OpenEyes - Access Control

Editors: G W Aylward
Version: 0.95:
Date issued: 4 October 2010

Target Audience

General Interest	✔
Heathcare managers	✔
Ophthalmologists	✔
Developers	✔

Amendment Record

Issue	Description	Authors	Date
0.9	Draft	G W Aylward	14 April 2010
0.95	Draft	G W Aylward	4 October 2010

Table of Contents

Introduction

The healthcare environment is complex, with a large range of professional staff, and a requirement for graded and hierarchical access to patient data. Security and confidentiality issues are paramount, and have an increasingly high profile. For this reason, a flexible but powerful authentication scheme is required to control access. For this reason, OpenEyes uses Role Based Access Control (RBAC) to handle permissions. With this approach, permission to view or modify patient records is based on a role, rather than on an individual. Users are not assigned permissions directly, but only acquire them through their role (or roles). This means that management of individual user rights becomes a matter of assigning appropriate roles to the user, which simplifies common operations, such as adding a user, or changing a user's department. RBAC was developed in the 1990s,[1] and was made the subject of an ANSI standard, last updated in 2004.[2]

RBAC concepts

The RBAC reference model has four components, Core RBAC, Hierarchical RBAC, Static Separation of Duty Relations, and Dynamic Separation of Duty Relations. For most anticipated purposes in OpenEyes, core and hierarchical RBAC will be sufficient (Figure 1). However, if required in the future, the additional elements can be built onto the foundation of core and hierarchical RBAC.

Core RBAC

Core RBAC consists of five basic data elements called users, roles, objects, operations, and permissions. These are defined as follows using examples from the healthcare environment;

Element	Description	Example
User	A user of the system	Joe Bloggs
Role	A job function	Clerk, Doctor, Nurse, Secretary
Object	An element of the EPR which can be viewed or modified	A diagnosis, a prescription, a booking for a surgical procedure
Operation	An action on an object	Adding, editing, deleting, viewing
Permission	A combination of an operation and an object	Adding a diagnosis, listing a patient for theatre

In addition core RBAC includes the concept of a session, in which the a subset of roles that are assigned to a user can be activated. This roughly corresponds to a user session as implemented when accessing a web server.

Hierarchal RBAC

Hierarchal RBAC adds role hierarchies to the core elements. Hierarchies are a natural means of structuring roles to reflect an organization’s lines of authority and responsibility.

