SECURE AUTHORISATION FOR WEB SERVICES

Sarath Indrakanti Vijay Varadharajan Michael Hitchens
Information and Networked Systems Security Research, Division of ICS
Macquarie University, Australia.
{sindraka, vijay, michaelh}@ics.mq.edu.au

Rajat Kumar
Microsoft Pty., Australia
rajk@microsoft.com

Abstract: In this paper, we investigate the authorisation service provided by Microsoft® .NET MyServices [1]. We propose modifications and extensions to eXtensible Markup Language (XML) [2] based data structures' schemas to support a range of commonly used access policies in commercial systems. We have developed the modified access evaluation algorithms that take the proposed extensions into account. The extensions proposed in this paper have been used in the specification and analysis of a practical application involving access control to electronic patient records in hospitals.

Key words: Web Services, Security, Access Control and Authorisation.

1. MICROSOFT .NET MYSERVICES

Microsoft .NET MyServices [1] is the name for a set of XML message interfaces delivered as part of the Microsoft .NET initiative. It is a platform for building user-centric Internet applications. It provides a private, secure "digital safe deposit box" where users can place their personal data. This digital storage box serves as the single, central location where users can store, update, and control all of their personal data. .NET MyServices aims to provide a fine granularity of control to the user (owner of the data) once the information is placed in the digital safe deposit box. The user may choose to share that information with friends, family, groups with which they have an association, and businesses. Additionally, users can sign up to receive alerts on a number of desktop and mobile devices. .NET MyServices is implemented as a set of web services that utilise industry standards including XML and SOAP.

These services can use .NET Passport [10] to authenticate users and .NET Alerts to notify users of important events through a wide range of client devices and to gain access to key pieces of data. .NET MyServices aims to provide ownership and control of personal data to individual users.

2. AUTHORISATION SERVICE

In .NET MyServices, the following items are used in the design of the authorisation system.

The definition of a role in terms of the permissions that it has on different objects is specified using a role-template. A role-template defines the set of methods allowed by the role-template and the Scope visible to each method. A role-template has a name (e.g. rt1)

The mapping between the role-template and the user is defined by the function "Role". Then the users have a list of roles that are authorised to access the content in documents associated with those users. This is defined in role-list, which describes what information to share, who to share with and how to share it.

A role-map describes what information is to be shared and how that sharing is to occur. It defines for each role (from the role-list), what the allowable methods are, and what scope of data is visible while using this method. The role-map is the same for all instances of a particular service, and is authored by the implementer of the service. Hence the objective of the role-map is to simplify the security authorisation layer as it appears to the end user. It does this by specifying the fixed set of access patterns that occur on a given service.

For schema and examples of role-list and role-map documents, please refer to [3].

3. PROPOSED EXTENSIONS TO AUTHORISATION SERVICE

We propose extensions to the existing authorisation system in .NET MyServices thereby enabling it to support the following range of access control policies - Conditions and Actions based access policies, Role based access policies, Static and Dynamic separation of duty, Delegation, Chinese-wall policy and Joint action based policies. In each case, we will first describe the access policy requirements to be supported by the web service, discuss the existing limitations and propose extensions to data structures and schemas and specify the authorisation service. In describing our extensions to the authorisation service, we will use the term "provider" to denote an entity that owns the content document (or web service data) and the term "user" to denote an entity that accesses the provider's document. Due to space limitations, we have not discussed extensions to implement Chinese-Wall policy, Delegation policy and 'limited number of accesses' policy in this paper. They can be found in [4] along with a detailed description of our extended authorisation model.

Through out this paper, when we use the element 'role' in role-list schema skeleton extensions, we implicitly mean that it is a role, a delegated-role or a dynamic-role. Delegated-role and dynamic-role are defined in the sections to follow.

