Three Approaches to Securing Your JSF-Spring-Hibernate Applications

Ray Lai
Staff Software Engineer, Intuit

Jaya Doraiswamy
Senior Engineer, ELM Resources

TS-4514

Goal

Demonstrate three different approaches to secure your JavaServer™ Faces applications-Spring-Hibernate applications and explain when to use and why

Agenda

- Business Challenges
- Three Security Approaches
  - Approach #1 Container Security
  - Approach #2 JSF-Security
  - Approach #3 Acegi Security
- Demo
- Summary

Agenda

Business Challenges

Three Security Approaches

- Approach #1 Container Security
- Approach #2 JSF-Security
- Approach #3 Acegi Security

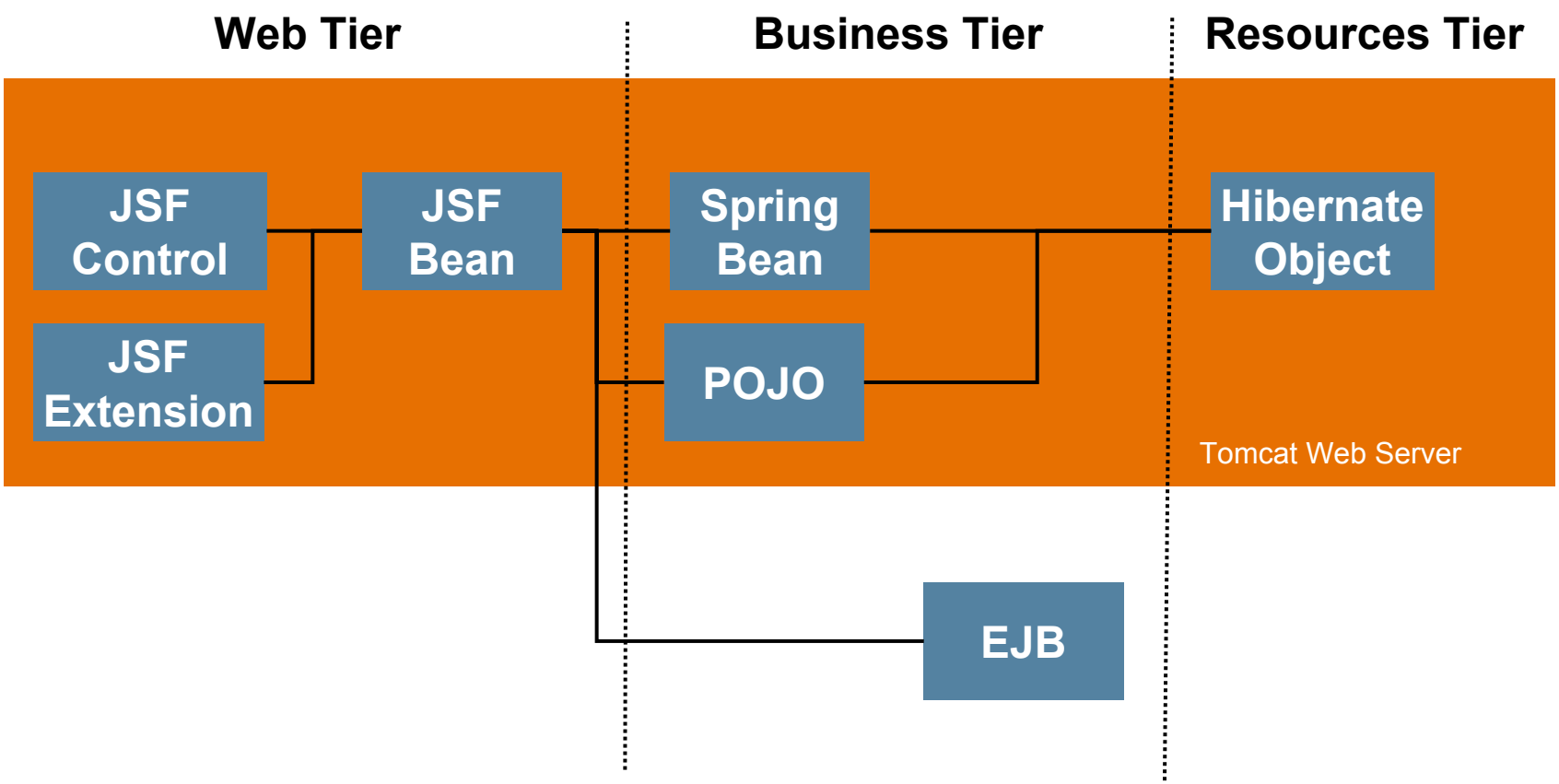
Demo

Summary

Business Challenges

- Why care?
  - 8,064 security vulnerabilities (2006)
  - US$52M financial loss due to security incidents (CSI/FBI 2006)
    - US$10.6M—Access control
- Who should care?
  - 4.5M+ Java™ platform developer base, 1M+ Spring and Hibernate developer base
- What is “secure”?
  - Security = ?
    - HTTPS basic authentication
    - Access control
