An Enterprise JavaBeans application constitutes the middle tier in the J2EE application development platform. Enterprise JavaBeans provide an object-oriented view of the database schema and implement the business logic of the application. One of the problems to be explored in Enterprise JavaBeans application development is the implementation of fine-grained access control for an Enterprise JavaBeans application. This project attempts to present a solution to this problem of user access control and examines the security framework required to facilitate user authentication and authorization within a J2EE framework.

The implementation of J2EE security in an Enterprise JavaBeans application is one of the areas of interest to the Information Integration Initiative (III). This report describes the work I have done during this semester to explore the issues involved in enforcing user access control policies through security management in an Enterprise JavaBeans application. The baseline for my project was the EJB application developed by Andrew Sterling Hanenkamp for the Information Integration Initiative.

II. ENTERPRISE JAVABEANS

Enterprise JavaBeans technology is a standard component of the J2EE architecture proposed by Sun Microsystems for building business applications using Java programming language. Enterprise JavaBeans allow application developers to concentrate on business logic without having to focus on low-level transaction, resource and security management issues. One of the main resources I used for this project is the Enterprise JavaBeans Specification Version 2.0 documentation provided by Sun Microsystems. Security management in EJB architecture is specified in Chapter 21. This project uses bean-managed persistence and the lifecycle for Bean-Managed Persistence Entity Bean is illustrated in Chapter 12.

The EJB development for this project was carried out using JDeveloper – a tool developed by Oracle Corporation that provides an integrated development environment for developing, testing and deploying J2EE components. This tool was suggested to me by Andrew Sterling Hanenkamp and few of my peers. JDeveloper provides an embedded version of the Oracle9i Application Server for deploying J2EE components. The embedded containers are known by the name OC4J (Oracle Containers for Java). Oracle9i Application Server was the target operational environment for my project and most of the work I have done especially with reference to deployment descriptors is specific to Oracle9iAS.

III. J2EE SECURITY

Security management in any EJB application requires certain tasks to be performed by two ‘entities’ operating in separate domains of the computational environment. The first entity is the application developer or bean provider and the second entity is the application assembler or deployer. The former writes the bean business logic and the later manages the target operational environment (application server) where the beans will be deployed. The bean provider is required to define certain ‘logical security roles’ for the application. Logical security roles are application specific and are used to control client access to various bean methods within the EJB application. The deployer is required to define certain ‘user’ and ‘role’ information in the target operational environment. This information will be used to authenticate the clients before they attempt to access the EJB application that is deployed on the server. The logical security roles defined in the application domain are then mapped on to the users/roles in the operational domain to provide for an integrated security framework for the EJB application.

Diagram 1 below illustrates the above security mapping and Diagram 2 presents a graphical representation of Enterprise JavaBeans in a J2EE Application framework.
Diagram 1: Security mapping in an EJB application

Logical security role ‘STUDENT’  Logical role ‘STUDENT’ mapped to user ‘joe’ defined in the operational environment.

User ‘joe’ stored in application server

Diagram 2: Enterprise JavaBeans in a J2EE Application framework

A. OC4J Security

As a starting point for this project I examined the information returned by accessing the isCallerInRole () and getCallerPrincipal () methods of the javax.ejb.EntityContext interface that is passed by the container to an entity enterprise bean instance after the bean has been created. These methods provide security information about the enterprise beans caller. The getCallerPrincipal () method returns information pertaining to the Java Authentication and Authorization Service (JAAS). JAAS is a Java package that enables applications to authenticate and enforce access controls upon users. Oracle9iAS supports JAAS by implementing a JAAS provider called JAZN. JAZN uses
certain repositories for secure, centralized storage, retrieval and administration of provider data. These repositories are called provider types and JAZN supports two kinds of provider types – a LDAP-Based Provider Type (Oracle Internet Directory) and a XML-Based provider type. The later has been used in this project. Both the JAAS providers enable different functionalities of JAAS and can be managed with certain tools. These tools have varying functionalities for each of the provider types. Detailed information about OC4J security and JAAS support in Oracle9iAS is provided in the Oracle9iAS Containers for J2EE Services Guide and Oracle9i Application Server Security Guide available on the Oracle Technology Network.

I. LDAP-Based Provider Type

The user repository is an Oracle Internet Directory. The Oracle Internet Directory runs as an application on the Oracle9i database server and is available with the Oracle9iAS Infrastructure. One of the goals of the KEAS project is to consolidate identification / authentication information into a central directory service. There is a way in which OC4J security using LDAP-Based Provider Type (Oracle Internet Directory) could be integrated with the central directory service of the KEAS system. The Oracle Internet Directory includes a component called the Oracle Directory Integration Service. This feature enables users to synchronize Oracle Internet Directory with other repositories in their enterprise repositories which in this case would be the central directory service of KEAS. The data in the Oracle Internet Directory can be managed with a combination of three tools mentioned below.

