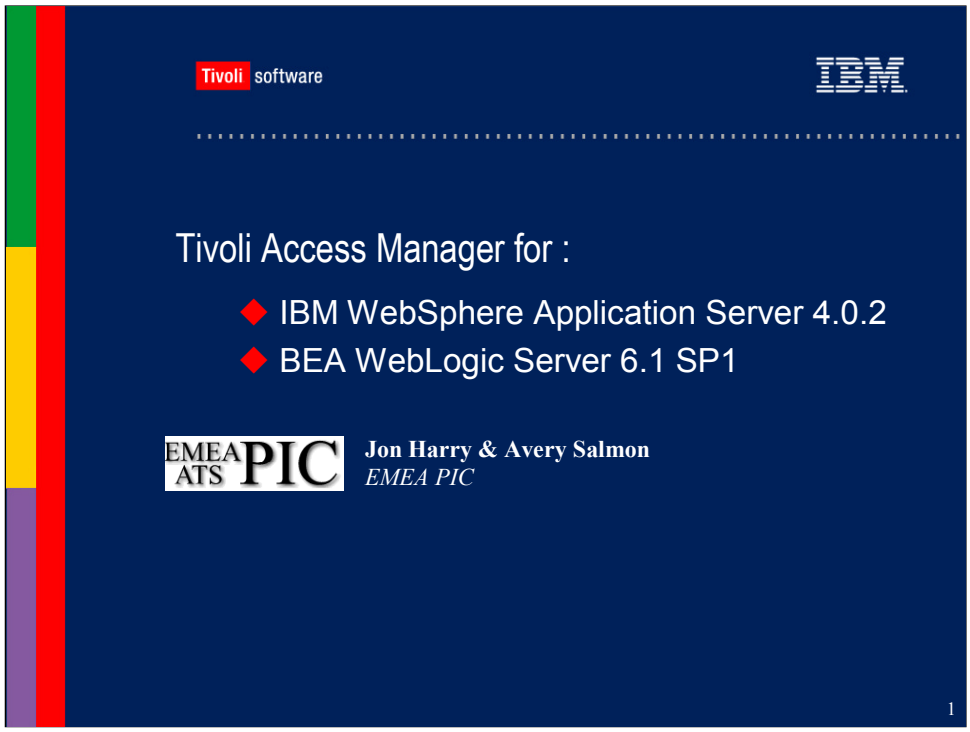


This presentation describes how Tivoli Access Manager can be integrated with IBM WebSphere v4.0.2 and BEA WebLogic Server 6.1 with SP1 to implement centralized authorization of J2EE applications.

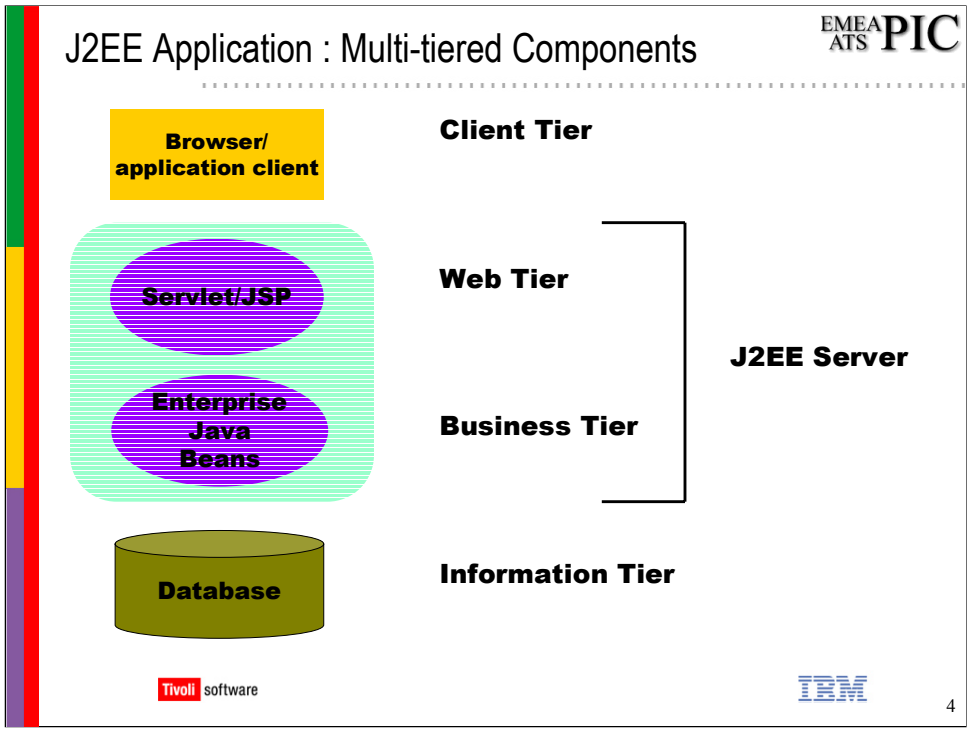
This function was released for Policy Director v3.8 but has been enhanced and modified for Access Manager v3.9.

This presentation is primarily a description of the functionality of AMWAS and AMWLS, which is the same between v3.8 and v3.9. However, there are additional slides and comments included for the differences in v3.8 and v3.9 function where appropriate.

The contents of the presentation are shown above.

This document contains an overview of the J2EE security model and then details of how Tivoli Access Manager integrates with two implementations of the model.


EMEA ATS PIC
J2EE Application & Security Overview
Tivoli software
IBM
3



Java 2 Enterprise Edition (J2EE) provides the standard for developing multi-tier, server-side java applications. J2EE builds upon features of Java 2 Standard Edition (J2SE) to add

•Enterprise JavaBeans™

•Java Servlets API

•JavaServer Pages™

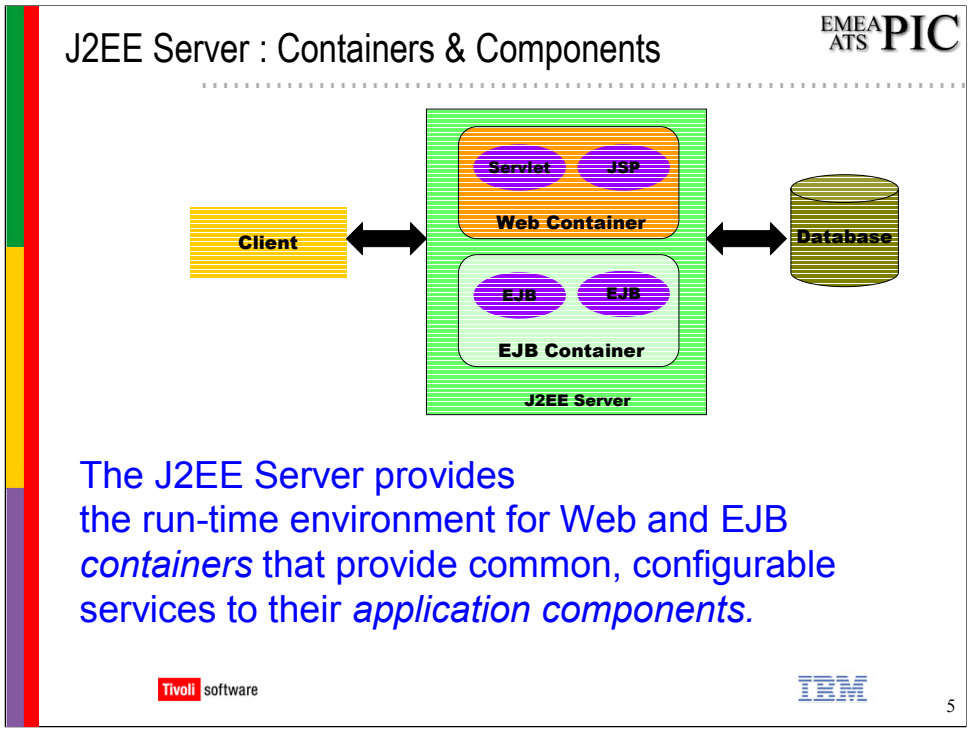
•XML technology.

The entire J2EE application is made up of *components* in the various tiers. All J2EE applications will contain one or both of the two server tiers.

Information on Web Applications from *Java™ Servlet Specification.*

Information on Enterprise Applications from *Enterprise JavaBeans TM Specification.*

The definitive source for Java specifications is **http://java.sun.com/products/**. A J2EE tutorial is available at **http://java.sun.com/j2ee/tutorial/1_3-fcs/index.html.**



The J2EE Server provides *container services* for the application components that reside on the server. Container services include:

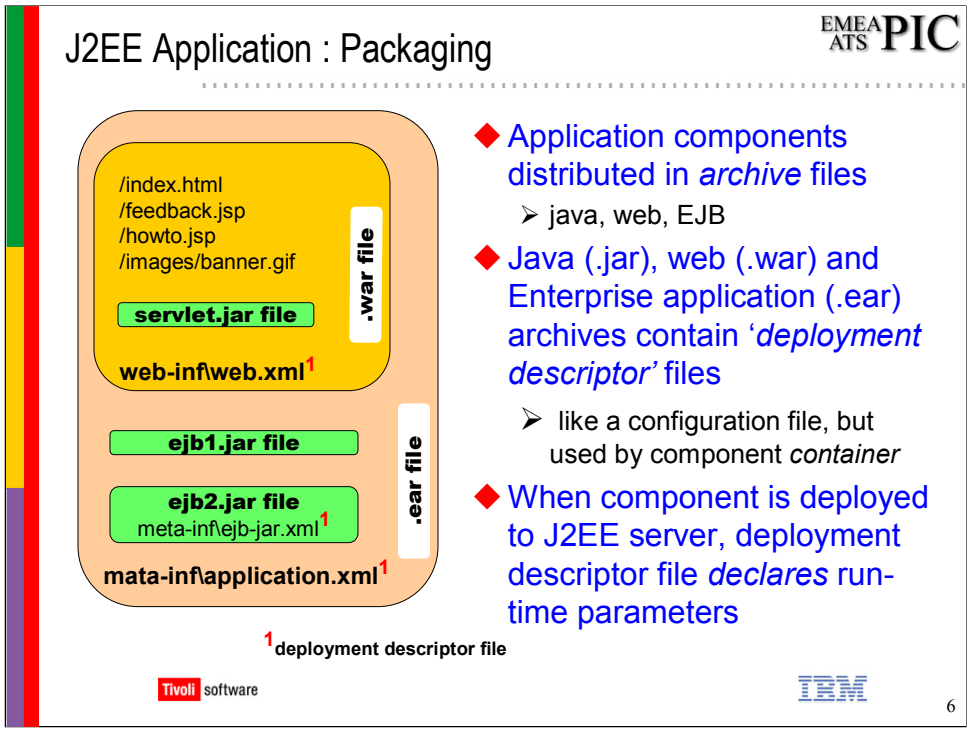
•**security**
•transaction management
•Java Naming and Directory Interface (JNDI) lookups
•remote connectivity to EJBs.

The J2EE server is the runtime portion of a J2EE product. A J2EE server provides EJB and Web containers:

•EJB container --manages the execution of enterprise beans for J2EE applications.
•Web container -- manages the execution of JSP pages and servlets components for J2EE applications.

The EJB and Web containers are implemented by the J2EE server as wrappers around each EJB, JSP and Servlet.

To download the Java Servlet and JavaServer Pages specifications go to http://www.jcp.org/aboutJava/communityprocess/final/jsr053/



The three types of packaging files shown here are:

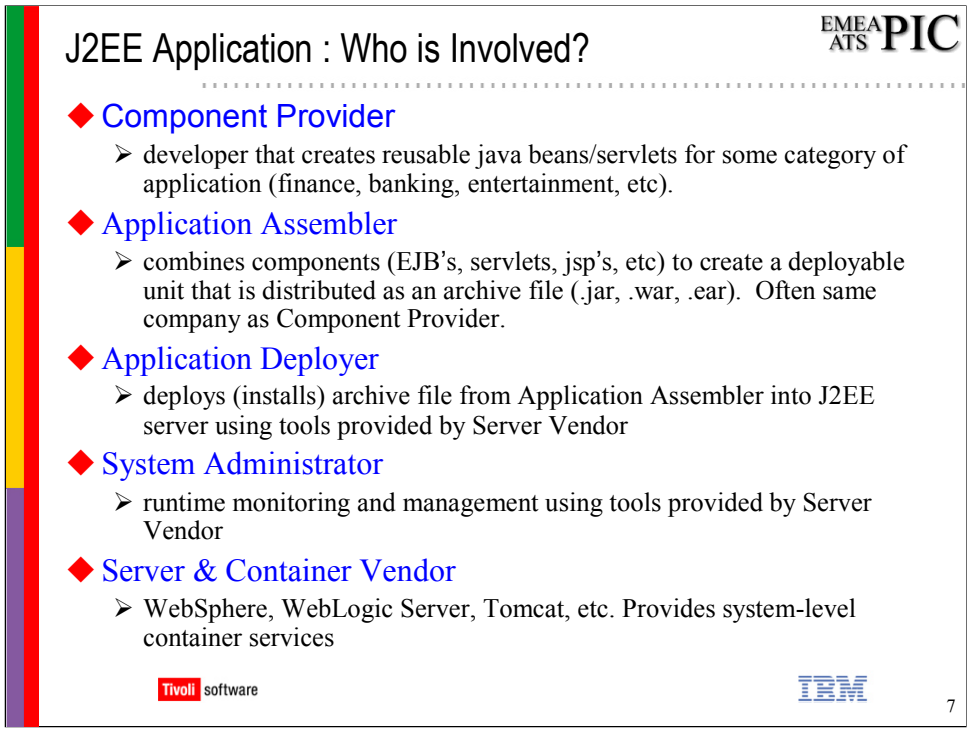
1. .jar file = java archive file : contains executable '.class' files, configuration '.properties' files, and META-INF/ejb-jar.xml **deployment descriptor** file.
2. .war file = web archive file : contains servlets,JSPs, static html & image files, java archive files, applets, and a **deployment descriptor** file with name web-inf\web.xml
3. .ear file = enterprise application archive file : contains .jar files with EJBs, .war files, and a **deployment descriptor** file with name `META-INF/application.xml`

Both .war and .ear files are standard .jar files with a different file extension. That all have the same format. WINZIP will open all of them.

The **deployment descriptor** file is an XML document that describes a component's deployment settings.

For example, an ejb-jar.xml file may describe transaction attributes and security authorizations for the EJB. At run time, the J2EE server reads the deployment descriptor and acts upon the component accordingly.

Application Servers also have their own proprietary XML files (still called deployment descriptors) containing vendor specific info.

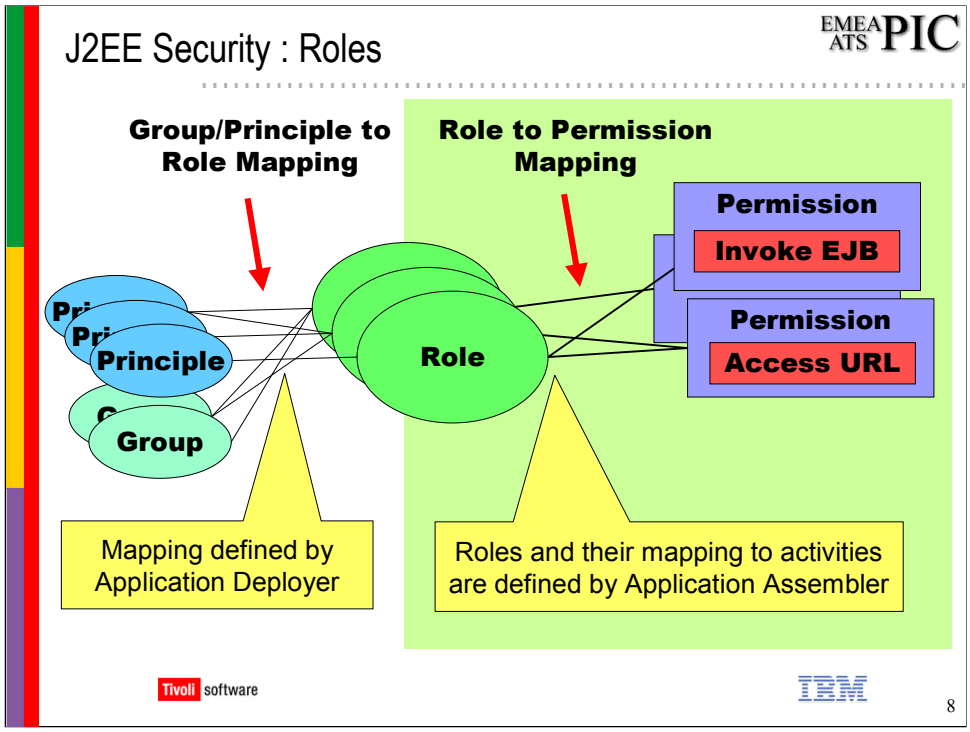


The J2EE model describes a number of different entities involved in the development, use and maintenance of a Web Application.

In terms of the integration of Access Manager into the J2EE environment, the Application Deployer and System Administrator are those who have to be most aware of the part that Access Manager will play.

The Application Deployer must ensure that Access Manager is populated with the information required for it to protect existing and new Web Applications deployed on the Application Server.

The System Administrator need**s** to use Access Manager's Administrative Console to permit/revoke access to the J2EE application resources.



A *security role* is a semantic grouping of permissions that users must have to successfully use the application in a particular way. The Application Assembler defines *security* roles. e.g "employee"

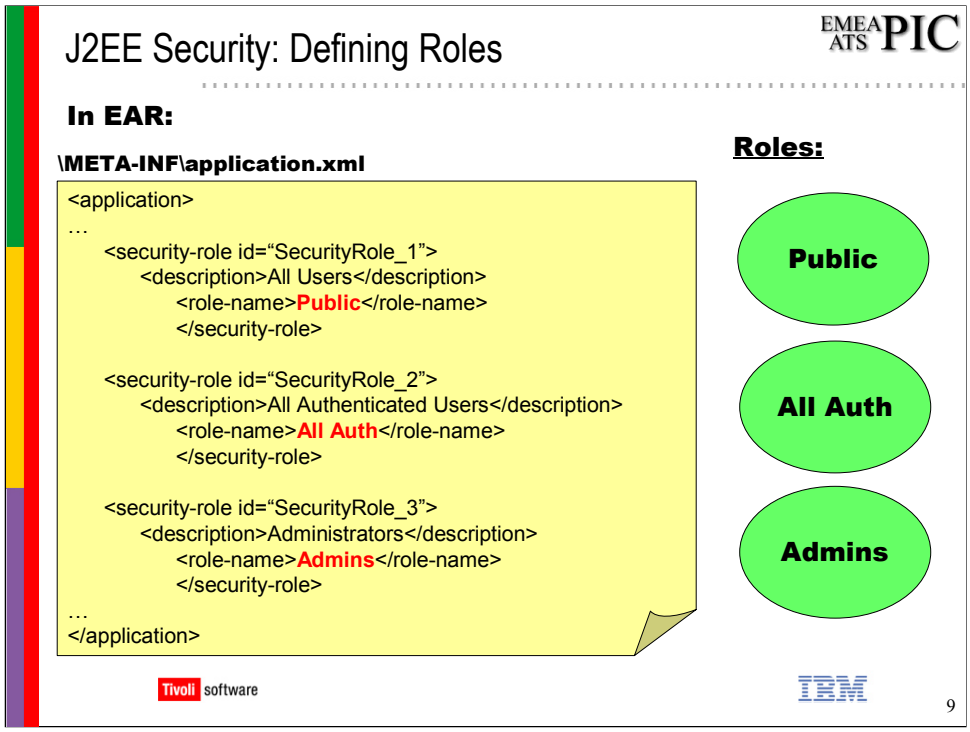
An EJB Application Assembler declares (in the deployment descriptor) *method permissions* for each security role. A method-permission specifies the role required to invoke a group of methods on the EJB.

A servlet Application Assembler declares (in the deployment descriptor) a security-constraint that specifies the roles required to access a given URL pattern. (May also specify the HTTP method to be constrained).

The creation of security roles make the Deployer's task easier. The roles present a simplified security view of the J2EE application for the benefit of the Application Deployer.

The Application Deployer is responsible for deciding which Principles (users) and Groups map to each Role.

Note: Principle/Group to Role mappings may be defined in server specific deployment descriptor extensions. These describe how the mapping is initially set up when the application is deployed.

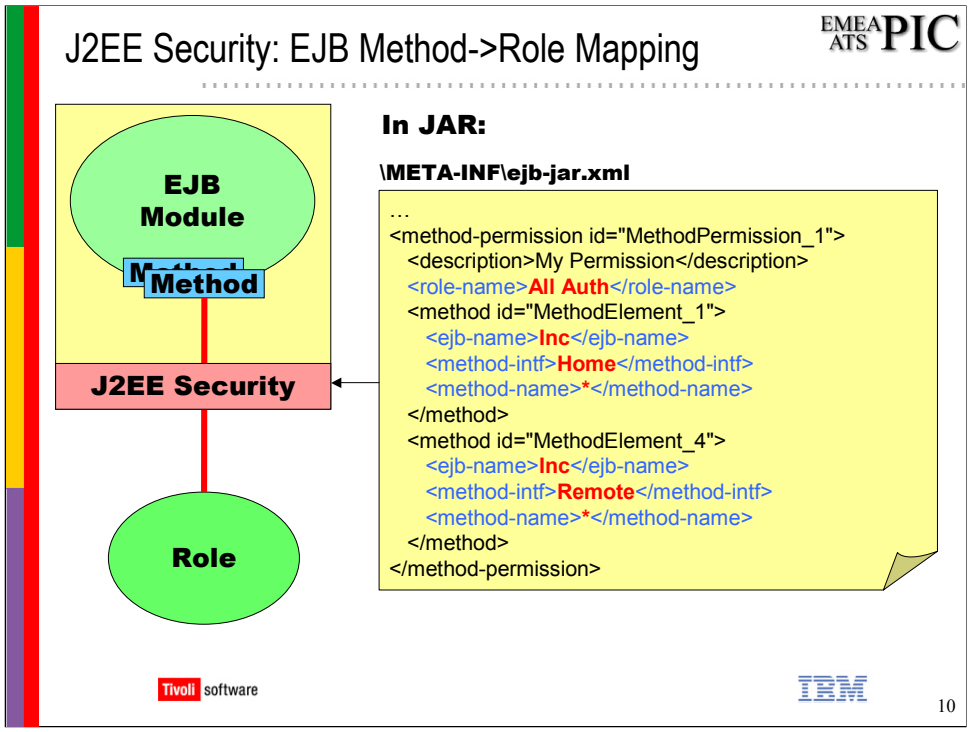


The slide above shows part of the *application.xml* deployment descriptor file that is found in an **Enterprise Application Resource (EAR)** file.

This information declares the roles that will be defined in the application. The description is used by the Application Deployer to determine which users should be assigned to which roles for the deployed environment.

