IBM Software Group

IBM WebSphere® Application Server V6

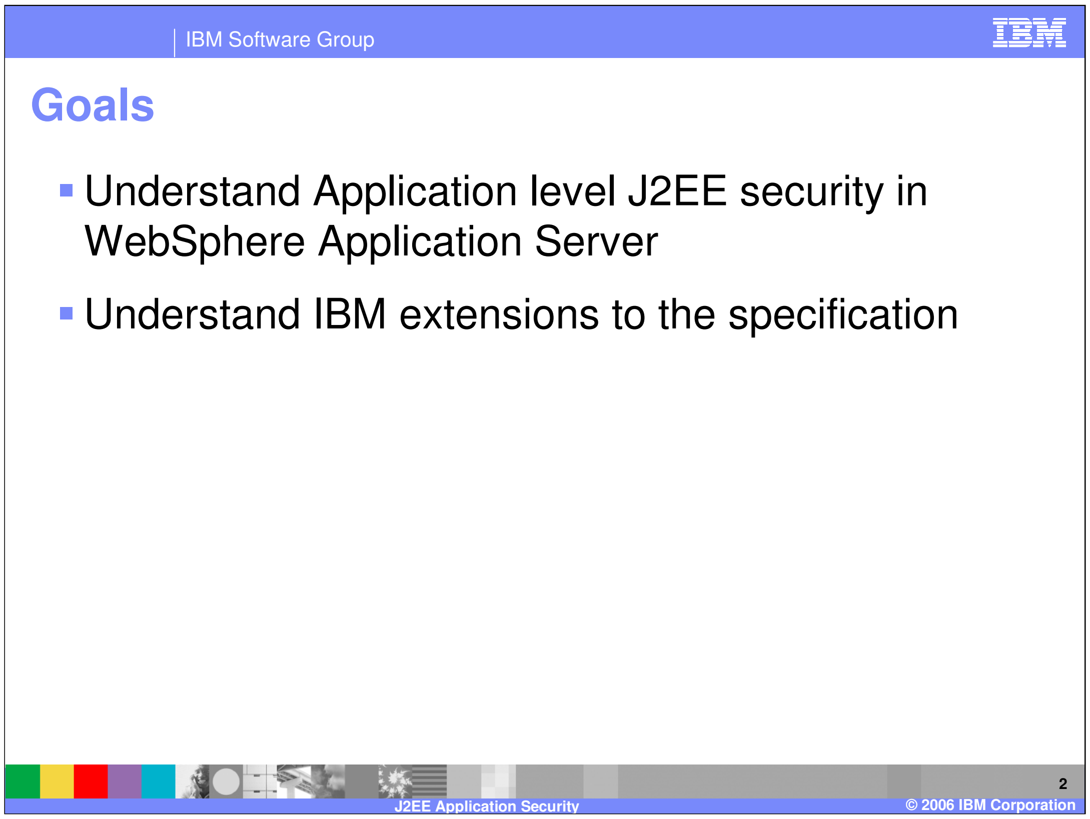
Java™ 2 Enterprise Edition (J2EE)

Application Security

@business on demand.

Updated June 8, 2006

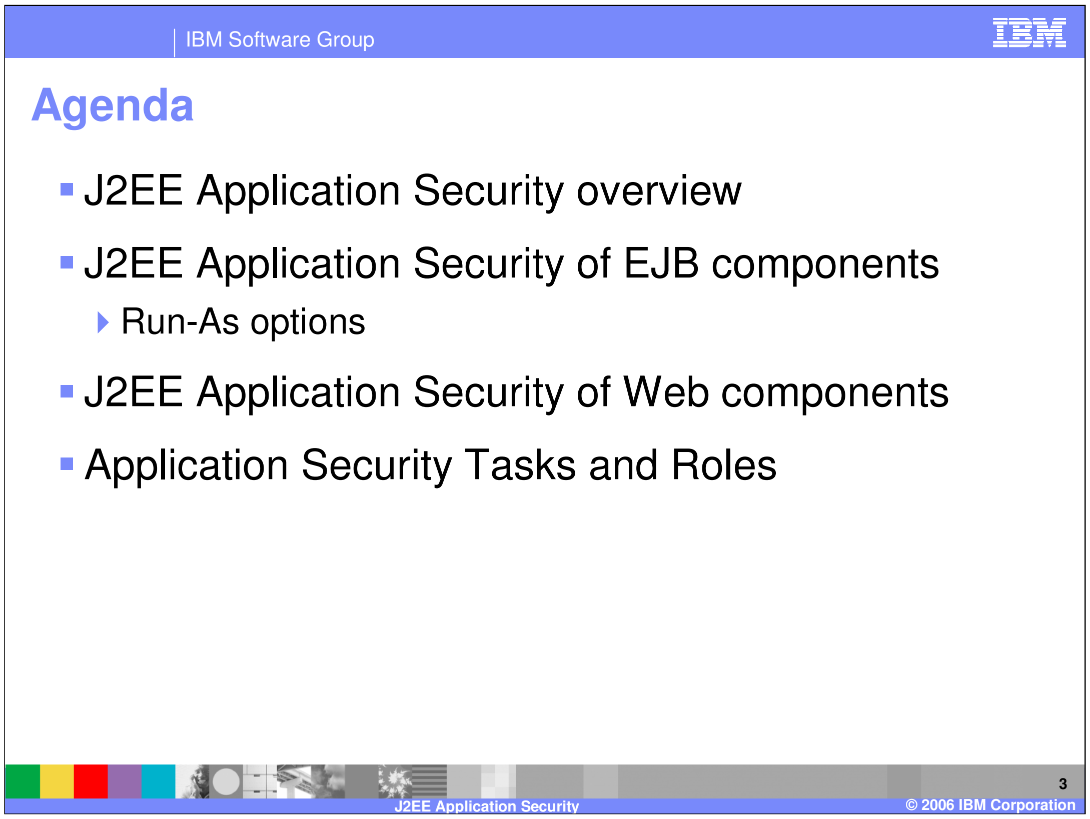
This presentation will focus on the J2EE Application Security.



Goals

- Understand Application level J2EE security in WebSphere Application Server
- Understand IBM extensions to the specification

The goal of this presentation is to help you understand the J2EE Security at the Application level. Some of the IBM extensions are also discussed here.



Agenda

- J2EE Application Security overview
- J2EE Application Security of EJB components
  - Run-As options
- J2EE Application Security of Web components
- Application Security Tasks and Roles

The agenda for this presentation is to provide an overview of J2EE Security, discuss application security as it relates to EJB and Web components, and discuss tasks and roles.

IBM Software Group

Section

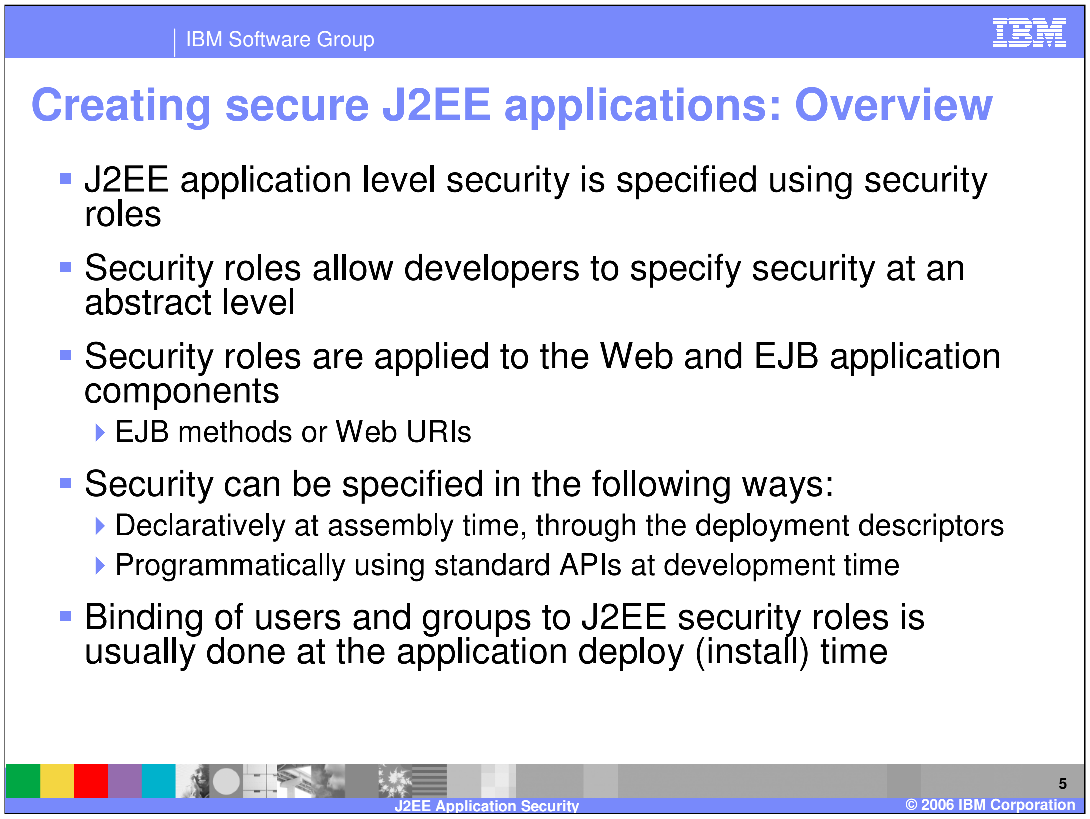
J2EE application security: Specifying authorization

