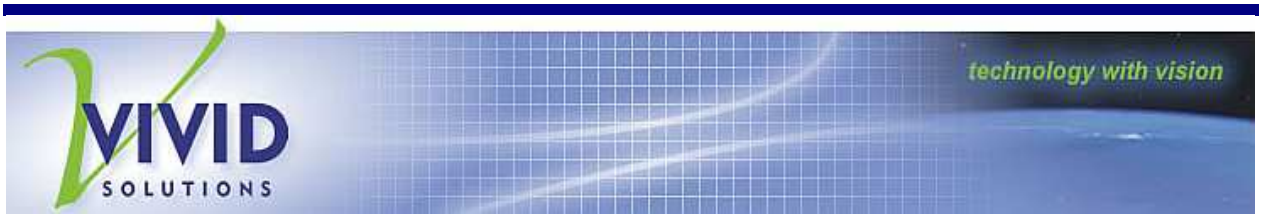

IMAP SECURITY

Technical Requirements

Client:	MSRM IMG
Date:	September 30, 2004
Revision:	5

Vivid Solutions Inc.
Suite #1A, 2328 Government St.
Victoria, BC V8T 5G5
Phone: (250) 385-6040
Fax: (250) 385-6046
Website: www.vividsolutions.com

Document Change Control

REVISION NUMBER	DATE OF ISSUE	AUTHOR(S)	DESCRIPTION
1		Jason Ross	Original draft
2	July 15, 2004	Jason Ross	Revised document after July 14th meeting at MSRM
3	August 15, 2004	Jason Ross	Updated documentation to reflect changed to Extension API
4	August 31, 2004	Jason Ross	Added configuration information for LRDW Security Extension
5	September 30, 2004	Jason Ross	Added documentation for the xacml.properties file configuration.

Table of Contents

1. XACML AND THE POLICY DATABASE STRUCTURE

As the set of policy tables were loosely based upon the XACML model, it should be relatively simple to use XACML to communicate with the database to perform policy requests for users to access resources as assigned in the database.

The database consists of the following tables: Policies, Subjects, Resources, Actions, and Obligations.

In order to create a WebADE extension that mines these tables for information, using XACML, the following Custom XACML components will be needed.

1.1 REQUEST PROCESSOR

This is the main component, which will be the exposed extension API. This piece will initialize the PDP, format requests from calls to the extension and pass them to the PDP, and parse results from the PDP, returning an authorization result and passing this back to the caller, or passing along errors, if they occur.

1.2 DATABASE POLICY FINDER MODULE

This is the component that will communicate with the database, translating the table records into policy classes to be used for the policy queries.

1.3 WEBADE ATTRIBUTE FINDER MODULE

Needed to lookup user attributes from WebADE. If the user only passes in a user id, this could be one way to get the user's groups, or other information needed to map them to entries in the Subject table. It may just be simpler for this extension to just make a lookup to the WebADE directly, but its something to think about, for future XACML code, if not this project.

1.4 LDAP ATTRIBUTE FINDER MODULE

Needed to lookup user and group attributes from LDAP. This will be used to look up the user's GUID for use in policy queries.

2. SAMPLE QUERIES

2.1 BASIC QUERY

Here is what a basic query looks like for the extension:

canViewLayer(layer, httpServletRequest)

layer: the layer object to validate resources for that action.
httpServletRequest: the HTTP Servlet Request object containing the user's identification as request header attributes.

This method will return a QueryResult object. It can also throw an exception if an error occurred while evaluating the policy.

2.1.1 QUERYRESULT OBJECT

The QueryResult object has an "isAuthorized()" method that returns a Boolean value indicating true if the user is permitted to perform the given action on the resource, or false otherwise.

If an authorization request is refused, the QueryResult object has an "getUnauthorizedResourceNames()" method that returns a list of strings describing the set of resources from the layer query that the user was refused access to. These strings are in the Resource URI format described in Section 4.2

A user will be authorized access to a layer if, for each resource defined for that layer (database tables or image layer), the user matches a policy defined to grant access to that layer. This policy match can be either based on the user's own unique GUID matching an entry in the policy table, or the user being a member of a group that is granted access in the policy table. A user can match on group if the group's unique GUID is allowed access to a resource, and the user is a member of that group or any group that is a member of that group (traversing the tree of sub-groups that are members of the target group down any level of depth).

2.1.2 LAYER OBJECT

The layer object will contain all information about the layer, as defined by the map service. This consists of a layer-type, and either a layer name or a set of database tables and views that are used by that layer.

If a layer is defined as a database layer, it will include a set of table and/or view names (including containing database and schema names) that will be used in the XACML query to determine if the user has access to that layer resource. If the user is refused access to any table or view required by the layer, access will be denied to the layer for that action.