Note: The basic J2EE standard does not include any automated way to map from roles to "real" users in the domain. The only hint the Application Deployer gets (except for product documentation) is the description of the role as shown above.

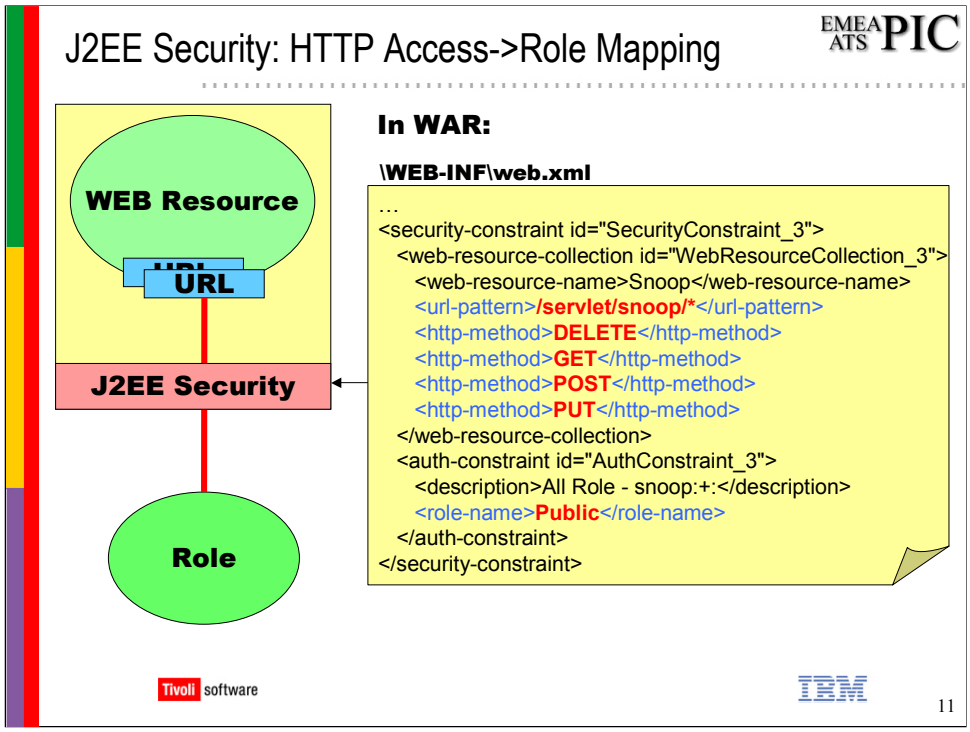
That said, both WebSphere and WLS have defined extensions in the EAR file that allow them to specify users (and special groups of users) to be assigned to the roles. These are discussed in the product sections.



Once the roles have been declared globally for an application they can be assigned permissions in the application components.

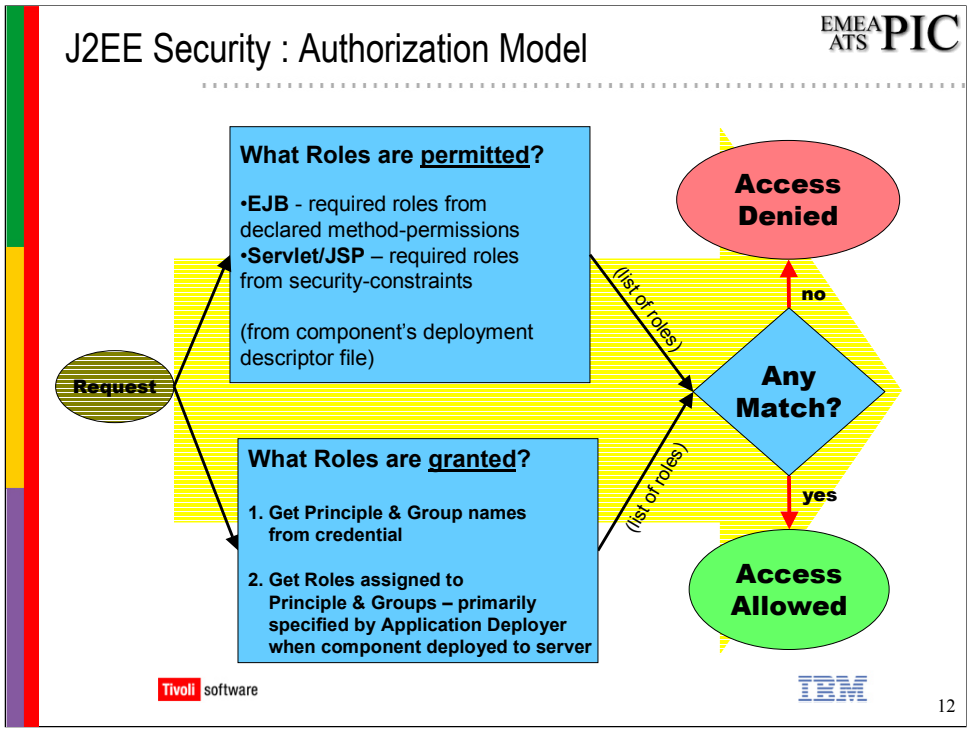
In EJB components, which are packaged as **Java Application Resource (JAR)** files, there is a deployment descriptor file, *ejb-jar.xml*, that defines which Roles can invoke which EJB methods. This information (which is imported into the Application Server from the deployment descriptor at deployment time) is used at run-time to determine which Roles are permitted access to the method being requested.

Once the permitted roles are known it just remains to determine if the requesting user is granted any of the permitted roles.



In Web components of an enterprise application, which are packaged in **Web Application Resource (WAR)** files, there is a *web.xml* deployment descriptor file that determines which Roles are permitted to access which web resources. This information (which is imported into the Application Server from the deployment descriptor at deployment time) is used at run-time to determine which Roles are permitted to access the Web resource being requested.

Once the permitted roles are known it just remains to determine if the requesting user is granted any of the permitted roles.



The foil above describes *Declarative Security* as implemented by the EJB or Web container. This security is done without the need for any coding by the application developer – the application assembler defines which roles can access which methods/web resources and this is enforced at runtime by the Application Server.

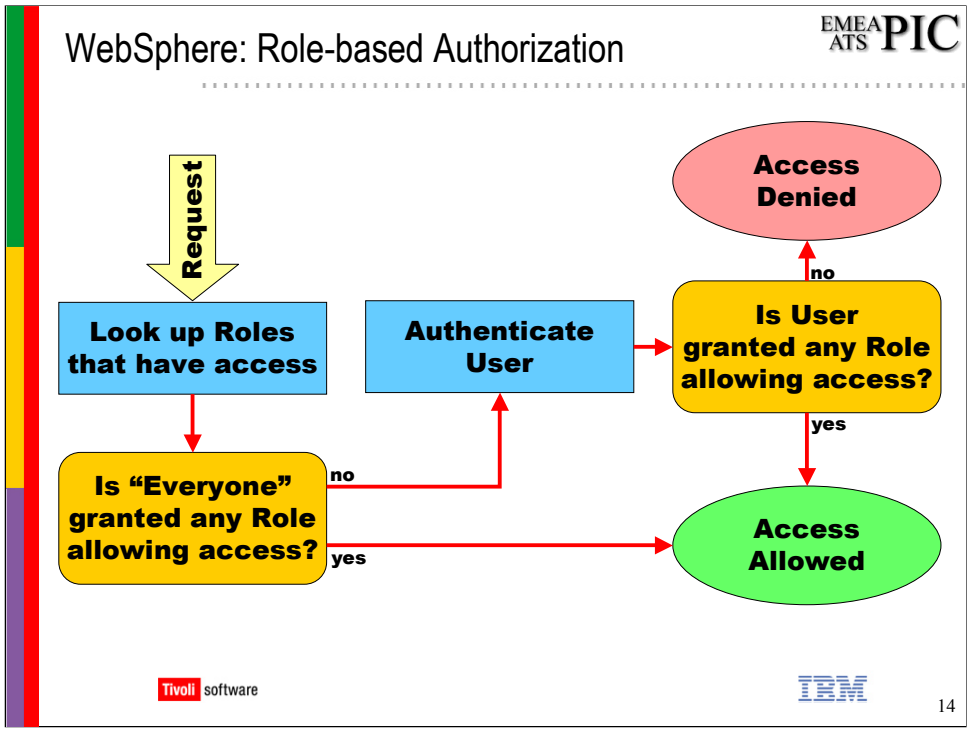
Note: Not all security policies can be expressed declaratively and the EJB and Servlet specifications also provide a simple programmatic interface that the Application Provider may use to check that the user has the necessary role for access:

•EJBs use: isCallerInRole(role-reference);

•Servlets/JSP: isUserInRole(role-reference);

A role-reference (which is just a level of abstraction to insulate the role names used by the application developer from those used by the assembler assember) is mapped to a role-name by the Application Assembler. The check for role membership is done using the same process as that which is used by the container.

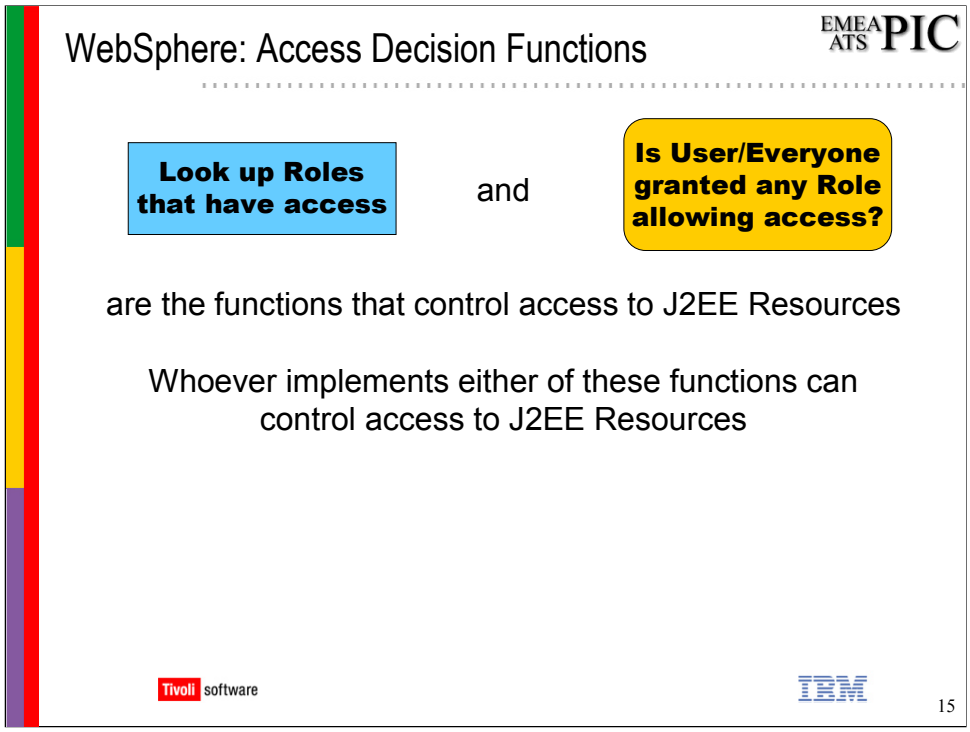

EMEA ATS PIC
WebSphere Security Overview
Tivoli software
IBM
13



When a request is received by an EJB or WEB container in IBM WebSphere Application Server (WAS) the process shown above is used to determine if access should be allowed or denied.

If no user is currently logged into WAS then a check is done to determine if the resource being requested is a public resource. This is done by checking if any of the Roles allowed access are mapped to "Everyone". If it is a public resource then access can be allowed with no further processing.

If the resource is not public then the user is authenticated by WebSphere (using any valid method) and then a check is done to determine if that user is a member of any of the Roles that are permitted access to the resource. If the user is a member of one or more permitted roles then access is allowed. If the user is not a member of any of the permitted roles then access is denied.

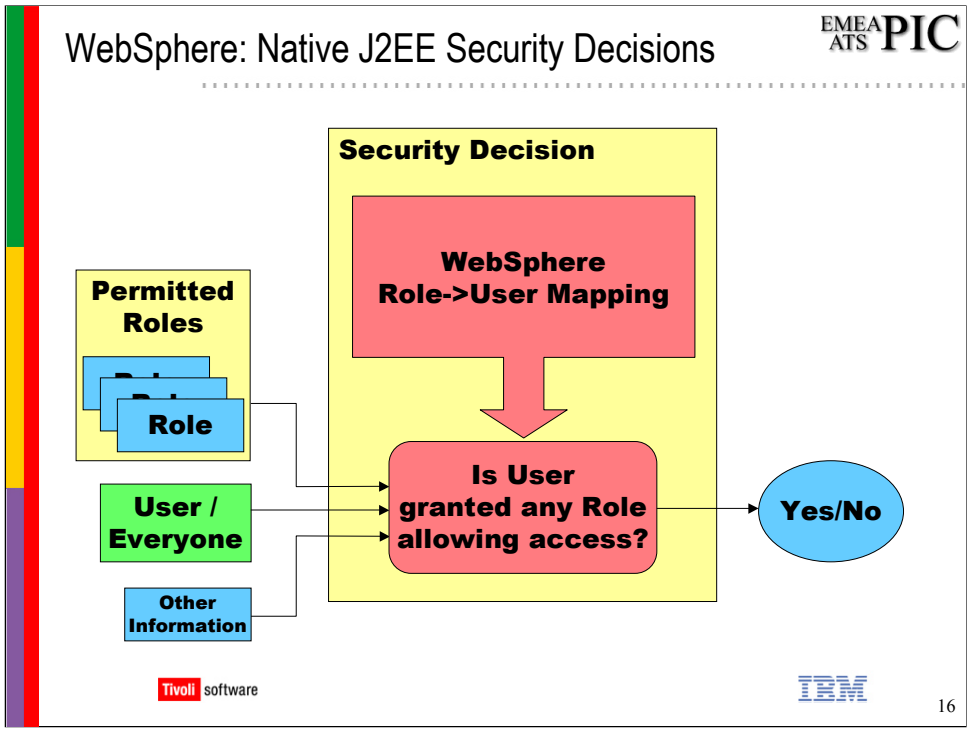


From the previous diagram it is clear that the access decisions are being made in two stages:

1) Determine if the user (or "everyone") is granted any of these roles
2) Determine which roles have access to the resource being requested

In a native WebSphere environment, WebSphere is responsible for both of these functions an thus has complete control over access to J2EE resources.

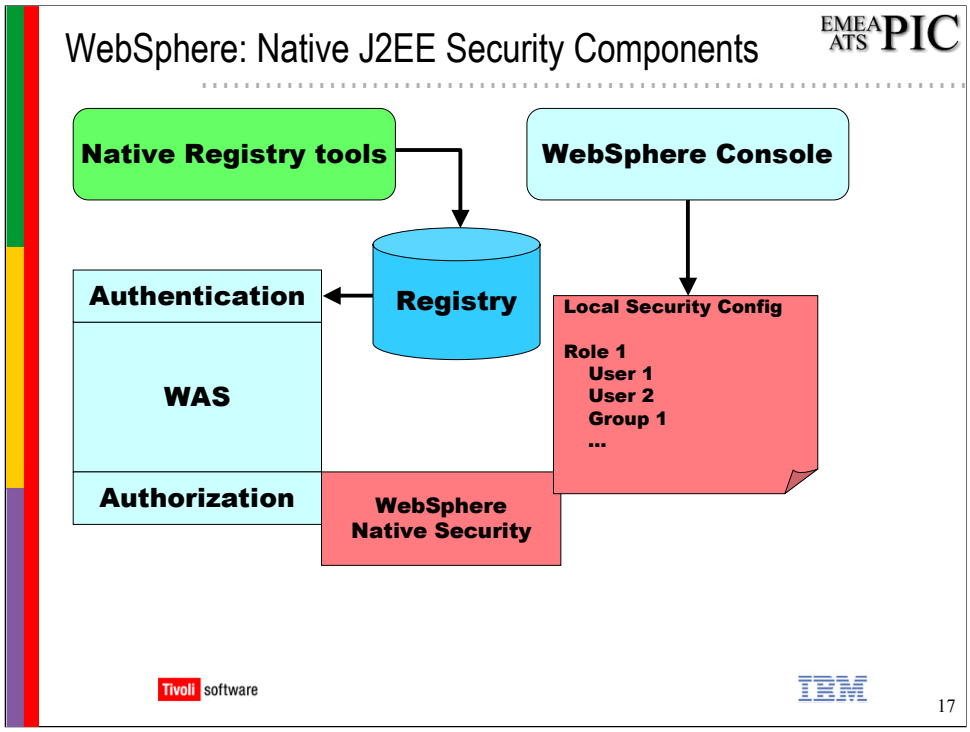
In an environment where Access Manager ***is*** integrated with WebSphere, WebSphere still maintains control over determining which roles have access to a given resource. Access Manager takes control of the "Is User/Everyone granted any Role allowing access" function and ***so*** has the final say in who gets access.



The diagram above shows the decision function used to determine if a user is granted any of the roles needed to access an object.

The inputs to the function are a list of roles (those permitted access to the resource in question), A user identifier (which could also be “everyone”) and other context information (such as the application name, server name etc.)

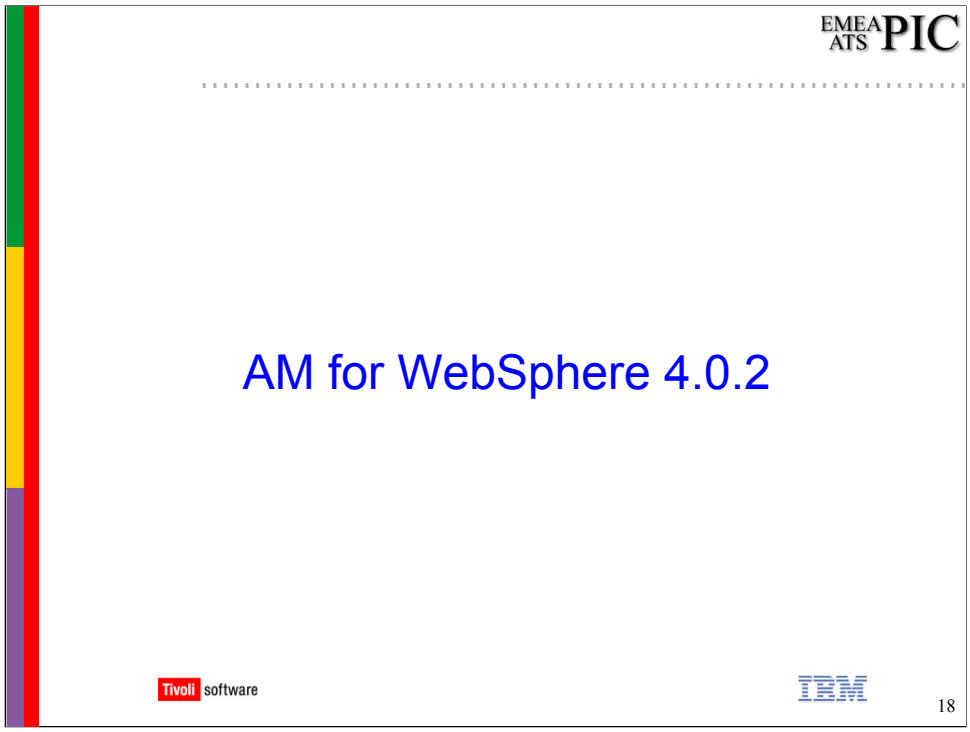
In the native WebSphere environment the internal WebSphere Role->User mapping table is used to determine if the user is a member of any of the permitted roles. This mapping is configured locally by the node administrator using the WebSphere Security Center.

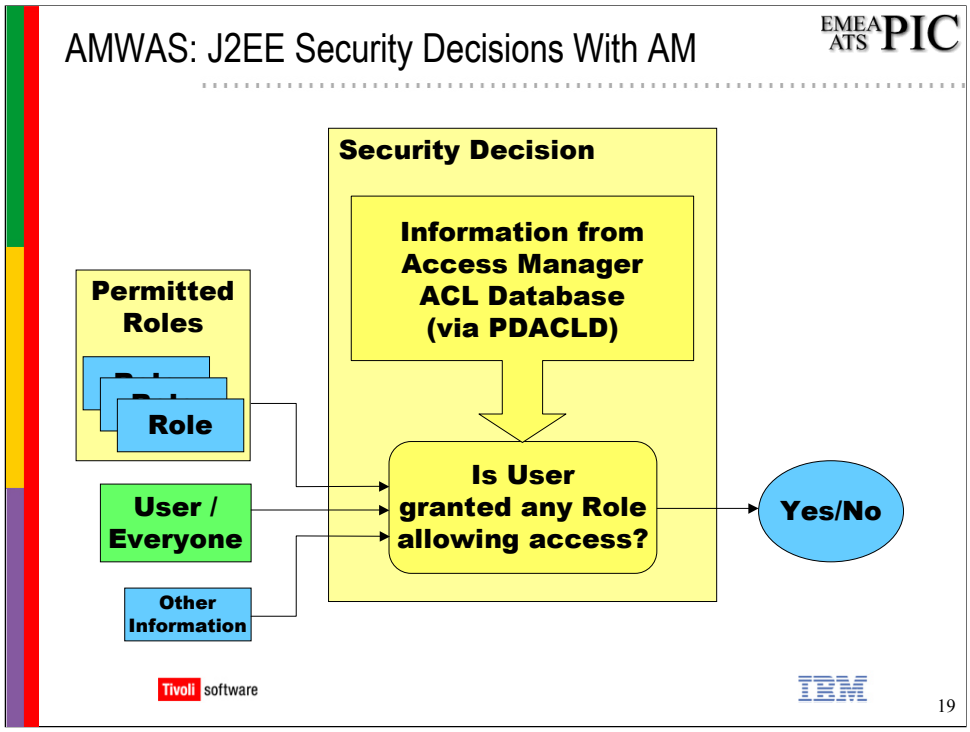


The diagram above shows the components involved with authentication/authorization in a WebSphere native environment.

Authentication is performed by WebSphere against the configured user registry. Note that WebSphere relies on the native registry tools for user management – it is only a consumer of user information.

Authorization (checking whether a user is granted any of the permitted roles) is done against the local security configuration – the local mapping of roles to users. The management of the roles is done using WebSphere tools.