J2EE Application Security

4

© 2006 IBM Corporation

This section will cover how to specify authorization information for your applications.



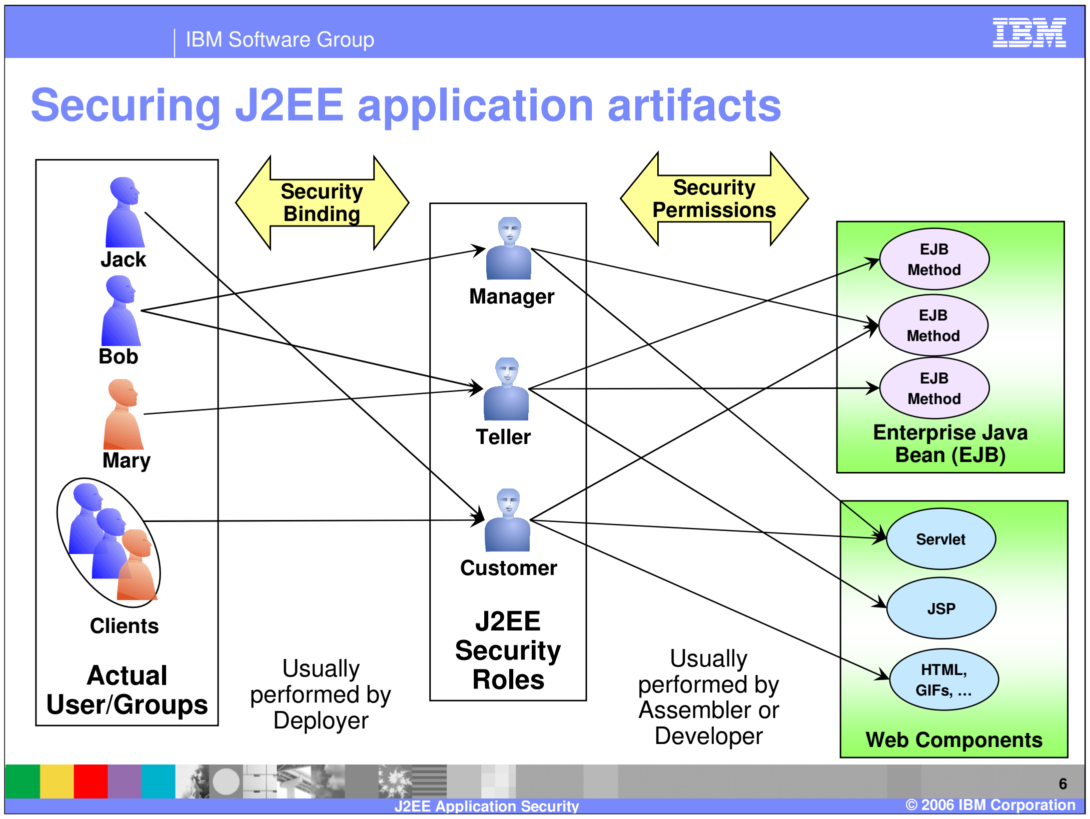
Creating secure J2EE applications: Overview

- J2EE application level security is specified using security roles
- Security roles allow developers to specify security at an abstract level
- Security roles are applied to the Web and EJB application components
  - EJB methods or Web URIs
- Security can be specified in the following ways:
  - Declaratively at assembly time, through the deployment descriptors
  - Programmatically using standard APIs at development time
- Binding of users and groups to J2EE security roles is usually done at the application deploy (install) time

An application developer may not know the actual users and groups that need access to the application. J2EE defines Security Roles that developers can use to provide authorization information. Developers provide access permission to the Security Roles for different parts of the application. They can do this Declaratively or Programmatically. This will be explained in the next few slides.

The binding or association of the roles to the actual users, groups or both is done by the System Administrator during application deployment on the Application Server, or after the application has been deployed.

J2EE Security roles apply to the entire application and all its modules. This information is saved in the Application and in the module deployment descriptors.



IBM Software Group

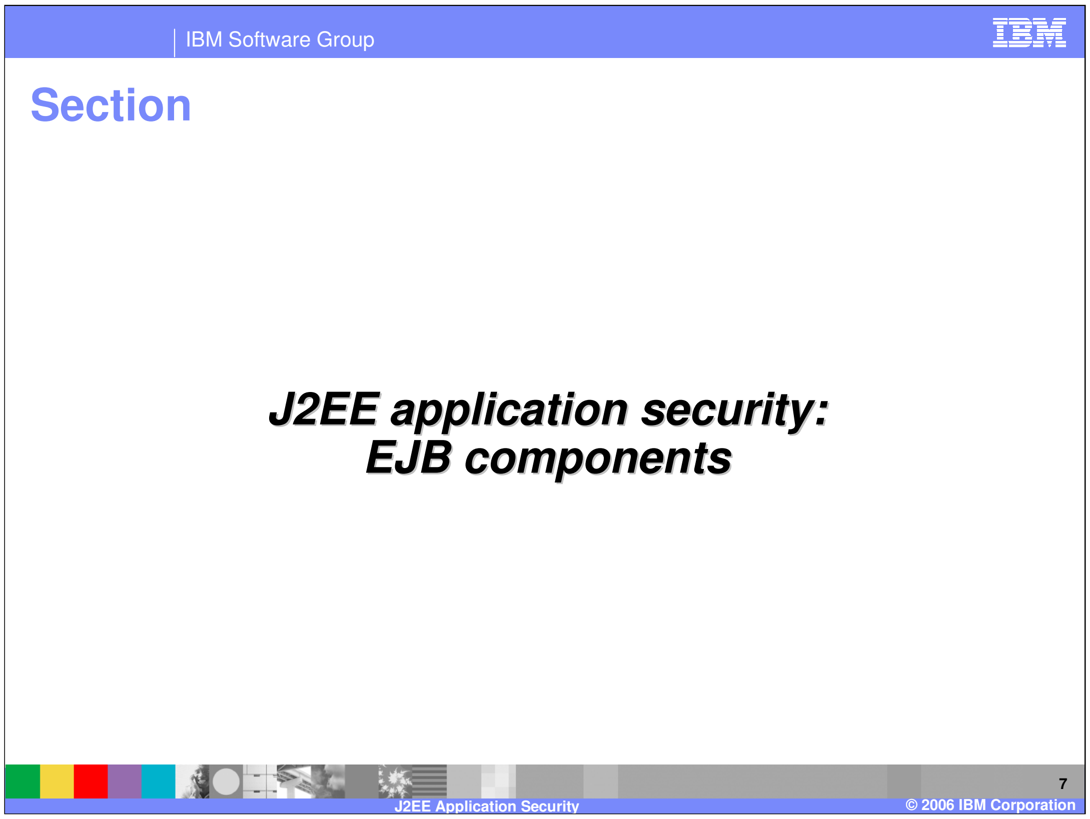
Securing J2EE application artifacts

6

J2EE Application Security

© 2006 IBM Corporation

This page shows the relevant part of the security process. The Developer or the Assembler will define the J2EE Security roles and specify what application artifacts can be accessed by these roles. The Deployer then binds the J2EE Security roles to the users or groups.



