

15

This role list includes one refined “scope” element and four “role” elements. The first “role” element means: fred running calendar@contoso.com can access all information, through all methods. Recall that role template “rt0” allows access to all information through all methods. The second “role” element means: fred running calendar@fabrikam.com only has read-only access to all information. The third “role” element means: barry running any application has read-only access to only public information, but only those pieces of public information that are categorized as golf related. This last access stipulation represents a refinement that was added to this role list to allow for more fine-grained control over access privileges. It is accomplished by reference to the local refined scope listed at the beginning of the role list. The last “role” element means: any and all users running the app@cpandl.com application have no access to any information held within this service.

The method 400 then includes an act of receiving a request from the requesting entity to perform at least one of the command methods, the request identifying the requesting entity (act 404). In one embodiment, the request identifies, at least in encrypted form, the user identifier, the application identifier, the platform identifier, and the credential type identifier.

The method 400 then includes an act of identifying a role definition corresponding to the requesting entity (act 405). First, the appropriate role list is identified by identifying the owner and type (e.g., content, role list, system) of the target data structure. Then, the user identifier, application identifier, platform identifier, and credential type identifier received in the request is matched against those similar fields in the “subject” element of the role definition. This may be accomplished via a database lookup.

The matching process may be as follows. In a first matching operation, the user identifier in the message is matched against “userId” attributes in the “subject” element of the role definitions to find a first set of matching role definitions. Once this first set of matching role definitions is identified, a second match operation is performed. In this second matching operation, the credential type identifier associated with the request is matched with the “credType” attribute in the “subject” element of the first set of role definitions to narrow to a second set of matching role definitions. If there are no role definitions in the second set of role definitions, then all role definitions from the first set having a “subject” elements containing the “credType” attribute are discarded keeping only those “subject” elements that do not contain a “credType” attribute to form the second set of role definitions.

Then, a third matching operation is performed in which the combined platform identifier and application identifier of the request are matched against the “appAndPlatformId” attribute of the second set of role definitions. This generates a third set of role definitions. If there are no role definitions in the third set of role definitions, then all role definitions from the second set having a “subject” elements containing the “appAndPlatformId” attribute are discarded keeping only those “subject” elements that do not contain an “appAndPlatformId” attribute to form the third set of role definitions. If a matching role element is not found, the request is failed with an authorization fault. Also, if the matching role definition contains an “expiresAt” element that indicates that the role definition has expired, then an error message is also returned.

Note that the role list structure allows for different role definitions even for the same user and the same application

16

should the user authenticate using different authentication methods and thus create different credential type identifiers. Thus, a user authenticating using a more secure authentication mechanism may be granted more extensive access than the same user using a less secure authentication mechanism.

The role list lookup may be farther optimized through the use of licenses. When an application sends a message containing an identity license, the authorization station 130 finds a role template and a refined, local scope corresponding to the request as described above. The authorization station 130 then places this in the request as an “authorized role”. Once the request has been fulfilled, the authorization station 130 sends a response back which includes an <authorizedRole/> element. When the application sends a subsequent request back to the same service, the request includes both the identity license and the authorized role license. During authorization, the authorization station 130 notices the authorized role license, and determines that it is valid and that it was properly issued to the identity sending the message. The authorization station 130 then uses the information contained within the authorized role element (i.e., the appropriate role template with the refined, local scope), instead of once again accessing the role list database. Thus, a database lookup process is avoided for subsequent accesses to the same service.

The method 400 includes an act of determining access permissions for the requesting entity with respect to the command method using the role definition corresponding to the requesting entity (act 406). In order to accomplish this, the role templates in the service’s role map are extracted and only the highest priority role template is kept. The specified command method is compared with the applicable role template to determine if the method is allowed. If it is not allowed, then an error message is returned. If the method is allowed, then the scope corresponding to that method (as referred to in the role template) is combined with any refined scope referenced by the role definition found within the role list. This information is then passed to the target service along with an authorization to proceed.

The present invention has the advantage of performing authorization in a standardized manner regardless of the target service that is desired. The service is only factored in when selecting an appropriate role map.

In addition, this is accomplished while providing a standardized set of templates that may be used for coarse grained control over access. Thus, applications that are not able to add further refined scopes to the role list may at least have some level of access control over the service’s data structures. Furthermore, those applications that can define more refined scopes may have those more refined scopes included in the role list documents to allow for more user-specific and refined control over access permissions. Accordingly, the present invention provides for a high level of control over access permissions in a manner that is relatively independent of the underlying service being targeted.

As a final advantage, note that scopes define views on a document. Thus, unlike conventional access control lists, the present invention facilitates access granularities below the document level. In other words, portions of documents may be viewed or operated upon, while other portions remain secure.

Having now described the principles of the present invention in detail, it is noted that the precise hardware configuration that implements the above-described features is not important to the present invention. For example, it is not