  - Authentication + access control **not** equal secure
    - Why?

Scope-Application Architecture



Security (e.g. Authentication, Authorization) Should Be End-to-end, Across Tiers

Can These Make My Application Secure?

Quality Attributes (What and Why)	How to Protect	Security Vulnerabilities
Confidentiality	Authentication Authorization	Broken authentication Password management Broken access control Session management
Integrity	Cryptography (e.g. encryption) Digital signature	Input data validation Message replay Data protection
Accountability	Non-repudiation via logging Audit trail	Error handling Logging and auditing
Availability	High availability	Denial of service

Authentication and Access Control Are Parts of the "End-to-end" Security

http://www.owasp.org/index.php/Category:Vulnerability

Agenda

Business Challenges

Three Security Approaches

- Approach #1 Container Security
- Approach #2 JSF-Security
- Approach #3 Acegi Security

Demo

Summary

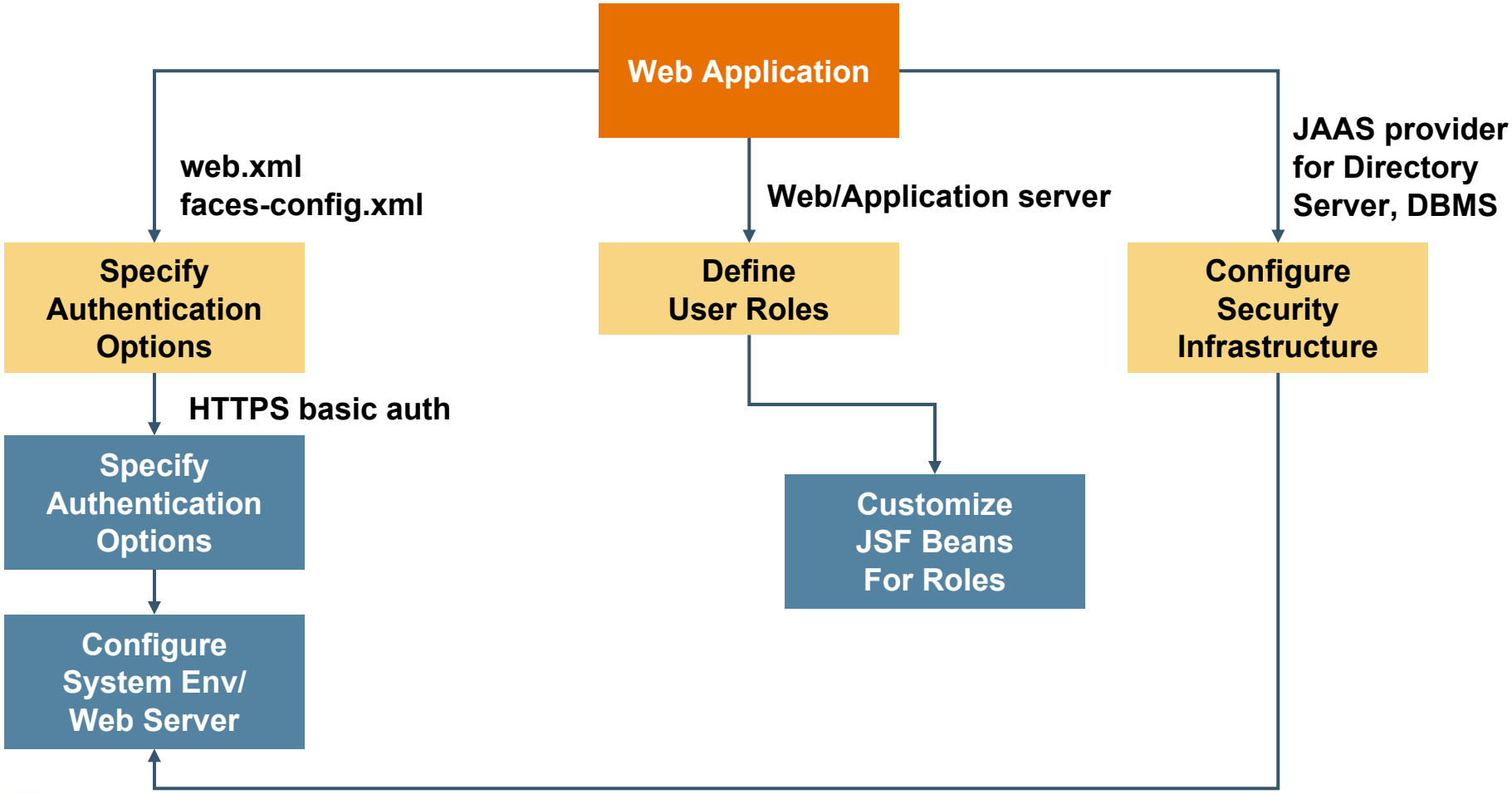
Container Security: What Is It?