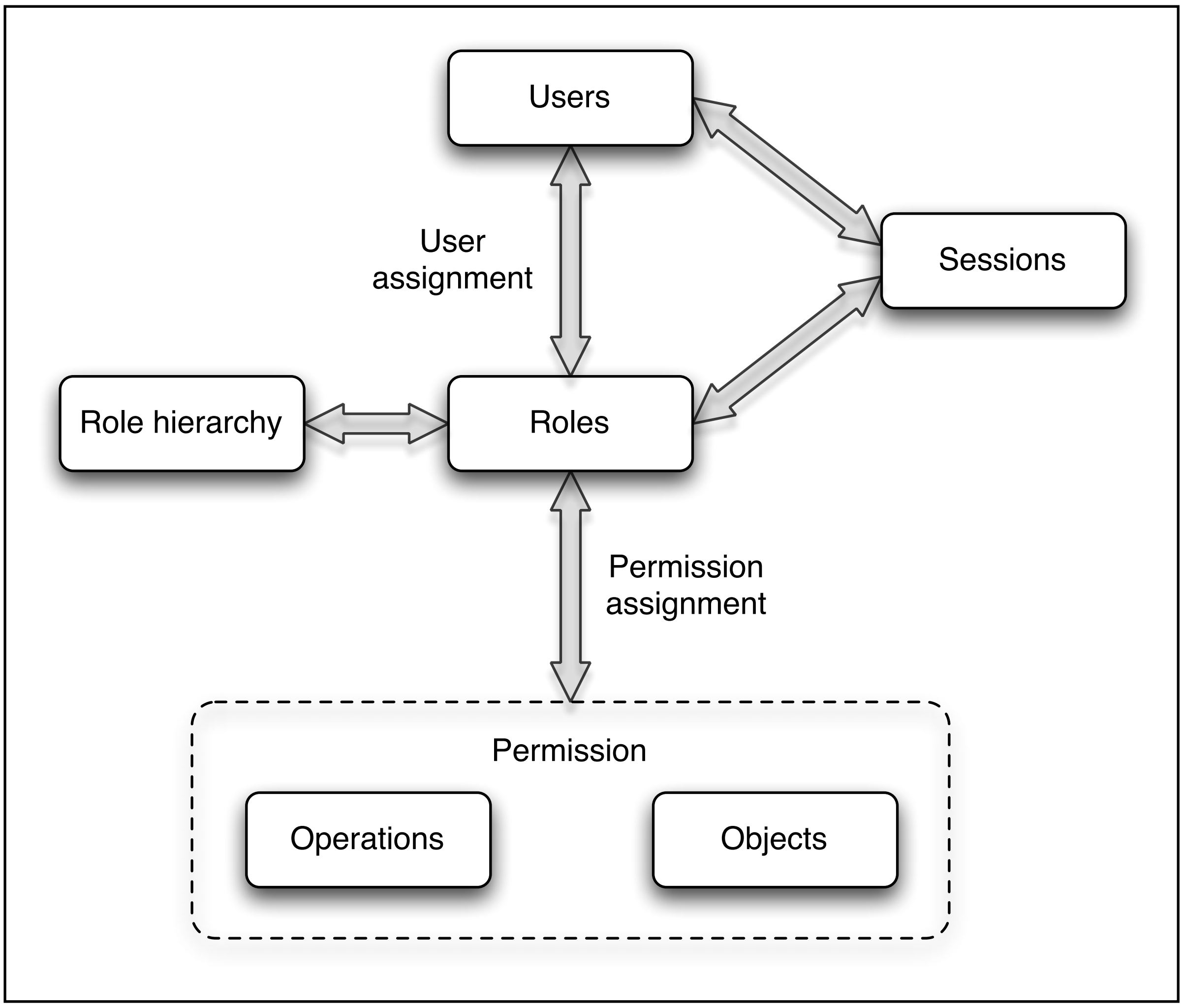

Figure 1. Outline of core and Hierarchal RBAC

Open Eyes RBAC Implementation

OpenEyes offers two implementations of RBAC, either of which can be used. They are described in the following table;

	LDAP	SQL
Requirements	A server running version 3 of the Lightweight Directory Access Protocol (LDAP).[3]	An additional database running on a SQL server
Advantages	Compatibility with existing authentication systems. Distributed authentication	Ease of set up and administration

Sessional data is stored using PHP Session variables. This means that accessing RBAC need only occur once for each session, at the time of logon (Figure 2). This reduces the overhead required when checking permissions for tasks performed during a user session.