For simplicity, we only show the schema skeleton for ‘role’ element. The same applies to both methodRef and jointMethodRef elements in role elements in the role-list document. For simplicity, we only show the schema skeleton for methodRef element. Every .Net MyServices provider has a content document that actually stores his/her data and an access control list (ACL) document that stores authorisation information. We suggest every provider provisioned for a .NET MyService should also be provisioned a log document along with the content document and ACL (role-list) for auditing purposes. We will not be discussing the auditing aspects in this paper.

To begin with, we propose a small modification to the role-map schema that helps to identify uniquely a method-scope pair in the role-template. This is achieved by including an “id” attribute. This new attribute is introduced so that this pair can be referred to in a role element in a provider’s role-list document. We will see later that this minor extension will be useful in the specification of fine-grained access control policies. An example with this minor extension (in bold) is shown in figure 1.

```
<roleMap>
      <hs:scope ... />
      <hs:roleTemplate name="rt0">
         <hs:method name="query" scopeRef="1" id = “1”/>
         <hs:method name="insert" scopeRef="1" id = “2”/>
      </hs:roleTemplate>
</roleMap>
```

Figure 1. Role-map extension with preliminary extension

We are now in a position to consider the extensions to schemas and data structures that would enable us to support the range of access control policies mentioned above.

3.1.1 Conditions and Actions

The existing authorisation service in .NET MyServices does not have the ability to specify conditions to methods when performing access control checks. Each role-template provides access to a number of methods on various scopes to a user. To further restrict access on each of those methods based on arbitrary conditions, we introduce a new element *methodRef.* This construct enables us to associate conditions and actions with a method. A method can have a condition and one or more actions associated with it. The modified schema skeleton for role-list document is shown in figure 2. The elements and attributes introduced to support Conditions and Actions based policies are described below.

```
<hs:roleList ... >1..1
   <hs:scope ... />
   <hs:role ... >0..unbounded .....
      <hs:subject .. />
      <hs:methodRef idRef = "..." >0..unbounded
         <hs:condition operation = "...">0..1 {any}
            <hs:predicate name = "...">0..unbounded
               <hs:parameter name= "..." value = "...">0..unbounded {any}
               </hs:parameter>
            </hs:predicate>
            <hs:condition operation = "..."/>0..unbounded
         </hs:condition>
         <hs: action id = "..." type = "...">0..unbounded
            <hs:predicate name = "...">0..unbounded
               <hs:parameter name= "..." value = "...">0..unbounded {any}
               </hs:parameter>
            </hs:predicate>
         </hs:action>
      </hs:methodRef>
      <hs:expiresAt>0..1</hs:expiresAt>
         {any}
   </hs:role>
</hs:roleList>
```

Figure 2. Modified schema skeleton for role-list document

/roleList/role/methodRef element refers to the ID of the method for this role in its role-template. Multiple method IDs can be referenced here. For instance, methodRef = "1,2". In this case the conditions and actions are performed for the methods with id 1 and 2. */methodRef/@idRef* attribute gives the reference to the method id in the role template referred to by this role.

/roleList/role/methodRef/condition element is a Boolean formula. If the formula evaluates to true then the access is granted. In general, the condition expression can involve several predicates involving standard logic operators such as 'and', 'or' and 'not'. Defining a condition element within another condition element can specify recursion. Each predicate may have one or more parameters.

A parameter can be specified in three ways: (a) as a string in its 'value' attribute (b) as a list of arbitrary elements and (c) as a single 'function' element. If a value attribute is specified, all the child nodes are ignored even though they are specified. Also, if some elements other than 'function' elements are specified, every 'function' element is ignored even though it is specified. Consider an example of a predicate: compareStr(s1,s2,s3) is a predicate where s1 is 'eq'(equal) or 'neq' (not equal) and s2 and s3 are two strings to be compared. This predicate compares two strings according to the operator. Consider an example of a function: getDate() is a function, which takes no parameters, and it returns a text node representing the current date.