If a layer does not use database tables (Example: picture layer), the name of the layer, as defined in the map-service, will be used to perform a XACML query.

3. LIMITATIONS

The XACML API can only be used to create policies and query these policies to grant or deny access to resources given a user, a resource, and an action. This API cannot be used to perform queries such as "Given this resource name and action to be performed on it, give me the list of users and groups that can perform this action". Essentially, you cannot use the API to mine the policy store to determine what policies exist within. You can only use it to verify a user's desired action on a resource.

4. IMF POLICY DEFINITIONS

There are some challenges unique to IMF when identifying policies that will control user access to layers within an IMF application. These challenges result from the way layers are named and identified within an IMF application.

4.1 MAP SERVICES

Each IMF application defines the layers that are available for users of that application via one or more map services. Map services are configured using ArcInfo standard XML documents (called AXL files), and are used to set the spatial configuration attributes for a layer as needed to retrieve the layer data from the spatial database.

4.1.1 MAP SERVICE LAYERS

The name of a layer is set in the AXL file, and only needs to be unique within this file. The result is many applications can contain a layer with the same name. For layers that use database tables and views, this is not an issue, as layer policy queries use these table names and views, not the layer name. But for all other layers, this poses a problem identifying the correct layer for policy queries.

Because of this, in order for policies to be defined to restrict users' access to these types of layers, the following data is needed to uniquely identify the layer:

- Application name
- Map service name
- Layer name

4.1.1.1 APPLICATION NAME

The application name will be a name or acronym that uniquely identifies the application.

4.1.1.2 MAP SERVICE (OPTIONAL)

For applications that have multiple themes (use separate map services, and thus separate AXL files), each map service should be assigned a unique name, to allow for layers in separate map services that have the same name to be uniquely identified.

As many applications will only use one map service, this is an optional attribute, defaulting to the value of "default" if it is not provided in a policy request.

4.1.1.3 LAYER NAME

This name must match the layer's "name" attribute, as defined in the map service's AXL file.

4.1.2 PUBLIC RESOURCES

There exist the possibility for public resources, accessible to all users. This is handled in the policy database by a subject with a GUID value of "Public". And resources with a policy linking it to the "Public" subject will always return true for policy requests.

4.1.3 DIRECTORY-WIDE RESOURCES

It is also possible to assign a resource to all users of a directory, such as IDIR and BCeID. . This is handled in the policy database by a subject with a GUID value of "IDIR" or "BCeID". Resources with a policy linking it to one of the directory subjects will always return true for policy requests for users who are members of that directory.

4.2 LAYER RESOURCE URI VALUES

Each table and view referenced by map service layers and layers of a map service that are not database-related will have an entry in the LRDW RESOURCE table. This table has the following columns:

COLUMN NAME	COLUMN DESCRIPTION
RESOURCE_ID	A unique numeric id value identifying the resource.
RESOURCE_TYPE	A name describing the resource type
RESOURCE_SUBTYPE	A name describing the resource sub-type
RESOURCE_VALUE	A URI identifying the resource
DESCRIPTION	A description of the particular resource.

When defining a table or view as an LRDW resource, these columns will be set as follows:

COLUMN NAME	DEFINITION
RESOURCE_ID	Generated by a sequence in the database
RESOURCE_TYPE	Set to map_service_resource (All lowercase).
RESOURCE_SUBTYPE	Set to table. (All lowercase)
RESOURCE_VALUE	Set to the name of the table or view (including database and schema names, in all uppercase, separated by colons). **Example:** "MYDATABASE:MYSCHEMA:TARGET_TABLE"
DESCRIPTION	Set by the policy administrator.

When defining a layer as an LRDW resource, these columns will be set as follows:

COLUMN NAME	DEFINITION
RESOURCE_ID	Generated by a sequence in the database
RESOURCE_TYPE	Set to map_service_resource. (All lowercase)
RESOURCE_SUBTYPE	Set to layer. (All lowercase)
RESOURCE_VALUE	Set to a URI with the following pattern (All uppercase): *APPLICATION_NAME*:*THEME*:*LAYER* where *APPLICATION_NAME* is the application acronym, *THEME* is the theme or AXL file name, and *LAYER* is the name of the layer, as defined in the AXL file.

DESCRIPTION	Set by the policy administrator.

4.3 OTHER NOTES

4.3.1 ACTIONS

Currently, the only action to be defined for IMF policy queries is "view". If a user has access to the layer within an application, they can perform any action using that layer that the application provides (view, export, etc.). In the future there may be more actions defined that allow an application to prevent some users from performing certain actions on a layer, while allowing them access to other actions on the same layer.