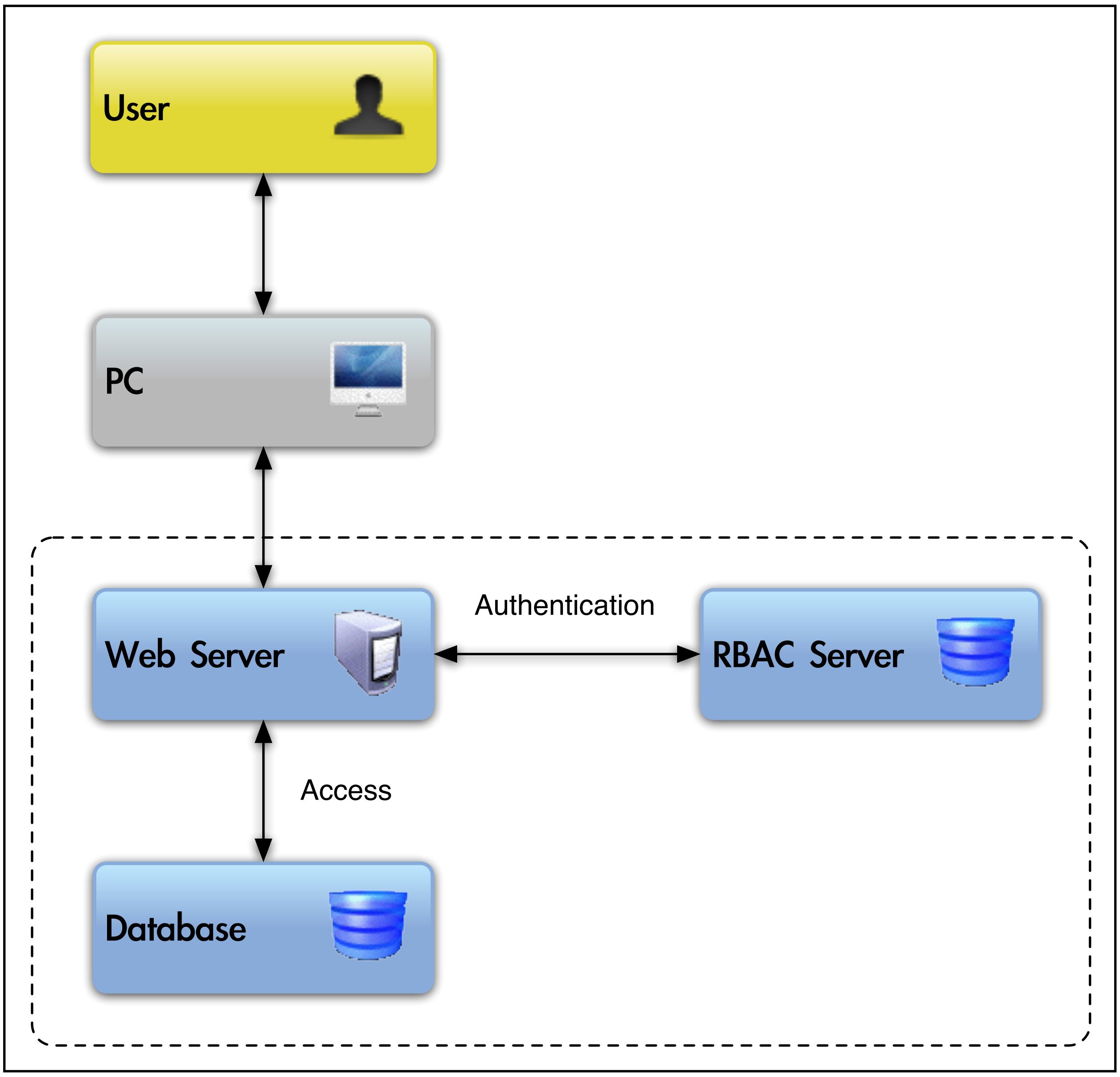

Figure 2. Authentication is achieved at logon by means of a single interrogation of an LDAP server, giving permissions which are valid for the duration of the session

Currently, all the roles that are stored on the RBAC server are activated at the start of a session, meaning that there is currently no difference between static and active roles. Logon creates session variables that can be interrogated by web server scripts for the purposes of determining permissions.

LDAP implementation

Hierarchical RBAC is implemented using a tree structure on the RBAC server. For the LDAP implementation, Users, roles, objects, operations, and permissions are each stored in their own subtree. The structure can be loaded into an LDAP server using the LDIF files described in Appendix 1. Use is made of standard LDAP schemas, but there are number of requirements which require a custom schema. See Appendix 2 for full details. Entries in each subtree have a number of attributes which are described as follows;

Users

Users are stored using the standard LDAP object class "inetOrgPerson". The attributes that are utilised are shown in the following table;

Attribute	Comment	Example
uid	Unique user identification	Joe Bloggs
cn	Common name	Joe Bloggs
sn	Surname	Bloggs
givenName	Forename	Joe
userPassword	Password	*********

Roles

The hierarchy of roles is represented by a tree structure within the roles subtree. This can be represented in text form very simply as in the following example;

```
Clinical Director
    Consultant
        Doctor

Head Nurse
    Nursing Sister
        Staff Nurse
            Nurse


Head Secretary
    Secretary
```

Assignment of a role to a user is accomplished by adding the user identifier (UID) to a role subtree entry. Permissions inherit from right to left in the above diagram, so that adding 'Jo Bloggs' to the Consultant subtree would result in Joe Bloggs inheriting all the existing permissions of a doctor as well as those of a consultant, but not any additional permissions assigned to the role of Clinical Director.