- Features
  - Container security—Security infrastructure provided by
    - Web server (Web container)
    - Application server (EJB™ software container)
  - Multiple authentication mechanisms, e.g. HTTPS basic authentication
  - Security providers, e.g. Java Authentication and Authorization Service (JAAS), LDAP, RDBMS, SAML
  - Role-based security
- Benefits
  - Leverage common, centralized security infrastructure, without requiring developers to build their own
  - No need to re-invent the wheel
  - Scalable, robust security

JavaServer Faces Platform Using Container Security

- Scenario:
  - To add authentication and role-based access control to JavaServer Faces platform Web pages
- What to do:
  - Use form-based authentication
    - Why? HTTP/S basic or digest authentication is not sufficient
  - Use role-based security to control access
    - Why? Business data is accessible by roles, not by specific users

Adding Security to JavaServer Faces Application



Specifying Security in web.xml

```
<login-config>
    <auth-method>FORM</auth-method>
    <form-login-config>
        <form-login-page>/formLogin.jsp</form-login-page>
        <form-error-page>/errorLogin.jsp</form-error-page>
    </form-login-config>
</login-config>

<security-constraint>
    <web-resource-collection>
        <web-resource-name>form-based-authentication-
policy</web-resource-name>
        <url-pattern>/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
        <role-name>borrower</role-name>
    </auth-constraint>
</security-constraint>
```

1. Specify form-based authentication

2. Specify roles that can access the Web pages or resources

Sample web.xml

Defining User Roles in Web Container

- Defining users and roles by:
  - Server console
  - Config files

Weblogic

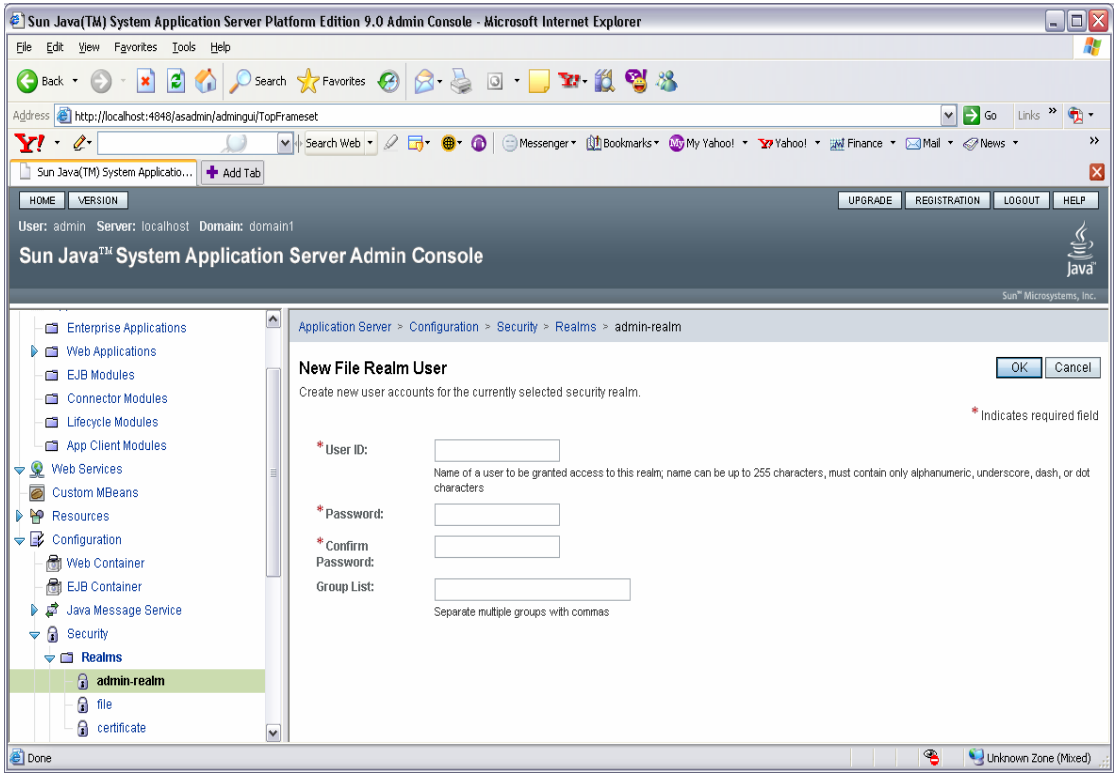
```
%BEA_HOME%\server\domain1\config
```

