scarcely*. Moreover, since the files are also enriched with those policies that have been triggered by other members of the same cluster, few accesses are enough to have a subject policy file that will satisfy most of the access requests.

It should be noted that *under a typical access request processing approach* when no dynamic update is involved, the overall complexity is estimated by the execution of tasks 1, 3 and 4 and thus:

$$O_{ARE} = O_{FSPF} + O_{SPB} + O_{Send} \approx cs_1 + O(mpoc \times mc) + cs_2 = \mathbf{O(mpoc \times mc)}$$

Under the dynamic update approach, two conditions may occur (in the worst case):

1. No policy is able to be triggered from subject policy file (thus the whole list of associated policies should be checked).
2. No policy can be triggered form the list of policies associated with the specified object category from the policy base, and in such a case access is denied.

Therefore, in the *worst case* executing all of the four tasks (according to formula (3)) will result in an overall complexity of:

$$O_{ARE} = O_{FSPF} + O_{SSPF} + O_{SPB} + O_{Send} \approx cs_1 + O(msoc \times mc) + O(mpoc \times mc) + cs_2 = \mathbf{O[(msoc + mpoc)mc]}$$

since both loops have to be executed. Since $msoc \ll mpoc$ then the complexity of the access request evaluation in the dynamic update approach converges to that of the typical approach and it will be:

$$O_{ARE} \approx \mathbf{O(msoc \times mc)}$$

In the *mean case* task 3 is not executed and therefore the complexity according to formula (4) will be:

$$O_{ARE} = O_{FSPF} + O_{SSPF} + O_{Send} \approx cs_1 + O(msoc \times mc) + cs_2 = \mathbf{O(msoc \times mc)}$$

Table 5 summarizes the complexities of the dynamic update approach and the typical approach in the worst and mean cases, respectively.

Table 5 – Complexity analysis of the dynamic update and the typical approach for the worst and mean case

	Type of approach	Complexity
Worst case	Dynamic update	$O[(msoc + mpoc)mc] \approx O(mpoc \times mc)$
	Typical	$O(mpoc \times mc)$
Mean case	Dynamic update	$O(msoc \times mc)$
	Typical	$O(mpoc \times mc)$

Considering the above example, as subjects access the system and dynamic update is executed, the subject policy file is enriched with policies that are most likely to be triggered. Therefore, it is expected that most of the times only the subject policy file will be scanned in order to find the appropriate policy to be triggered (mean case). In such a case the complexity is $O(msoc \times mc)$, since only the subject policy file should be scanned. Thus, it is expected that in most of the cases the complexity of the access request evaluation will be significantly reduced and in the worst case (which will occur rarely) it will converge to the typical approach.

5. Conclusions and future work

In this paper, we have proposed an approach for improving access control execution time by using clustering. We have tried to address the needs of subscription environments where subjects' attributes are given through credentials. The dynamic update approach clusters subjects according to their interests, which are identified by their access history. The members of a specified cluster are likely to trigger the same policies and therefore they are associated with them and upon a request there is no need to scan the whole policy base in order to find the appropriate policy to trigger.

According to a complexity analysis we found that the proposed approach results in improved times. Due to the above encouraging results and the few research work conducted in the speeding up field, the proposed module is an immediate future plan. We have already begun the implementation of the subscription and the clustering module. Moreover, we have defined the format of the involved files (subject policy file, access log file, etc.) and we have checked the performance of the dynamic update approach by also implementing a first access control module protecting confined environments protecting XML files where few subjects can have access. Since the first results are really promising, after we have implemented the whole access control and dynamic update module we will try to improve it in order to work for even open environments where subjects are unknown. Furthermore, since lately there is interest in controlling access to web services, these promising results may find an interesting application to environments protecting them.

REFERENCES

Baldi P, Frasconi P, Smyth P. Modeling the Internet and the web. Wiley; 2003.

Bertino E, Castano S, Ferrari E. Securing XML documents with author-X. IEEE Internet Computing May–June 2001;21–31.

Carminati B, Ferrari E. AC-XML documents: improving the performance of a web access control module. In: Proceedings of the 10th ACM symposium of access control models and technologies. Sweden: Stockholm; 2005.

Castano S, Ferrari E. Protecting datasources over the web: policies, models and mechanisms, web-powered databases. Idea Group Publishing; 2003. p. 299–330.

Chakrabarti S. Mining the web. Morgan Kaufmann; 2003.

Jain AK, Murty MN, Flynn PJ. Data clustering: a review. ACM Computing Surveys 1999;31(3):264–323.
Jeng H-J, Chen Z, Ma W-Y. A unified framework for clustering heterogeneous web objects. In: Proceedings of the third international conference on web information systems engineering; 2002.
Jiang J. Using latent semantic indexing for data mining. Thesis presented for the Master of Science Degree, The University of Tennessee, Knoxville; 1997.
Middleton SE, Shadbolt NR, De Roure DC. Ontological user profiling in recommender systems. ACM Transactions on Information Systems 2004;22(1):54–88.
Murata M, Tozawa A, Kudo M, Hada S. XML access control using static analysis. In: Proceedings of the 10th ACM conference on computer and communications security, Washington, DC, USA; 2003.
Pallis G, Stoupa K, Vakali A. Storage and access control issues for XML documents, Web information systems. Idea Group Publishing; 2004. p. 104–40.
Stoupa K, Vakali A. Policies for web security services, book chapter in web and information security. In: Ferrari E, Thuraisingham B, editors. Idea Group Publishing; 2006. p. 52–72.
Wang Q, Makaroff DJ, Edwards HK. Characterizing customer groups for an E-commerce website. In: Proceedings of ACM conference on electronic commerce (EC' 2004), New York, USA; 2004.
Ward P, Smith CL. The development of access control policies for information technology systems [Elsevier]. Computers and Security 2002;21(4):356–71.
Winslett M, Ching N, Jones V, Slepchin I. Using digital credentials on the world-wide web. Journal on Computer Security 1997;5: 255–67.
Xie Y, Phoha VV. Web user clustering from access log using belief function. In: Proceedings of first international conference on knowledge capture, Victoria, British Columbia, Canada; 2001.
Yang C, Li C. Access control in a hierarchy using one-way hash functions [Elsevier]. Computers and Security 2004;23(8): 659–64.

Konstantina E. Stoupa received her B.Sc. degree in Computer Science from the Department of Informatics at the Aristotle University of Thessaloniki where she is currently a Ph.D. graduate student. She has also received the M.B.A. degree from the University of Macedonia in Thessaloniki, Greece. She has published papers on Access Control using XML, Delegation and Revocation of Authorizations and Roles, and Storage Subsystems in international journals and conferences. Her primary research interests include Access Control especially in distributed, widely accessed environments. She is also a teaching member in the Information Management Department of Technological Educational Institute of Kavala.

Athena I. Vakali received a B.Sc. degree in Mathematics from the Aristotle University of Thessaloniki, Greece, a M.Sc. degree in Computer Science from Purdue University, USA (with a Fulbright scholarship) and a Ph.D. degree in Computer Science from the Department of Informatics at the Aristotle University of Thessaloniki. Since 1997, she is a faculty member of the Department of Informatics, Aristotle University of Thessaloniki, Greece (currently she is an Assistant Professor). Her research interests include design, performance and analysis of storage subsystems and data placement schemes for multimedia and Web based information. She is currently working on Web data management and she has focused on XML data storage issues. She has published several papers in international journals and conferences.