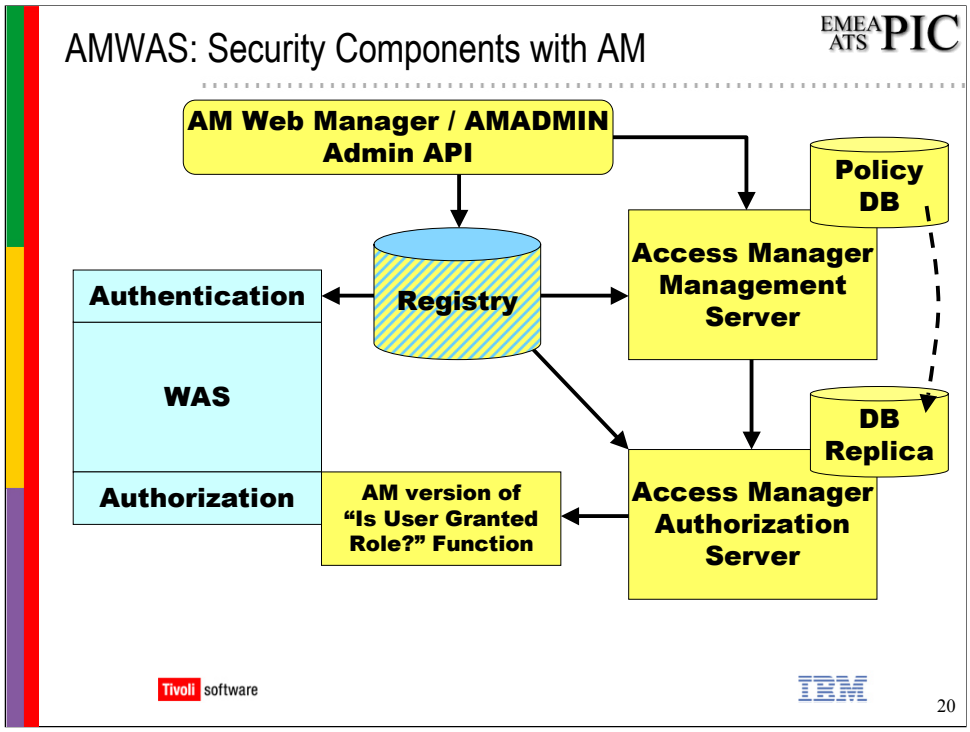

EMEA ATS PIC
AM for WebSphere 4.0.2
Tivoli software
IBM
18



When Access Manager is integrated with WebSphere Application Server, the function that determines if a user is granted any permitted roles is handled by Access Manager.

The inputs to the function are the same as in the native WebSphere environment but the role membership is now determined by information in the Access Manager ACL database (which is managed centrally by AM).

In addition to the role/user information passed in, Access Manager also allows some of the context information to be taken into account when making the access decision. More details later.



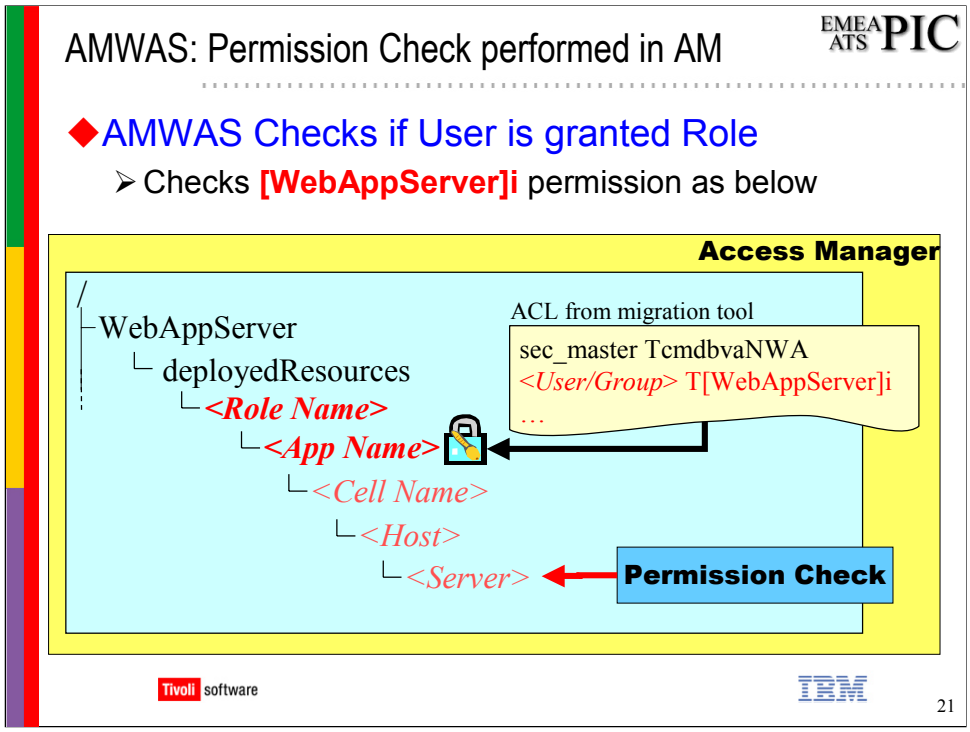
The diagram above shows how Access Manager integrates with WebSphere to perform J2EE security.

In the case above, WebSphere authenticates users against a registry (probably LDAP) that is shared with AM. This not only means that there are no problems with user synchronisation but also means that Access Manager tools can be used to centrally manage the users (rather than using native registry tools).

For authorization, WebSphere determines which roles are permitted to access the requested resource and then Access Manager is called to determine if the user is granted any of these roles.

The role->user mapping information is stored in the Access Manager ACL database. This information can be centrally managed using Access Manager tools, that interact with the AM Management server (PDMgrd). The information is replicated to the Access Manager Authorization Server (PDACLD) where it is accessed by the WebSphere AM integration code (AMWAS).

Note: It is possible for AMWAS to work without a shared a user registry. In this case WAS users must exist in both registries.

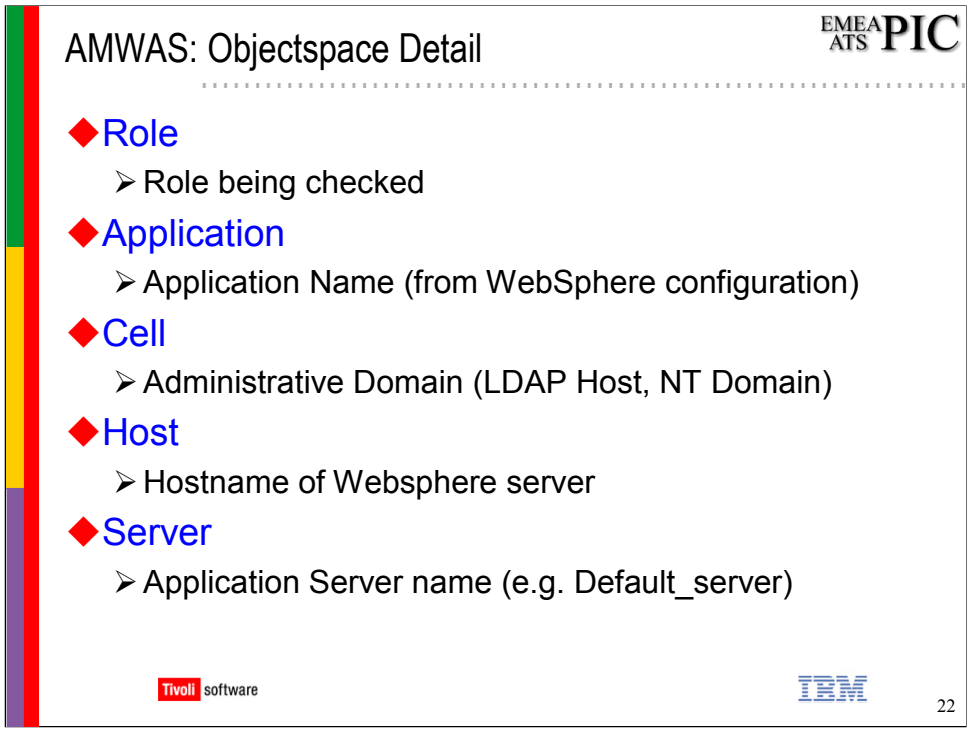


The diagram above shows the permission check that is performed in Access Manager to determine if a User is granted a Role. The user is deemed to be granted the role if they (or a group they are a member of) have the (i)nvoke permission - in the WebAppServer action group – on the object being checked.

By default the only objects created in the objectspace by the AMWAS migration tool (discussed later) represent the Role and the Application using the role. As shown above, the ACL created by the migration tool is attached to the Application object.

The permission check performed by AMWAS is done on an object that includes other information (Cell Name, Host, Server - discussed on the next page) in its name. This gives the check the potential to be very granular if the AM administrator wants to manually create the additional objects and attach ACLs to them. By default these objects do NOT exist and the check depends (by ACL inheritance) only on the Role Name and the App Name.

Note: The administrator can move the ACLs created by the migration tool (see later) but this may cause problems if the migration tool is re-run for the same application. Where possible it is better to create new ACLs.



The information considered when determining if a user should be granted a role is as follows:

Role – The role being checked

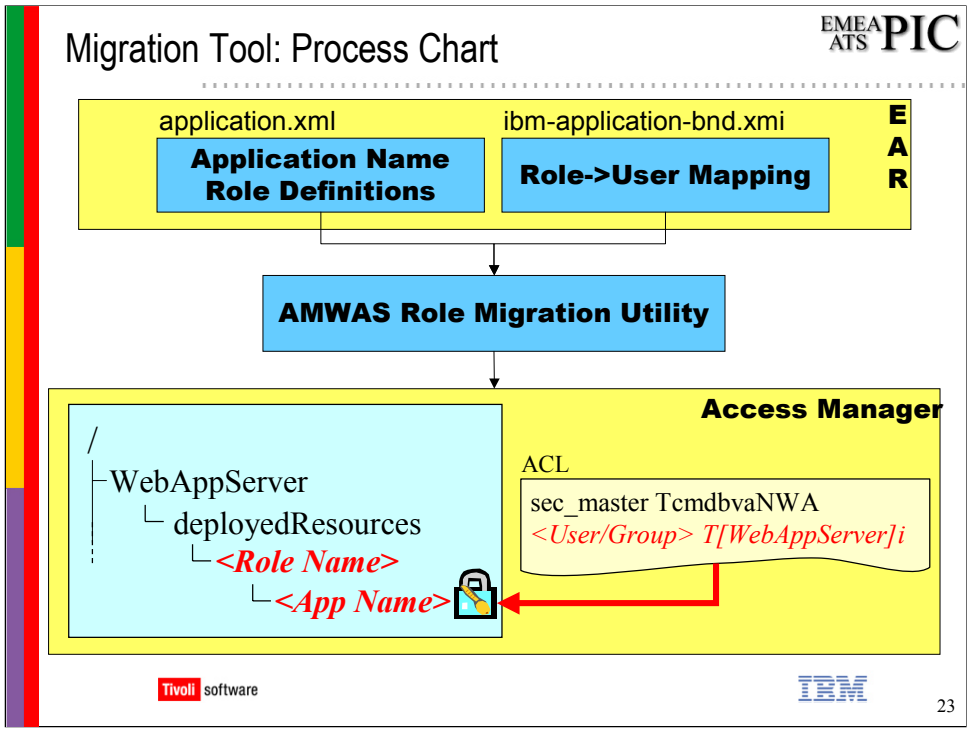
Application – The name of the Enterprise Application being accessed.

Cell – The administrative domain – the registry the user was authenticated against. If using LDAP user registry then Cell will be LDAP host in format <DNS Name>:<port>. E.g. myhost.ibm.com:389. If using NT User Registry then Cell will be NT Domain name. E.g. MYDOMAIN.

Host – The hostname of the WebSphere Node. E.g. myhost.

Server – The Application Server that is providing the resource.

It is clear from the above that Access Manager could, with a little customization of the objectspace, allow very fine-grained control over access to J2EE resources. It would be possible, for example, to only allow a user to access a resource if they are connecting to a specific application on a particular application server (on a specific host). In contrast the access control can also be made very course for simple administration.

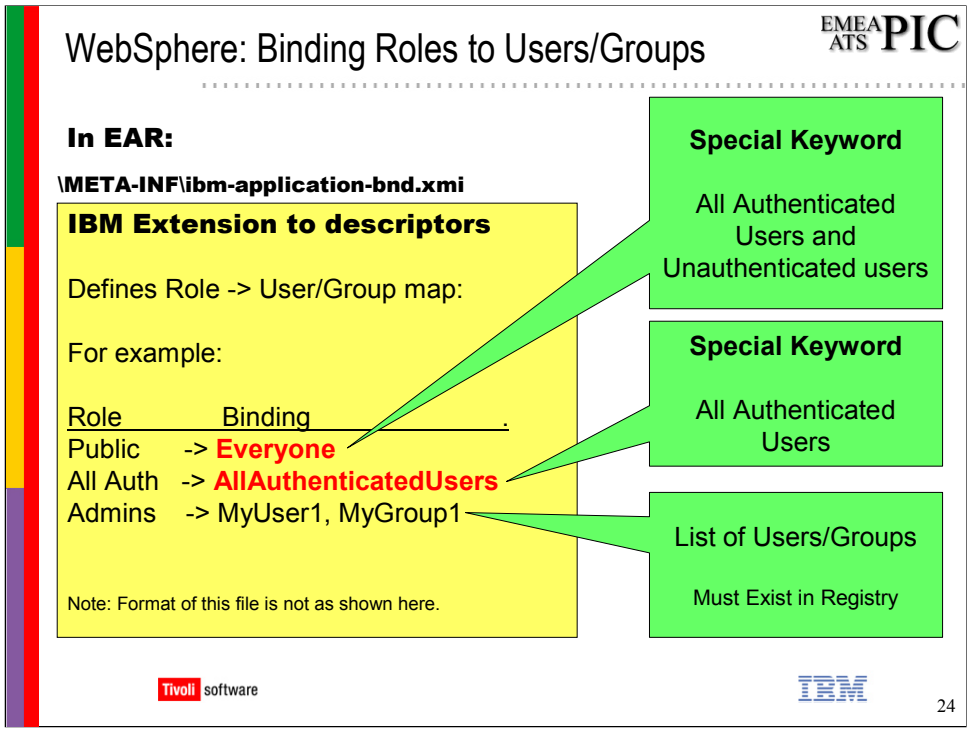


In order for a check to be done in Access Manager, the appropriate objects (and actions) must be created in the ACL database. AMWAS provides a migration tool that automates this process.

The migration tool reads information about an application from the EAR file and creates the appropriate objects in the objectspace. It uses the *application.xml* file to determine the application name and the roles that are used by the application– these are created as shown above.

In addition to information about the roles, the migration tool can also interpret a WebSphere extension to the the J2EE descriptors that describes how to map users, groups and certain special user types to the application roles (see next page). If this information is present in the EAR file's deployment descriptors then entries will be added to the ACL associated with the role to reflect this.

The first time the migration tool is used it creates the WebAppServer objectspace, the deployedResources object, the WebAppServer action group and the (i)nvoke action. It also creates a group called *pdwas-admin* that represents WAS administrators and adds the WAS admin user to that group.



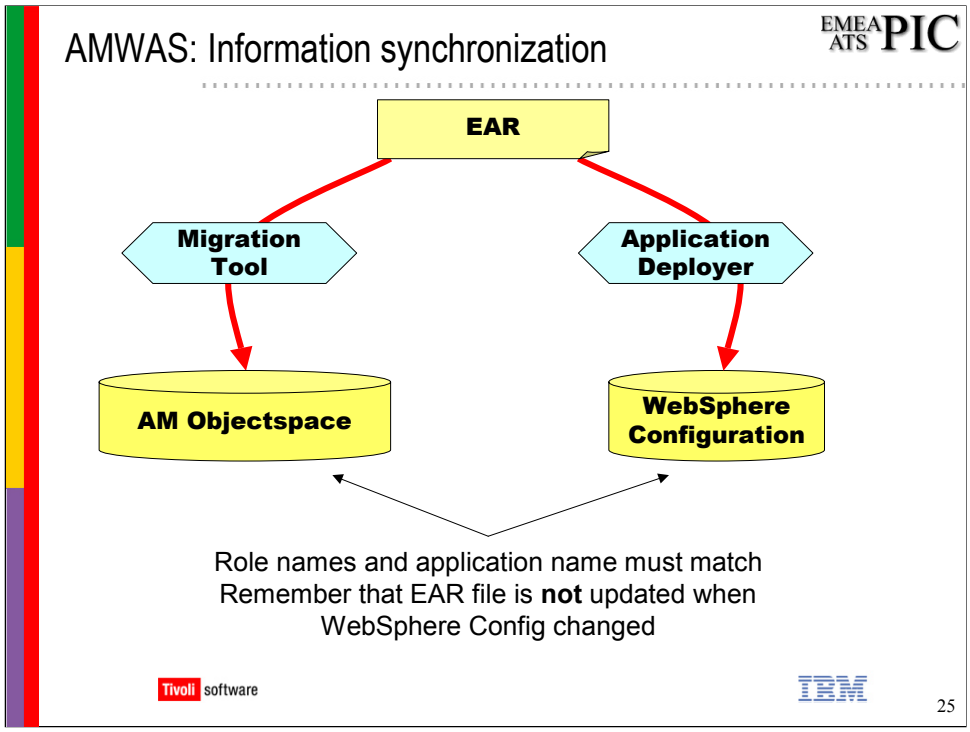
WebSphere provides an extension to the EAR file descriptors that describes how roles should be mapped to users. This information is used by the AMWAS migration tool to add appropriate entries to the ACLs that grant roles.

There are three types of entries in this extended information that the migration tool recognises:

1) Special Keyword “Everyone”. This means that everyone (authenticated and unauthenticated users) should be granted this role. In response, the migration tool gives the (i)nvoke permission to the Access Manager “Unauthenticated” and “Any Other” pseudo-groups.
2) Special Keyword “AllAuthenticatedUsers”. This means that all authenticated users should be granted this role. In response, the migration tool gives the (i)nvoke permission to the Access Manager “Any Other” group.
3) A list of Groups and/or Users. This means that if these groups/users exist in the registry they should be granted the role. In response, the migration tool gives the (i)nvoke permission to the specified users and/or groups.

This is how the ibm-application-bnd.xmi file is formatted:

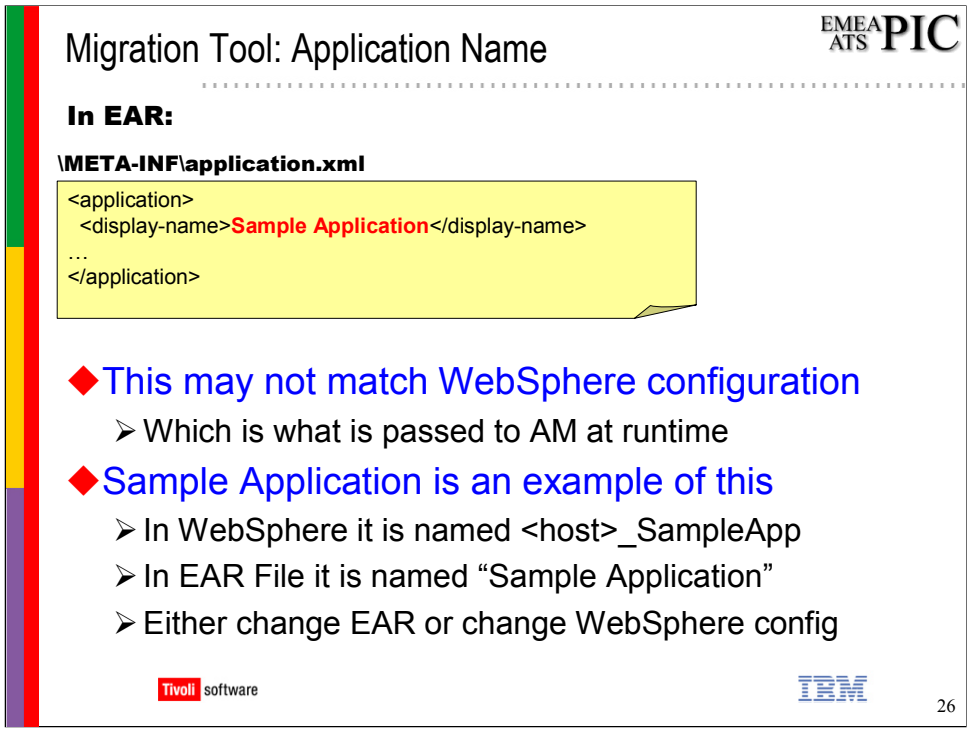
```
…
 <authorizations xmi:id="RoleAssignment_1">
    <role href="META-INF/application.xml#SecurityRole_1"/>
    <specialSubjects xsi:type="applicationbnd:AllAuthenticatedUsers" xmi:id="AllAuthUser_1"
name="AllAuthenticatedUsers"/>
  </authorizations>
  <authorizations xmi:id="RoleAssignment_2">
    <role href="META-INF/application.xml#SecurityRole_2"/>
    <specialSubjects xsi:type="applicationbnd:Everyone" xmi:id="Everyone_1" name="Everyone"/>
  </authorizations>
  <authorizations xmi:id="RoleAssignment_3">
     <role href="META-INF/application.xml#SecurityRole_3"/>
     <users xmi:id="User_1" name="cn=MyUser1, o=ibm,c=gb"
accessId="user:myhost.pic.uk.ibm.com:389/cn=MyUser1, o=ibm,c=gb"/>
    <groups xmi:id="Group_1" name="cn=MyGroup1, o=ibm,c=gb"
accessId="group:myhost.pic.uk.ibm.com:389/cn=MyGroup1, o=ibm,c=gb"/>
  </authorizations>
```



One important concept to understand is that (for WebSphere at least) the EAR file that the application is imported from is **not** updated to reflect changes made to the application via the WebSphere console after it is deployed into WAS. This means that it is possible that the information being loaded into AM by the migration tool (the application name and the role->user mapping) will not match what is configured in the WebSphere server.

In order to ensure that a previously deployed application is migrated correctly into Access Manager it is good practice to "export" the application to an EAR file using the WebSphere console and then immediately run the migration tool against that file – this will ensure AM and WebSphere are synchronised.

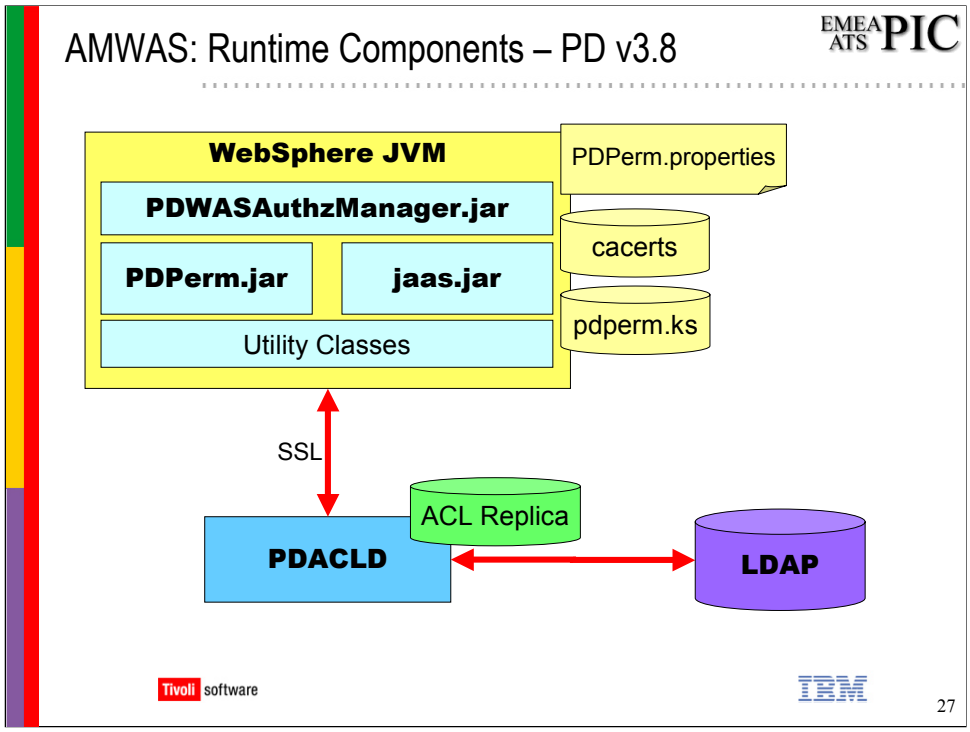
Note: Once an application has been migrated to Access Manager it is important not to change the application name in the WebSphere console.



