



BEA
world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Building Secure Web Services with BEA WebLogic Workshop

David Remy

Director, Product Design

Bea Systems, Inc.

Disclaimer

This information represents work in progress
This information is NOT a commitment by BEA
This information is subject to change

Learning Objectives

- As a result of this presentation, you will be able to:
 - Understand how *Weblogic Server* security relates to *Weblogic Workshop* security
 - Develop a *Weblogic Workshop* web services security strategy using transport security, message security, and/or role based security
 - Understand the new *Weblogic Workshop 8.1* WS-Security implementation

Speaker's Qualifications

- **David Remy** is a Director of Product Design for Weblogic Workshop at BEA Systems, Inc. focused on Weblogic Workshop security, web services, and xml technologies.
- 16 years of industry experience prior to joining BEA including most recently as a co-founder of the security firm GeoTrust, Inc.
- Certified as a Computer Information Systems Security Professional (CISSP) as well as having a Sun Certified Java Programmer certification.
- Currently co-authoring the book "Understanding Web Services Security" (June 2003) for Addison Wesley.

Presentation Agenda

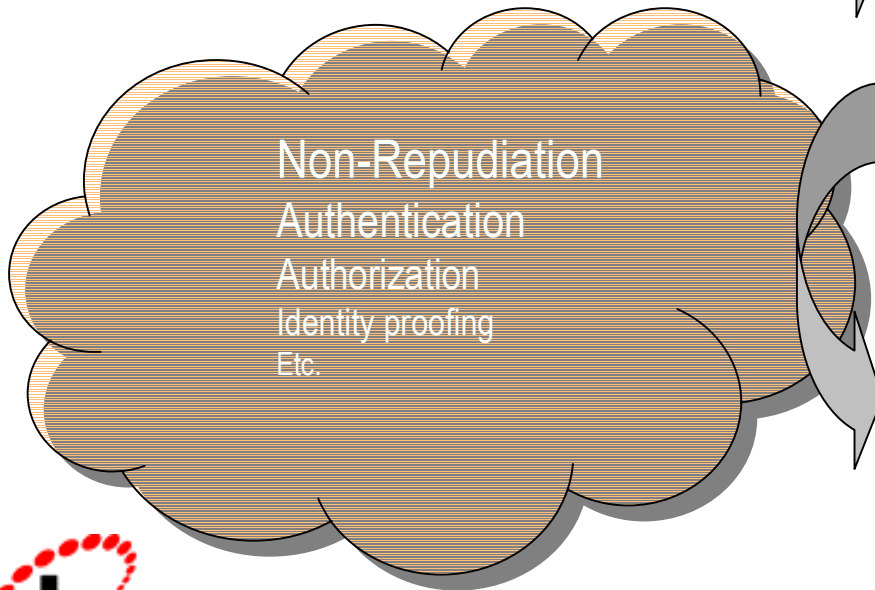
- Security from the top
- Weblogic Server Security
- Weblogic Workshop Security
 - Transport Level Security
 - Message Level Security
 - Role Based Security
- Summary
- Q&A

Security from the top

Confidentiality

Integrity

Availability



Non-Repudiation
Authentication
Authorization
Identity proofing
Etc.

Protecting information assets from internal and external compromise in a cost effective manner.

Cost effective = risk analysis

- What to protect
- How much \$\$ can be lost
- How much \$\$ to spend protecting

Security Policy

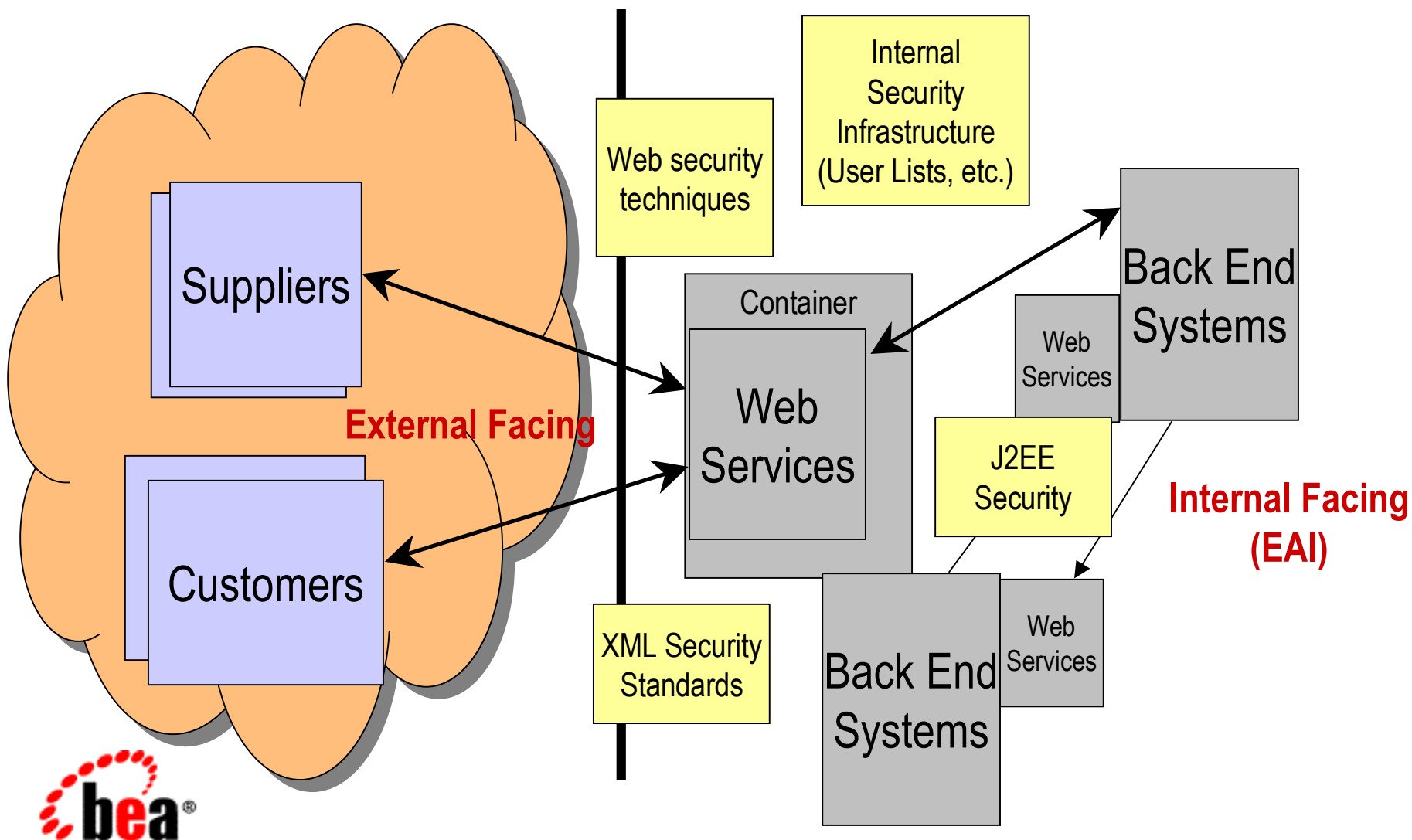
Confidentiality * Integrity * Availability Mechanisms

- Cryptography
- Authentication techniques
- Access control
- Monitoring & auditing
- Intrusion Detection
- Vulnerability Analysis
- Separation of duties
- Load Balancing
- Firewalls
- Etc., Etc. Etc.