/roleList/role/methodRef/action specifies actions that need to be performed when an access request is made to the method(s) through methodRef. Each action has an ID so that it can be uniquely referenced (using Xpath). The type attribute is used to define whether the action is only performed when the access is granted (on_access) or is always performed whether or not access is granted (always). The predicates in

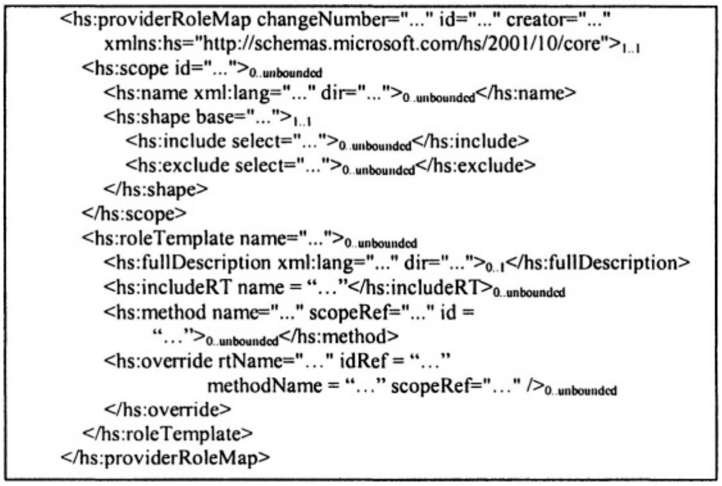
action expressions are similar to those defined in conditions except that there should be at least one predicate element.

3.1.2 Role Hierarchy

One of the advantages of the role-based (RBAC) [5][9] approach is that it can model the privileges in an organisation more effectively. A simplistic view of an organisation structure is a hierarchical ordering of responsibilities, with the senior positions encompassing all the privileges of the junior positions, with some extra privileges. Such a role hierarchy can be achieved by providing hierarchy in role template definition. At present, most of the services in .NET MyServices use only about 3-5 role templates and there is no hierarchy. We propose the following extensions to .NET MyServices authorisation service to enable role based access control and role hierarchy. Each service should define a dynamic role-map at the user level. The role-map should be static at the web service level but the provider should be able to change it when needed. Therefore, every provider should have his/her own role-map along with the web service role-map. Let us call the provider's role-map the provider-role-map (XML document name = providerRoleMap). New scopes can now be defined in this provider-role-map instead of in the provider's role-list. We also remove all scope definitions from the role-list for a provider and move them to the provider-role-map for consistency. Role-templates' ID in provider-role-map should be anything other than rT0, rT1, rT2, rT3 and rT99 as these are the default role-template IDs. Every method in the default web service role-map and provider-role-map should have an ID number attribute to uniquely identify the method.

When a provider is provisioned to a service, the provider has his or her own provider-role-map document along with the content, system, log and acl (role-list) documents. The XML Schema skeleton for providerRoleMap is shown in figure 3.

The schemas for scope, role-template and their child elements and their attributes are the same as for those defined for the web service role-map. Other elements and attributes introduced to implement role hierarchy policy are described below.



```
<hs:providerRoleMap changeNumber="..." id="..." creator="..."
    xmlns:hs="http://schemas.microsoft.com/hs/2001/10/core">1..1
  <hs:scope id="...">0..unbounded
    <hs:name xml:lang="..." dir="...">0..unbounded</hs:name>
    <hs:shape base="...">1..1
      <hs:include select="...">0..unbounded</hs:include>
      <hs:exclude select="...">0..unbounded</hs:exclude>
    </hs:shape>
  </hs:scope>
  <hs:roleTemplate name="...">0..unbounded
    <hs:fullDescription xml:lang="..." dir="...">0..1</hs:fullDescription>
    <hs:includeRT name = "..."</hs:includeRT>0..unbounded
    <hs:method name="..." scopeRef="..." id =
          "...">0..unbounded</hs:method>
    <hs:override rtName="..." idRef = "..."
               methodName = "..." scopeRef="..." />0..unbounded
    </hs:override>
  </hs:roleTemplate>
</hs:providerRoleMap>
```