JBoss

```
%JBOSS_HOME%\server\default\conf\props\jbossws-roles.properties and
jbossws-users.properties
```

Tomcat

```
%TOMCAT_HOME%\conf\tomcat-users.xml
```



Note: Password in clear text in the config file

Tip #1: Role—Based Data Security

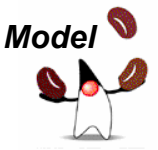
- Problem
  - Adding access control to specific data on JavaServer Faces pages
- Solution
  - Customize a managed bean (e.g. **`UserRoleBean`**) to retrieve username (**`getRemoteUser`**) and roles (**`isUserRoleIn`**) from the HTTP request context
  - Define roles in the Web container and in the web.xml
  - Use **`rendered`** attribute to control whether user can access data element in the JavaServer Faces platform Web page, e.g. render credit card number if role is administrator

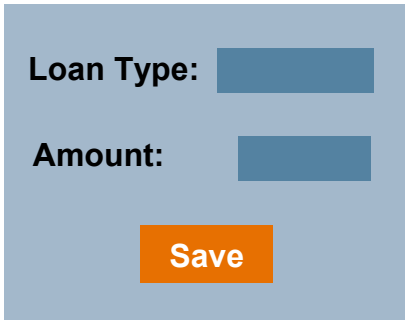
Tip #1: Role–Based Data Security

You want administrator role to add/save loan.
Otherwise, the "save" button won't show.

In your JSF managed bean:

```
public String addLoan() {
   // insert your processing logic
   return "save-loan";
}
```





addPersonalLoan.jsp

In your JSF page:

```
<h:panelGroup rendered="#{userRoleBean.borrower}">
  <h:commandButton value="Save"
     action="#{loanBean.addLoan}"
</h:panelGroup>
```

Tip #1: Role–Based Data Security

```
protected String username;
protected boolean borrower = false;

public String getUsername() {
    String username = FacesContext.getCurrentInstance().
        getExternalContext().getRemoteUser();
    return username;
}

public void getRemoteRole() {
   …
   if (FacesContext.getCurrentInstance().
      getExternalContext().isUserInRole("borrower")) {
      this.borrower = true;
     …
   } …
}
```

1. Get username

2. Derive user role

Tip #2: JavaServer Faces Platform With Form-Based Authentication

- Problem
  - Form-based login does not work well with JavaServer Faces platform pages
  - Symptoms: Blank page, link does not work
- Solution
  - Use `<verbatim>` tags to include HTML forms
  - Don't interleave JavaServer Faces platform form `<f:form>` and HTML `<form>`—They are not compatible

Tip #2: JavaServer Faces Platform With Form-Based Authentication

```
<f:verbatim>
      <form method="POST" action="./../../j_security_check">
      <h2>User Login</h2>
      <hr /> <br />
</f:verbatim>

<h:outputLabel for="username" id="username" value="Username:" />
<h:inputText id="j_username" />

<f:verbatim>
      <br />
</f:verbatim>

<h:outputLabel for="password" id="password" value="Password:" />
<h:inputSecret id="j_password" />

<f:verbatim>
      <br /><p />
</f:verbatim>
<h:commandButton id="loginButton" value="Login" />
<f:verbatim>
      </form> ...
```

1. Use <verbatim> to embed the HTTP form-based authentication (J_security_check)

2. Use JavaServer Faces platform tag <inputText> for uername and <inputSecret> for password

Don't Interleave HTTP <form> and JSF <f:form>

Container Security—Strengths and Limitation

Strengths	Limitations
• Centrally maintained • Support of JAAS login module various security providers (e.g. LDAP)	• Fine granularity • Support of non-Web/non-EJB architecture applications

Container Security—Bullet-Proof?