Apply

Web Services

The Web Services Security Landscape



Security Roles

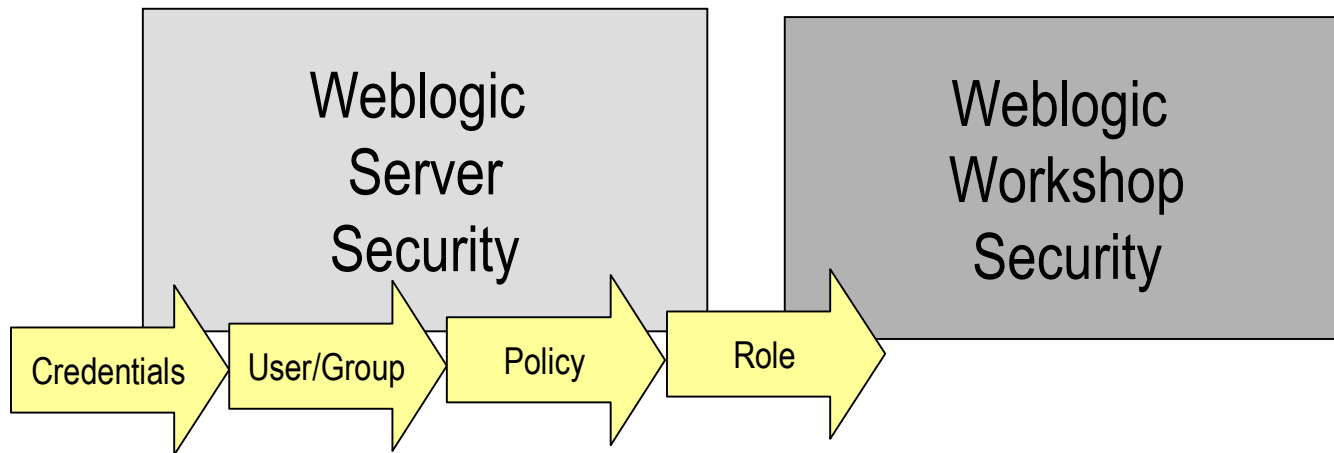
- Administrator

- Test/QA

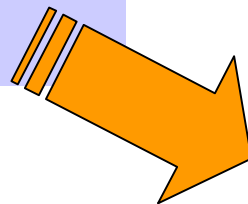
- Developer

- Develops to security policy
- Works primarily with Roles

Weblogic Server and Weblogic Workshop – Partners in Security



Administrators typically control Weblogic Server security ... however ...



Workshop developers work primarily at the role level

Developers need to be familiar with Weblogic Security Framework to dev and test secure web applications/services ...

Weblogic Server and Weblogic Workshop – Partners in Security

Weblogic Security Framework

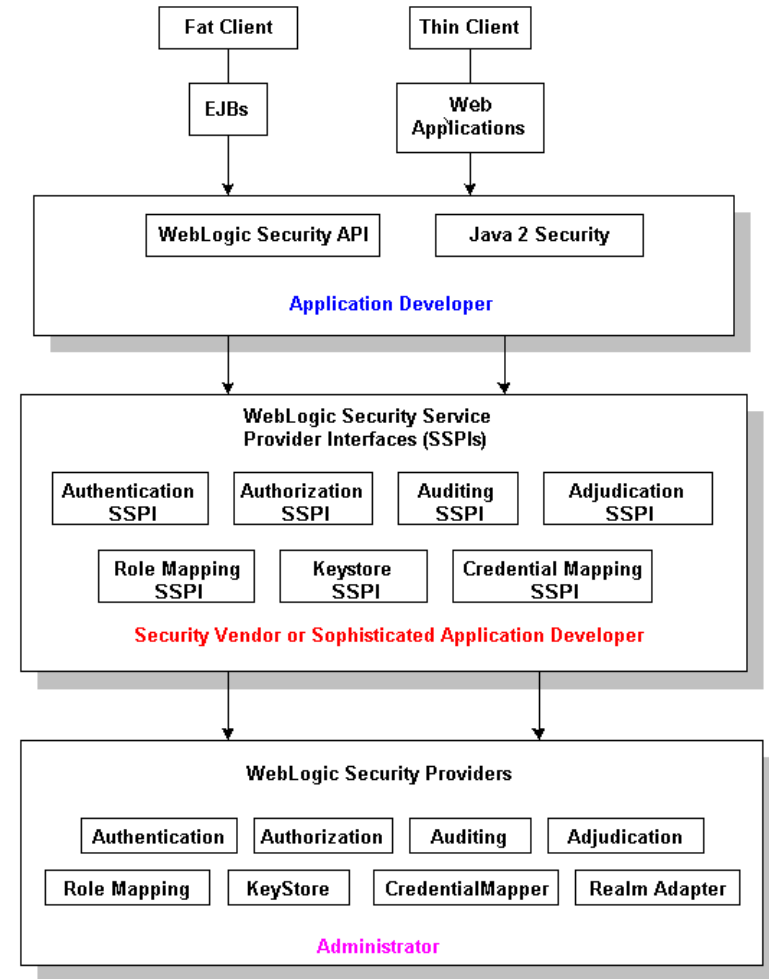
Security Framework

- Authentication
- Key Management
- Authorization
- Role Mapping
- Etc.

Weblogic Console

Console

- Policy Management
- User Management



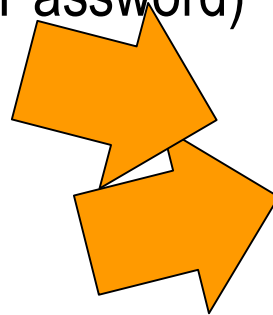
Three major security mechanisms:

Transport security (http)

- One Way SSL
- Basic Auth (Username Password)
- Two Way SSL

Role Based Security (application)