IBM Software Group

Section

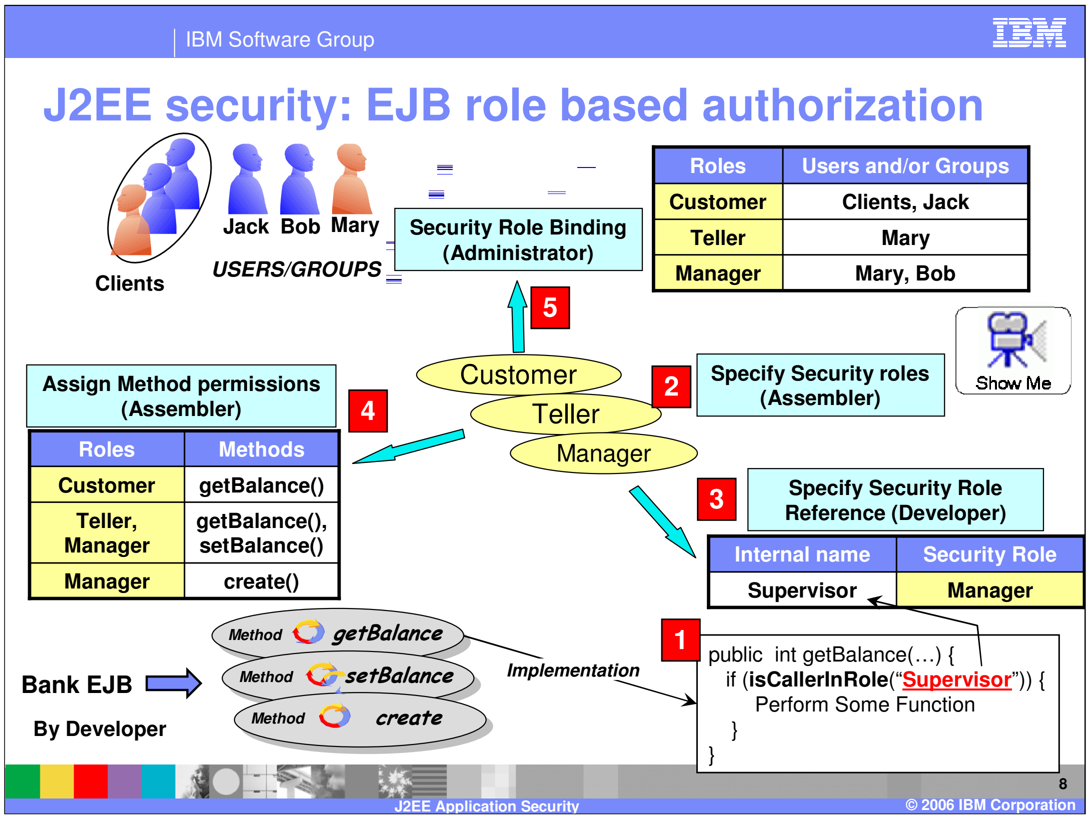
J2EE application security: EJB components

J2EE Application Security

© 2006 IBM Corporation

7

The next section will discuss how to specify authorization information for EJB components.



J2EE security: EJB role based authorization

Roles	Users and/or Groups
Customer	Clients, Jack
Teller	Mary
Manager	Mary, Bob

Roles	Methods
Customer	getBalance()
Teller, Manager	getBalance(), setBalance()
Manager	create()

Internal name	Security Role
Supervisor	Manager

```
public  int getBalance(…) {
   if (isCallerInRole("Supervisor")) {
         Perform Some Function
     }
}
```

The goal is to enable a developer or an assembler to specify who can access the methods within an EJB.

In some cases, the developer may not be the same person as the application Assembler.

A developer may not know the actual Security roles that can be defined later by an Assembler. In that case, the developer can use an internal security role name when making programmatic API calls to check for security, as shown in Step 1. It will be up to the Assembler to tie the internal role name to the actual security role by defining a security role reference, as shown in step 3.

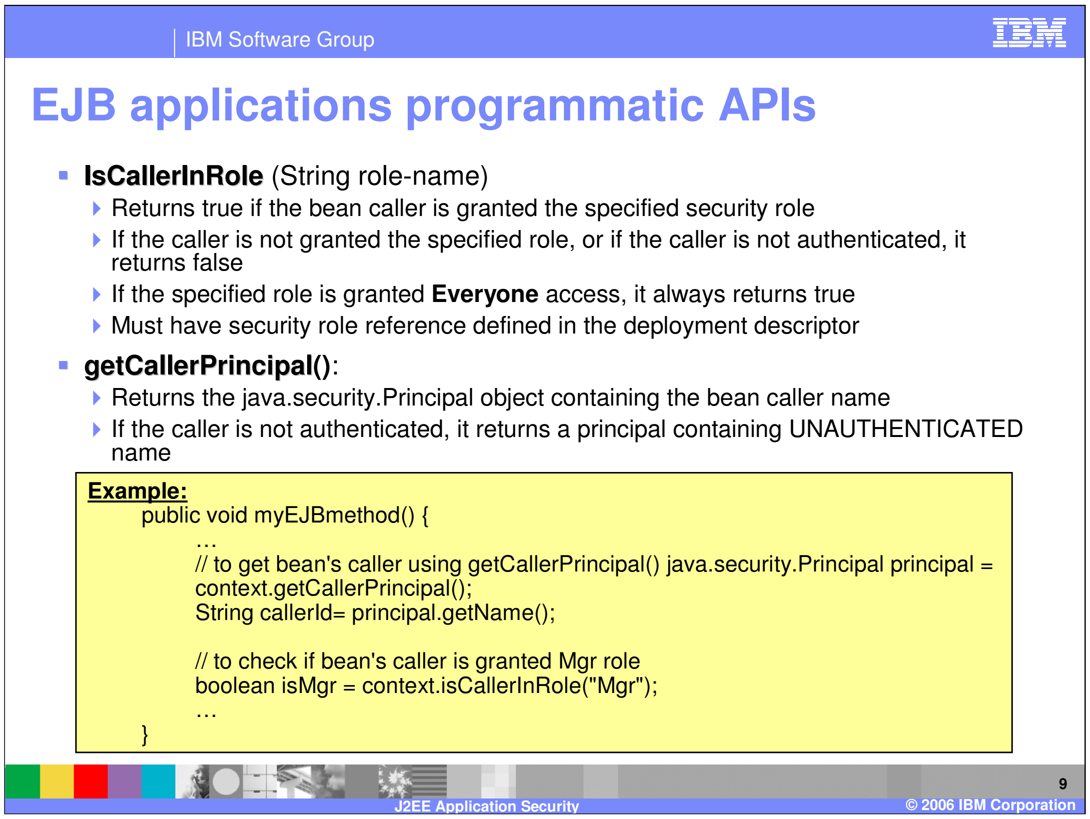
In the example, "Supervisor" is the internal local name. Later, the Assembler will have to map the internal name to the Role defined by the Assembler.

In the life cycle of the application, step 2 shows an Assembler defining a set of J2EE Security Roles for the application.

Now that the real Security Roles have been defined, the assembler must map any internal Role name used by the Developer to the Security Role reference, as shown in Step 3.

The Assembler can assign EJB method permissions, as shown in step 4. This is associating the methods to the Security roles. Once done, any user or group that is mapped to the Security role will have permission to access the method. You can use wild cards to specify multiple methods.

The application is now ready for deployment into an Application Server. During deployment, as shown in step 5, the System Administrator can bind the security roles to the users or groups that are in the Security User Registry for the Application Server. The Administrator can change the role to "user" or "group" binding at any time after the install.