- Authentication
  - Depending on HTTP/S basic authentication
  - Depending custom JAAS provider
- Authorization
  - Business logic via POJO/Web services (not via JavaServer Faces platform pages)
- Managing security changes
  - What if roles are changed
  - Audit security changes
- Oops
  - Can some users read credit card # (or SSN#), and some cannot?
  - Handling dynamic roles
  - If I switch from Tomcat to Sun Java™ system or Weblogic, what changes do I need to make?

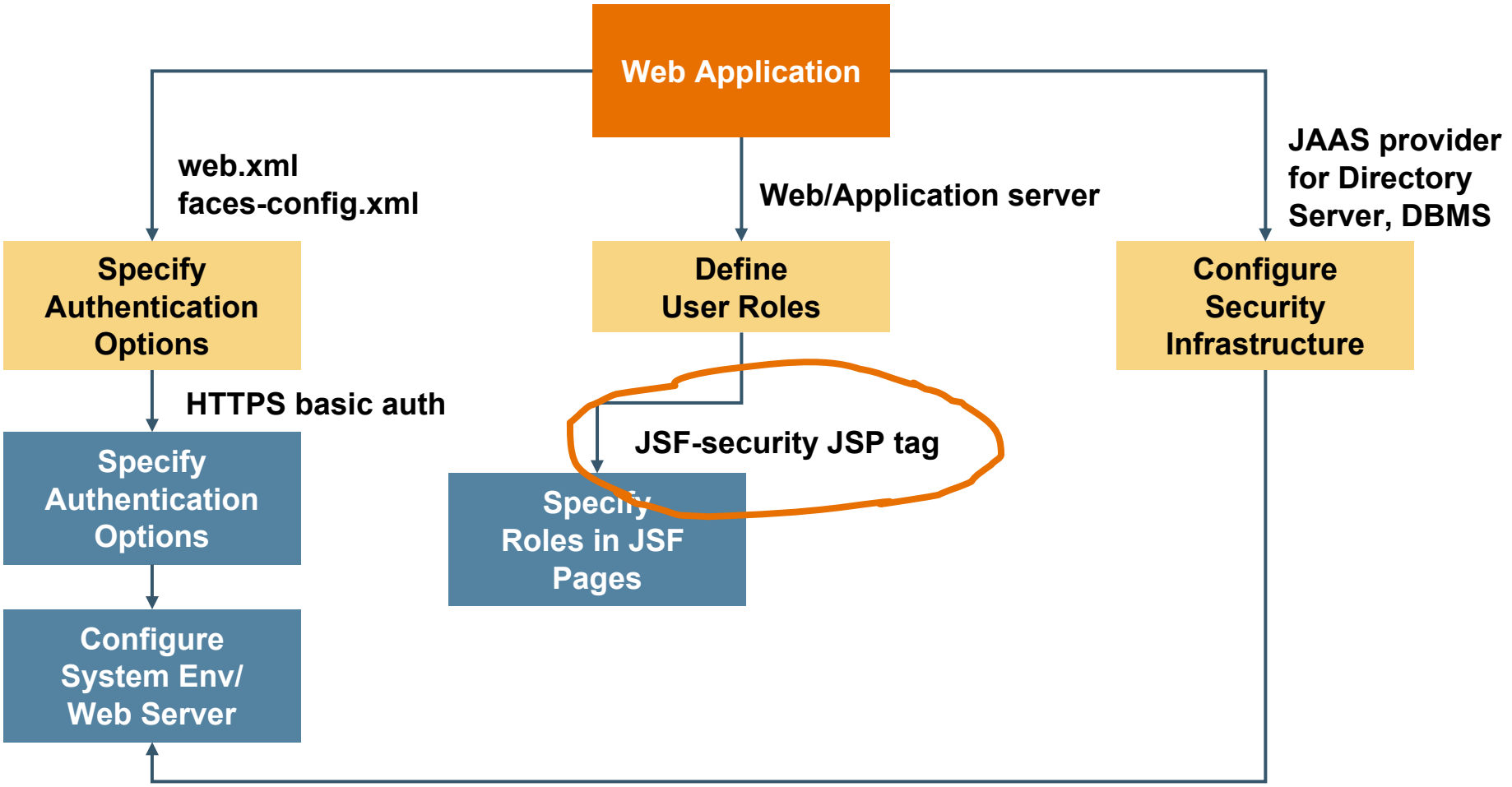
JSF-Security: What Is It?