5. USING THE WEBADE LRDW SECURITY EXTENSION

5.1 ADDING THE EXTENSION TO A WEBADE APPLICATION

To make the LRDW Security Extension available to a WebADE application, the following steps must be followed:

1) Add the "lrdw-security.jar" to the web application's WEB-INF/lib directory.
2) Add the extension to the list of extensions in the WEB-INF/classes/WebADE.xml file.
3) Place a properly configured xacml.properties file at WEB-INF/classes/ xacml.properties
4) Create a WebADE Role and Proxy Control entries in the WebADE database for use by the extension.
5) Retrieve the extension instance from the "Application" singleton.

5.1.1 WEBADE.XML SETTINGS

All that is required to add the LRDW Security Extension to the WebADE.xml file is to add the following line:

<extension class="ca.bc.gov.msrm.webade.security.lrdw.XACMLExtension" />

A properly configured WebADE.xml file should look like the following:

```
<?xml version="1.0" encoding="ISO-8859-1"?>

<application>
  <security class="ca.bc.gov.mof.webade.security.oc4jjaas.OC4JSecurityInfo" />
  <extension class="ca.bc.gov.msrm.webade.security.lrdw.XACMLExtension" />
</application>
```

5.1.2 "XACML.PROPERTIES" FILE SETTINGS

This file should contain the connection settings for the active directories for IDIR and BCEID. To properly configure this file, you will need a valid username and password for both IDIR and BCEID. The "ldap.username.1" and "ldap.username.2" values should be set to *username*@idir and *username*@bceid. The "ldap.password.1" and "ldap.password.2" values should be set to the login passwords or the IDIR and BCEID users, respectively. The remainder of the properties in this file should be set as listed below.

```
ldap.directory.server.name.1=IDIR
ldap.provider.url.1=ldap://idir.bcgov/
ldap.search.base.1=ou=BCGOV,dc=IDIR,dc=BCGOV
ldap.username.1=user@idir.bcgov
ldap.password.1=password

ldap.directory.server.name.2=BCEID
ldap.provider.url.2=ldap://lentil.bceid/
```

```
ldap.search.base.2=OU=BCEID,DC=bceid
ldap.username.2=user@bceid.
ldap.password.2=password
```

5.1.2.1 USER AND GROUP CACHING (OPTIONAL)

In addition to the mandatory XACML properties, there are two optional properties, controlling the number of hours user and group data is cached in memory, to reduce the amount of active directory queries. By default users data is cached for 1 hour, and group data is cached for 6 hours. These values can be changed by setting the "ldap.user.hours.cache.time.1" and "ldap.user.hours.cache.time.2" settings for user data cache time (in hours), and "ldap.group.hours.cache.time.1" and "ldap.group.hours.cache.time.2" properties for group data caching (in hours). An example of a xacml.properties file with these optional parameters is below:

```
ldap.directory.server.name.1=IDIR
ldap.provider.url.1=ldap://idir.bcgov/
ldap.search.base.1=ou=BCGOV,dc=IDIR,dc=BCGOV
ldap.username.1=user@idir.bcgov
ldap.password.1=password
ldap.user.hours.cache.time.1=1
ldap.group.hours.cache.time.1=6

ldap.directory.server.name.2=BCEID
ldap.provider.url.2=ldap://lentil.bceid/
ldap.search.base.2=OU=BCEID,DC=bceid
ldap.username.2=user@bceid.
ldap.password.2=password
ldap.user.hours.cache.time.2=1
ldap.group.hours.cache.time.2=6
```

5.1.3 WEBADE DATABASE ENTRIES

The LRDW Extension depends on entries in the Role and Proxy_Control tables with the following values:

Role Table

COLUMN NAME	COLUMN VALUE
APPLICATION_ACRONYM	*Target application acronym*
ROLE_NAME	IMAP
ROLE_DEFINITION	LRDW Extension Role

Proxy_Control Table

COLUMN NAME	COLUMN VALUE
ROLE_NAME	IMAP
APPLICATION_ACRONYM	*Target application acronym*
DB_USERID	*Database username used to connect to LRDW database*
DB_PASSWORD	*Database password used to connect to LRDW database*
CONNECTION_INFO	*JDBC database URL of LRDW database*

DB_DRIVER	ORACLE

5.1.4 RETRIEVING THE EXTENSION INSTANCE

To retrieve the extension, simply use code similar to the following:

```
IMapApplication imapApp = (IMapApplication)
                    WebADEActionServlet.getApplication(session.getServletContext());
webadeExt = (XACMLExtension)imapApp.getWebADEExtension(XACMLExtension.class);
```