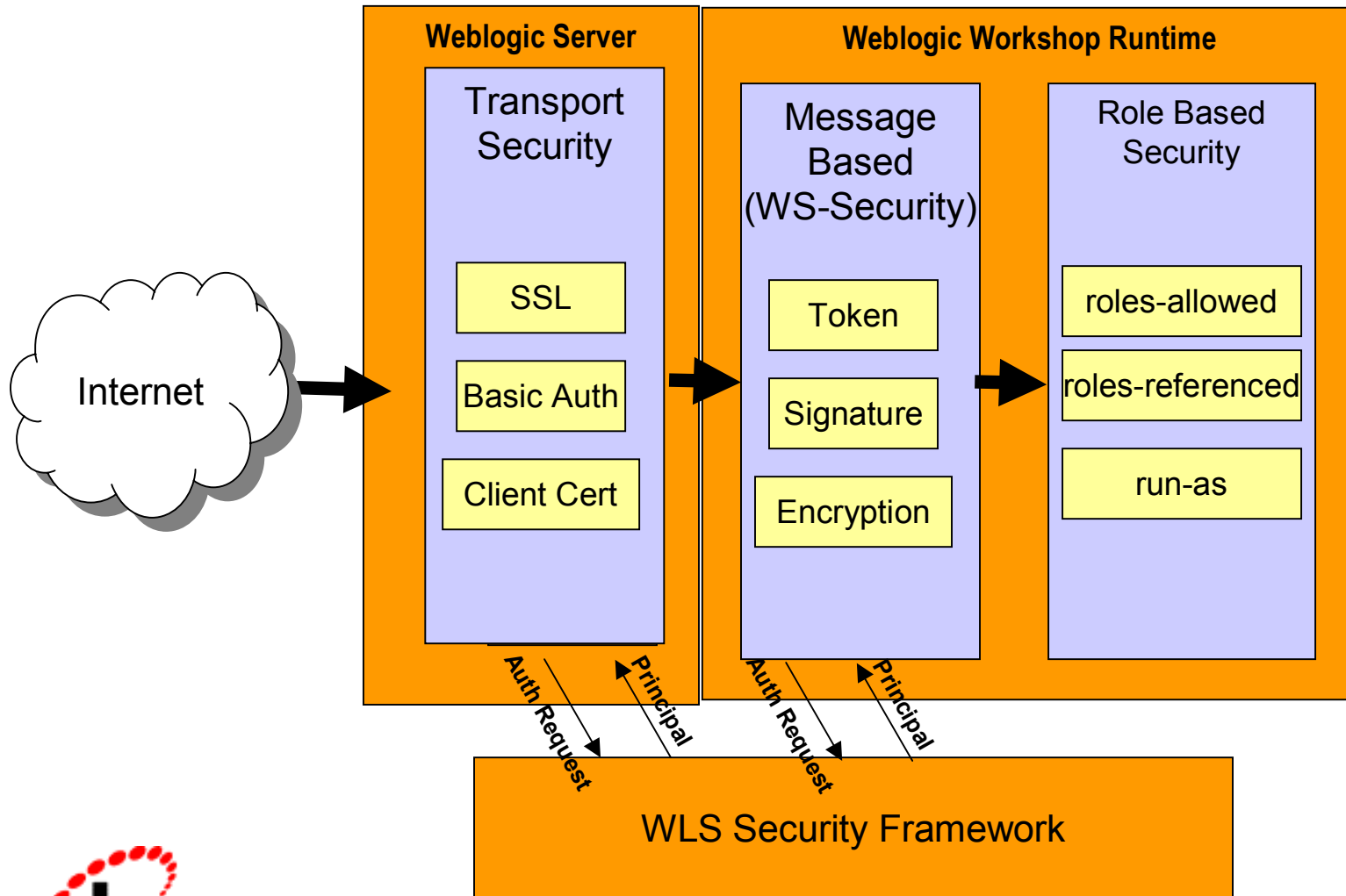
- roles-required
- roles-referenced
- run-as



Message Based Security

- Tokens
(Authentication/Authorization)
- XML Encryption
- XML Signature

Weblogic Server/Workshop Security Runtime





BEA **G** world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Transport Based Security

Transport Security 1 (http)

What is it?

SSL/TLS

- Encryption
- Authentication of server (sort of)

Basic Auth

- Username Password
- By default in the clear, unless combined with SSL

Client Auth (Two way SSL) (Mutual Auth)

- Combines with one way SSL (above)
- Client must provide X509 Certificate from trusted authority

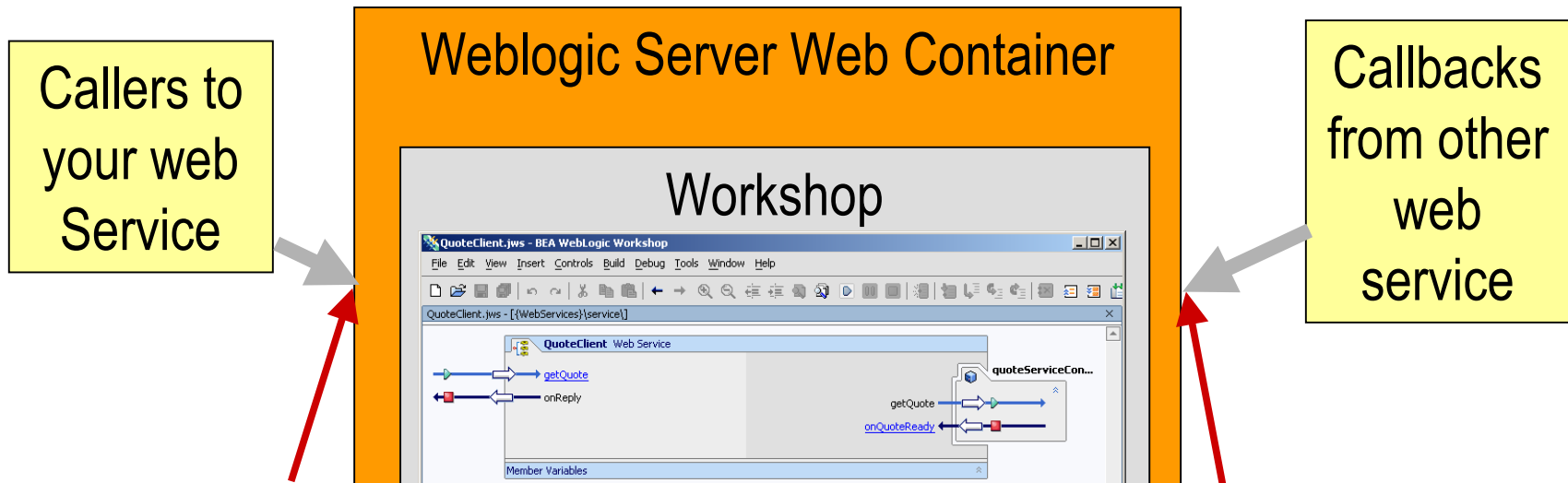
Advantages:

- Mature – tried and true
- Supported by most clients and all web containers
- Understood by most administrators

Disadvantages:

- Point to point. **Information in the clear after endpoint.**
- Intermediaries (firewalls) cannot read content

Transport Security in Weblogic Workshop - Inbound



- Container handles handshake, credentials gathering, encryption, based on web.xml
- All inbound to your web service treated the same – caller or callback

Transport Security in Weblogic Workshop - Inbound

Modify web.xml:

```
<web-app>
  . . .
  <security-constraint>
    <web-resource-collection>
      <web-resource-name>myjws</web-resource-name>
      <description>MyJws</description>
      <url-pattern>/my.jws</url-pattern>
      <http-method>POST</http-method>
    </web-resource-collection>
    <login-config>
      <auth-method>CLIENT</auth-method>
    </login-config>
    <user-data-constraint>
      <description>SSL required</description>
      <transport-guarantee>CONFIDENTIAL</transport-guarantee>
    </user-data-constraint>
  </security-constraint>
  . . .
</web-app>
```