Figure 3. Provider-role-map schema skeleton

providerRoleMap/roleTemplate/includeRT element is used to specify hierarchy. A role-template can inherit another role-template's privileges with this element. See figure 4. */includeRT/@name* attribute is used to refer to the role-template whose privileges can be inherited.

providerRoleMap/roleTemplate/override element is used to override either a scope or method name in a method element in a role-template. This can be used to implement role hierarchy as shown in scenarios below. */override/@rtName* attribute refers to the role-template (that is inherited) in which the overriding has to be done. If the mentioned role-template is not inherited with an includeRT element, override element is ignored. */override/@idRef* attribute is provided to refer to the method element (using its ID) from the inherited role-template that is to be overridden. */override/@methodName* is an optional attribute. It is to be used only when the method in the inherited role-template is to be overridden. */override/@scopeRef* is an optional attribute. It is to be used only when the scope in the inherited role-template is to be overridden.

Note that the role templates inherited can either be the default role templates from the web service role-map or from provider-role-map. To achieve role hierarchy, a suitable new role-template (with needed hierarchy definition) in the provider-role-map is created. Then it is to be referred in the suitable role element(s) in provider's role-list. For illustration purposes, a few scenarios are shown in figure 4. Role-templates rt10, rt20 and rt30 are defined. rt30 inherits privileges from both rt10 and rt20. Method with ID '4' from rt10 is overridden in rt30 to 'create'. 'insert' method's scope from rt20 is overridden to 2 in rt30.

```
<hs:roleTemplate name="rt10">
   <hs:method name="query" scopeRef ="1" id = “1”/>
   <hs:method name="insert" scopeRef="3" id = “2”/>
   <hs:method name="replace" scopeRef="3" id = “3”/>
   <hs:method name="delete" scopeRef="3" id = “4”/>
</hs:roleTemplate>
<hs:roleTemplate name="rt20">
   <hs:method name="query" scopeRef ="5" id = “1”/>
   <hs:method name="insert" scopeRef="5" id = “2”/>
</hs:roleTemplate>
<hs:roleTemplate name="rt30">
   <hs:includeRT name = “rt10”>
   <hs:includeRT name = “rt20”>
   <hs:override rtName="rt10" idRef= “4”
                        methodName = “create” />
    <hs:override rtName = “rt20” idRef = “2” scopeRef ="2" />
   <hs:method name="query" scopeRef="4" id = “1”/>
   <hs:method name="insert" scopeRef="4" id = “2”/>
   <hs:method name="replace" scopeRef="4" id = “3”/>
</hs:roleTemplate>
```

Figure 4. Role-template example with role hierarchy policy

3.1.3 Dynamic Separation of duty

Dynamic separation of duty [6] can only be determined during system operation. This is whereby a user having chosen a specific action or role at run time is not authorised to perform another action or assume another role. To begin with, the user is allowed to perform either of the actions or assume either of the roles. We achieve dynamic separation of duty by adding a new element called ‘dynamicRole’ to the schema of role-list. The modified schema skeleton for role-list document is shown in figure 5.

Extensions to the role-list schema are described below:

/roleList/dynamicRole element is introduced to support dynamic separation of duty policies. Most of the child elements and attributes for this element are exactly the same as for ‘role’ element. Therefore, only the new added elements and attributes are described here. */dynamicRole/RT* - A minimum of 2 (or more) RT elements should be used in each dynamicRole element. This element encapsulates both role-template and the scope on which it is used along with the conditions and actions on the methods. The user with a dynamic-role has access to any of the role-templates and on any of the scopes provided by the dynamic-role. The role-template - scope pairs are mutually exclusive. Once the user accesses a role-template on a scope (through a RT element), s/he is not allowed to access any other privileges provided by his/her other RT elements. */dynamicRole/RT/scopeRef* element stores the scope to which the user has access with the role-template in this RT element. */dynamicRole/RT/roleTemplateRef* element stores reference to one of the role-templates’ ID to which the user has access.