- Oracle Enterprise Manager – This is a GUI tool that is used to perform two JAAS Provider tasks – manage JAAS policy and manage Java permissions. It manages this information in the Oracle Internet Directory. Oracle Enterprise Manager is a component of the Oracle9i Application Server.
- JAZN Admin tool – A command line interface tool that enables administrators to create and manage users, realms (user communities), roles and policies. It uses the JAAS Provider API packages.
- JAZNUserManager class and JAAS Provider API – JAZNUserManager class is used by OC4J to implement JAZN. The LDAP-Based Provider environment can be managed programmatically by using JAZNUserManager and the JAAS Provider API’s.

II. XML-Based Provider Type

An XML-based provider is typically a file called jazn-data.xml. All the ‘user’, ‘role’ and ‘permission’ information required by the application server to grant access to EJB clients is stored in this file. The information in the repository is used to verify the identity of a user attempting to access a J2EE application on the server.

A user ‘joe’ having password ‘welcome’ can be defined within the jazn.com realm (default realm used by Oracle9iAS) in jazn-data.xml in the following manner:

<jazn-data>
<jazn-realm>
    <realm>
        <name>jazn.com</name>
        <users>
            <user>
                <name>joe</name>
                <description>student</description>
                <credentials>!welcome</credentials>
            </user>
        </users>
    </realm>
</jazn-realm>
</jazn-data>
J2EE Security and Enterprise JavaBeans

The `<credentials>` element denotes the password that will be used to authenticate the user. Setting the `<credentials>` element data with the ‘!’ (as illustrated above) or with `<credentials clear = “true”>welcome<credentials>` enables you to use clear (readable) passwords in `jazn-data.xml` file for the first time. When the file is read and persistence occurs, the password in `jazn-data.xml` is **obfuscated** and becomes unreadable.

I wrote some basic provider data (for the scope of this project I just had to create some users and roles) by editing the `jazn-data.xml` file using a text editor.

There is however, scope for further work to be done in the area of managing the XML-based provider type. The JAZN Admintool can be used to manage XML-based provider data from the command prompt. I have not been able to use this tool to manage provider data in `jazn-data.xml` file. Another area for further work would be to develop programs using the JAZNUsrManager class (which is used to implement JAZN) to create realms, users and roles; grant roles to users and assign permissions to specific users and roles in the XML-based provider type. I did not concentrate on the above 2 areas as it would have meant focusing further on the server side of OC4J security.

B. Declarative Security

The security view of the enterprise beans is contained in the J2EE deployment descriptor `ejb-jar.xml`. This file also contains structural and referential information of the bean classes. A security view consists of a set of logical security roles. A logical security role is a semantic grouping of permissions that a given type of users of the application must have in order to successfully access use the application resources (in this case EJB methods). Security roles ‘FACULTY’ and ‘STUDENT’ for the application can be defined in `ejb-jar.xml` as given below:

```xml
<assembly-descriptor>
  <security-role>
    <role-name>FACULTY</role-name>
  </security-role>
  <security-role>
    <role-name>STUDENT</role-name>
  </security-role>
</assembly-descriptor>
```

Method permissions can then be defined by specifying the methods of the enterprise beans’ home and remote interface that each security role is allowed to invoke. The security view of the security is thus defined.

```xml
<method-permission>
  <role-name>FACULTY</role-name>
  <method>
    <ejb-name>Student</ejb-name>
    <method-name>findByPrimaryKey</method-name>
  </method>
</method-permission>
```

The FACULTY security role is allowed to access the ‘findByPrimaryKey’ method of the ‘Student’ bean. More than one security role can be listed for a `<method-permission>`. If we wanted to allow the STUDENT security role access to the ‘findByPrimaryKey’ method of the ‘Student’ bean, then we can modify the above as

```xml
<method-permission>
  <role-name>FACULTY</role-name>
  <role-name>STUDENT</role-name>
  <method>
    <ejb-name>Student</ejb-name>
    <method-name>findByPrimaryKey</method-name>
  </method>
</method-permission>
```
In general each `<method-permission>` element includes a list of one or more security roles and a list of one or more methods. For each `<method-permission>`, all listed security roles are allowed to invoke all listed methods. It is possible that some methods are not assigned to any security roles. These methods can be accessed by anyone.