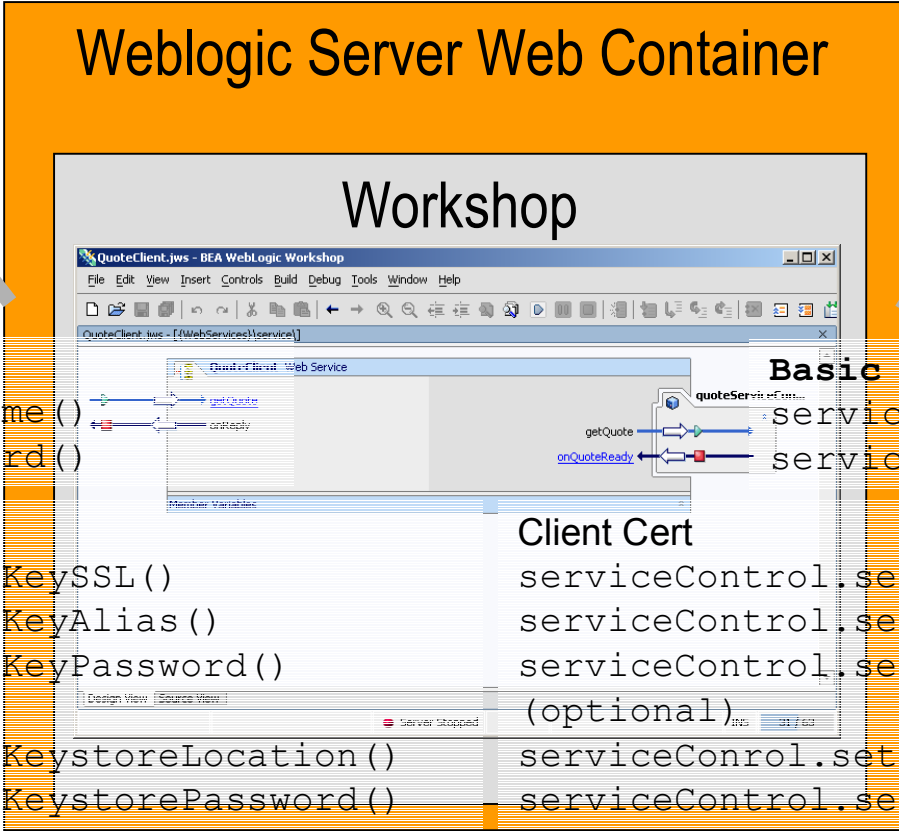
Client Auth



Transport Security in Weblogic Workshop - Outbound

Callbacks from your web service

Calling another web service



Basic Auth

```
callback.setUsername()  
callback.setPassword()
```

Client Cert

```
callback.setClientKeySSL()  
callback.setClientKeyAlias()  
callback.setClientKeyPassword()  
(optional)  
callback.setClientKeystoreLocation()  
callback.setClientKeystorePassword()
```

Basic Auth

```
serviceControl.setUsername()  
serviceControl.setPassword()
```

Client Cert

```
serviceControl.setClientKeySSL()  
serviceControl.setClientKeyAlias()  
serviceControl.setClientKeyPassword()  
(optional)  
serviceControl.setClientKeystoreLocation()  
serviceControl.setClientKeystorePassword()
```



- Solid, strong enterprise security
- Most likely to be interoperable with other parties
- Less granular than message level security
 - Can't differentiate callbacks
 - Can't differentiate request/response
- Point to point means that message itself is not secure ...



BEA **G** world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Message Based Security

WS-Security implementation

Advantages

- Allows the message to be self protecting
- Portions of message can be secured to different parties
- More granular (callbacks, request/response)

Disadvantages

- Immature, still in Oasis Technical committee
- Complex, encompassing many other standards including XML Signature, XML Encryption, and more

History and status

- Submitted as a note by Microsoft, IBM, and Verisign to the W3C (April 2002)
- Oasis Security TC (July 2002)
- Still in committee

Highlights

- New soap security header
- Defines no new security mechanisms, refers to other such as XML Signature, XML Encryption, and SAML
- Specifies some best practices where applicable
- Extremely flexible, able to support new security requirements going forward

Signature

- XML Signature
- Can sign any portion of message
- Can have multiple signatures
- Sign with private key

WS-Security Example Message

Security Header

```
<wsse:Security
xmlns:wsse="http://schemas.xmlsoap.org/ws/2002/04/secext">
```

UserName Token

```
<wsse:UsernameToken Id="MyID">
  <wsse:Username>frank</wsse:Username>
  <wsse:Password>password</wsse:Password>
</wsse:UsernameToken>
```

BinarySecurityToken

```
<wsse:BinarySecurityToken
  ValueType="wsse:X509v3"
  Id="X509Token"
  EncodingType="wsse:Base64Binary">
  MIIEZzCCA9CgAwIBAgIQEmtJZc0rqrKh5i...
</wsse:BinarySecurityToken>
```

WS-Security Example Message XML Encryption

EncryptedKey

- EncryptedKey wraps the symmetric key.
- Common usage for web services.

```
<xenc:EncryptedKey>
  <xenc:EncryptionMethod
    Algorithm="http://www.w3.org/200
1/04/xmlenc#rsa-1_5"/>
  <ds:KeyInfo>
    <ds:KeyName>
      CN=Hiroshi Maruyama, C=JP
    </ds:KeyName>
  </ds:KeyInfo>
  <xenc:CipherData>
    <xenc:CipherValue>
      d2FpbmdvbGRfE0lm4byV0...
    </xenc:CipherValue>
  </xenc:CipherData>
  <xenc:ReferenceList>
    <xenc:DataReference URI="#body"/>
  </xenc:ReferenceList>
</xenc:EncryptedKey>
```

Reference to
X509 Certificate
(contains public
key)

```
<S:Body>
  <xenc:EncryptedData
    Type="http://www.w3.org/2001/04/xmlenc#Element" Id="body">
    <xenc:EncryptionMethod
      Algorithm="http://www.w3.org/2001/04/xmlenc#3des-cbc"/>
    <xenc:CipherData>
      <xenc:CipherValue>d2FpbmdvbGRfE0lm4byV0...
      </xenc:CipherValue>
    </xenc:CipherData>
  </xenc:EncryptedData>
</S:Body>
```

WS-Security Example Message XML Signature

```
<ds:Signature>
  <ds:SignedInfo>
    <ds:CanonicalizationMethod
      Algorithm="http://www.w3.org/2001/10/xml-exc-c14n#" />
    <ds:SignatureMethod
      Algorithm="http://www.w3.org/2000/09/xmldsig#rsa-sha1" />
    <ds:Reference uri="body">
      <ds:DigestMethod
        Algorithm="http://www.w3.org/2000/09/xmldsig#sha1" />
      <ds:DigestValue>LyLsF094hPi4wPU...</ds:DigestValue>
    </ds:Reference>
  </ds:SignedInfo>
  <ds:SignatureValue>
    Hp1ZkmFZ/2kQLXDJbchm5gK...
  </ds:SignatureValue>
  <ds:KeyInfo>
    <wsse:SecurityTokenReference>
      <wsse:Reference URI="#X509Token" />
    </wsse:SecurityTokenReference>
  </ds:KeyInfo>
</ds:Signature>
```

Reference to
what is signed

The signature
itself

Reference back to
BinarySecurityToken

WS-Security in Workshop

Take a simplified, most applicable, path through WS-Security

Tokens

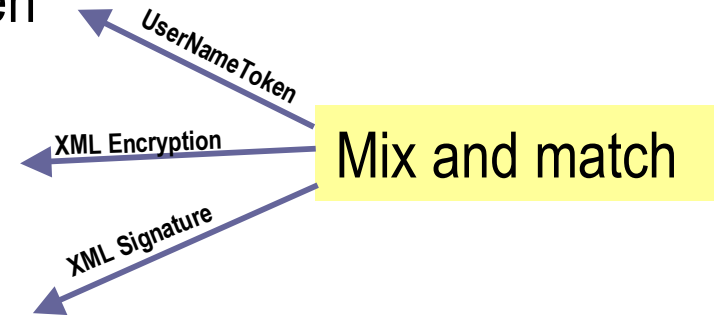
- UserNameToken
- BinarySecurityToken - X509Token*