As mentioned on the previous page, the application name must match in WebSphere and in the Access Manager objectspace. If there is a mismatch then no access to the application will be possible because the permission checks in Access Manager will be done against the wrong object.

In WebSphere the application name can be seen in the Administration Console under the "Enterprise Applications" tab. In Access Manager the application name can be seen in the objectspace under each of the appropriate Role objects.

The sample application that is optionally installed with WebSphere is given a different name in the WebSphere configuration than what appears in the SampleApp.ear file. In order to make the Sample Application work successfully with AMWAS you must either edit the *application.xml* file, re-export the application to a new EAR file or change the name of the application in WebSphere to match what is in the EAR file (Sample Application).



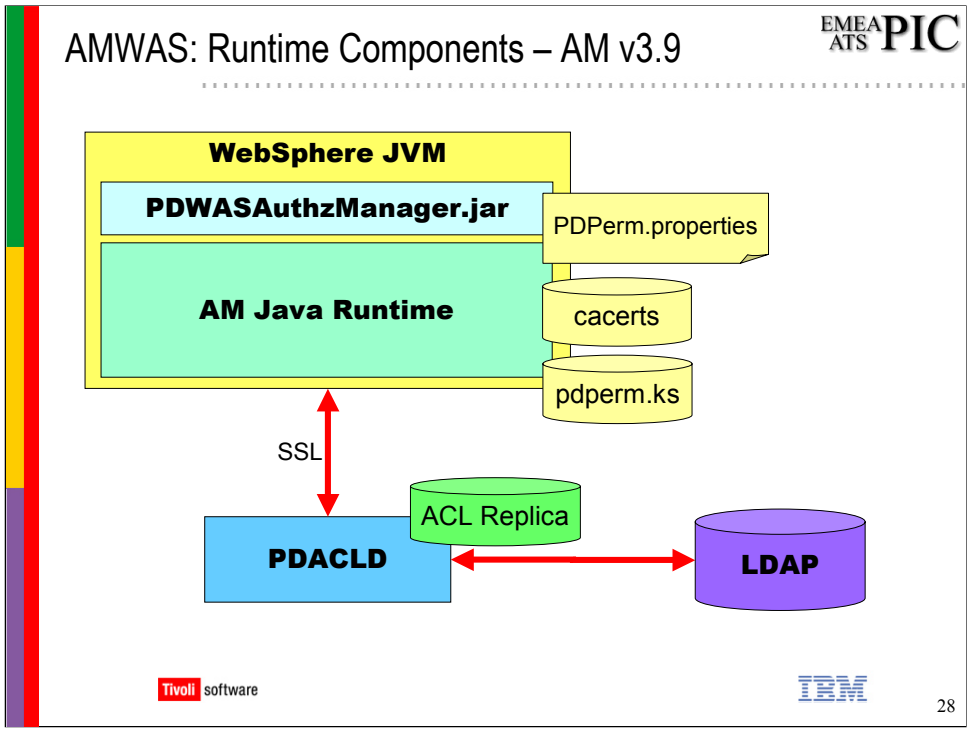
The diagram above shows the components of PDWAS v3.8 that run as part of WebSphere.

When PDWAS v3.8 is installed the PDWAS jar file and the PDPerm and JAAS jar files are installed into the WAS JVM lib directory. When the PDWAS configuration script is run PDPerm is registered with Policy Director which creates the PDPerm.properties file and the pdperm.ks file.

For those who are familiar with the PDPermission class the configuration script simply executes the com.tivoli.mts.SvrSslCfg class to perform this registration.

During runtime, the PDWAS class is called by WebSphere to determine role membership. Credential acquisition and permission checks are performed by a call to the PDACLD server.

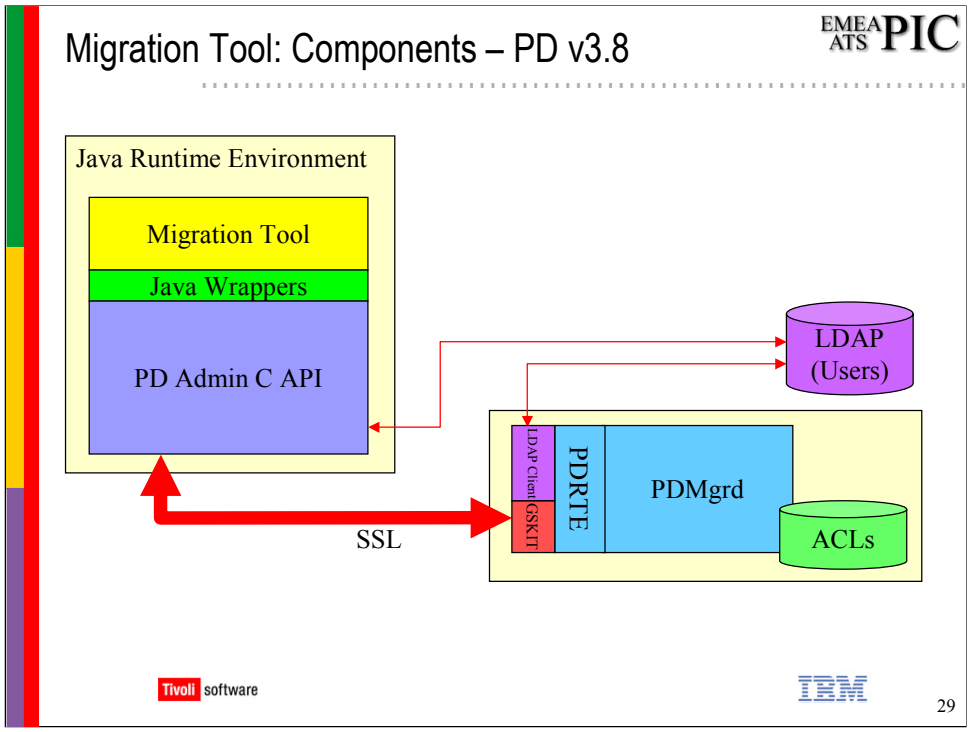
Note: PDWAS does NOT require the PD Runtime component (or its pre-requisites) to be installed on the local machine.



In Access Manager v3.9 the product packaging has been changed so that the functions required for a JVM to communicate with Access Manager are shipped together as the AM Java Runtime (AMJRTE) component. This component must be chosen at install time if AMWAS v3.9 is to be used.

AMJRTE includes a command-line utility for configuring a JRE environment for use with Access Manager. It copies the required AM security jar files into the <jre_home>/lib/ext and creates properties files for these.

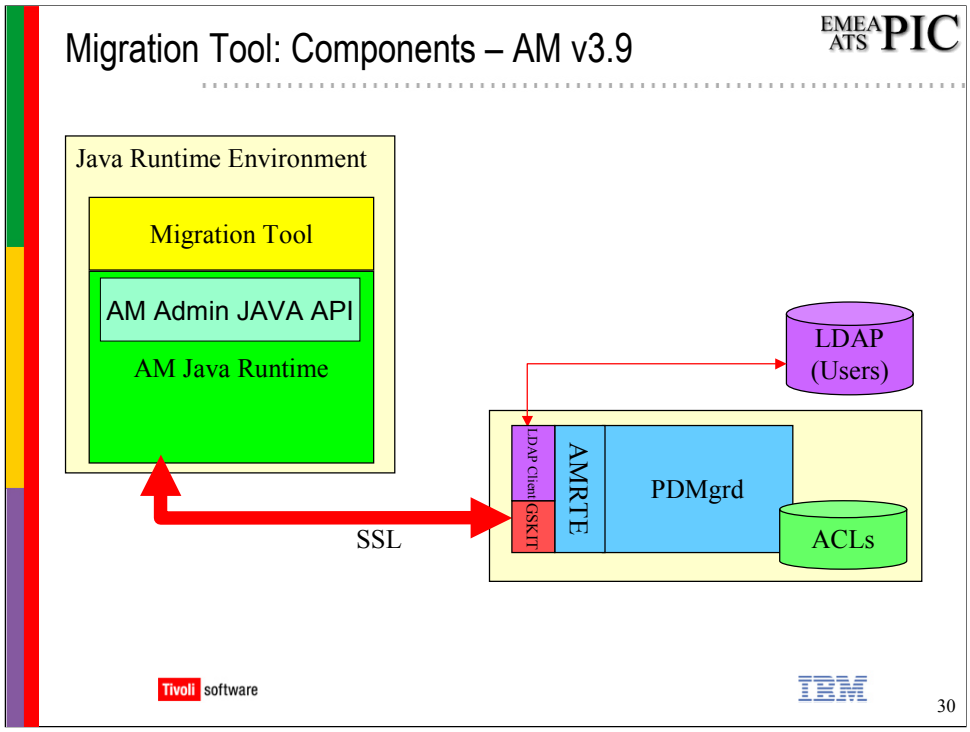
Note: AMWAS v3.9 does NOT require the AM Runtime component (or its pre-requisites) to be installed on the local machine – it only needs the AMJRTE component to be installed and configured.



The PDWAS v3.8 migration tool is a standalone java application (it does not run inside WebSphere Application Server). It uses a set of java wrappers to the Policy Director C Administration API. This API communicates directly with LDAP and the PD Management Server to perform updates to the user registry and the ACL database.

The migration tool requires that the PD Runtime component (and its pre-requisites) be installed and configured on the local machine. This is how communication with PDMgrd and the user registry is achieved.

Note: Although the migration tool does not require WebSphere to run, WebSphere v4.0.2 does provide all of the required modules for its operation.



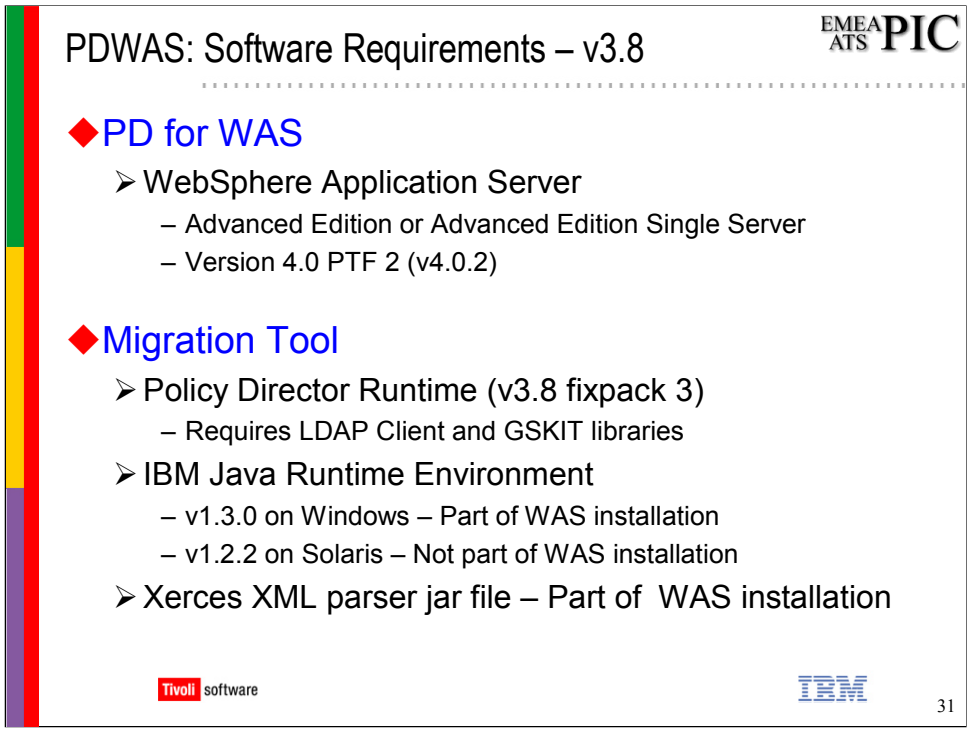
The AMWAS v3.9 migration tool is quite different.

It uses the new pure java admin API for Access Manager to communicate with PDMgrd without the need for AMRTE or any of its pre-requisites.

The AM Java Admin APIs are part of the AM Java Runtime so this must be installed and configured before the migration tool can be used.

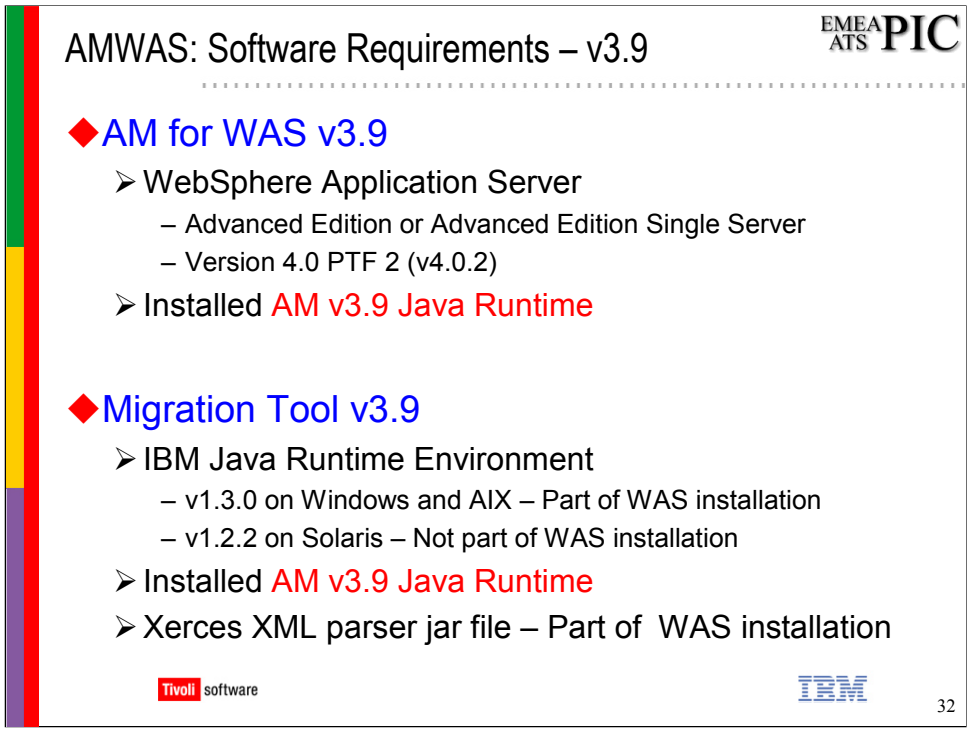
The Migration tool does not have to be run with WebSphere – it is a standalone java application and can run in any supported JVM that has AMJRTE configured against it. However, it is easy to run it in the WebSphere JVM because this provides all of the components required and must be configured with AMJRTE anyway in order to run AMWAS.

Note: Although the migration tool does not require WebSphere to run, WebSphere v4.0.2 does provide all of the required modules for its operation.



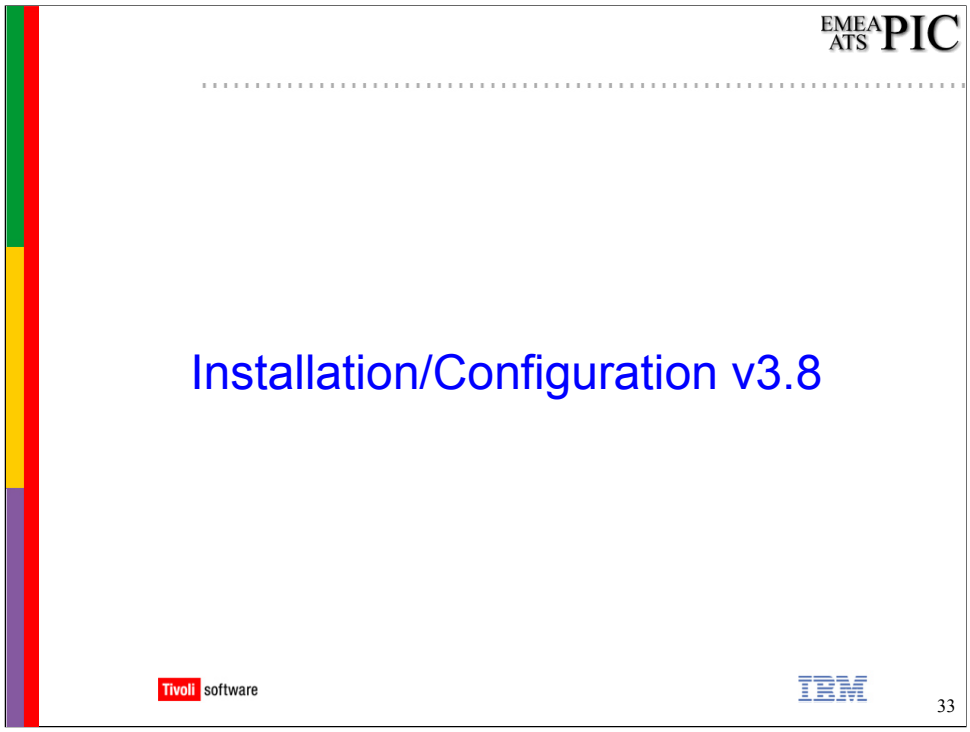
The chart above shows the pre-requisite software for the PDWAS runtime and the PDWAS migration tool.

Note that most of the JAVA pre-requisites for the migration tool are provided by WAS v4.0.2 so this may provide the easiest way to meet them. The only exception is JDK 1.2.2 that is required for Solaris. This must be obtained from some other source.



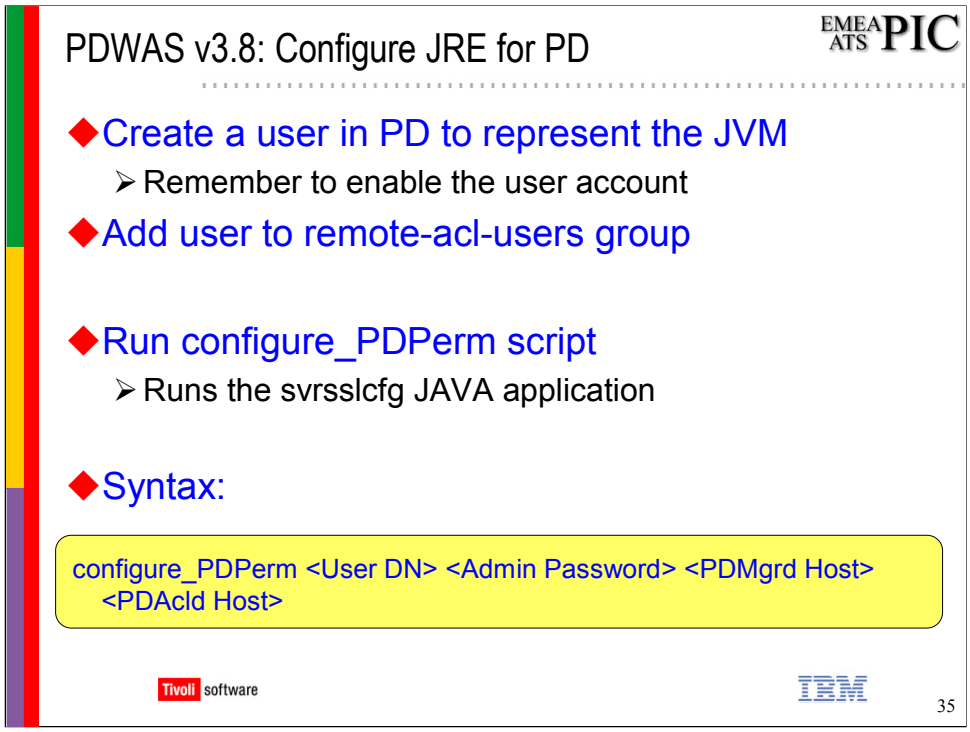
The chart above shows the pre-requisite software for the AMWAS runtime and the AMWAS migration tool.

Note that most of the JAVA pre-requisites for the migration tool are provided by WAS v4.0.2 so this may provide the easiest way to meet them. The only exception is JDK 1.2.2 that is required for Solaris. This must be obtained from some other source.


EMEA ATS PIC
Installation/Configuration v3.8
Tivoli software
IBM
33

The following slides provide a brief summary of the installation and configuration of PDWAS and the migration tool. For details please reference the product documentation.

The Install_PDPerm script is found in the directory where the PDWAS installation package is extracted. It uses the WAS_HOME environment variable to copy the PDWAS files into the correct location. It also copies its own configuration script into the WebSphere bin directory where it is part of the PATH.



The process for configuring the WAS JRE for communication with Access Manager is no different from configuring any other JRE.

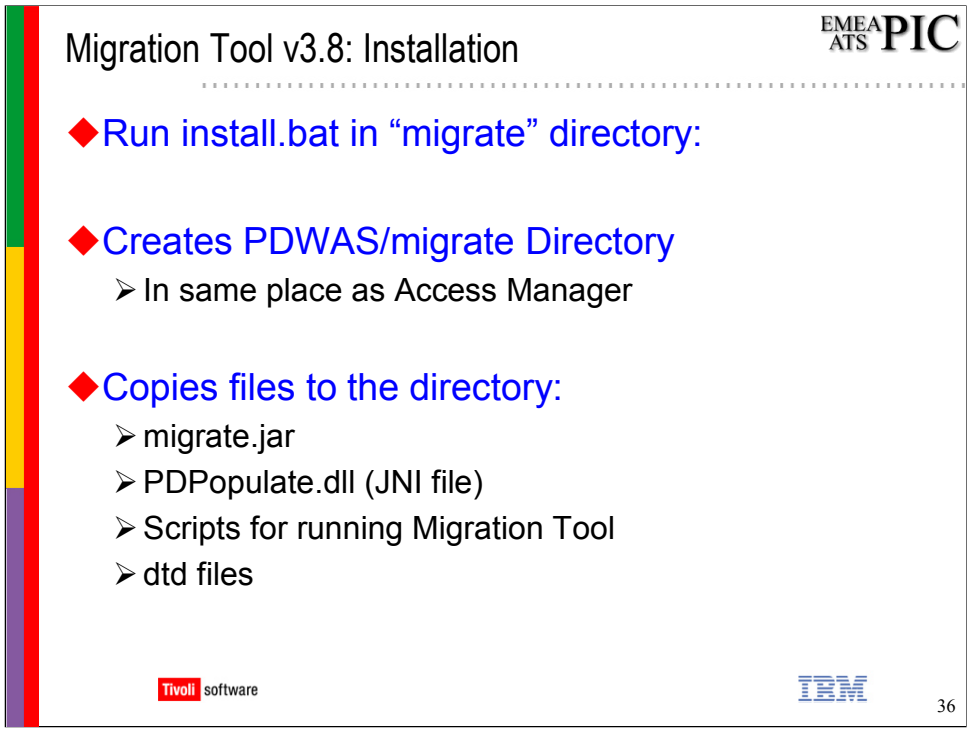
First, a user must be created in Access Manager to represent the JVM. Remember to enable the account! Since the JVM will be a remote user or PDACLD, the user must be a member of the *remote-acl-users* group.