It is important to distinguish between roles (as defined in RBAC) from jobs, even though the two are superficially similar, particularly in the above example. Since permissions are assigned to roles rather than users, permission is granted to a user by assigning that user to a role. For example, consider a permission to write a letter on a patient. This permission could be assigned to the role secretary, to which a user who had the job of secretary would be assigned. However, if a user assigned to the role of doctor was also permitted to write letters, then this can be achieved by assigning the role of secretary to that user. This will work as long as there are no permissions in the role of secretary which should not be assigned to that user. An alternative solution is to assign the permission to write a letter to the role of doctor.

Objects

Objects are stored using a custom LDAP object class “oeRbacObject” with the single property oeRbacNameAtribute.

Operations

Operations are stored using a custom LDAP object class “oeRbacOperation” with the single property oeRbacNameAtribute.

Permissions

Permissions are stored using a custom LDAP object class “oeRbacPermission” with the following properties;

Attribute	Comment	Example
oeRbacNameAttribute	Combination of operation and object	Add Diagnosis
oeRbacOperationAttribute	An operation on data	Add
oeRbacObjectAttribute	An object in the database	Diagnosis

SQL implementation

The SQL implementation of RBAC uses a set of tables stored in a separate database (OpenEyesRBAC). These are described as follows.

Users

Basic users information, including hashed passwords, are stored in a users table, the structure of which is given in Appendix 2.

Roles

The hierarchy of roles is represented by a tree structure within an SQL table. This uses a left and right node methodology to implement a tree in a table. NB if a role is added to the table, then the tree structure needs to be regenerated.

Objects, Operations and Permissions

These are stored in separate tables. Entries in the permissions table are simple a list of all possible combinations of the operations and objects.

RBAC functions

OpenEyes RBAC (OERBAC) currently provides a subset of functions for core and hierarchical RBAC as defined in the full reference model.[2] The following table gives a full list of the available functions.

Component	Function	Notes	OERBAC
Core RBAC	AddUser	Creates a new user	✔
	DeleteUser	Deletes a user	✔
	AddRole	Creates a new role	✔
	DeleteRole	Deletes a role	✔
	AssignUser	Assigns a user to a role	✔
	DeassignUser	Deletes a role assignment	✔
	GrantPermission	Grants permission to a role	✔
	RevokePermission	Removes a permission from a role	✔
	CreateSession	Creates a session	✔
	DeleteSession	Deletes a sessions	✔
	AddActiveRole	Add an active role	✖
	DropActiveRole	Drops an active role	✖
	CheckAccess	Checks whether user has permission	✔
	AssignedUsers	Users assigned to a given role	✖
	AssignedRoles	Roles assigned to a given user	✖
	RolePermissions	Permissions granted to a given role	✔
	UserPermissions	Permissions granted to a given user via roles	✔
	SessionRoles	Roles granted to a user in a given session	✔
	SessionPermissions	Permissions granted to a user in a session	✔
	RoleOperationsOnObject	Operations a role can perform on an object	✖
	UserOperationsOnObject	Operations a user can perform on a given object	✖

Component	Function	Notes	OERBAC
Hierarchical RBAC	AddInheritance	Adds inheritance relationship	✔
	DeleteInheritance	Removes inheritance relationship	✔
	AddAscendant	Adds new role in hierarchy	✔
	AddDescendant	Adds new role in hierarchy	✔
	CreateSession	Creates a user session	✔
	AddActiveRole	Adds new role to existing session	✖
	AuthorizedUsers	List of users with a given role	✖
	AuthorizedRoles	List of roles for a given user	✖
	RolePermissions	Permissions granted to a given role	✔
	UserPermissions	Permissions granted to a given user via roles	✔
Static Separation of Duty Relations	Various		✖
Dynamic Separation of Duty Relations	Various		✖

References

1. Ferraiolo, D.F. and Kuhn, D.R. (October 1992). Role-Based Access Control. 15th National Computer Security Conference. pp. 554–563.
2. Role Based Access Control. (ANSI ® INCITS 359-2004) American National Standards Institute, Inc. 2004.
3. RFC 4510 - Lightweight Directory Access Protocol (LDAP) Technical Specification Roadmap

Appendix 1 - Setting up LDAP

The Schema file

The attributes used are described in a schema file called ‘oerbac.schema’. This should be copied into the appropriate place for OpenLDAP to read;

```
‣ sudo cp /Users/bill/Databases/OpenEyes/RBAC/oerbac.schema /opt/
  local/etc/openldap/schema/
```

NB. You will have to restart OpenLdap to make it read the new schema file

Loading data into the LDAP database

LDIF files are a convenient way of loading data into an LDAP server. There are several LDIF files provided with OpenEyes as described in the following table.