EJB applications programmatic APIs

- **IsCallerInRole** (String role-name)
  - Returns true if the bean caller is granted the specified security role
  - If the caller is not granted the specified role, or if the caller is not authenticated, it returns false
  - If the specified role is granted **Everyone** access, it always returns true
  - Must have security role reference defined in the deployment descriptor
- **getCallerPrincipal()**:
  - Returns the java.security.Principal object containing the bean caller name
  - If the caller is not authenticated, it returns a principal containing UNAUTHENTICATED name

Example:

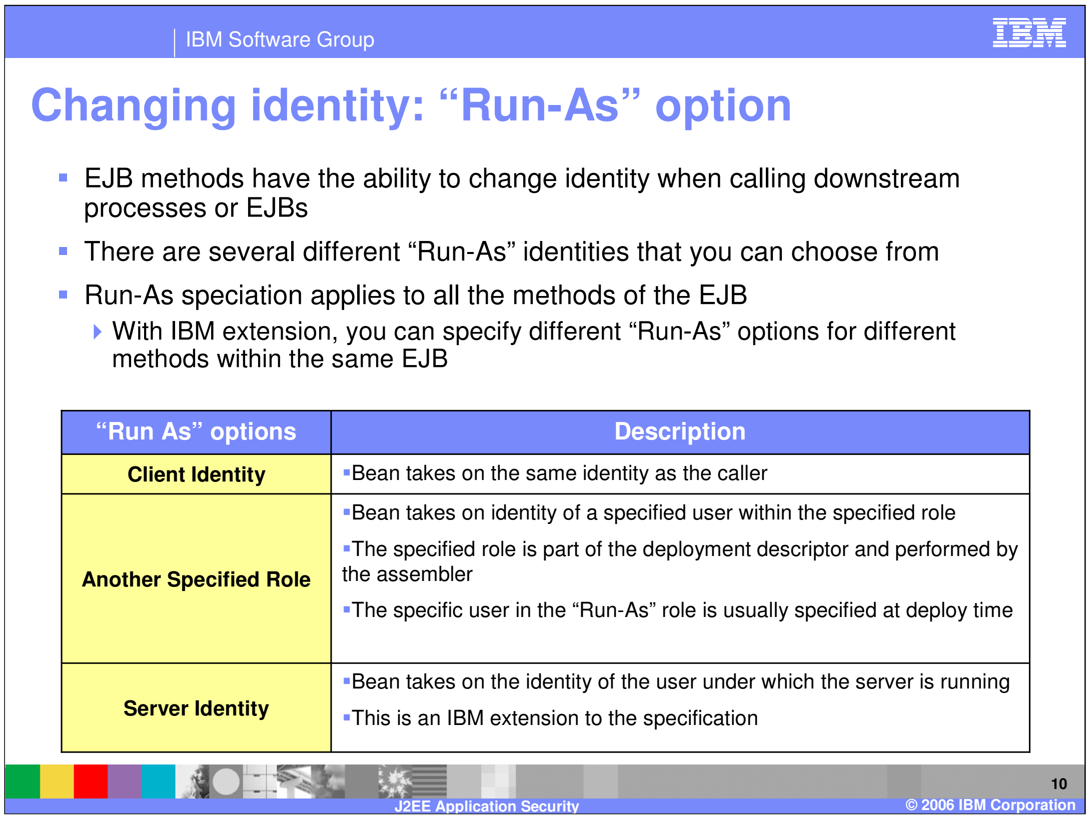
```
    public void myEJBmethod() {
        …
        // to get bean's caller using getCallerPrincipal() java.security.Principal principal =
        context.getCallerPrincipal();
        String callerId= principal.getName();

        // to check if bean's caller is granted Mgr role
        boolean isMgr = context.isCallerInRole("Mgr");
        …
    }
```

J2EE Security defines two APIs that can be used by EJB developer:

The **IsCallerInRole()** method is used to determine if the authenticated caller is in a specific Role defined by the application. This allows the developer to make decisions where they might want a certain part of the code to be allowed to run only for a user in a specific Role.

The **getCallerPrincipal()** method is used to return the authenticated caller.

The example provided shows how the developer could use these APIs.



IBM Software Group

Changing identity: “Run-As” option

- EJB methods have the ability to change identity when calling downstream processes or EJBs
- There are several different “Run-As” identities that you can choose from
- Run-As speciation applies to all the methods of the EJB
  - With IBM extension, you can specify different “Run-As” options for different methods within the same EJB

“Run As” options	Description
Client Identity	▪Bean takes on the same identity as the caller
Another Specified Role	▪Bean takes on identity of a specified user within the specified role ▪The specified role is part of the deployment descriptor and performed by the assembler ▪The specific user in the “Run-As” role is usually specified at deploy time
Server Identity	▪Bean takes on the identity of the user under which the server is running ▪This is an IBM extension to the specification

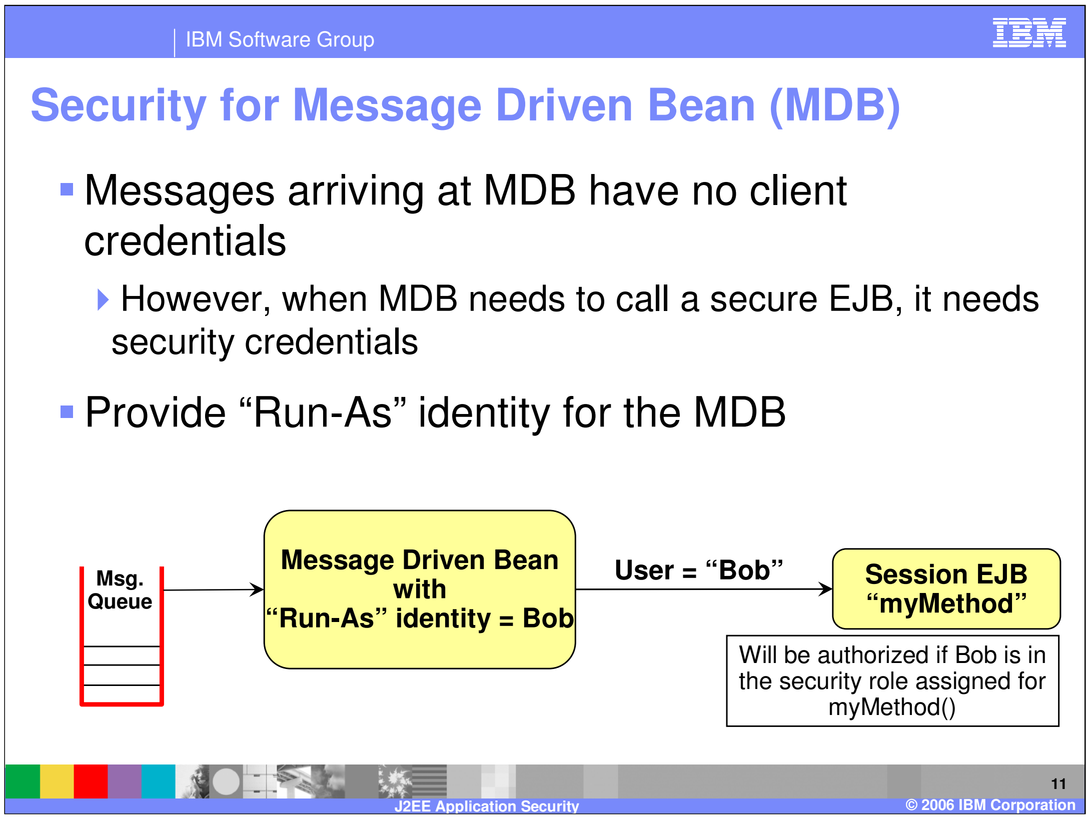
10
J2EE Application Security
© 2006 IBM Corporation

The “Run-As” option is a way to change the identity of the caller from an EJB when calling a downstream EJB.

An example of this idea is where a client cannot directly call a downstream EJB, but can call an upstream EJB, which can call a downstream EJB by changing its identity.