Once the user is created run the configure_PDPerm script with the syntax shown above (the usage shown on the command line is misleading). This sets up the classpath and calls the *com.tivoli.mts.SvrSslCfg* class to register the JVM with Access Manager. If you are familiar with this process you may prefer to call this class yourself.

<UserDN> is the DN of the user you have created to represent the JVM

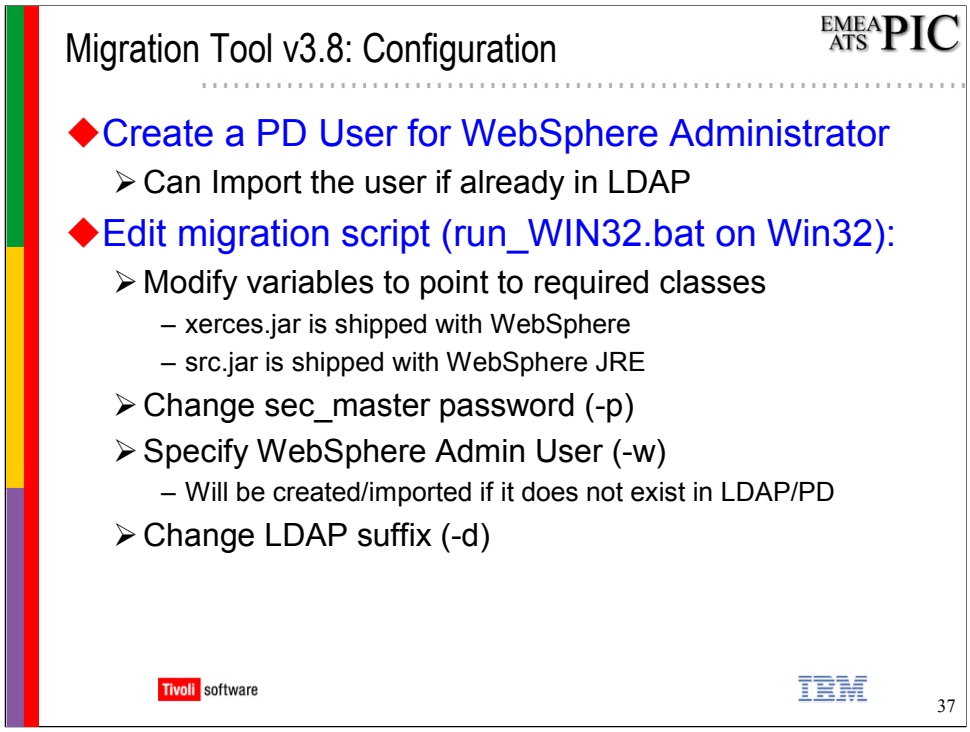
<Admin Password> is the password of sec_master

Note: The configure_PDPerm command gives no message on successful completion – only errors are reported.



To set up the migration tool, run the *install* script found in the migrate directory under where the installation package was extracted.

The script copies the migration utility files into a directory called PDWAS/migrate in the same place as Access Manager (/opt on UNIX, c:\program files\tivoli) on Windows.



Before running the migration tool for the first time a few things must be done.

First, create a user in PD that will represent the WebSphere admin user. If you have already defined an LDAP user then use the import facility of PDADMIN. Don't forget to enable the account!

Note: If you specify an admin user that does not exist when using the Migration Utility then it will cretae/import the user for you. If it is a new user you will need to set the users password using PDADMIN or WPM.

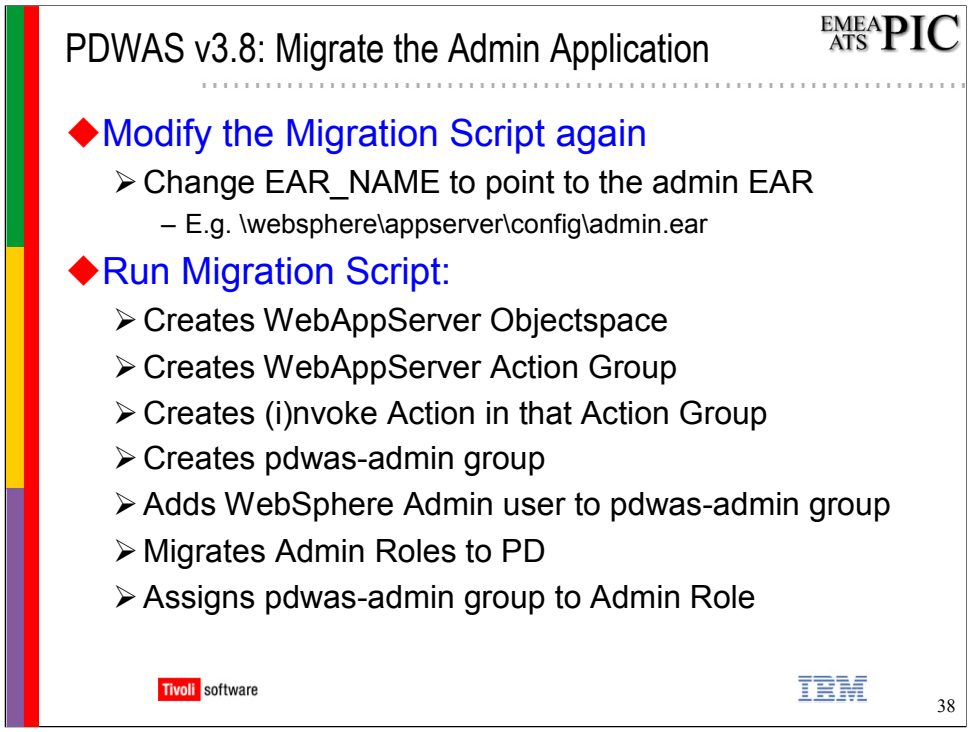
Next, the script that calls the migration JAVA class must be modified to identify the classes it needs and to provide the PD information required. (see product documentation for details).

If you have WebSphere v4.0.2 installed on the same machine then you can use the following:

XML_PARSER_PATH= WAS_HOME/lib/xerces.jar

JDK_PATH=WAS_HOME/java/jre/lib/src.jar

JDK_DIR=WAS_HOME/java/jre



The first application that **must** be migrated to Access Manager before it is integrated to WAS is the administrative console.

Edit the migration script and change the EAR_NAME variable to point to the admin.ear file/directory. This is found in WAS_HOME/config

Now run the migration script. Assuming that it completes with no errors you are now ready to activate PD authorization in WebSphere. If you want to be sure all is ready you can check the PD objectspace and ACLs.

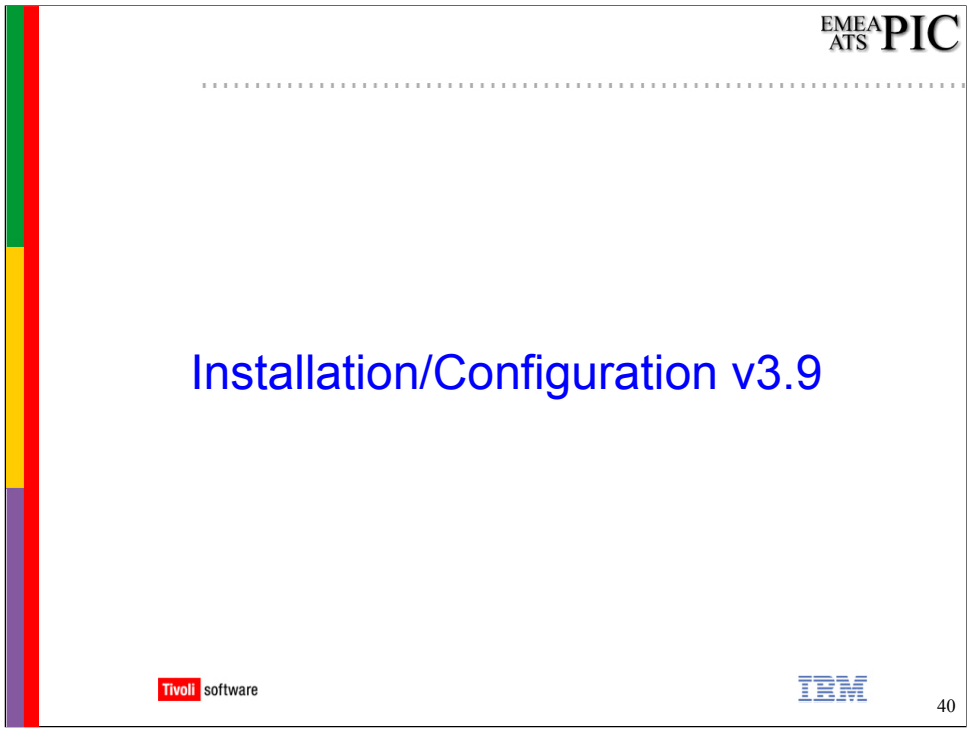
Note: To migrate other applications simply change the EAR_NAME variable and run the migration script again. The variable can either point to an EAR file or to a directory containing the contents of an expanded EAR file.

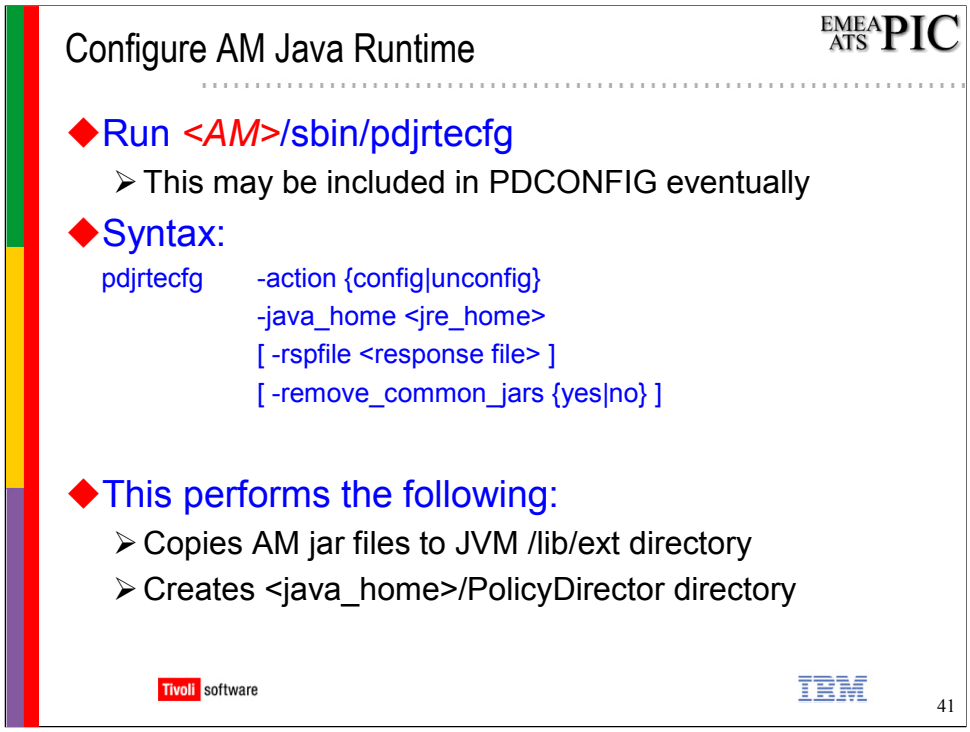
Known Problem: At the time of writing there can be a problem with the ACL attached to the **\WebAppServer\deployedResouces\AdminRole\admin** object. You may need to manually attach the **_WebAppServer_deployedResources_AdminRole_admin** ACL to replace the **_WebAppServer_deployedResources_AdminRole_admin_ACL** ACL. Migrating another application will also fix the problem.

In order to instruct WAS to call PD for role-based authorization decisions edit the sas.server.props file and the sas.server.props.future file (if it exists) adding the line shown above. It is good practice to make a backup copy of the files before making changes.

Restart the WebSphere node to activate the new configuration.

Note: With this line in place WebSphere authorizes itself using Access Manager. If your configuration is not correct then the WAS adminserver will probably fail to start. If this happens use the log files to determine the problem (discussed later). To bring WAS up again without PD integration back out the changes made to the properties files.

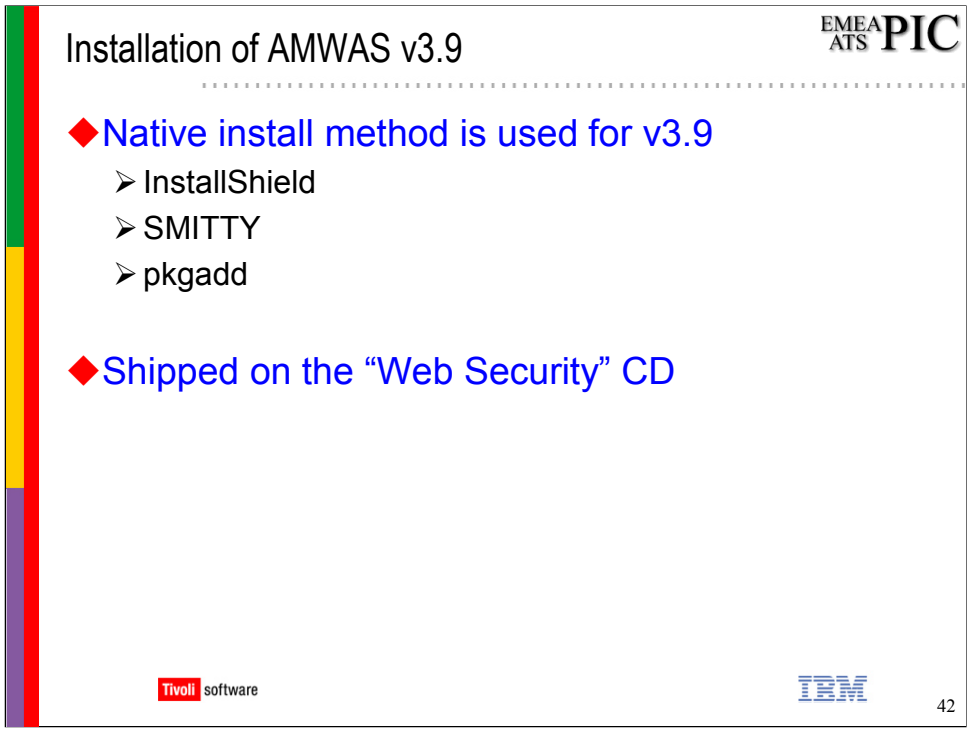

EMEA ATS PIC
Installation/Configuration v3.9
Tivoli software
IBM
40



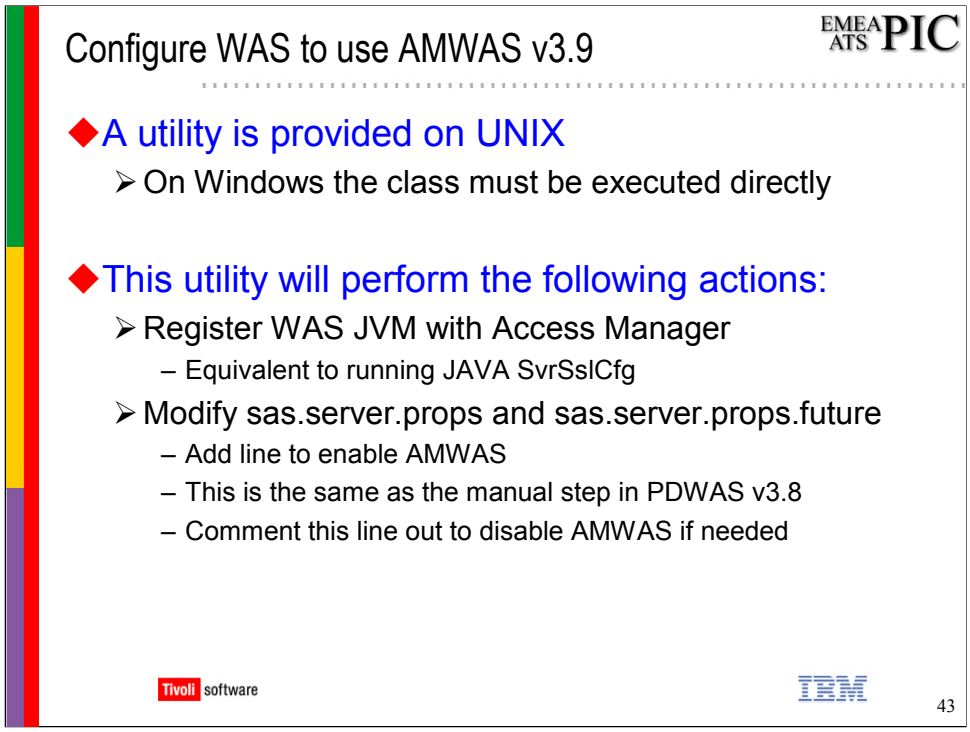
The new Java Runtime component in AM v3.9 includes a command-line configuration utility that should be used to configure a JVM to use Access Manager.

This copies JAR files over for PDPermission classes, AM ADMIN Java API, AM JLog extensions and files required for using Access Manager with JAAS.

The *remove_common_jars* option allows the caller to specify if Access Manager should overwrite JAR files that it finds in the /lib/ext directory during configuration or delete these files during unconfigure.



Installation packages are provided for all platforms for AMWAS v3.9.



The class can be called manually using the following syntax:

java -Dpdwas.home=<PDWAS Home> -Dwas.home=<WAS HOME> PDWAScfg -action {unconfig | config} -remote_acl_user <Remote ACL user> -secMaster_pwd <password> -pdmgrd_host <PDMGRD Hostname> -pdacld_host <PDACLD Hostname> [-pdmgrd_port <PDMGRD Port>] [-pdacld_port <PDACLD Port>]

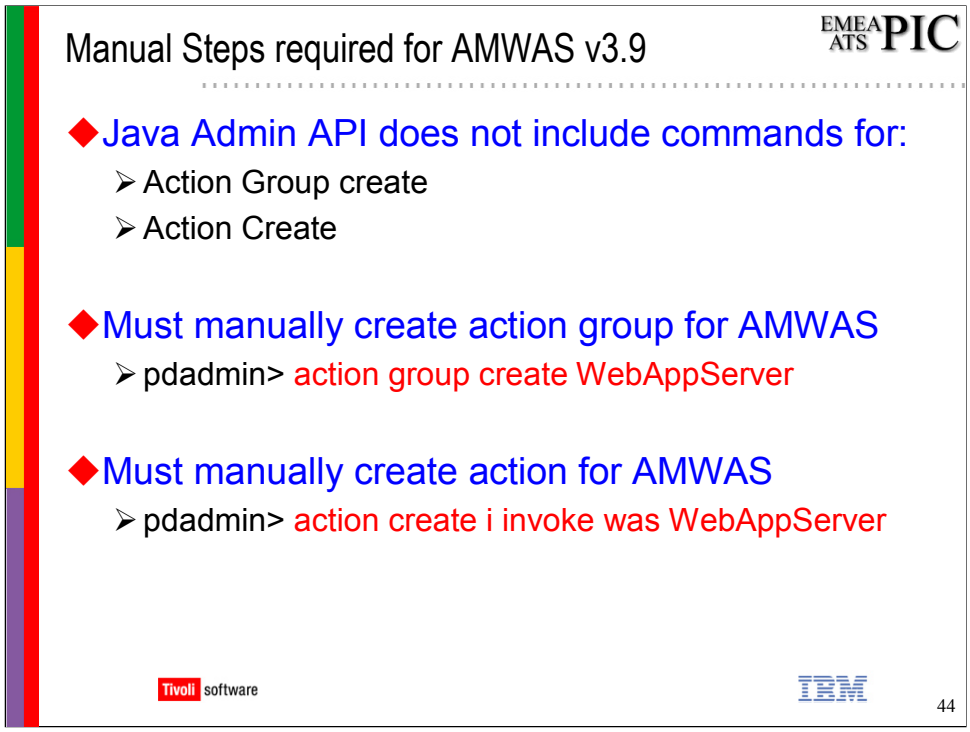
This requires that AMJRTE component be installed and configured and that the classpath contain:

<PDWAS>/sbin

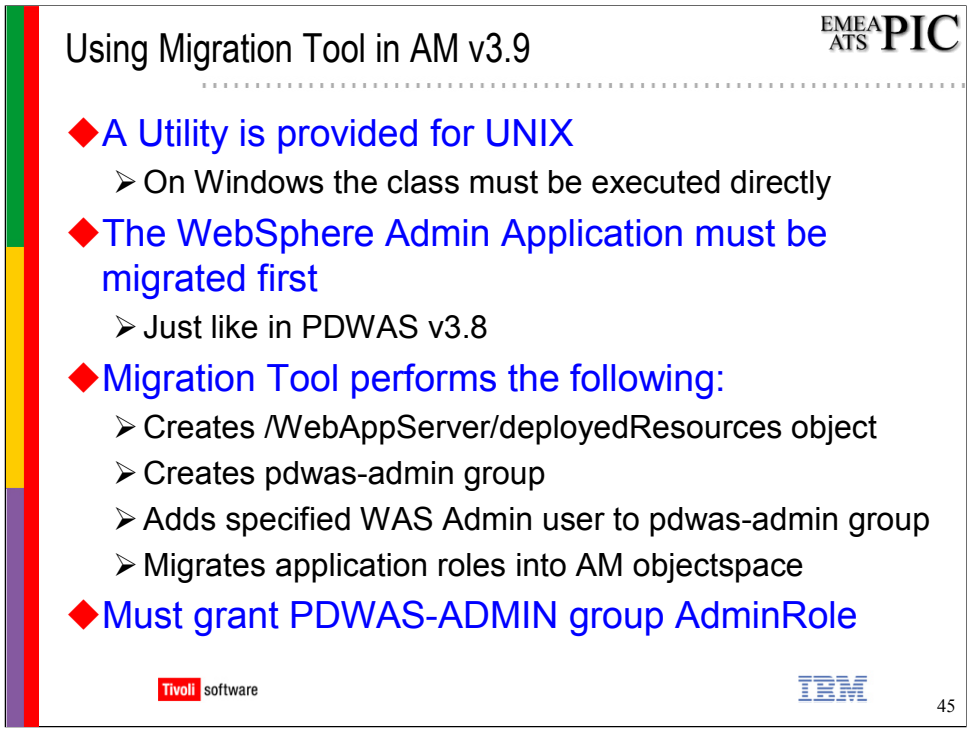
<PDWAS>/PDWASAuthzManager.jar

<PDWAS>/lib

It is important that the java.exe used is the one that has been configured for AM Java Runtime.