XML Encryption

- Body
- Conversation Header

XML Signature

- Body
- Conversation Header



WS-Security Annotations

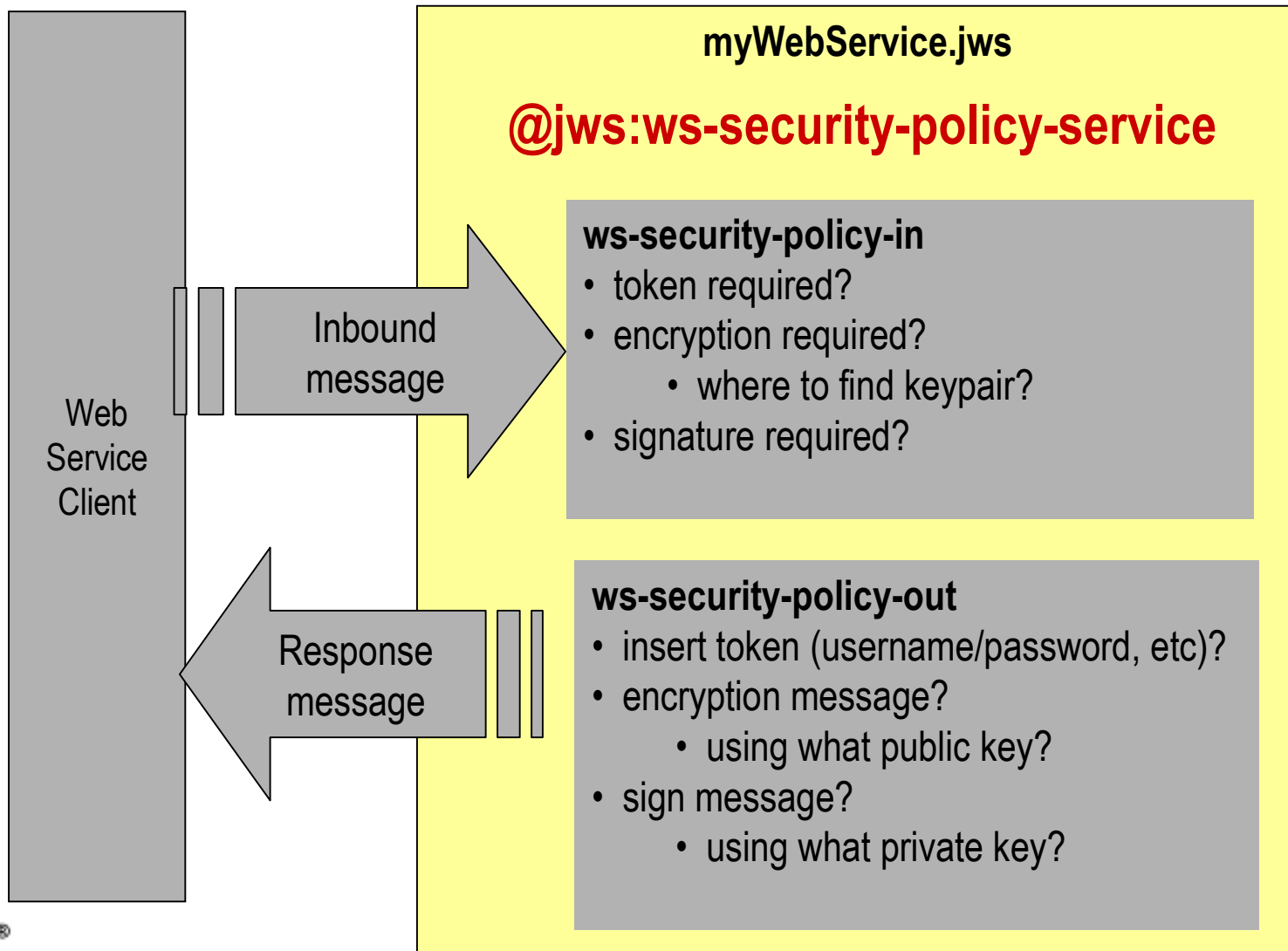
Two new annotations

`@jws:ws-security-policy-service file="mySecurityPolicy.xml"`

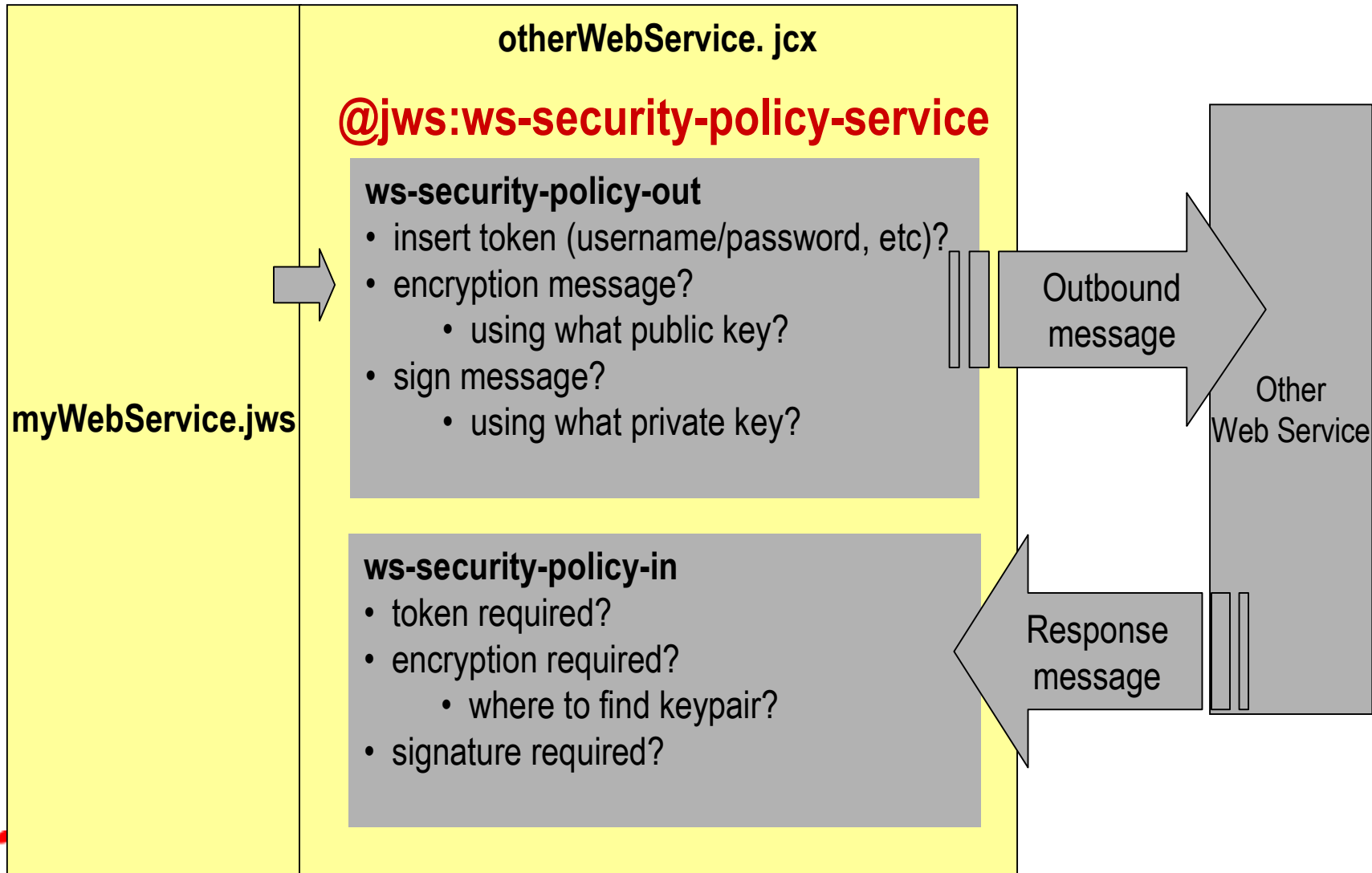
`@jws:ws-security-policy-callback
file="myCallbackSecurityPolicy.xml"`

```
<?xml version="1.0" encoding="UTF-8"?>
<wsSecurityPolicy xmlns="http://www.bea.com/2003/03/wsse/config"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.bea.com/2003/03/wsse/config
D:\dev\knex\src\com\bea\wlw\runtime\jws\wssecurity\config\WSSecurity-policy.xsd">
  <wsSecurityIn>
    <token tokenType="username"/>
    <encryptionRequired>
      <decryptionKey>
        <alias>myPublicKey</alias>
        <password>myPassword</password>
      </decryptionKey>
    </encryptionRequired>
    <signatureRequired>true</signatureRequired>
  </wsSecurityIn>
  <wsSecurityOut>
    <userNameToken>
      <userName>testUser</userName>
      <password type="TEXT">testPassword</password>
    </userNameToken>
    <encryption>
      <encryptionKey>
        <alias>myPublicKey</alias>
      </encryptionKey>
    </encryption>
    <signatureKey>
      <alias>myPrivateKey</alias>
      <password>myPrivateKeyPassword</password>
    </signatureKey>
  </wsSecurityOut>
  <keyStore>
    <keyStoreLocation>/myKeyStore.jks</keyStoreLocation>
    <keyStorePassword>myKeyStorePassword</keyStorePassword>
  </keyStore>
</wsSecurityPolicy>
```