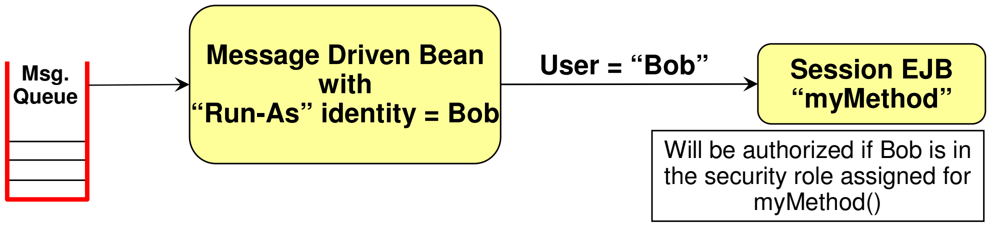
There are three “Run as” options, as shown in the table. The downstream EJB can be called as the same identity as the original client identity, or can be called as the Server identity, which is the user ID under which WebSphere Application Server is running. The last option is to run as another specified role.

The Server Identity “Run as” option is an IBM extension, carried forward from Version 4 and Version 5.


IBM Software Group

Security for Message Driven Bean (MDB)

- Messages arriving at MDB have no client credentials
  - However, when MDB needs to call a secure EJB, it needs security credentials
- Provide “Run-As” identity for the MDB



Msg. Queue
Message Driven Bean with “Run-As” identity = Bob
User = “Bob”
Session EJB “myMethod”
Will be authorized if Bob is in the security role assigned for myMethod()

11

J2EE Application Security

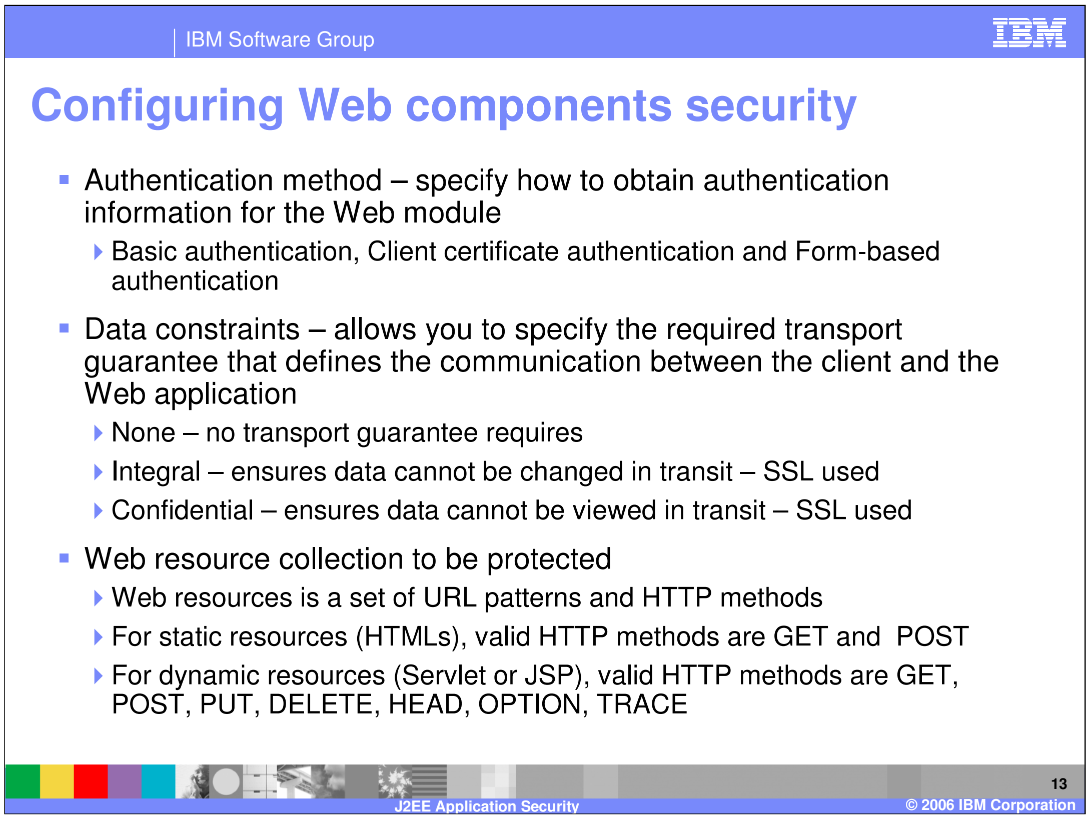
© 2006 IBM Corporation

Message Driven EJBs, or MDBs, are called by the EJB container when there is a message in the queue. As such, there is no client identity. So if a MDB needs to call another downstream EJB that needs authentication, providing “Run-As” on the MDB allows the Assembler to send an identity to the downstream EJB, as shown in the example.

Section

J2EE application security:
Web components (Servlets/JSPs/HTML)

The next section will discuss how to specify authorization information for Web components.



Configuring Web components security

- Authentication method – specify how to obtain authentication information for the Web module
  - Basic authentication, Client certificate authentication and Form-based authentication
- Data constraints – allows you to specify the required transport guarantee that defines the communication between the client and the Web application
  - None – no transport guarantee requires
  - Integral – ensures data cannot be changed in transit – SSL used
  - Confidential – ensures data cannot be viewed in transit – SSL used
- Web resource collection to be protected
  - Web resources is a set of URL patterns and HTTP methods
  - For static resources (HTMLs), valid HTTP methods are GET and POST
  - For dynamic resources (Servlet or JSP), valid HTTP methods are GET, POST, PUT, DELETE, HEAD, OPTION, TRACE

Configuring Web Component security is a little more involved than EJB components.

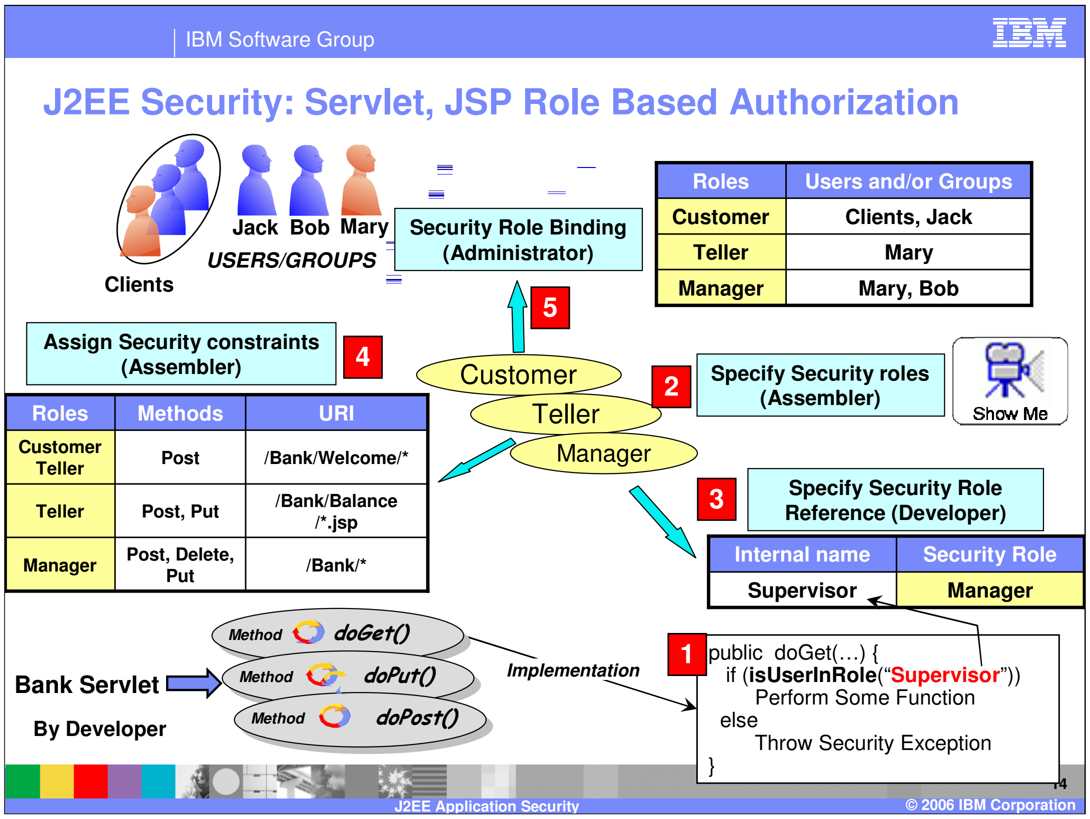
The Web component needs to let the server know how the Web client should provide authentication: with basic authentication, client certificate or form based custom login.