The Pure Java version of the AM ADMIN API does not include functions that allow the creation of action groups and actions. As a result two manual steps are required before AMWAS v3.9 can be used.



The class can be called manually using the following syntax:

```
java -Dpdwas.lang.home=%WAS_HOME%\lib;%PDWAS_HOME%\nl
s\java com.tivoli.pdwas.migrate.Migrate -j <EAR to
Migrate> -a <Sec Master User> -p <Sec Master
password> -w <WAS admin user> -d <userdomain/suffix>
-c <URL to PDPerm.properties file>
```

For LDAP and AD, -d should be the directory location where users should be created (e.g. o=ibm,c=gb). For Domino give the domain (e.g. pic)

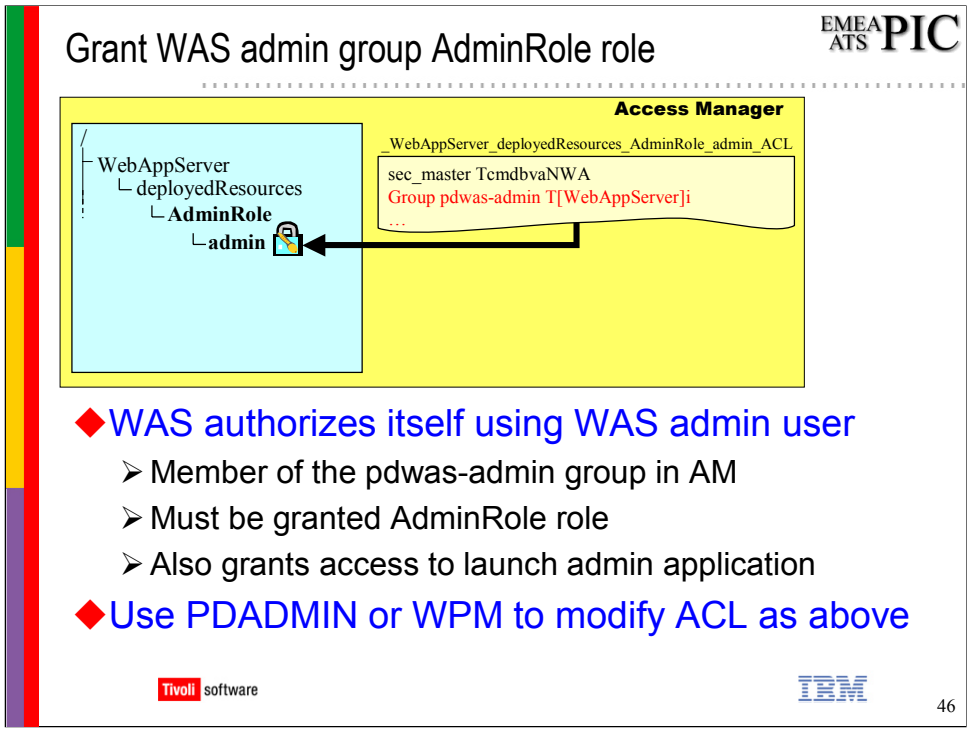
This requires that AMJRTE component be installed and configured and that the classpath contain:

<PSWAS>/lib/migrate.jar

xerces.jar (XERCES parser – can be found in <WAS>/lib)

It is important that the java.exe used is the one that has been configured for AM Java Runtime.

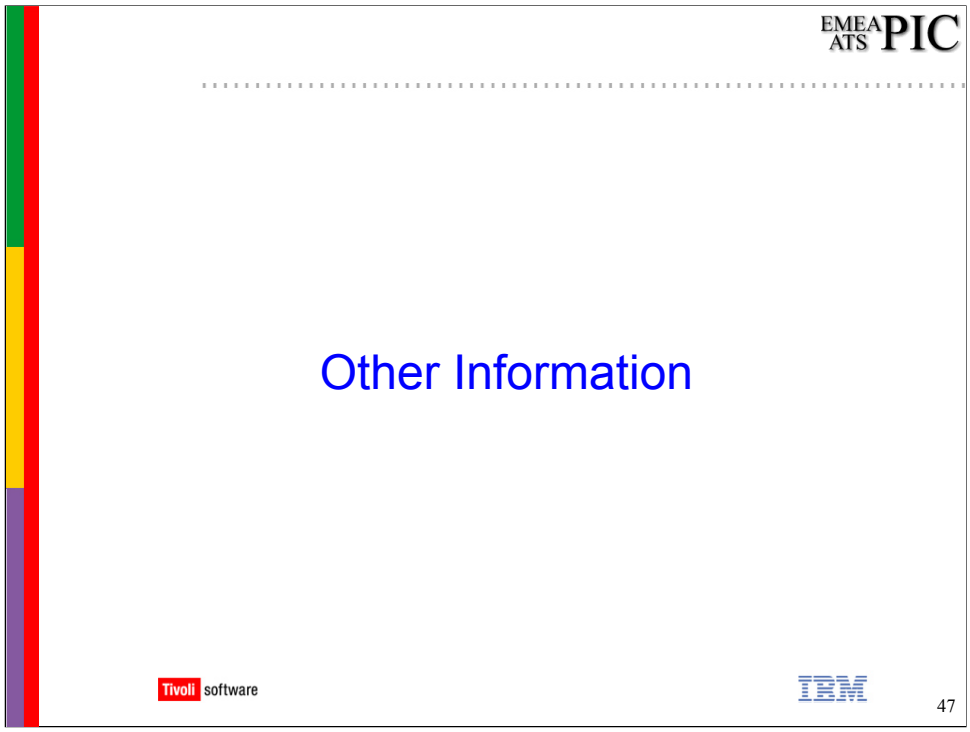
The migration tool does NOT add the pdwas-admin group to the ACL attached to that AdminRole role. This MUST be done before AMWAS can be used.

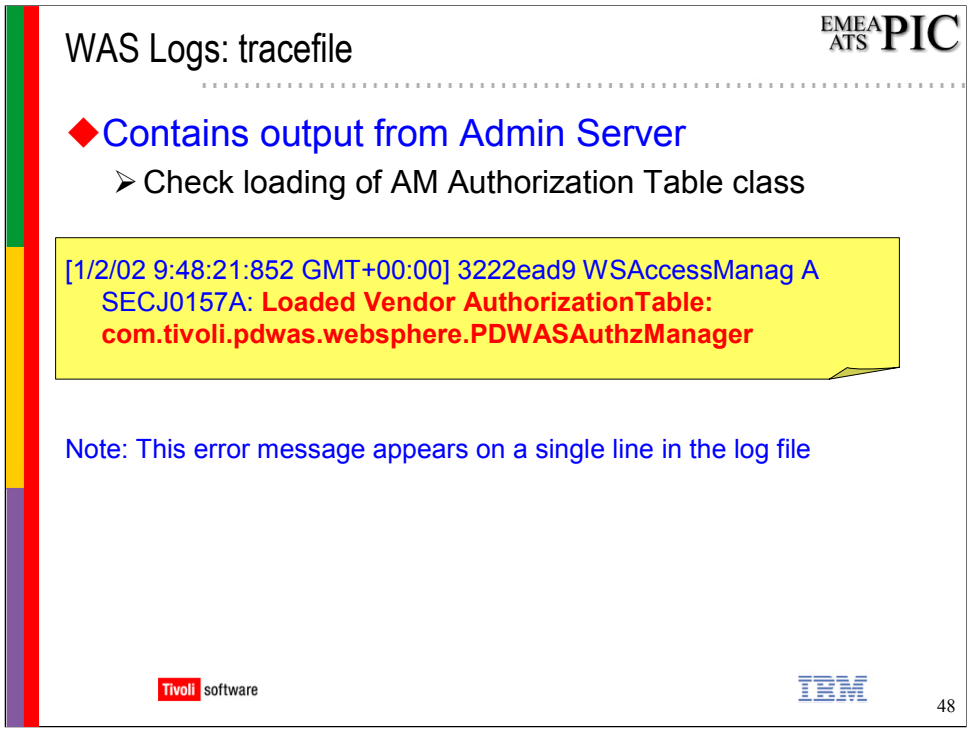


When WebSphere is running it authorizes itself based on the access granted to the "Security Server ID" specified in the WAS security configuration. This user (that is specified during AMWAS configuration and therefore added to the pdwas-admin group) requires the AdminRole role to be granted otherwise WAS cannot start.

In order to grant this required role, use PDADMIN or WPM to modify the ACL attached to the AdminRole role for the admin application. This ACL is called _WebAppServer_deployedResources_AdminRole_admin. Grant the pdwas-admin group T (Traverse) and [WebAppServer]i (invoke) permissions.

This ACL setup means that any user that is made a member of the pdwas-admin group can use the WAS admin console - the example of using AM to manage WAS security.


EMEA ATS PIC
Other Information
Tivoli software
IBM
47

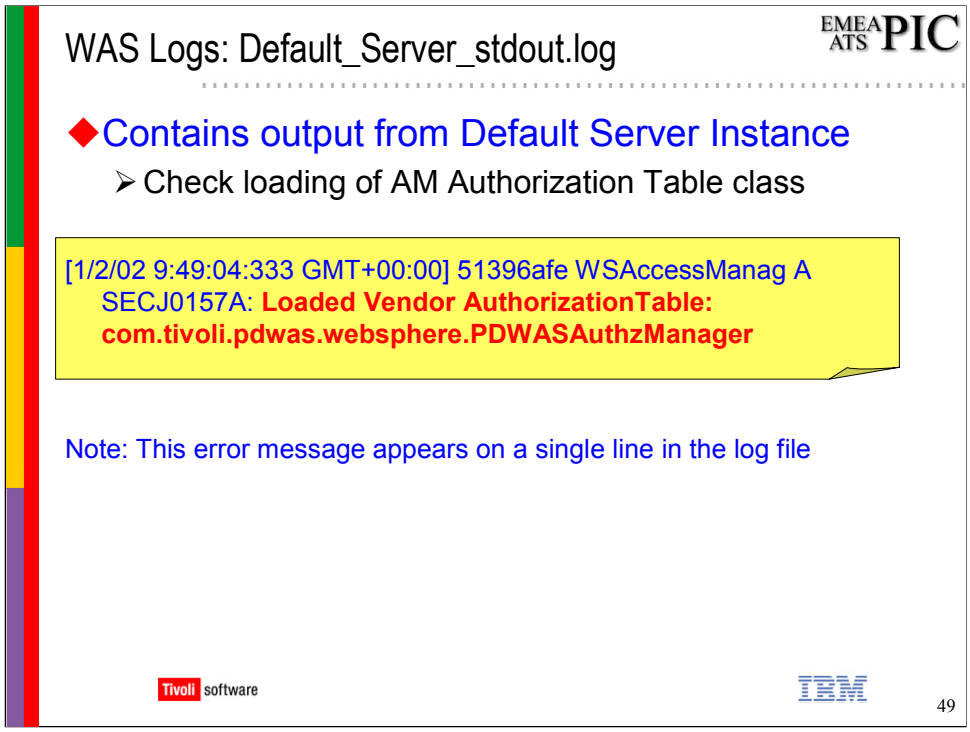


Once the sas.server.props files have been modified the following should be seen in the tracefile log file (found in WAS_HOME/logs). This file contains the output from the adminserver.

If the line is NOT seen then it means that the AM integration code is not being loaded.

The main causes of this are:

1) Typo in the sas.server.props or sas.server.props.future configuration file – they are case-sensitive !
2) PDWASAuthzManager.jar file is not in the classpath
3) There is a sas.server.props.future file but it was not modified. This can be seen because the line added to sas.server.props file vanishes at WAS startup.

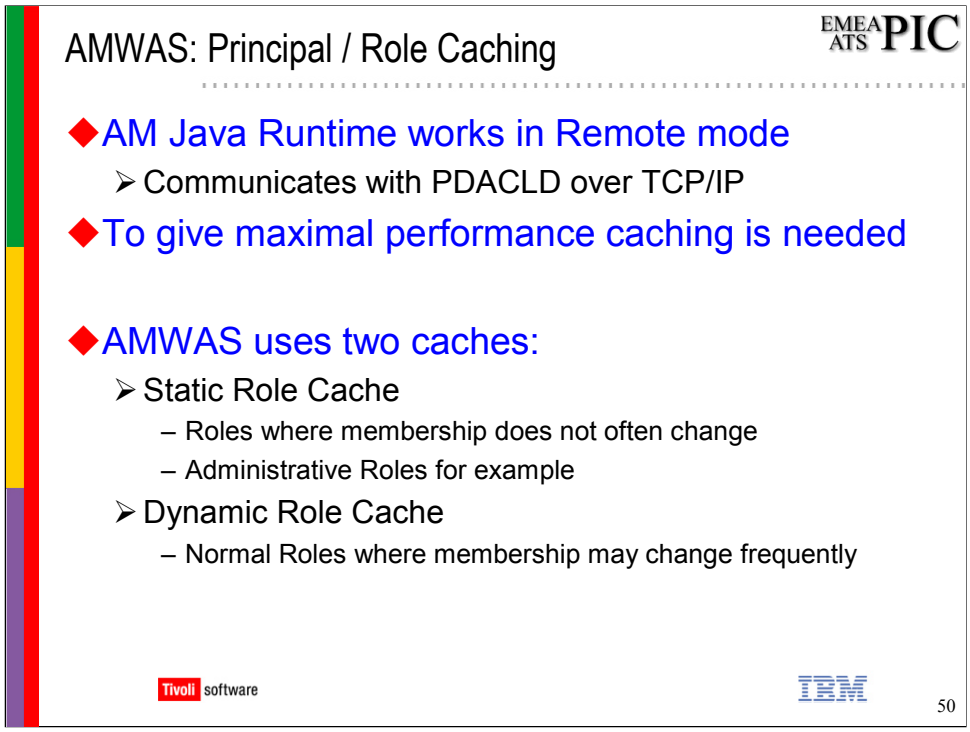


Once the sas.server.props files have been modified the following should be seen in the Default_Server_stdout.log log file (found in WAS_HOME/logs). This file contains the output from the default application server instance.

If the line is NOT seen then it means that the AM integration code is not being loaded.

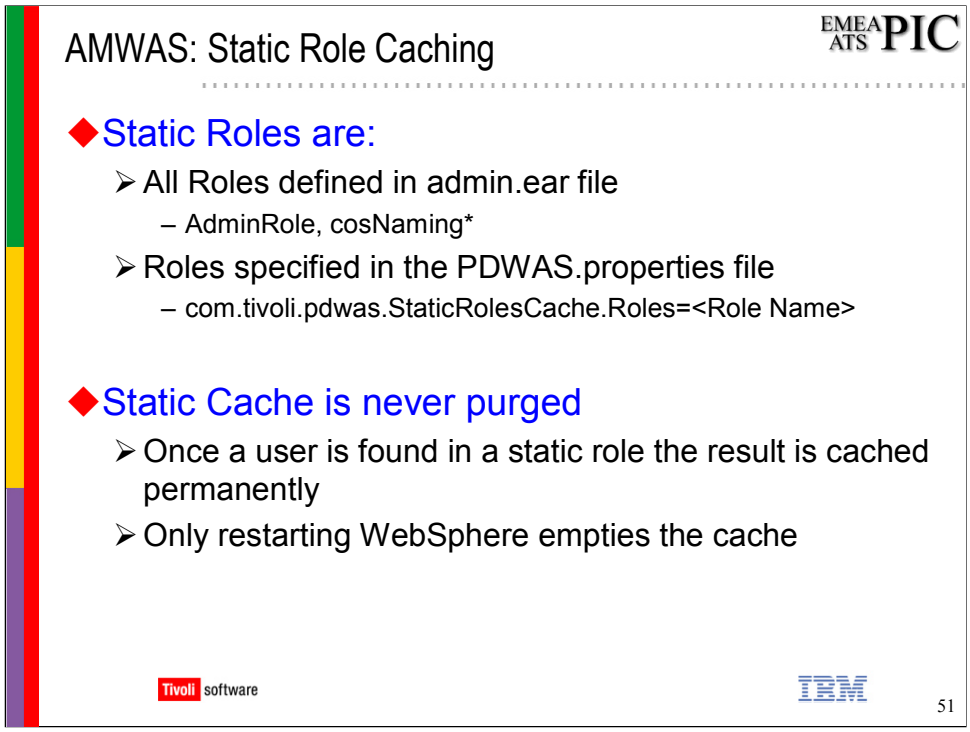
The main causes of this are:

1. Typo in the sas.server.props or sas.server.props.future configuration file – they are case-sensitive !
2. PDWASAuthzManager.jar file is not in the classpath
3. There is a sas.server.props.future file but it was not modified. This can be seen because the line added to sas.server.props file vanishes at WAS startup.



As with all Access Manager JAVA authorization the AMWAS java classes operate as a “remote mode” application to Access Manager. This means that authorization decisions are made by a TCP/IP call to PDACLD.

In order to get maximum performance in this environment caching is required. AMWAS implements two levels of caching which are discussed in the following pages.

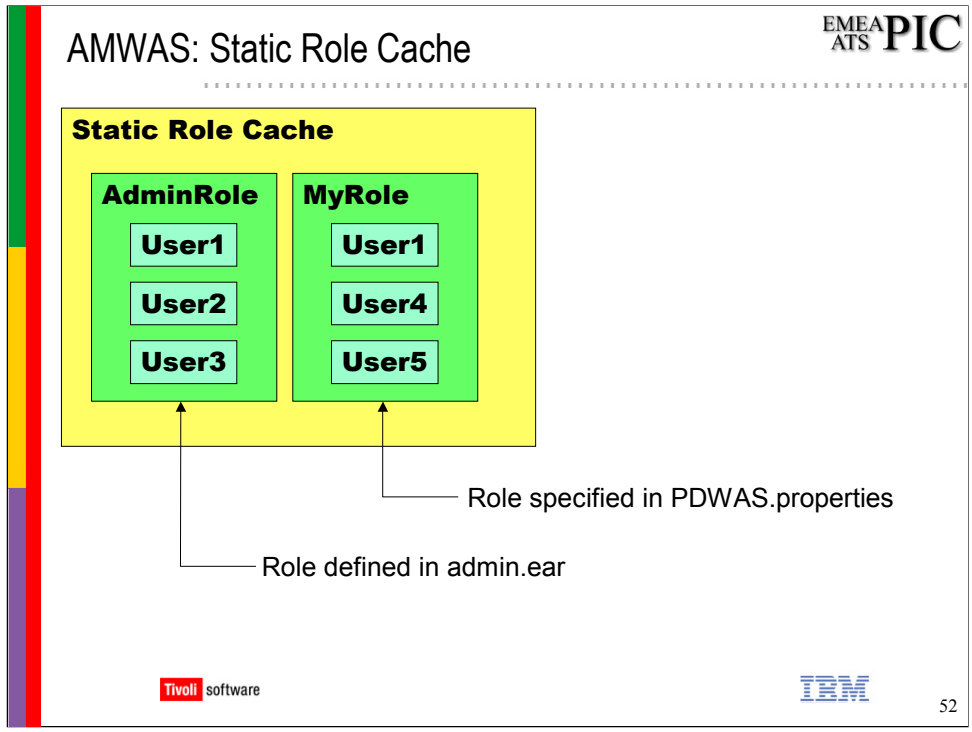


The static cache contains information on the members of the WAS administrative roles. The speed at which administrative roles can be processed is very important because WebSphere itself is authorized (in order to load classes for example) against these roles.

The Admin roles are determined by their inclusion in the *application.xml* file in admin.ear. This file is read on AMWAS initialisation. It is also possible to add other roles to the static cache (if their membership does not change) by adding lines to the PDWAS.properties file (as shown above).

To make the static cache as fast as possible it is not purged during runtime. This has the effect that it is not possible to remove a user from the administration role without restarting the WAS server. Experience indicates that removing users from the admin role is a rare occurrence.

Note: If a user is added to the administration role then this will be immediately effective.



The diagram above shows the structure of the static cache.

The cache is indexed by role name and returns a list of the users in the role.

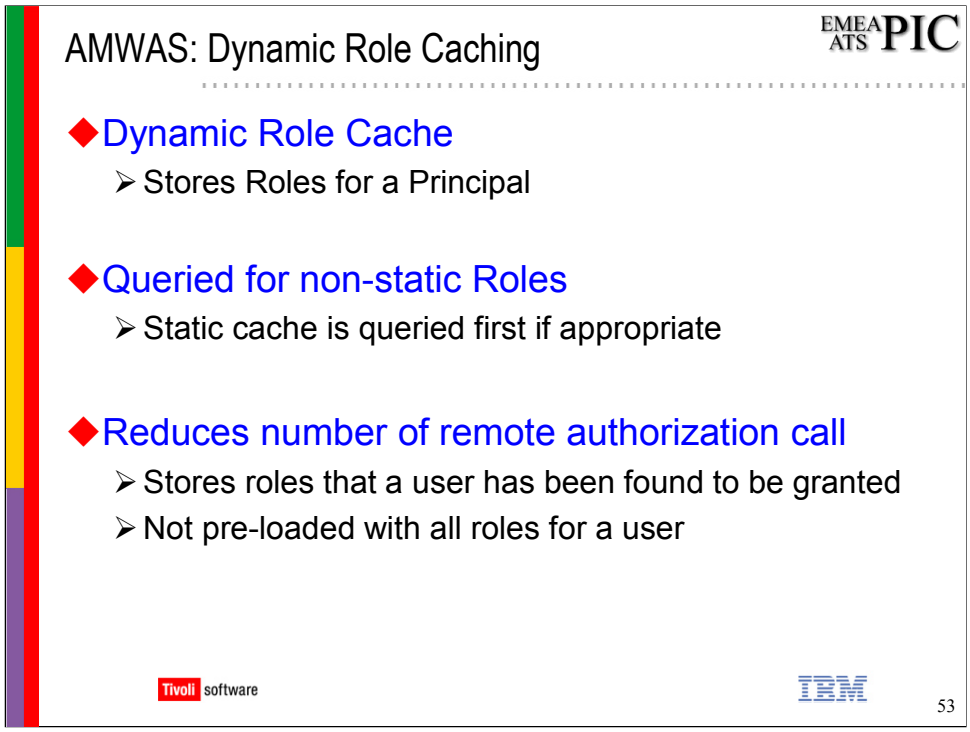
Each time a user is identified (through PDPermission calls) as being a member of a static role their name is added to the membership list for the role.