Understanding @jws:ws-security-policy-service in a jws



Understanding @jws:ws-security-policy-service in a jbc



ws-security-policy file example

```

<wsSecurityPolicy xmlns="http://www.bea.com/2003/03/wsse/config">
  <wsSecurityIn>
    <token tokenType="username"/>
    <encryptionRequired>
      <deryptionKey>
        <alias>myPublicKey</alias>
        <password>myPassword</password>
      </deryptionKey>
    </encryptionRequired>
    <signatureRequired>true</signatureRequired>
  </wsSecurityIn>
  <wsSecurityOut>
    <userNameToken>
      <userName>testUser</userName>
      <password type="TEXT">testPassword</password>
    </userNameToken>
    <encryption>
      <encryptionKey>
        <alias>myPublicKey</alias>
      </encryptionKey>
    </encryption>
    <signatureKey>
      <alias>myPrivateKey</alias>
      <password>myPrivateKeyPassword</password>
    </signatureKey>
  </wsSecurityOut>
  <keyStore>
    <keyStoreLocation>/myKeyStore.jks</keyStoreLocation>
    <keyStorePassword>myKeyStorePassword</keyStorePassword>
  </keyStore>
</wsSecurityPolicy>

```

Specify inbound policy

Specify outbound policy

Optionally specify keystore



Message Level Security (WS-Security) Wrap-up

- Class level annotations, not operation level
- WSDL ws-security info
- Remember request and response are each soap messages
- SSL with username token
- Familiarity with PKI a must ...

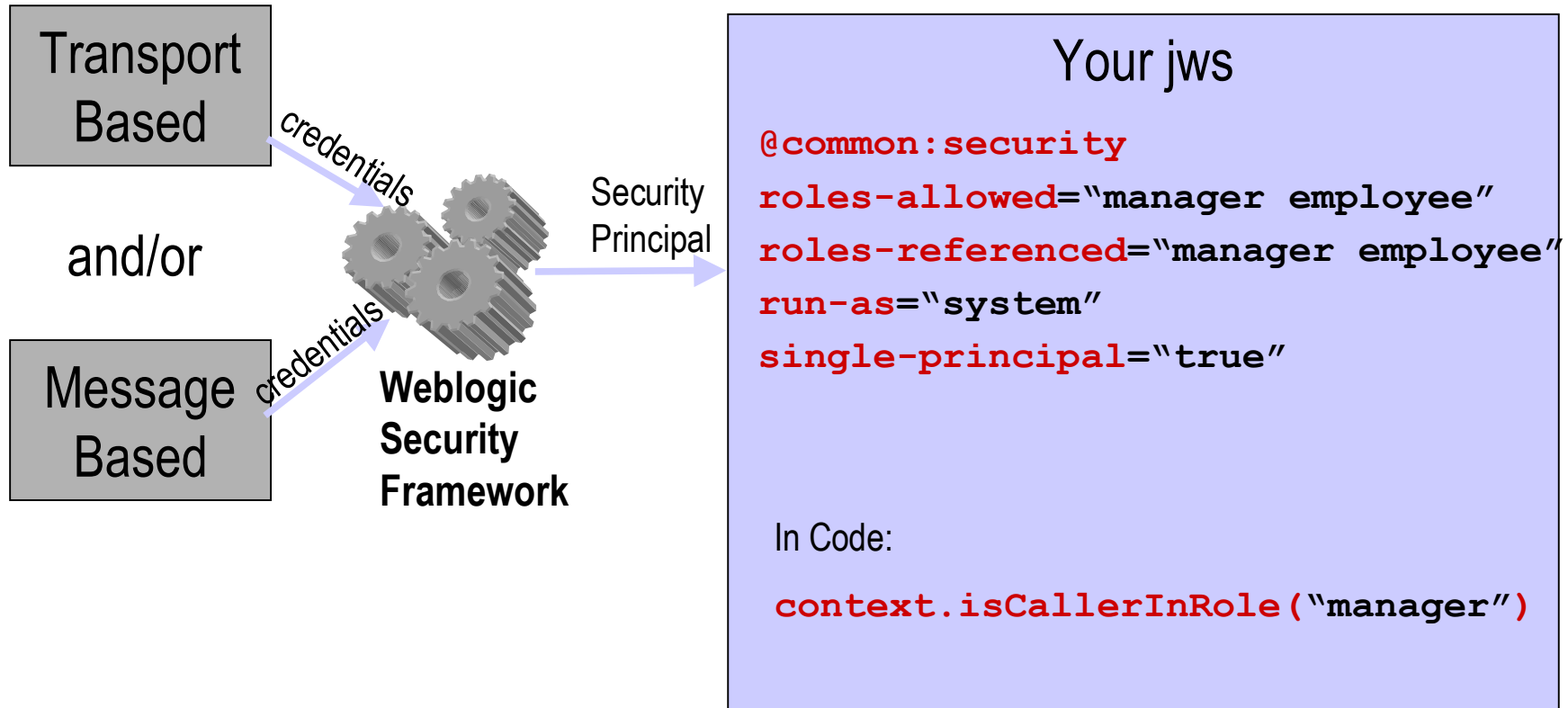


BEA **G** world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Role Based Security

Role Based Security



Using roles-allowed

```
@wlv:security roles-allowed="<role-name> [...<role-name>]"
```

- Optional (any role can access if not present)
- Class level (applies to **all** operations in the jws)
- JWS Operation level (union with class level roles-allowed)

Semantically Equivalent to EJB:

```
<method-permission>
```

```
  <unchecked/>
```

Semantically Equivalent to EJB:

```
<assembly-descriptor >
```

```
  <method-permission>
```

Semantically Equivalent to EJB:

```
<method-permission>
```

```
  <role-name>
```

```
    READONLY
```

```
  </role-name>
```

```
<method>
```

```
  <ejb-name>EmployeeBean</ejb-name>
```

```
  <method-name>getFirstName</method-name>
```

```
</method>
```

```
<method>
```

```
  <ejb-name>EmployeeBean</ejb-name>
```

```
  <method-name>getLastName</method-name>
```

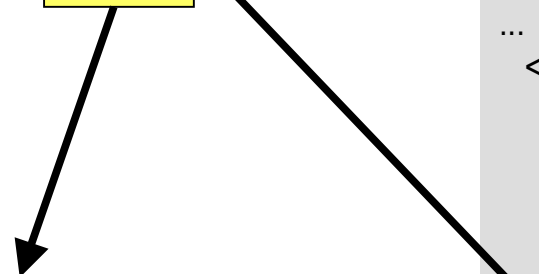
```
</method>
```

```
</method-permission>
```

Using roles-referenced and isCallerInRole()

```
@wlv:security roles-referenced="<role-name> [...<role-name>]"
```

The list of roles that have been referenced in your **code**.