Next, the Web component Developer needs to specify whether integrity or confidentiality is required. This is specified by the Data constraints.

Finally, the Assembler need to specify who has access to the set of Web resources within the component. The Web resource is a set of URLs and the HTTP method. Note that a Web resource is accessed using a URL and one of the six methods specified.

The Assembler can then authorize the different J2EE Security roles to have permission to the Web resource

All this information is stored in the Web Deployment descriptor.



J2EE Security: Servlet, JSP Role Based Authorization

Clients
Jack Bob Mary
USERS/GROUPS
Security Role Binding (Administrator)
5
Assign Security constraints (Assembler)
4
Customer
Teller
Manager
2
Specify Security roles (Assembler)
Show Me
3
Specify Security Role Reference (Developer)
1
Bank Servlet
By Developer
Method doGet()
Method doPut()
Method doPost()
Implementation

Roles	Users and/or Groups
Customer	Clients, Jack
Teller	Mary
Manager	Mary, Bob

Roles	Methods	URI
Customer Teller	Post	/Bank/Welcome/*
Teller	Post, Put	/Bank/Balance /*.jsp
Manager	Post, Delete, Put	/Bank/*

Internal name	Security Role
Supervisor	Manager

```
public  doGet(…) {
  if (isUserInRole("Supervisor"))
      Perform Some Function
  else
      Throw Security Exception
}
```

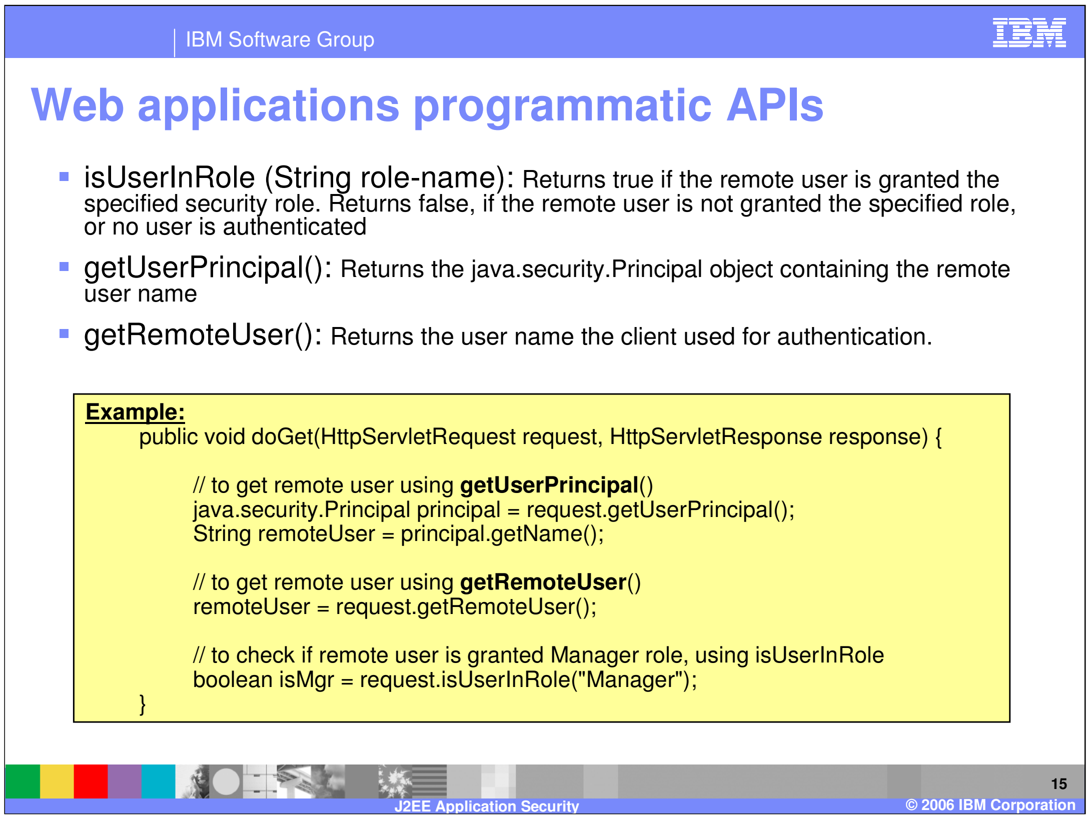
Pause this presentation and click the Show Me icon for a demonstration on how to add J2EE security roles for Web applications using the tools.

Applying J2EE Security to Web components is very similar to EJBs.

So, steps 1, 2 and 3 are similar to what was explained for the EJB methods.

The difference is step 4. For EJB, the permission is on EJB methods, whereas for Web components, it is on the Web resource. As indicated in the previous slide, the Web resource is the collection of the URL and the HTTP methods. Once Web resources are defined, the J2EE Security Roles can then be assigned to the Web resource collection, thereby creating the security constraints.

Again, step 5 of binding the J2EE security roles to the users or groups is similar to EJBs.



Web applications programmatic APIs

- isUserInRole (String role-name): Returns true if the remote user is granted the specified security role. Returns false, if the remote user is not granted the specified role, or no user is authenticated
- getUserPrincipal(): Returns the java.security.Principal object containing the remote user name
- getRemoteUser(): Returns the user name the client used for authentication.

Example:

```
    public void doGet(HttpServletRequest request, HttpServletResponse response) {

        // to get remote user using getUserPrincipal()
        java.security.Principal principal = request.getUserPrincipal();
        String remoteUser = principal.getName();

        // to get remote user using getRemoteUser()
        remoteUser = request.getRemoteUser();

        // to check if remote user is granted Manager role, using isUserInRole
        boolean isMgr = request.isUserInRole("Manager");
    }
```

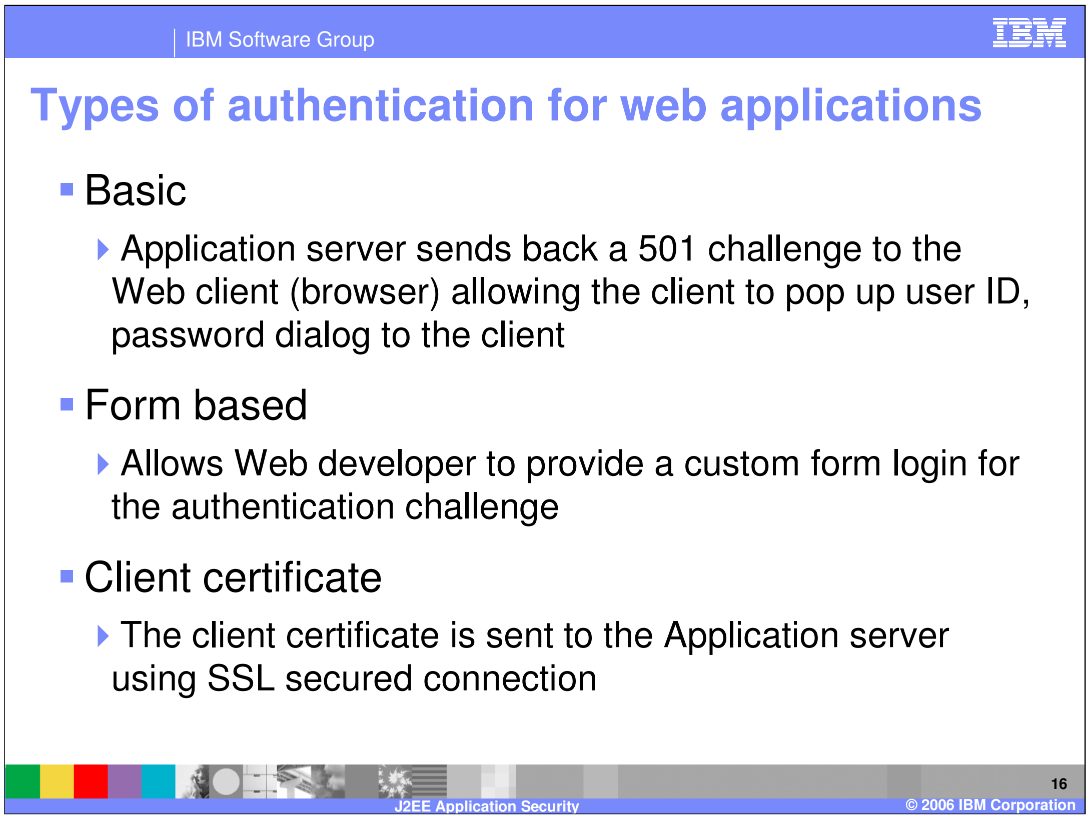
J2EE Security defines two APIs that can be used by Web developer. Note that these are similar to the APIs used by EJB developers, but not the same.