- Features
  - Open source project to enable data level access control on a JavaServer Faces platform page by roles
  - Use of JavaServer Pages™ (JSP™) technology tag to provide data level access control
- Benefits
  - Leverage container security; no need to re-invent the wheel
  - Openness-Support of JSP technology tag

JSF-Security

- Scenario
  - To make certain data elements in JavaServer Faces platform Web pages accessible by specific user role
    - Data elements can be readable, editable or executable
    - Don't want to write different sets of JavaServer Faces platform pages or JavaServer Faces platform beans
- What to do
  - Add JavaServer Faces platform-security tag (**`securityScope`**) to the specific JavaServer Faces platform controls / data fields on the JavaServer Faces platform Web page
  - Server configuration—Add **`jsf-security.jar`** to your user lib (or server lib)

Adding Security Using JSF-Security



Example #1: JavaServer Faces Platform Page Using JSF-Security

```
<div class="appBody">
<h:outputText value="#{bundle['resBundle.loanType']}"
      rendered="#{(securityScope.userInRole['servicer']) &&
      userHandler.isUserStateNew}"/>
<h:outputText value="#{bundle['resBundle.ssn']}"
      rendered="#{(securityScope.userInRole['borrower'])
      && !userHandler.isUserStateNew}"/>
</div>

<h:commandLink styleClass="body"
      rendered="#{securityScope.userInRole['borrower']}"
      action="#{ploanBean.updateLoan}" immediate="true">
<h:outputText value="#{resBundle.apply}" />
</h:commandLink> |
```

You can reference to static user roles with simple condition

JSF-Security: Strengths and Limitations

Strengths	Limitations
• Leverage container security • Declarative role-based security for JavaServer Faces platform components	• Lack of dynamic user roles • Support of non-Web or non-EJB architecture applications

JSF-Security: Bullet-Proof?

- Authentication
  - Depending on HTTP/S basic authentication
  - Depending custom JAAS provider
- Authorization
  - Invoking business logic directly via POJO/Web services (not via JavaServer Faces platform pages)
- Managing security changes
  - What if roles are changed
  - Audit security changes
- Oops
  - What about user-defined roles that don't exist in the container?
  - Complex conditions cannot be handled in simple boolean logic
  - Whenever we add new user roles to the container, do we need to re-deploy the application, or re-boot the server?

Same as Container Security!

Acegi Security Framework

- What is it?
  - Open source project by Ben Alex (2003)
- Who uses it?
  - Spring developers, Mule, Fortify, etc.
- Why?
  - Provide comprehensive (web, non-web, non-EJB) security services
  - Separate the security processing logic from the business logic
- Features
  - Filters for securing Web Application
  - Use Aspect-oriented Programming (AOP) for securing Business Service Methods

Acegi Security Framework

Example: http://localhost:9990/acegidemo1/pages/loan.jsf

User login

index.jsp

invokes

FilterChainProxy

Checks for all filters to invoke

invokes

PathBasedFilterInvocation DefinitionMap

Check if the URL matches a pattern specified in config file

HttpSession

HttpSessionContextIntegrationFilter

Create new SecurityContext and associate it with SecurityContextHolder

AuthenticationProcessingFilter

SecurityContextHolder AwareRequestFilter

AnonymousProcessingFilter

Populate SecurityContextHolder with anonymous token

ExceptionTranslationFilter

FilterSecurityInterceptor

login.jsf

Acegi Security Framework

Objects Provided

- HttpSessionContext IntegrationFilter
  - Holds SecurityContext between HTTP request
- SecurityContext
  - Provides various implementation
- SecurityContextHolder
  - Holds current security context of the application
  - Has Authentication object
- UserDetailService
  - Interface that accept a String and return a instance of UserDetails
  - Builds the Authentication Object
- Authentication
  - Holds current Principal Object
  - Returns an array of GrantedAuthority objects
- UserDetails
  - Holds detail about the logged in user
- Granted Authorities
  - Holds information about various roles
  - Used for web, method, domain authorization