Programmatic access to security info:

```
context.isCallerInRole("role-name")  
context.getCallingPrincipal().getName()
```

Objective:

Independence of code from target environment role names

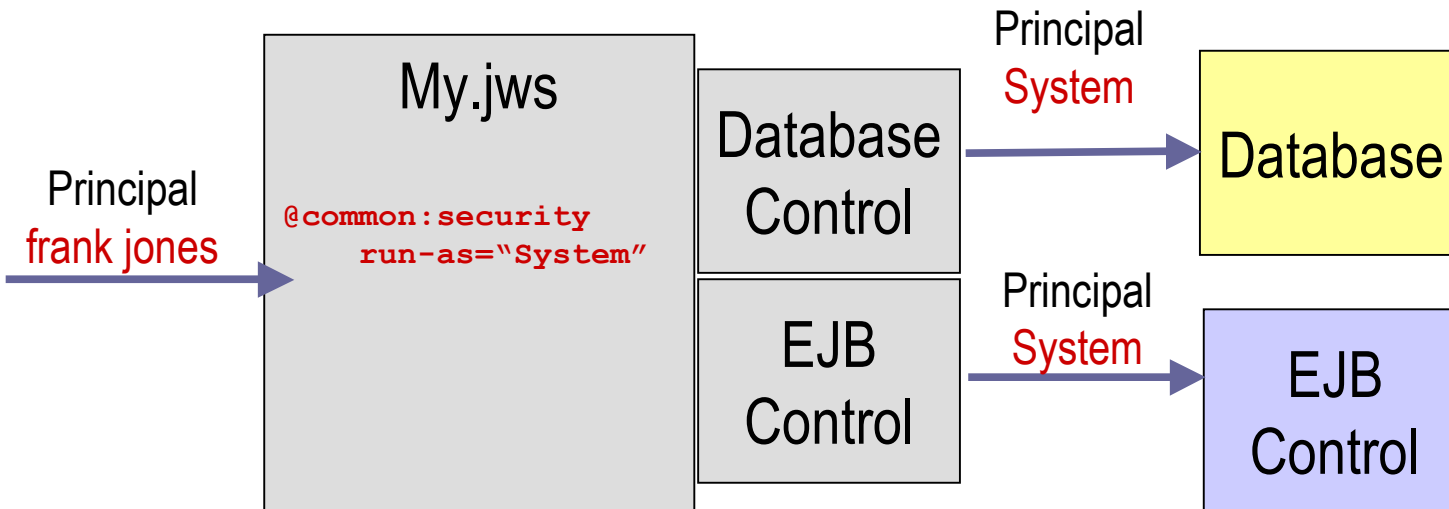
Semantically equivalent to EJB:

```
...  
<enterprise-beans>  
...  
<session>  
  <ejb-name>Op</ejb-name>  
  <ejb-class>sb.OpBean</ejb-class>  
  ...  
  <security-role-ref>  
    <role-name>role1</role-name>  
  </security-role-ref>  
  ...  
</session>  
...  
</enterprise-beans>  
...
```

Using run-as

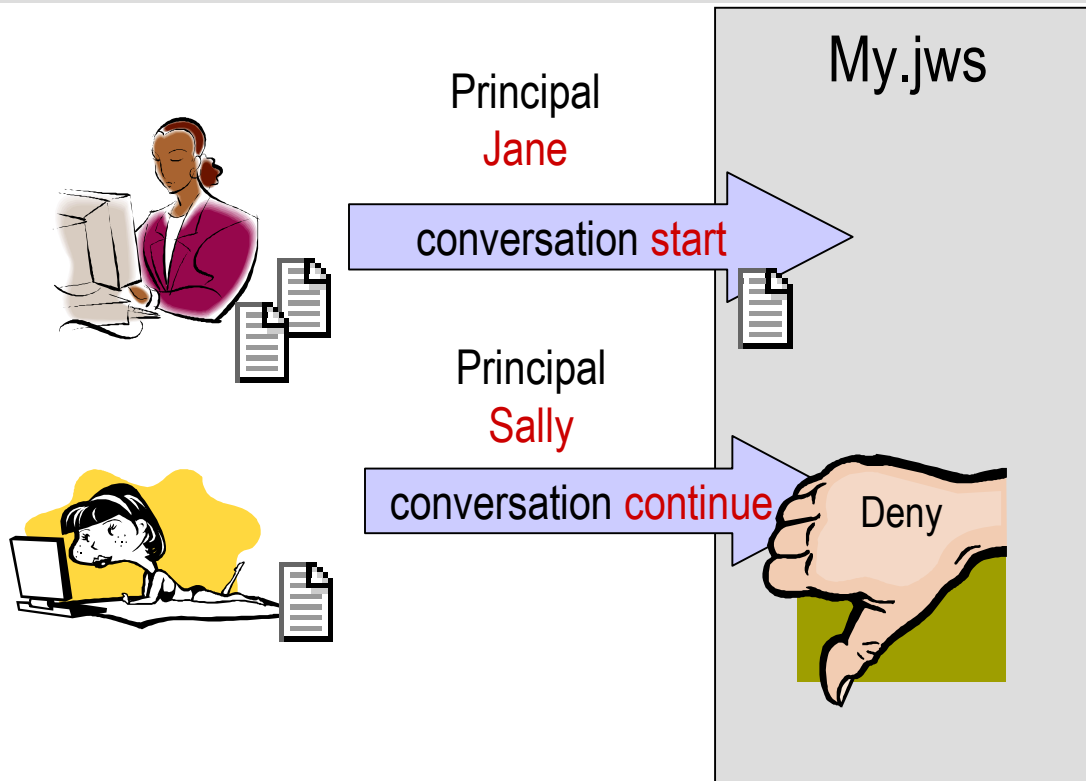
```
@wlw:security run-as="<role-name> [...<role-name>]"
```