/roleList/dynamicRole/usedRT/ element is used to store information to enable dynamic separation of duty. */usedRT/idRef* element stores the ID reference to one of the RT elements (in this dynamic-role). Until the first time the user accesses any of the privileges in his/her RT elements, this is a null element. Once the user accesses

privileges provided by any of the RT elements, the ID of that RT element is written here by the authorisation service.

```
<hs:roleList ... >1..1
  <hs:scope ... />
  <hs:role ... />
  <hs:dynamicRole scopeRef="..." roleTemplateRef="..." ....>0..unbounded
     .......
    <hs:subject ... />
    <hs:RT id = "...">2..unbounded
      <hs:scopeRef>0..1</hs:scopeRef>
       <hs:roleTemplateRef>1..1</hs:roleTemplateRef>
      <hs:methodRef idRef = "..." >0..unbounded
           <hs:condition operation = "...">0..1              ...
   </hs:condition>
       <hs:action id = "..." type = "...">0..unbounded .....
   </hs:action>
       </hs:methodRef>
    </hs:RT>
    <usedRT>1..1
      <idRef>1..1</idRef>
    </usedRT>
    <hs:expiresAt>0..1</hs:expiresAt>
    {any}
  </hs:dynamicRole>
</hs:roleList>
```

Figure 5. Modified schema skeleton for role-list document

All the methods in every RT element in a dynamicRole element implement at least the 'checkAccess' predicate, which does the following:

When the user accesses one of the role-templates (provided by the RT elements) on the scope allowed for the first time, it sets the idRef element in usedRT to the RT's ID.

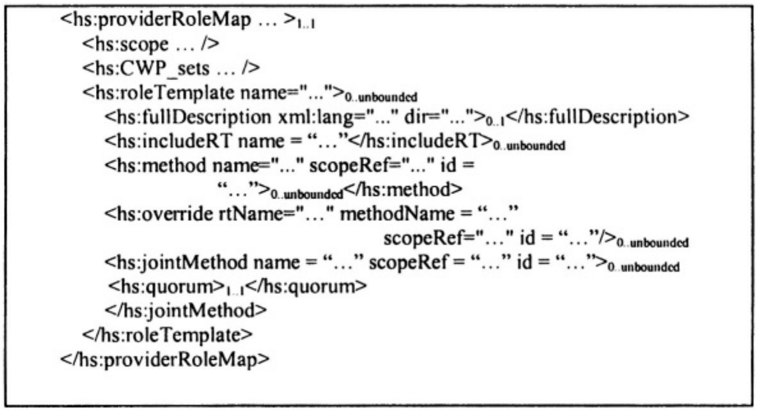
Next time a user with dynamicRole element tries to access a scope with a role-template provided by any other RT elements, access is denied. Access to RT with ID stored in usedRT element is only allowed.

3.1.4 Joint Action Based Policies

Joint action based policies [7] are used in situations where trust in individuals needs to be dispersed. Often this arises because individuals are trusted according to their expertise, which in turn maps the concept of trust to a specific set of actions. In delegation, there is a partial or complete granting of privileges, whereas in joint actions agents may acquire privileges, by working together in tandem, which none posses in isolation.

Consider the following example. A patient is admitted to the hospital if both the patient and a doctor agree. That is, the policy is that the doctor and patient jointly need to agree for the action "admission of the patient to the hospital" to be authorised. Modified XML schema skeletons for provider-role-map and role-list documents are shown in figures 6 and 7 below.