The **IsUserInRole()** method is used to determine if the authenticated caller is in a specific Role defined by the application. This allows the Developer to make decisions where they might want a certain part of the code to be allowed to run only for a user in a specific Role.

The **getUserPrincipal()** method is used to return the Principal of the authenticated caller.

The **getRemoteUserl()** method is used to return the remote web client user.

The example on this page shows how developers can use these APIs.


IBM Software Group

Types of authentication for web applications

- Basic
  - Application server sends back a 501 challenge to the Web client (browser) allowing the client to pop up user ID, password dialog to the client
- Form based
  - Allows Web developer to provide a custom form login for the authentication challenge
- Client certificate
  - The client certificate is sent to the Application server using SSL secured connection

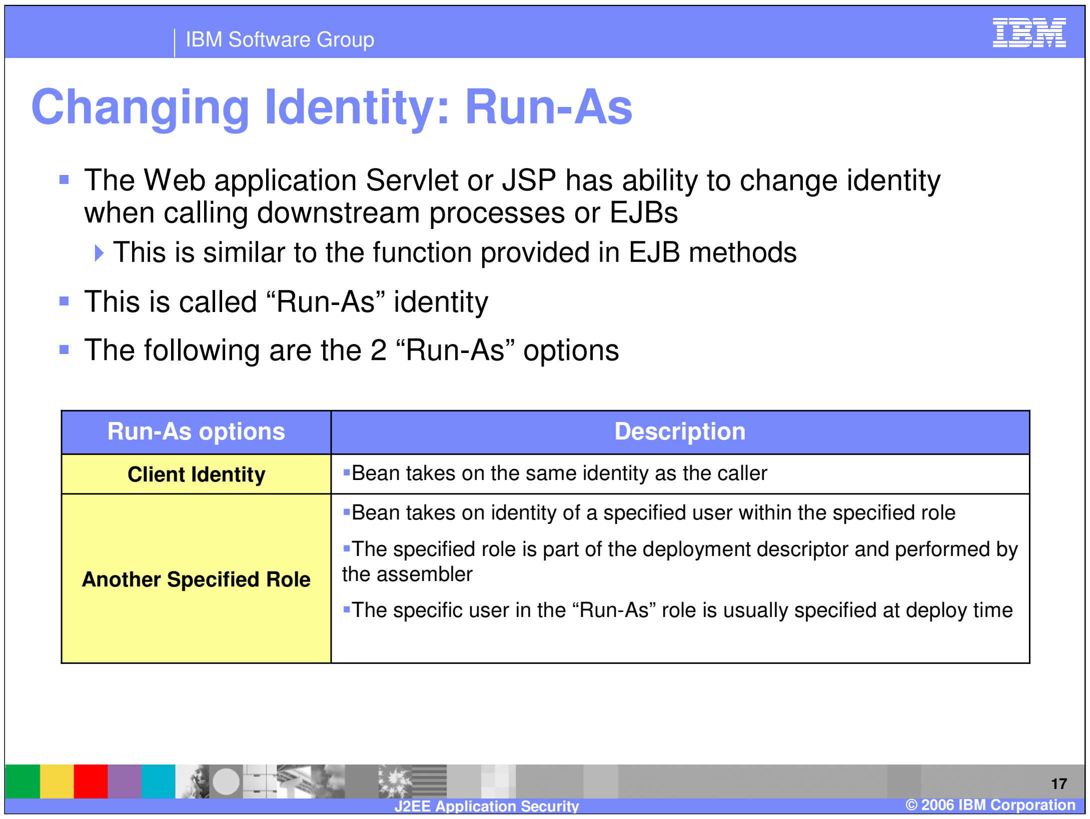
16

J2EE Application Security

© 2006 IBM Corporation

There are three types of authentication that a Web client can perform to access the Web applications. It is the Assembler that specifies the authentication method that is to be used by the Web client.

This information is stored in the Web module deployment descriptor. It is used by the Web container to challenge the Web client for authentication.



IBM Software Group

Changing Identity: Run-As

- The Web application Servlet or JSP has ability to change identity when calling downstream processes or EJBs
  - This is similar to the function provided in EJB methods
- This is called “Run-As” identity
- The following are the 2 “Run-As” options

Run-As options	Description
Client Identity	▪Bean takes on the same identity as the caller
Another Specified Role	▪Bean takes on identity of a specified user within the specified role ▪The specified role is part of the deployment descriptor and performed by the assembler ▪The specific user in the “Run-As” role is usually specified at deploy time

17

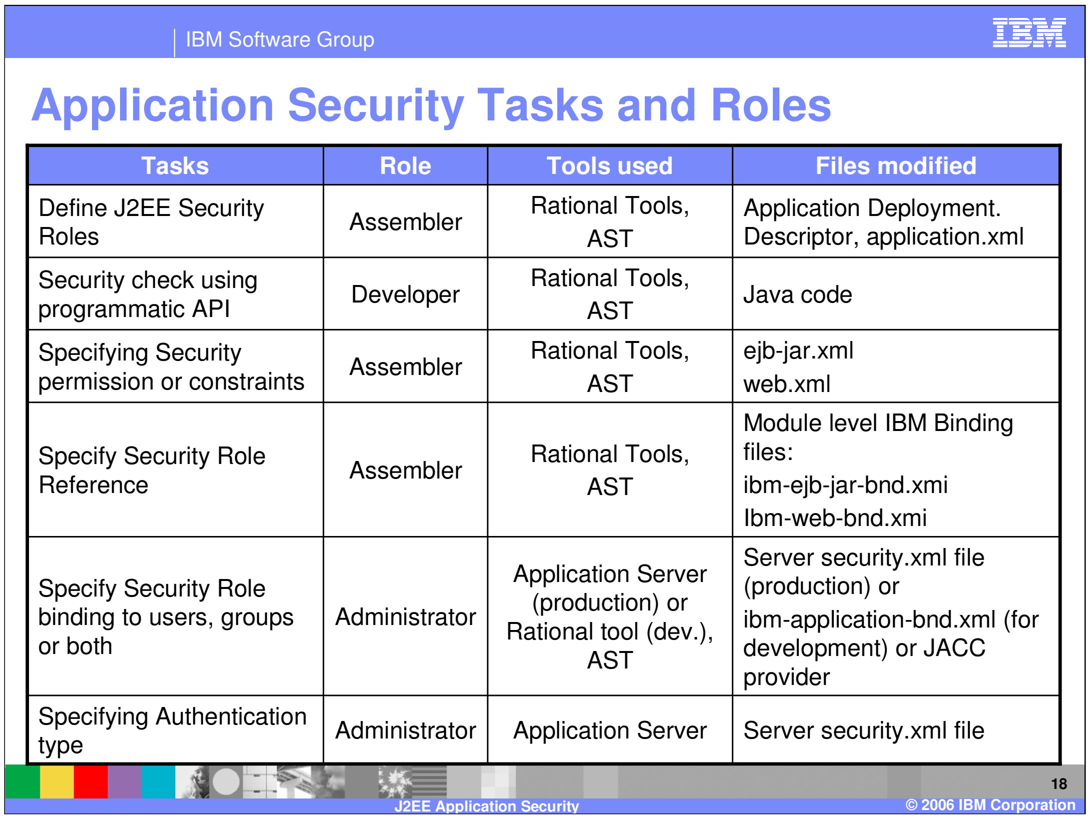
J2EE Application Security

© 2006 IBM Corporation

“Run As” is a way to change the identity of the caller from a servlet or JSP when calling a downstream EJB.