The identity that a jws will run-as when it makes calls



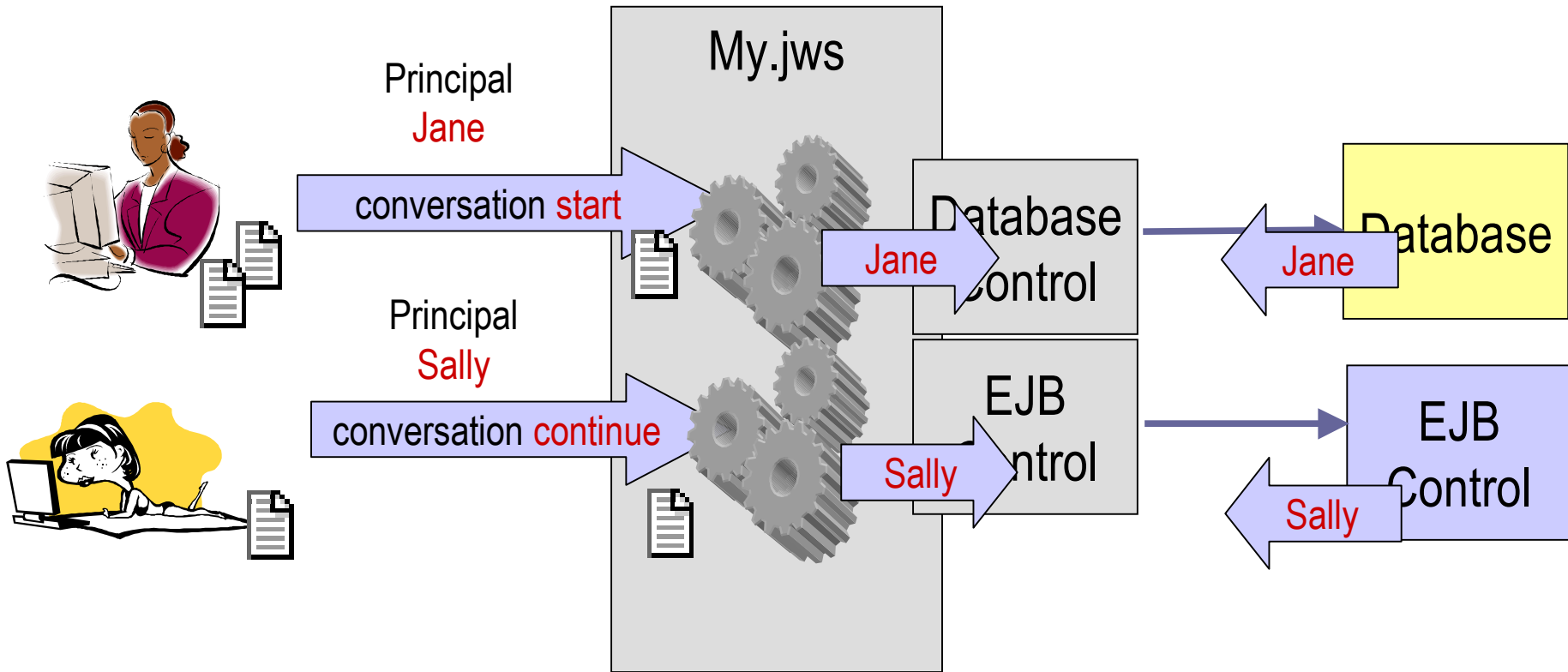
Conversations and security annotations single-principal

```
@wlv:security single-principal="true" | "false"
```



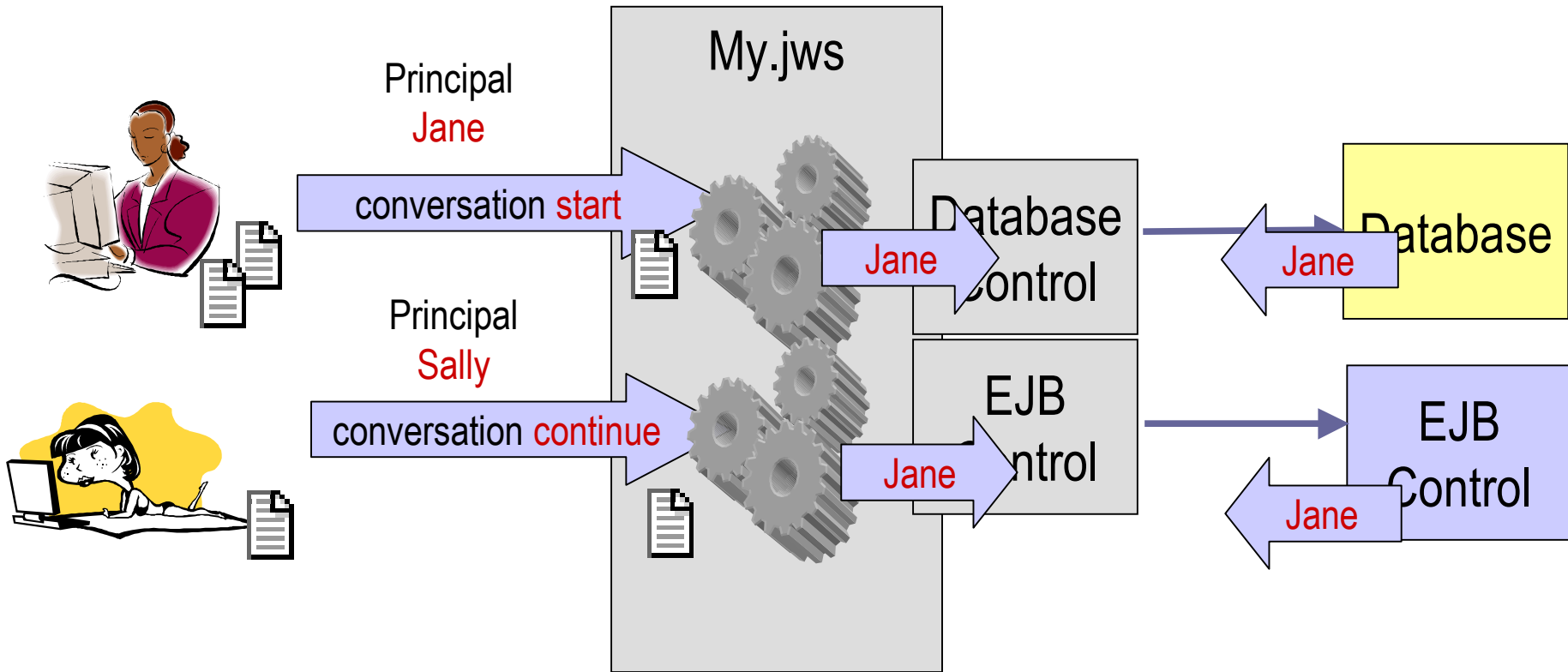
Conversations and security annotations run-as="<start-user>"

Without run-as="<start-user>"



Conversations and security annotations run-as="<start-user>"

With run-as="<start-user>"



Role Base Security Summary

- Role security -> **@common:security** annotation
- There are 4 role annotations:
 - roles-allowed
 - roles-referenced
 - run-as
 - single-principal
- You can also use `context.isCallerInRole("role")` within your code – but **remember to add to roles-referenced**

Workshop Security Summary

- Weblogic Server Security Framework takes care of most of the work to accept credentials and bind principal
- Three major workshop security mechanisms:
 - Transport Security
 - Message Level Security (WS-Security)
 - Role Based Security
- Transport security is mature and viable for many, if not most situations
- WS-Security is less mature but very flexible and powerful
- As a developer you must be aware of Transport and Message based security – but focus within your jws itself will be on roles

Best of luck!

Disclaimer

This information represents work in progress
This information is NOT a commitment by BEA
This information is subject to change

A large graphic in the top-left corner consisting of three overlapping circles in shades of red and orange. Small yellow triangles are positioned at the intersections of the circles.

BEA **G** world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Q&A



BEA **G** world 2003

THE 8TH ANNUAL BEA TECHNOLOGY CONFERENCE

Thank you!