Schema extensions to provider-role-map document are as follows: *providerRoleMap/roleTemplate/jointMethod* element is added to a role-template. It defines a joint method and its quorum number. Its name, scopeRef and id attribute schemas are the same as method element's schema. They are not redefined here. */jointMethod/quorum* element gives the authorisation system the quorum number for this joint method. Quorum is the minimum number of users that have access to this joint method via the same role-template and call the joint method for it to be performed. Till the quorum is reached, a temporary flag is set in their role element as defined below.



```
<hs:providerRoleMap ... >1..1
  <hs:scope ... />
  <hs:CWP_sets ... />
  <hs:roleTemplate name="...">0..unbounded
    <hs:fullDescription xml:lang="..." dir="...">0..1</hs:fullDescription>
    <hs:includeRT name = "..."</hs:includeRT>0..unbounded
    <hs:method name="..." scopeRef="..." id =
                 "...">0..unbounded</hs:method>
    <hs:override rtName="..." methodName = "..."
                                    scopeRef="..." id = "..."/>0..unbounded
    <hs:jointMethod name = "..." scopeRef = "..." id = "...">0..unbounded
     <hs:quorum>1..1</hs:quorum>
    </hs:jointMethod>
  </hs:roleTemplate>
</hs:providerRoleMap>
```

Figure 6. Modified schema skeleton for provider-role-map document

Schema extensions to role-list document are as follows: */roleList/role/jointMethodRef*: There are as many jointMethodRef elements in a role element as there are joint-methods in the role-template it is referring to. It is similar to a methodRef element expect that it has a boolean flag element. All other elements are exactly the same as those defined in methodRef schema. */jointMethodRef/@idRef attribute* refers to the joint method's ID and is used to refer to the joint-method in this role's role-template.
/roleList/role/jointMethodRef/condition/flag element is used to store if joint access request is made. */condition/flag/reducedScope* is an optional element. It may so happen that a user should gain authorisation only to perform a joint action on a further reduced scope than that permitted by her role-template. This scope element solves such purpose. Its shape element's schema is the same as shape element's schema for scope element in role-map. */condition/flag/value* element is by default set to false. When the request for the joint-method comes in from the user (with this role element), this element is set to true. Until the quorum is reached only the flag in the joint-method elements in different users' role elements is set to true. Once the quorum is reached, the actual action is performed on the content document of the provider.

```
<hs:roleList ...>1..1
    <hs:scope .../>
    <hs:role ... >0..unbounded ....
     <hs:subject .../>
    <hs:jointMethodRef idRef = "..." >0..unbounded
      <hs:condition operation = "...">0..1
        <hs:flag>1..1
          <hs:reducedScope>0..1
      <hs:shape base="...">1..1
              <hs:include select="...">0..unbounded</hs:include>
              <hs:exclude select="...">0..unbounded</hs:exclude>
       </hs:shape>
          </hs:reducedScope>
          <hs:value>1..1</hs:value>
          <hs:flag/>
          <hs:predicate name ..../>1..unbounded
          <hs:condition operation = "...">0..unbounded </hs:condition>
          <hs:action.../>0..unbounded
    </hs:jointMethodRef>
    <hs:methodRef ... />0..unbounded
    <hs:expiresAt>0..1</hs:expiresAt>
    {any}
  </hs:role>
</hs:roleList>
```

Figure 7. Modified schema skeleton for role-list document

A new predicate 'checkQuorum' is defined for all services that need to implement joint action based policy. Every jointMethodRef element in a role element must at least call this predicate in its condition. 'checkQuorum' predicate does the following:

- Whenever a call to a joint method is made, it checks the number of flags that are set to true in the role elements that have access to this joint method via the same role-template the caller has access to.
- If the number of flags set to true is more than or equal to the quorum number mentioned in the joint method definition in the role-template, then the actual change is made to the content document. Or else the flag in the caller's jointMethodRef element is set to true.