An example is where a client cannot directly call a downstream EJB, but can call a servlet, which can call a downstream EJB by changing its identity.

There are two “Run As” options, as shown in the table. The downstream EJB can be called as the same identity as the original Web client identity that called the Servlet, or the identity can be changed by the Servlet Assembler by providing “Run As” as another specified role.


IBM Software Group

Application Security Tasks and Roles

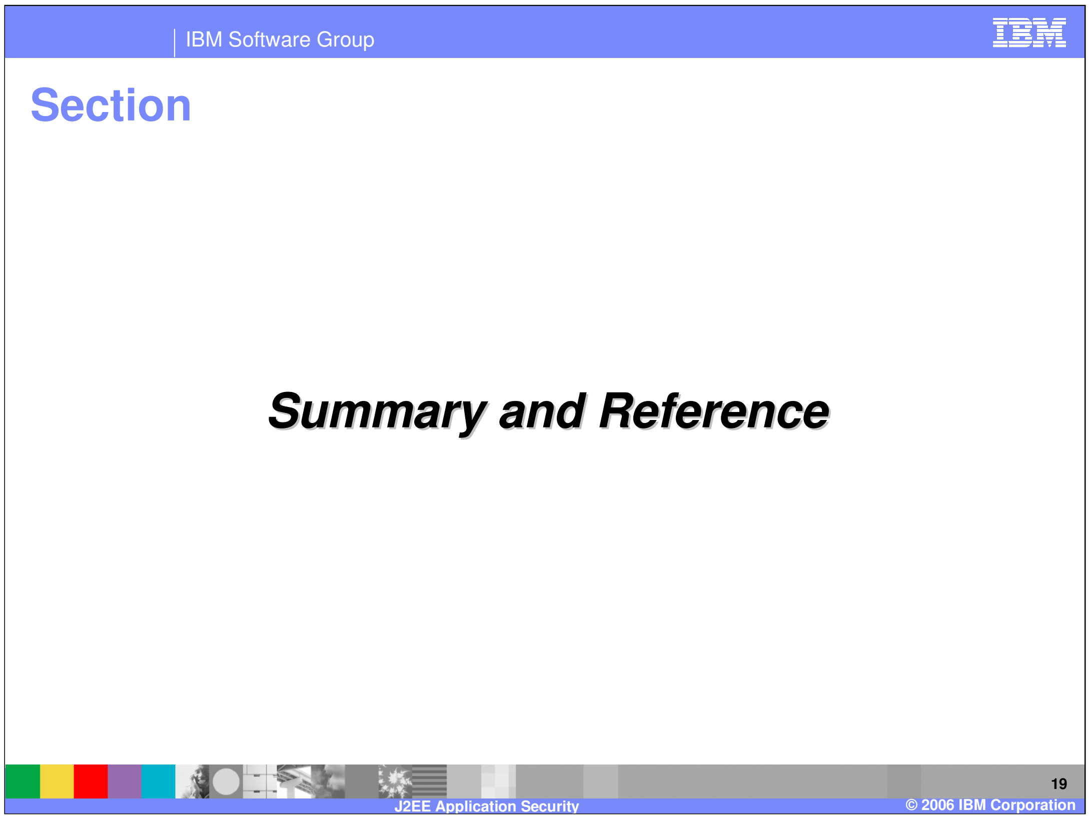
Tasks	Role	Tools used	Files modified
Define J2EE Security Roles	Assembler	Rational Tools, AST	Application Deployment. Descriptor, application.xml
Security check using programmatic API	Developer	Rational Tools, AST	Java code
Specifying Security permission or constraints	Assembler	Rational Tools, AST	ejb-jar.xml web.xml
Specify Security Role Reference	Assembler	Rational Tools, AST	Module level IBM Binding files: ibm-ejb-jar-bnd.xmi Ibm-web-bnd.xmi
Specify Security Role binding to users, groups or both	Administrator	Application Server (production) or Rational tool (dev.), AST	Server security.xml file (production) or ibm-application-bnd.xml (for development) or JACC provider
Specifying Authentication type	Administrator	Application Server	Server security.xml file

18

J2EE Application Security

© 2006 IBM Corporation

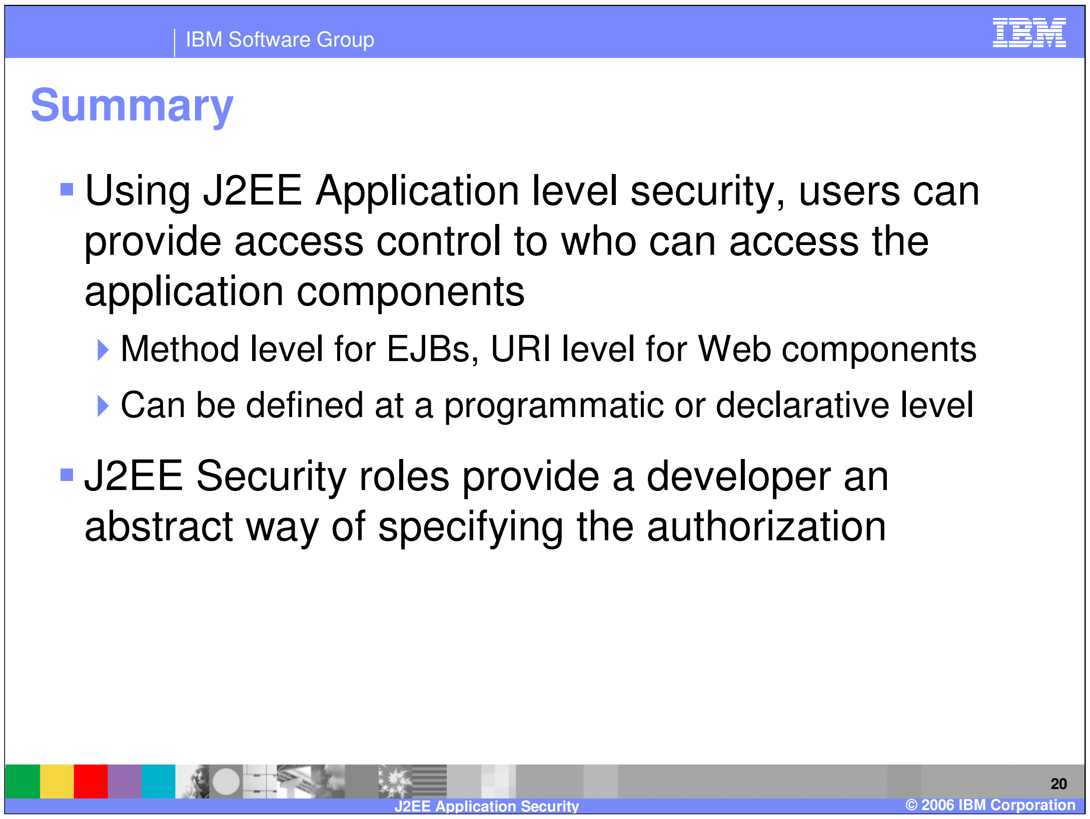
The table provides the summary of different tasks related to the J2EE Security Roles, the role that typically performs the task, the tools used to performs the task, and the files that get modified.



Section

Summary and Reference

Next section will provide the Summary.



Summary

- Using J2EE Application level security, users can provide access control to who can access the application components
  - Method level for EJBs, URI level for Web components
  - Can be defined at a programmatic or declarative level
- J2EE Security roles provide a developer an abstract way of specifying the authorization

In summary, this presentation has focused on J2EE Security roles and how they are used to authorize the J2EE Application artifacts: EJBs and Web applications.

Trademarks, copyrights, and disclaimers