Now the logical security roles define in `ejb-jar.xml` have to be mapped on to the actual ‘users’ and ‘roles’ (who will be accessing the ejb application) defined in the `jazn-data.xml` repository of the application server. The mapping is specified in the OC4J specific deployment descriptor `orion-ejb-jar.xml`. Suppose we want the user ‘joe’ to access the ‘findByPrimaryKey’ method of the student bean then the mapping specified in `orion-ejb-jar.xml` would be

```
<security-role-mapping name="STUDENT">
    <user name="joe"/>
</security-role-mapping>
```

Now if the following is specified in `ejb-jar.xml` (only the FACULTY security role to access the ‘create’ method of the ‘student’ bean) then the user ‘joe’ will be refused access to the ‘create’ method of the student bean by the server

```
<method-permission>
    <role-name>FACULTY</role-name>
    <method>
        <ejb-name>Student</ejb-name>
        <method-name>create</method-name>
    </method>
</method-permission>
```

C. Programmatic Security

Since not all security policies can be expressed declaratively, the EJB architecture provide programmatic access to a caller’s security context by using the `isCallerInRole (String roleName)` and `getCallerPrincipal()` methods of the `javax.ejb.EntityContext` interface. The `isCallerInRole (String roleName)` tests whether the current caller of the enterprise bean has been assigned to a given logical security role or not. Both the above methods can be used within a bean method to impose role-based access restrictions on callers. Section 12.1.5 of the EJB 2.0 specification defines the methods of an entity bean class in which enterprise bean instances can access the methods of the `javax.ejb.EntityContext` interface. If an entity bean instance attempts to invoke a method of the `javax.ejb.EntityContext` interface in a particular entity bean class and the access is not allowed then the OC4J container throws a `java.lang.IllegalStateException`

One of the requirements of this project was that if a bean provider does not use any of the recommended security features, then the bean he deploys should not be allowed any database access whatsoever. I have attempted to use programmatic security to fulfill this requirement. All of the entity bean classes in this application extend a class called `KatsBase` that implements the `EntityBean` class. The `setEntityContext (EntityContext context)` method of the `EntityBean` class is called in the `KatsBase` class. This method passes a reference to the `EntityContext` interface to the entity bean instance and the container invokes this method after it creates the bean instance and before it allocates it to the pool of available instances. Thus the bean instance is not yet associated with any EJB object. Any resource that is going to be utilized by the bean during its life-cycle needs to be set up in this method. Thus a jdbc connection to the database is established in this method.

Ideally the `isCallerInRole (String roleName)` method could be used before establishing the jdbc connection to check if the caller is in the desired security role. If he is not in the desired security role, the jdbc connection could be denied and the bean instance could be terminated at the beginning of its life-cycle. However the EJB 2.0 specification does not allow the `isCallerInRole ()` method to be called in the `setEntityContext` method.

Thus, the bean provider would have to compulsorily use the `isCallerInRole ()` method to check for caller identity in all of the other `EntityBean` class and business methods before any data access is performed. EJB 2.0 specification allows the `isCallerInRole ()` method to be called in all the other methods. If the caller is not in the desired security
role then the jdbc connection will be closed and the bean will be terminated. Thus even tough the jdbc connection is initially set up; data access operations are always prevented from being carried out if the above check is performed. Hence an illegitimate caller is prevented access to the database.

IV SUMMARY

User access control in an Enterprise JavaBeans application has been implemented using a combination of declarative security and programmatic security. This solution is intended to aid the development of EJB application security at the Information Integration Initiative program. In addition to providing a detailed look at the XML-Based provider type in OC4J (JAAS) security, this project gives a general overview for using the LDAP-Based provider type for authentication and authorization purposes. The project implementation required creation of different scenarios as test cases to test access control. These scenarios have been illustrated in the appendix attached to this report. The security mapping was performed in the XML based deployment descriptors utilized by JDeveloper when deploying the EJB application.

The EJB application provides the middle tier in the J2EE application framework and this project concentrates on security management involving EJB clients. The final implementation of the Information Integration Initiative is likely to be a full fledged enterprise application that would involve a web-tier interacting with the EJB layer. Hence the EJB security management would have to be integrated with the web tier security management to provide an integrated security system form the entire application. JDeveloper-the development tool used in this project can also be used to manage the web tier application and the associated security features. The skills developed during Enterprise JavaBeans development and EJB security implementation using JDeveloper can thus be extended to incorporate the web development and web tier security.

Also, there is scope for further work to be done in the area of managing JAAS provider types or server side OC4J security. As mentioned in the report, there are certain tools that can be used to manage the data used for authentication and authorization in the JAAS repositories. Investigating this area of work would build up expertise in server side (OC4J) security management to complement the application or client side security management featured in this project.