4. AUTHORISATION EVALUATION

The .NET MyServices authorization algorithm is redefined here to into account of the extensions proposed to the authorisation service as follows. Please see [1] for original authorisation algorithm.

The user, application and platform identities as well as the credential type are determined from the incoming message. The matching role from the provider's role-list document is then located. If a matching role or dynamic-role is not found, then the request fails with an authorisation fault. If a matching role is found and if it contains an expiresAt element, then the role is checked to see if it has expired. If not, then the role-template that is referenced by the role is found from the role-map or provider-role-map. This template is checked to see if the method requested is allowed. Using the scope from the role-map or provider-role-map (referenced by role-template), combined with the optional scope referenced by the role, the node set that is visible to this message is determined.

If a matching dynamic-role is found, and if it contains an expiresAt element, then the dynamic-role is checked to see if it has expired. If not, the idRef element's value of usedRT in the dynamic-role element is checked. If the idRef element's value is null, the role-template and scope referenced by first RT element are located from the role-map or provider-role-map that is referenced by the dynamic-role. This template is then used to determine whether the method or joint-method requested is allowed. The scope is used to determine if the operation requested is within that scope. If either the method is not allowed or the operation is not within the scope, then the role-template and scope referenced by second RT element are located. Then whether the method or joint-method requested is allowed by the template or not is determined as well as whether the operation requested is within that scope for all other RT elements. If none of the role-templates and scopes allows access to that requested method on the requested scope, then the message fails with an authorisation fault. The usedRT's idRef element is set to the value of the 'id' attribute of the RT element that contains role-templates and scopes with privileges requested in the message. If usedRT's idRef element's value refers to an RT element's ID, and if either the method is not allowed by the role-template in the RT element or if the operation requested is not within the scope allowed, the message fails with an authorisation fault. Using the scope from the role-map or provider-role-map, combined with the optional scope referenced by the RT element in the dynamic-role, the node set visible to this message is computed.

If the method requested is a normal method, all conditions are evaluated and if any one returns false, then a fail message is sent. All the actions with 'type' set to "always", are performed. All the actions with 'type' set to "on-access" are only preformed if condition is true. If the method requested is a joint-method then there exists at least one predicate (checkQuorum) in the condition element. The condition is evaluated. If it returns false, the message fails with an authorisation fault. If not, all the actions with 'type' set to "always", are performed. All the actions with 'type' set to "on-access" are only performed, if condition is true. The requested operation is then performed ensuring that the user has no access to information outside the scope computed above.

5. PRACTICAL APPLICATION

We have applied the proposed extensions to the authorisation service by developing an access control system for electronic patient records. In this section, we only briefly describe some of the various access policies mentioned above in Section 4 that have been specified in this system. For a detailed description of the system and the authorisation policy specification, refer to [4].

In our demonstration, we assume that an electronic Patient Record System (PRS) is used by several hospitals in a metropolitan city. Hence each hospital is the provider and patients are users. A patient record is stored as a XML document. In this paper, we will not describe the structure of the patient record. We are currently preparing a separate paper on this complete system.

Examples of the role hierarchy policies specified in this system - A Receptionist Rose can create a patient's name in a blank medical record. A Nurse Nancy can read a patient's general information and has Receptionist privileges. A Doctor Alice can read entire medical record. Doctor also has Nurse and Receptionist privileges. A Doctor in charge Bob can not only read but also write to record and informed consent sections. However, he can only write once into the informed consent section. He also has Doctor's privileges. This scenario is achieved by introducing new role-templates and then creating the required role hierarchies using these templates as described above in Section 4.

We have specified both static and dynamic separation of duty policies. In the static case, for instance in the role hierarchy example above, the duties of a nurse, doctor and doctor in charge are statically separated by creating role-template elements for them in the provider-role-map and then referring to them (or directly referring to the default role-template elements from the role-map) in the role-list for the provider – in this case all the hospitals. In the case of dynamic separation of duty, we have policies such as doctors Doug and Lee can write a report after a medical procedure or certify it but not both. That is, one doctor should not be able to both write a report and certify it. Depending on the action taken by a doctor, access is controlled in a dynamic fashion.