The static cache is checked first for any static role in the list of permitted roles.

The static cache can be disabled by adding the following line to the PDWAS.properties file:

com.tivoli.pdwas.EnableStaticRoleCaching=[true|false]

Please note that this may have a significant performance impact.

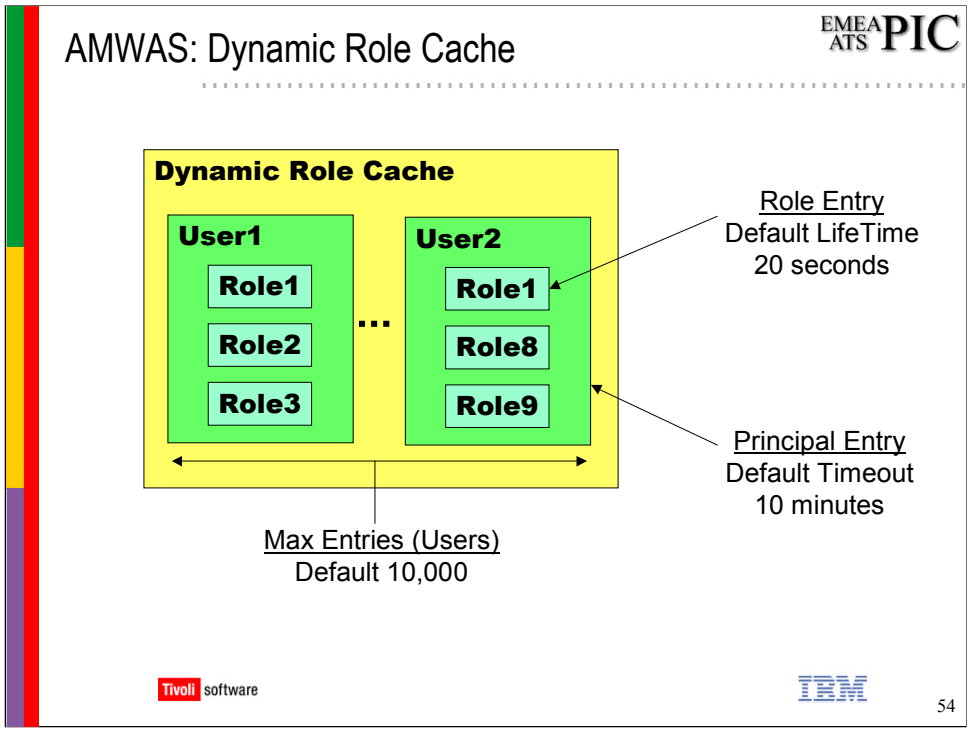


The dynamic role cache is used to store the roles that have been identified as granted to a particular user (principal). This cache is used to store information about all non-static roles and is simply intended to reduce the number of requests that have to be made "over the wire" to PDACLD.

Whenever a PDPermission check is done to determine if a user is granted a non-static role the result is stored under the users entry in the dynamic cache.

Each time the user requests an object the list of permitted roles for the object can be compared to the list of roles currently in the users dynamic cache entry. If there is any match then the user can be granted access and no further processing is required. If there is no match then only the permitted roles that are not in the cache need to be checked.

Note: The fact that a user is **not** granted a role is **not** cached – this would make for a big cache! (Actually this "not granted" information *is* stored for the "unauthenticated" user – for speed – but it is the only exception).



The diagram above shows the structure of the dynamic role cache. This cache is indexed by username and returns a list of roles the user has been previously found to be granted.

The dynamic cache is kept fresh by always removing role entries after a short time and also by removing entire user entries if they are not used. As such the dynamic cache is different from the static cache – a users membership of a role will be re-checked (by default) every 20 seconds.

Properties of the dynamic cache can be changed by modifying the PDWAS.properties file (see product documentation for details):

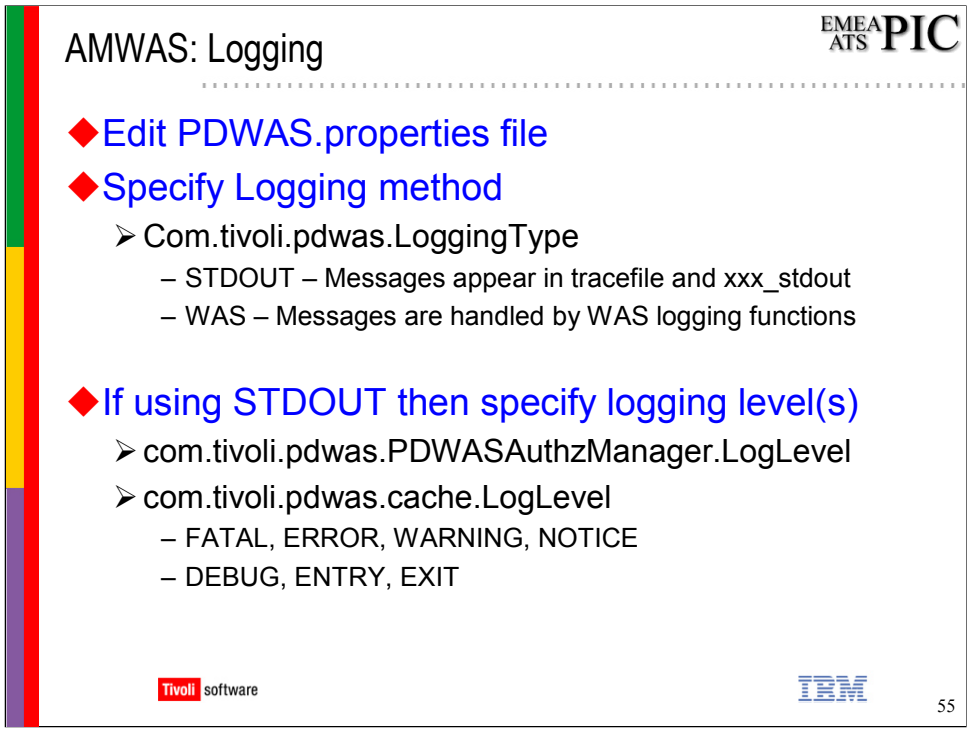
com.tivoli.pdwas.EnableDynamicRoleCaching=[true|false]

com.tivoli.pdwas.DynamicRoleCache.MaxUsers=<Number of Entries>

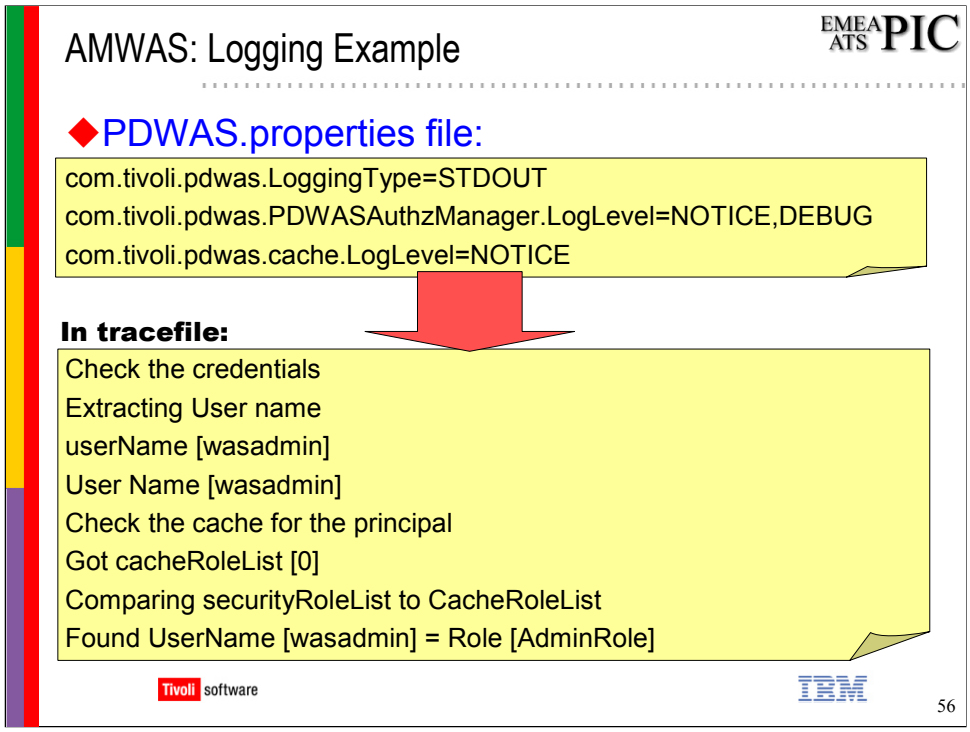
com.tivoli.pdwas.DynamicRoleCache.PrincipalLifeTime=<Number of minutes>

com.tivoli.pdwas.DynamicRoleCache.RoleLifeTime=<Number of seconds>

com.tivoli.pdwas.DynamicRoleCache.NumBuckets=<Concurrent threads>

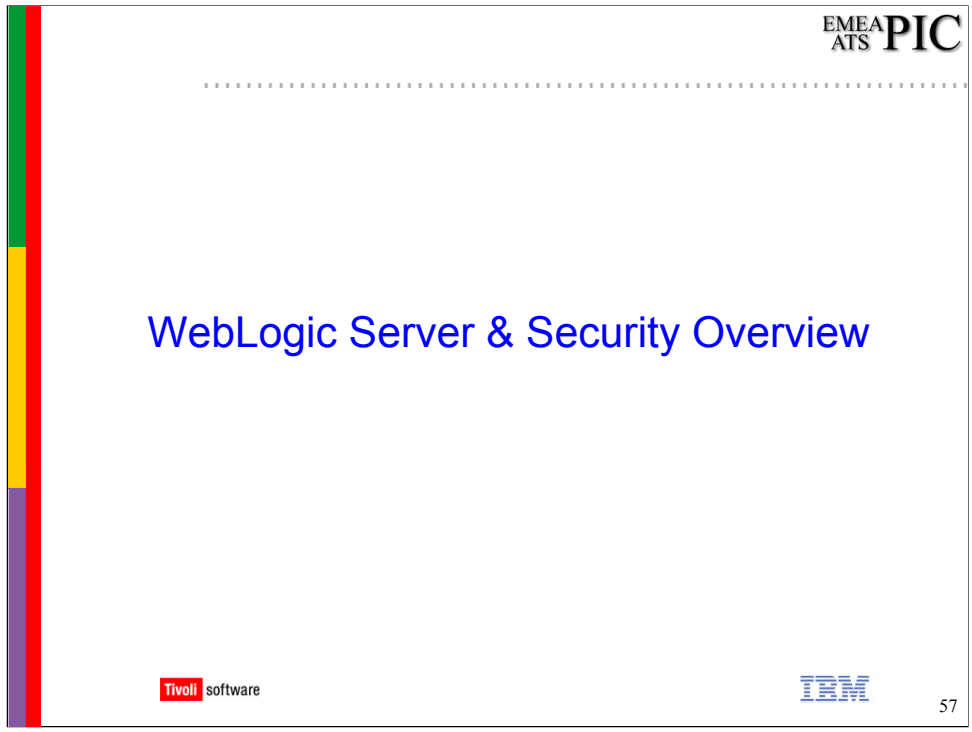


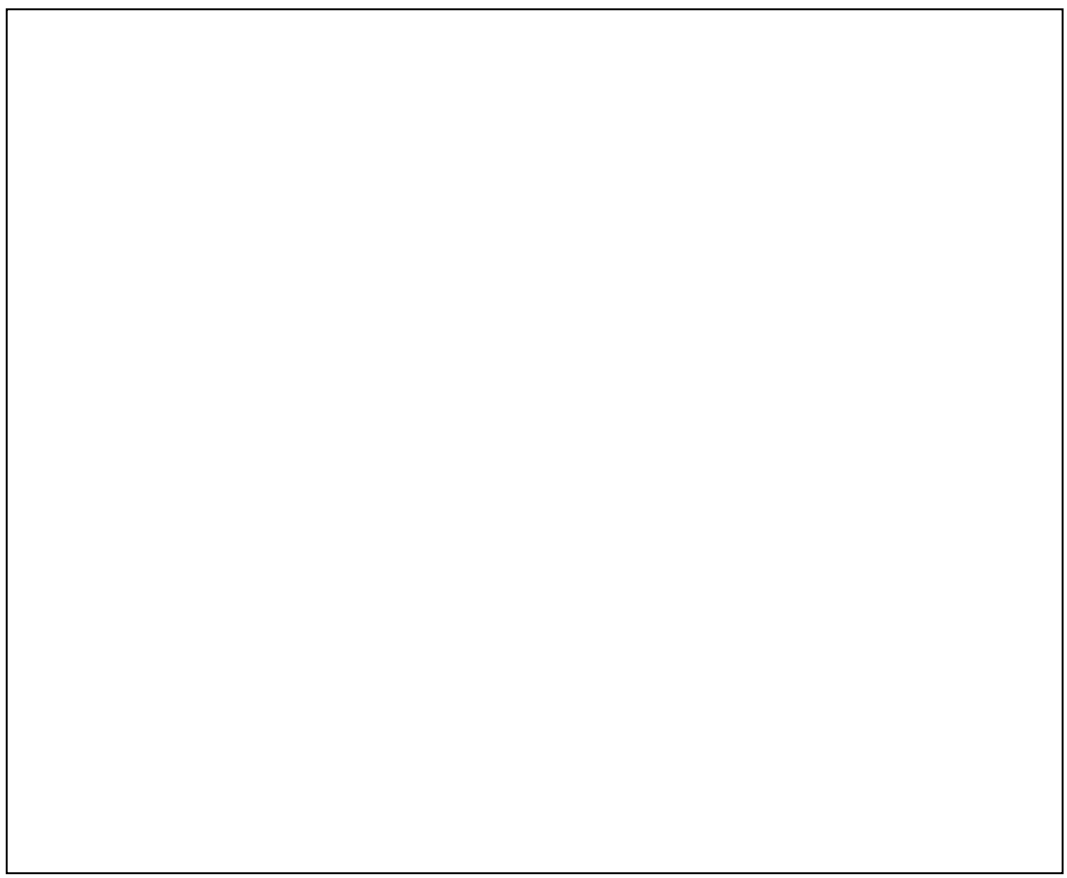
AMWAS can be instructed to output debug messages for troubleshooting. This is specified in the PDWAS.properties file.

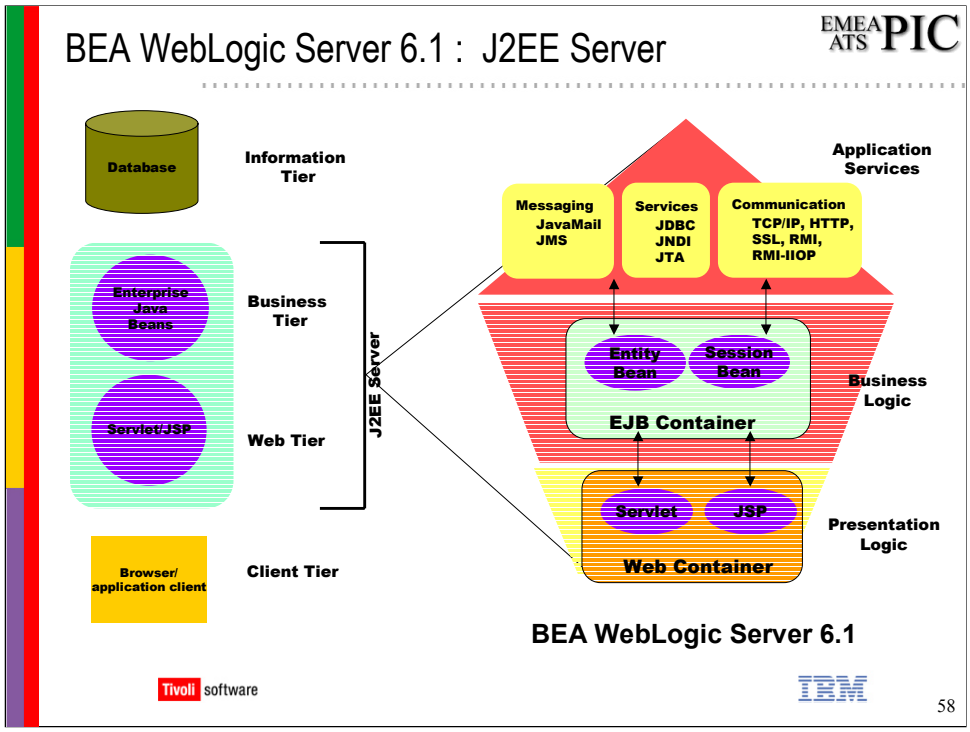


The example above shows a logging configuration in the PDWAS.properties file and the resulting output in the WAS adminserver “tracefile”.

This log **does not** show the individual permissions that are being checked in the Access Manager ACL database. If you need to see this then turn on auditing for the PDACLD server. This must be done in pdacld.conf and in an appropriately placed Protected Object Policy (POP).


EMEA ATS PIC
WebLogic Server & Security Overview
Tivoli software
IBM
57

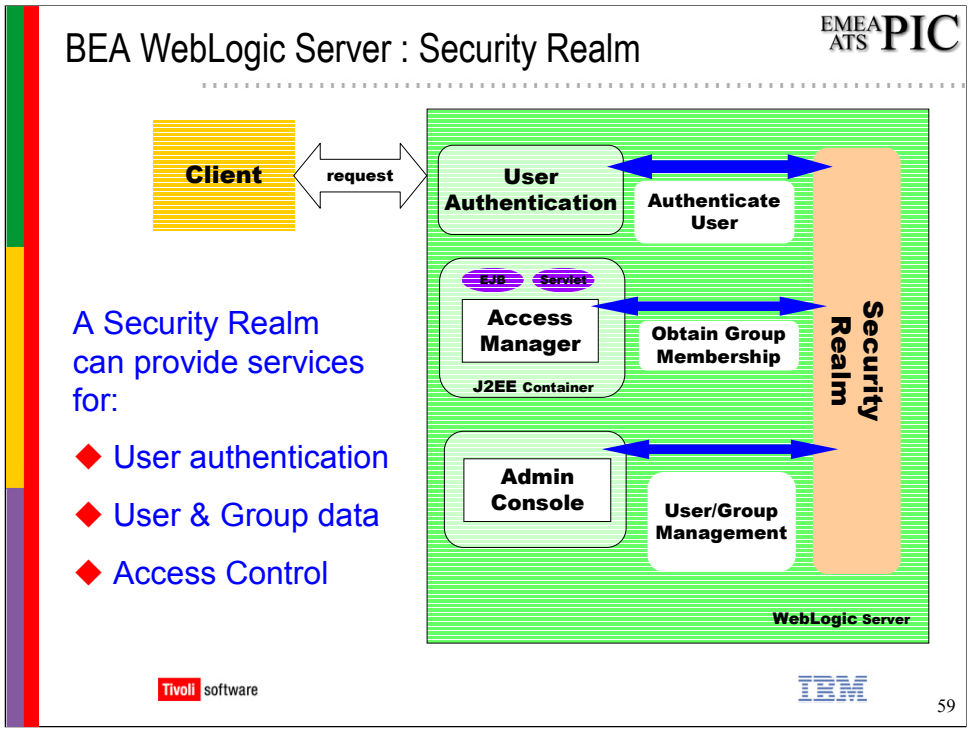




WebLogic Server 6.1 contains Java 2 Platform, Enterprise Edition (J2EE) technologies. J2EE is the standard platform for developing multitier enterprise applications based on the Java programming language. The technologies that make up J2EE were developed collaboratively by Sun Microsystems and other software vendors.

J2EE applications are based on standardized, modular components. WebLogic Server provides a complete set of services for those components and handles many details of application behaviour automatically, without requiring programming.

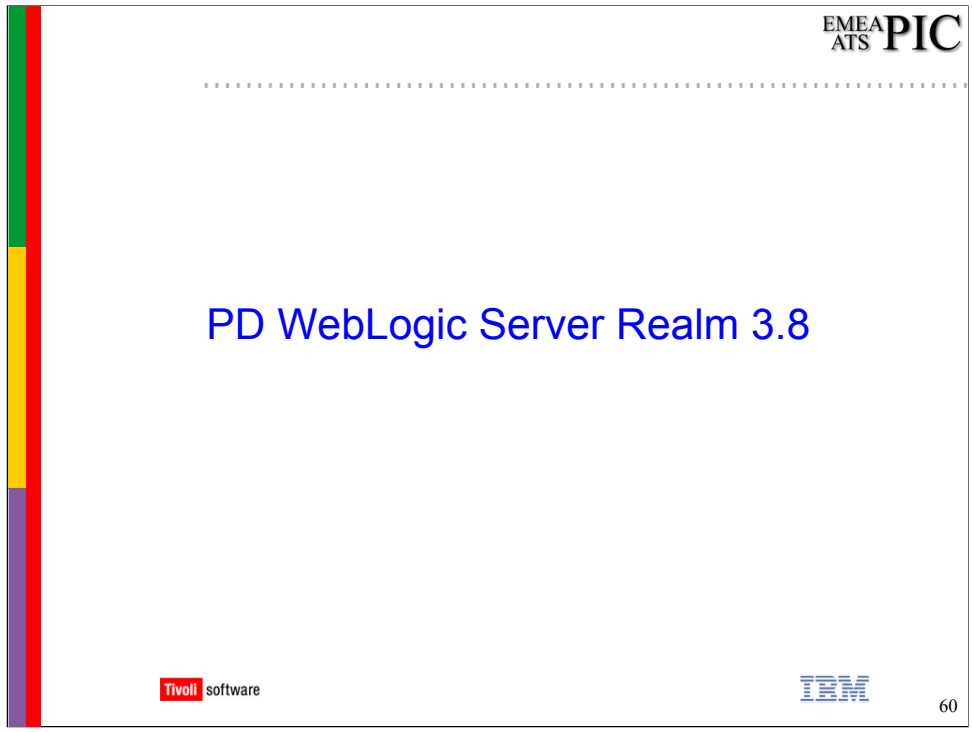
WebLogic Server 6.1 with J2EE 1.2 Plus Additional J2EE 1.3 Features provides EJB 2.0, JSP 1.2, Servlet 2.3, and J2EE Connector Architecture 1.0.



A WLS Security Realm represents a logical grouping of Users, Groups, ACLs, (not implemented by PD Realm) and permissions for the purpose of protecting WebLogic Server resources.

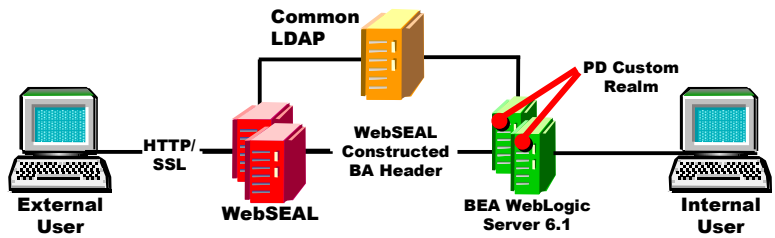
WLS 6.1 provides a default File Realm and these Alternative Realms:
•LDAP Security Realm (supports Open LDAP, Netscape iPlanet, Microsoft Site Server, and Novell NDS.)
•Windows NT Security Realm
•UNIX Security Realm
•RDBMS Security Realm (sample)
•Custom Realm – this is what is provided by Access Manager 3.8 for WebLogic Server 6.1.