File	Description
structure.ldif	Lays out the overall database structure with subtrees for users, roles, operations, objects, and permissions
roles.ldif	Roles arranged in a hierarchy
operations_and_objects.ldif	Operations and objects
users.ldif	A sample range of users
permissions.ldif	Example assignment of permissions to roles

Importing LDIF files

The following instructions describe how to import an LDIF files into an LDAP database. Make a directory for LDIF files (for example in /opt/local/var).

```
‣ cd /opt/local/var
‣ sudo mkdir data_files
‣ cd data_files
```

Put the LDIF files in this directory (eg by using the following command)

```
‣ sudo cp /Users/bill/Databases/OpenEyes/RBAC/LDIF\ files/*.ldif /
  opt/local/var/data_files
```

Then load them one by one (in the order given in the table) into the LDAP server using the terminal with the following command, where ‘secret’ is the password for the directory manager.

```
‣ ldapadd -f structure.ldif -x -D "cn=Manager,dc=openeyes,dc=com"
  -w secret
‣ ldapadd -f roles.ldif -x -D "cn=Manager,dc=openeyes,dc=com" -w
  secret
‣ ldapadd -f operations_and_objects.ldif -x -D
  "cn=Manager,dc=openeyes,dc=com" -w secret
‣ ldapadd -f users.ldif -x -D "cn=Manager,dc=openeyes,dc=com" -w
  secret
‣ ldapadd -f permissions.ldif -x -D
  "cn=Manager,dc=openeyes,dc=com" -w secret
```

Generating Permissions

Permissions are simply a pairing of object and operation, so rather than import them from an LDIF file, they are generated using the php script “generateLDAPPermissions.php”.

Appendix 2 - table structures for SQL RBAC

Users

Field	Type	Comments
user_id	SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
uid	VARCHAR(40)	The user id which is used to logon to the system. In most scenarios, this will be the same as the username used to logon to the operating system
is_active	BOOL	A boolean value indicating whether the user is active or not (Inactive users must have a valid user_id in order to link previous records stamped for that user
forename	VARCHAR(20)	First name of the user, for display along with surname
surname	VARCHAR(40)	Last name of the user, for display along with forename
pass	VARCHAR(40)	Used to store the hashed version of the user's password

Roles

Field	Type	Comments
role_id	SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
parent_id	SMALLINT UNSIGNED	Pointer to this role's parent
left_node	SMALLINT UNSIGNED	Pointer to the role immediately to the left
right_node	SMALLINT UNSIGNED	Pointer to the role immediately to the right
name	VARCHAR(40)	Name of the role
service_code	CHAR(2)	Optional code for service
firm_code	CHAR(2)	Optional code for firm

Objects

Field	Type	Comments
object_id	SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
name	VARCHAR(80)	Name of the object
description	VARCHAR(500)	Description of the object
right_node	SMALLINT UNSIGNED	Pointer to the role immediately to the right
name	VARCHAR(40)	Name of the role
service_code	CHAR(2)	Optional code for service
firm_code	CHAR(2)	Optional code for firm

Operations

Field	Type	Comments
operation_id	SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
name	VARCHAR(40)	Name of the operation

Permissions

Field	Type	Comments
permission_id	SMALLINT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
operation_id	SMALLINT UNSIGNED	Foreign key pointing to operation
object_id	SMALLINT UNSIGNED	Foreign key pointing to object
name	VARCHAR(120)	Name of the operation

Roles_permissions

Field	Type	Comments
role_permission_id	INT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
role_id	SMALLINT UNSIGNED	Foreign key pointing to role
permission_id	SMALLINT UNSIGNED	Foreign key pointing to permission

Users_roles

Field	Type	Comments
user_role_id	INT UNSIGNED NOT NULL AUTO_INCREMENT	Primary key
user_id	SMALLINT UNSIGNED	Foreign key pointing to user
role_id	SMALLINT UNSIGNED	Foreign key pointing to role