In terms of the Chinese-wall policy, the PRS has patients grouped into hospital name and department categories as shown below.

Set A: Departments (Cancer research, Cardiology, Neurology)

Set B: Hospitals (ABC Hospital, PQR Hospital, XYZ Hospital)

An example in our system occurs when a hospital implements Chinese-wall policy on trainee doctors when they access the patient records. Trainee doctors may do research by accessing patients records from only one of the departments from Set A and from only one of the hospitals in Set B. Once a trainee doctor (Tom) accesses a document from a category say Cardiology from Set A, he is not allowed to access any other category from that set. Similarly if Tom accesses a record from ABC Hospital, he is not allowed to access patient records from any other hospital from Set B. Note that here we have patient records from multiple hospitals.

Our system has different examples of the delegation policy. A simple one occurs when a doctor Alice delegates part of her privileges to her personal assistant Patricia. Note that this is a case of partial permanent delegation. This is achieved by adding a delegated-role to the appropriate provider's role list. An example of a joint action policy in our system is as follows: A discharge medical report can only be certified by two doctors jointly, for instance, the duty doctor in the hospital and the patient's surgeon. Once again this is achieved using the constructs specified in Section 5 involving role-templates and role elements and a Boolean flag.

6. CONCLUDING REMARKS

In this paper, we have considered the authorisation model in .NET MyServices and discussed the limitations of the authorisation service in terms of the policies that can be supported. Then we proposed new extensions to the XML data structures' schemas that could help to support the specification of a range of commonly used access policies in commercial systems. In particular, we have provided new extensions for supporting conditional authorisation, role hierarchy, dynamic separation of duty, Chinese-wall policy, joint action based policies and limited number of accesses to objects. We have developed the modified access evaluation algorithms that take the proposed extensions into account. We have then discussed the application of the modified authorisation service to an electronic Patient Record System. We are also currently investigating the extensions proposed in this paper to extend the XACL [8] and creating a distributed authorisation service for XML Web Services.

7. ACKNOWLEDGEMENTS

Vijay Varadharajan would like to thank Australian Research Council and Microsoft Pty. Australia for their support.

REFERENCES

[1] Microsoft .NET MyServices Specification 2001, Microsoft Press

[2] T. Bray et al. 'Extensible Markup Language (XML) 1.0' (Second Edition), World Wide Web Consortium (W3C), http://www.w3.org/TR/REC-xml (October, 2000)

[3] The .NET Security Service, Chapter 7, pp. 141-157, .NET MyServices Specification, MS Press, 2001.

[4] S. Indrakanti: 'Authorisation Service in .NET MyServices', Technical Report, Dept. of Computing, Division of ICS, Macquarie University, Dec.2002.

[5] R. Sandhu et al, 'Role based access control models', IEEE Computer, 1996, pp. 38-47

[6] R. Simon and M. Zurko: 'Separation of duty in role-based environments.' Proceedings of the 10th computer security foundations workshop, Rockport, MA, USA, 1997, IEEE CS Press, pp. 183-194.

[7] V. Varadharajan and P. Allen: 'Joint Action based Authorisation Schemes', ACM 30, *Oper. Syst. Rev.,* 1996, 3, pp. 32-45

[8] M. Kudo and S. Hada: 'XML Document Security based on Provisional Authorisation', Proceedings of the ACM Conference on Computer and Communications Security (CCS), Nov 2000, Greece.

[9] M. Hitchens and V. Varadharajan: 'Design and specification of role based access control policies'. IEE Proceedings - Software, Vol.147, No.4, August 2000, pp.117-129

[10] Microsoft .NET Passport Web Site, http://www.passport.net