Note: The WLS Realm interface allows a Custom Realm to implement all or a sub-set of the Realm function; the interface includes complete access control capability that could be used as a replacement for J2EE role-based authorization. The functions shown inside the 'WebLogic Server' above are those provided by the PD 3.8 Realm for WLS.


EMEA ATS PIC
PD WebLogic Server Realm 3.8
Tivoli software
IBM
60

PD WLS Realm : Overview

EMEA ATS PIC



- Single Signon with WebSEAL
- Provides use of SecureWay LDAP or other Access Manager User Registry for WLS
- User/Group Administration from WLS Console or PD WPM
- User-to-group assignments control J2EE authorization decisions of WLS

Tivoli software

IBM

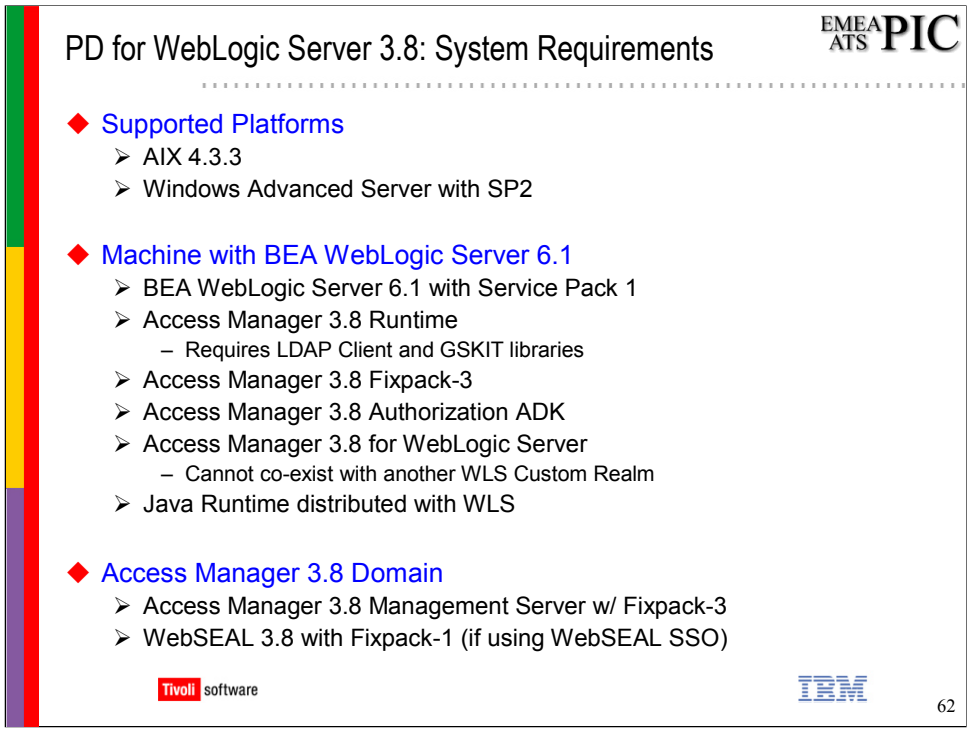
61

More detail on WebSEAL SSO is given in another slide.

In regard to the LDAP registry, it is worth noting that WLS does not provide a Directory Server – the customer must purchase from another vendor. In many cases, especially for Solaris customers, this means iPlanet which has a per-entry charge. By purchasing Access Manager 3.8 you acquire SecureWay directory that can be used with BEA WLS 'for free'.

The WLS Console does not provide as much control of user/group definition as Access Manager. Many customers may prefer using PD WPM, pdadmin> command line or PD Admin API for management of Users/Groups in the registry.

When using PD WLS configured for WebSEAL SSO, it is important to **not allow** internal users to have direct access to WLS. The diagram shows 'internal users' connected to WLS, but this access should not be allowed if configured for WebSEAL SSO.



The PD 3.8 WLS Custom Realm installation package is available as a software download from:
http://www.tivoli.com/secure/support/downloads/secureway/policy_dir/downloads.html

A valid login and password is required to access this site. IBM Internal users can access:

http://www-internal.tivoli.com/secure/support/downloads/secureway/policy_dir/downloads.html

AM for WebLogic Server 3.9: System Requirements

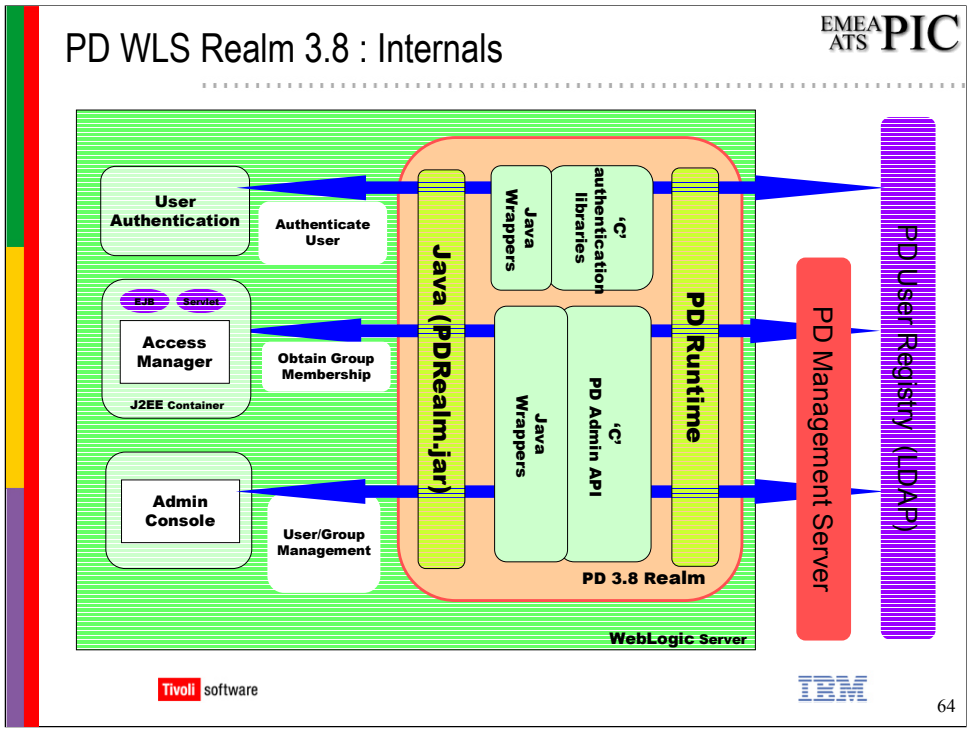
EMEA ATS PIC

- Supported Platforms
  - IBM AIX 4.3.3 and 5.1.0
  - Sun Solaris 7 and 8
  - HP-UX 11.0
  - WinNT 4.0 with SP 6a
  - Win2K Advanced Servers with SP 2
  - Red Hat Linux 7.1 (Intel only)
- Machine with BEA WebLogic Server
  - BEA WebLogic Server 6.1 with Service Pack 1
  - Java Runtime distributed with WLS
  - Access Manager Runtime & Java Runtime
  - AM 3.9 for WebLogic Server
    - Cannot co-exist with another WLS Custom Realm
- Access Manager 3.9 Domain
  - Policy Server
  - WebSEAL (if using WebSEAL SSO)

Tivoli software

IBM

63



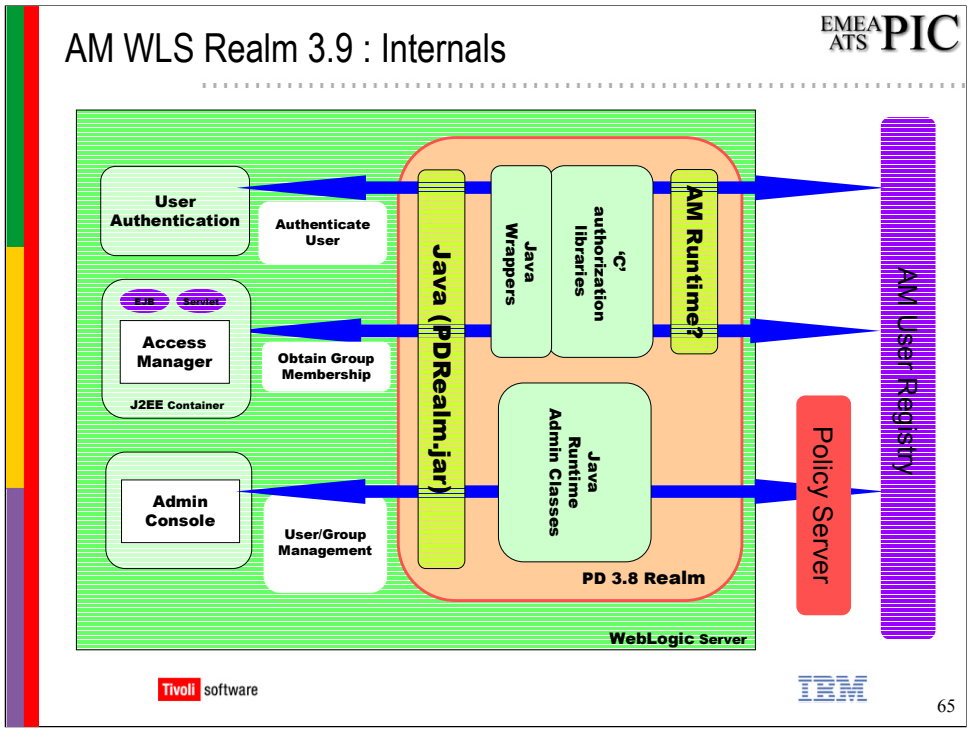
This foil shows the implementation details of PD WLS Realm 3.8.

PD 3.8 WLS Realm implements a Java interface provided by BEA WLS 6.1. (For more detail see, “Writing a Custom Security Realm” with the BEA WLS 6.1 documentation at http://edocs.bea.com/wls/docs61/security/prog.html#1041025)

These are JNI classes that utilize the ‘C’ APIs and libraries provided by Access Manager for Authentication and Administration.

Use of PD 3.8 for WLS requires the configuration of a PD 3.8 application using the **svrsslcfg** utility provided by PD Base. PD 3.8 WLS does not use the authorization policy database. The svrsslcfg command, therefore, should specify ‘**remote**’ as the server_type (ie, an application that does not need a local copy of the authorization policy database.). Further, PD 3.8 Realm does not require a PD Authorization Server (PDAcld) as part of its configuration. The “authorization API” is only used for user authentication.

An **important point** is that PD WLS requires access to the PD Management Server. Unlike WebSEAL, PD for WebSphere, and PD Authorization Server, PD 3.8 WLS Realm cannot operate without access to PD Manager. This should be carefully considered in the deployment fail-over strategy.



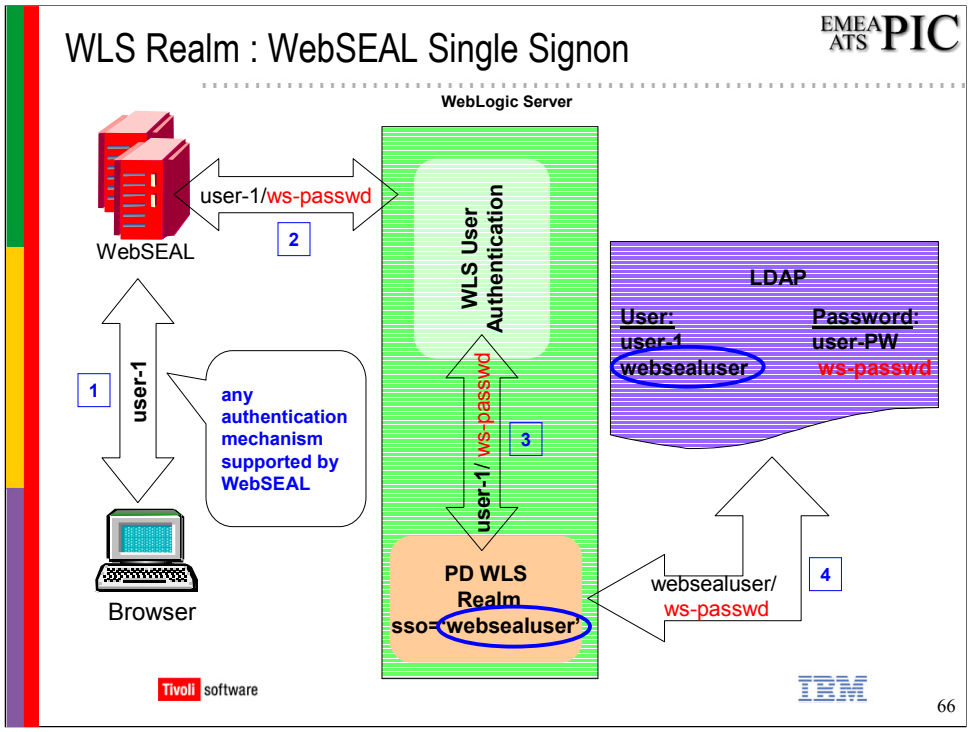
This foil shows the implementation details of AM 3.9 for WLS.

Like 3.,8, the integration with WLS is provided by implementing the java Realm interface. (For more detail see, "Writing a Custom Security Realm" with the BEA WLS 6.1 documentation at http://edocs.bea.com/wls/docs61/security/prog.html#1041025)

The WLS Realm uses Java JNI classes that utilize the 'C' APIs and libraries provided by Access Manager Authorization API. These are used for authentication and to determine a user's group membership.

Because AM 3.9 for WLS uses the authorization API (aznAPI) an server identity must be created using the **svrsslcfg** utility provided by AM Base. AM 3.9 WLS does not, however, use the authorization policy database so the **svrsslcfg** command should specify '**remote**' as the server_type (ie, an application that does not need a local copy of the authorization policy database.). Further, AM 3.9 does *not* require a PD Authorization Server (PDAcld) as part of its configuration. The "authorization API" is only used for user authentication and to determine a user's group membership.

Failover to multiple LDAP servers (as configured in *ldap.conf* of AM Base) is supported.



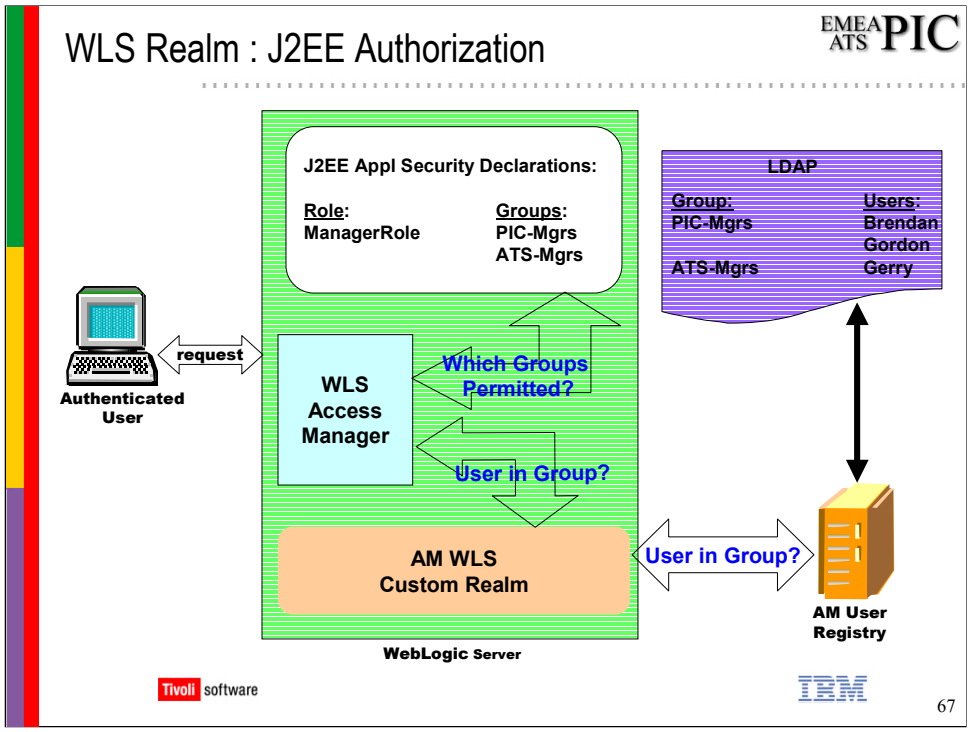
The process flow for SSO via WebSEAL is the same for both 3.8 and 3.9:

1. user authenticates to WebSEAL using any authentication mechanism supported for WebSEAL and submits a request for some WLS resource
2. WebSEAL has been configured with a junction to the WebLogic Server using the '-b supply' option. WebSEAL passes the request to WLS with a Basic Authentication header that contains:
   - the authenticated user-ID (user-1 in the diagram)
   - the value of '`basicauth-dummy-passwd`' in webseald.conf
3. WLS passes the user-Id & password to the PD Realm for authentication
4. PD Realm verifies (with the registry) that the given password is for the configured WebSEAL Single SSO User. This verification provides the Trust Relationship between WLS Realm and WebSEAL.

If (4) is successful, PD WLS authenticates the given user-id to WLS and WLS builds a credential for that user.

Note, the authentication of the webseal SSO password ('ws-passwd' in diagram) is only done **once.** After authentication the webseal-SSO password is cached in the realm (masked). Subsequent authentication requests compare the input password with the authenticated, cached value. When not equal the authentication is repeated. (this allows the webseal-SSO password to be changed without restarting WLS.)

Note: The WebSEAL Single SSO User. must be configured with 'account-valid=yes', but should have no access rights.



The logic for WLS J2EE authorization decisions is (not surprisingly) like that described in the introductory J2EE section.

The WLS Access Manager performs these actions:

1. Access the J2EE Security declarations to determine the Roles that are permitted access.
2. Access the J2EE Security declarations to determine which Principals (Users and Groups) have been mapped to those roles.
3. Access the Security Realm to determine if the current user is a member of any group that is mapped to the role.

- If the current user is mapped to a permitted Role, access is allowed.

- If the current user is a member of any group mapped to a permitted Role, access is allowed.

- Otherwise access is denied.

For example, using the diagram above, if access is allowed only for the ManagerRole, then only users Brendan, Gordon and Gerry will be given access.

Configuration

EMEA ATS PIC

- Configure AM Java Runtime to JVM of WLS (AM 3.9 only)
  - required for use of AM Java Admin classes
- PD WLS Realm requires registration as a PD application server
  - allows use of Authentication Services (PD 3.8 & AM 3.9)
  - allows use of group memebership queries (AM 3.9)
- Use svrsslcfg utility provided with PD Auth ADK
  - full details in User Guide
  - recommend creating a remote-mode application server
- PD Authorization Server (pdacld) not required
  - PD WLS Realm does not use PD authorization database

Tivoli software IBM

68

Full configuration details can be found in the User Guide provided as part of the web download.

Configuration – WLS Custom Realm

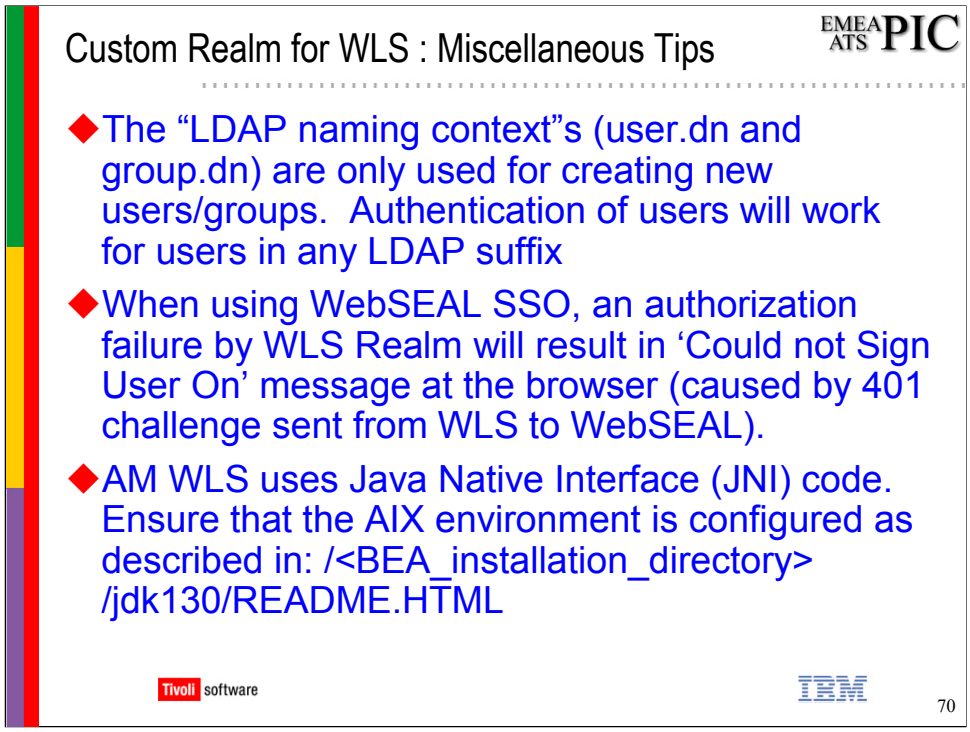
EMEA ATS PIC

- An example of Custom Realm Configuration Data:
  - aznapi.conf.file=c:\bea\PDWLSRealm\pdwlsrealm.conf
  - pdadmin.user.name=wlsadmin
    - user used with PD Admin API
  - pdadmin.password=secure99
  - group.dn=c=gb
  - user.dn=c=gb
    - these two used only for user/group create via WLS Console
  - pdrealm.tracing=true
    - files found in <wls-home>/config/<domain>/logs
  - webseal.sso.configured=true
  - connection.pool=20
  - pdrealm.registry.listing=true (not recommended for production)
  - wls.admin.user=websealuser
    - used with WebSEAL 'dummy' password to confirm WS as origin of request (I.e., basis of trust)

Tivoli software IBM

69

Full configuration details can be found in the User Guide.



These comments apply to both PD 3.8 and AM 3.9 for WLS.

AM 3.9 WLS Availability

EMEA ATS PIC

- AM 3.9 WLS is not included on CD package.
- Will be available for web download by GA data of 17th May.

Tivoli software

IBM

71


EMEA ATS PIC
Summary
Tivoli software
IBM
72

PD for WAS and WLS : Capabilities

EMEA ATS PIC

- Centralized User & Group Management in Access Manager User Registry
- Centralized Authorization Management for J2EE Applications
- Single Sign-on to Application Servers
- Migration Utility for WebSphere Applications

Tivoli software

IBM

73