JavaServer Faces Platform Using Acegi Security Framework

- Scenario
  - User gets authenticated and authorized using `acegijsf` tag
    - Example: borrower (not lender) wants to access the loan page, or to change specific data
- What to do
  - Configure the acegi security context file with required Filters
  - On the page wrap the functionality using `acegijsf` tag

Example #2: JavaServer Faces Platform Page Using Acegi Security

```
<f:verbatim>
      Your username is:
</f:verbatim>

<acegijsf:authentication
    operation="username">
</acegijsf:authentication>

<acegijsf:authorize ifAllGranted="BORROWER">
    <f:verbatim> <br /> <p />
        Your role is borrower.
    </f:verbatim>
</acegijsf:authorize>
```

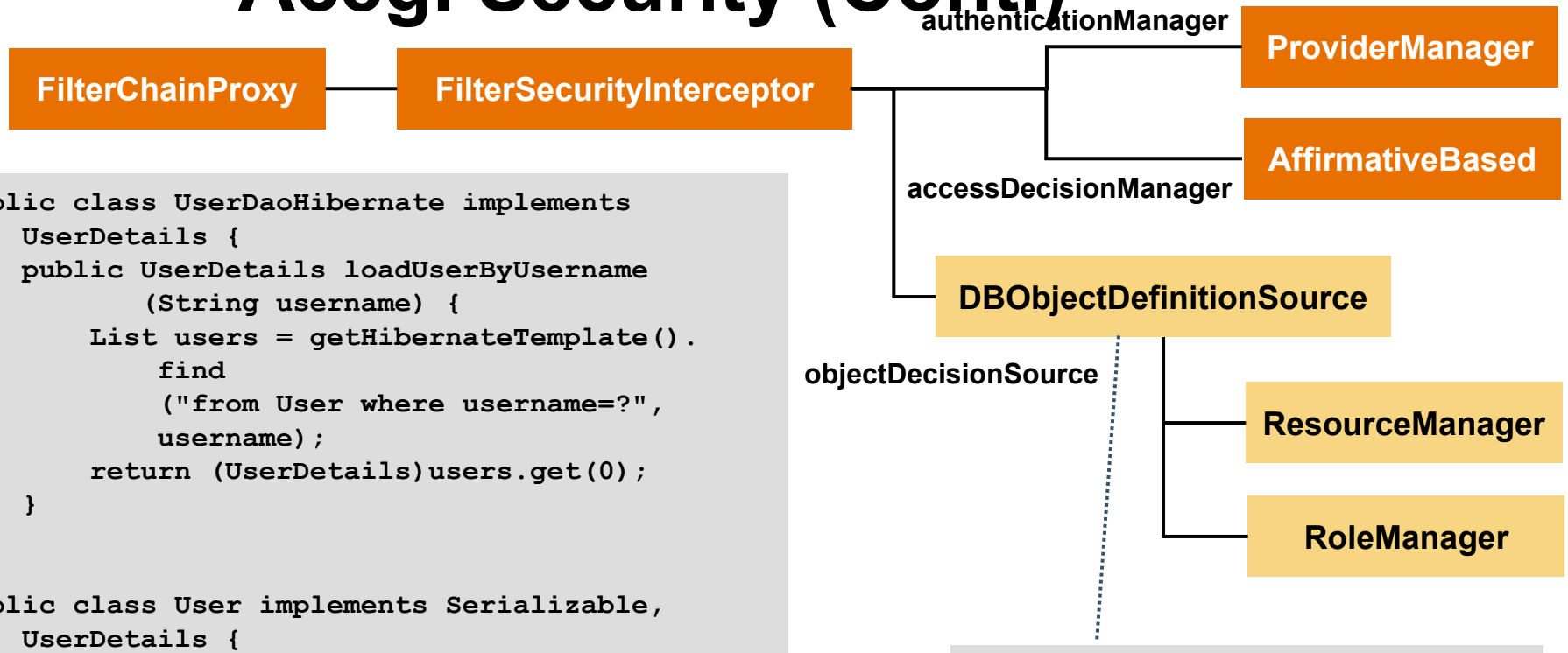
Sample output:

Your username is: mary
Your role is borrower

Tip #3: Dynamic Roles Using Acegi

- Problem
  - Admin defines custom roles, resources, users and associate user to roles and resources to roles at run-time
  - When user logs in, they can view only available resources
- Solution
  - Acegi can store/derive roles based on different conditions dynamically at run-time, instead of statically specify the roles at design time

Tip #3: Dynamic Roles Using Acegi Security (Cont.)




```
public class UserDaoHibernate implements
    UserDetails {
    public UserDetails loadUserByUsername
           (String username) {
        List users = getHibernateTemplate().
            find
            ("from User where username=?",
            username);
        return (UserDetails)users.get(0);
    }
}

public class User implements Serializable,
    UserDetails {
    public GrantedAuthority[] getAuthorities() {
        Set<Role> roles = getRoles();
        GrantedAuthority[] authorities =
            roles.toArray
            (new GrantedAuthority[0]);
        return authorities;
    }
}
```

extend **PathBasedFilterInvocationDefinitionMap**
Add the custom roles as a **SecurityConfig** object to **ConfigAttributeDefinition**
Override **addSecureUrl** method with your custom resources and roles

Tip #4: Securing Methods Using AOP

Problem

A database `saveXXX()` method can be called only by authorized user

It uses the Method Interceptor AOP concept which defines the saveUser method to only allow access to an ADMIN or BORROWER

Solution

```
<bean id="userManager" parent="txProxytemplate">
    ...
    <property name="preInterceptors">
        <list>
            <ref bean="userSecurityInterceptor" />
            <ref bean="userManagerSecurity" />
        </list>
    </property>
</bean>
<bean id="usermanagerSecurity"
      class="org.acegisecurity.intercept.method.
      aopalliance.MethodSecurityInterceptor">
      ...
      <property name="objectDefinitionSource">
          <value>
          com.service.UserManager.saveUser=BORROWER,ADMIN
          com.service.userManager.removeUser=ADMIN
          </value>
      </property>
</bean>
```

Acegi: Strengths and Limitations

Strengths

- Security bundled with application, without dealing with Java 2 Platform, Enterprise Edition (J2EE™ platform) realms
- High portability to different server environment
- Secure business method calls
- Separate security code from business logic
- Pluggable authentication models (e.g. LDAP, DAO)
- Rich authorization capabilities

Limitations

- A steep learning curve
- Spring dependent
- XML based configuration can get complicated to configure
- Single sign on through CAS

Agenda

Business Challenges

Three Security Approaches

- Approach #1 Container Security
- Approach #2 JSF-Security
- Approach #3 Acegi Security

Demo

Summary

DEMO

Agenda

Business Challenges

Three Security Approaches

- Approach #1 Container Security
- Approach #2 JSF-Security
- Approach #3 Acegi Security

Demo

Summary

Comparing Frameworks: Key Differentiators

How to Secure My Application	Container Security	JSF-Security	Acegi Security
System Set-up	Server-dependent, e.g. security realm	Server-dependent	Server independent
Granularity	Web-tier	Web-tier	Web-tier Business tier Domain objects
Flexibility	Static role	Static role	Dynamic role
Integration	Depends on security container	Depends on security container	LDAP, RDBMS, JAAS, CAS
Deployment	Within a security container	Within a security container	Deploy along with EAR/WAR file

When to Use

How to Secure My Application	Container Security	JSF-Security	Acegi Security
Roles	Static roles	Static roles	Dynamic roles
Portable between servers	Difficult	Difficult	Excellent
Support of multiple authentication mechanisms	Good	Good	Good
Adding access control to both Web pages and business methods	No	No	Excellent
Separate security code from business logic	No	Partially done using managed beans	Yes

Conclusion: Things to Remember

- Applying the right authentication and role-based security **not** equal “secure”
- Container security, JavaServer Faces platform-security and Acegi framework address different security problems
  - No simple answer to “which is the best” approach
  - Acegi-Spring Security provides a solution to secure business methods and domain objects
- How to make your codes vendor independence, e.g. separation of security logic from business logic

For More Information

Container Security

- http://java.sun.com/j2ee look for Java EE security topics

JavaServer Faces Platform-Security

- http://jsf-security.sourceforge.net/

Acegi

- http://www.acegisecurity.org/

Frameworks Comparison

- http://www.diva-portal.org/diva/getDocument?urn_nbn_no_ntnu_diva-1285-1__fulltext.pdf
- http://www.xenonsoft.com/resources/docs/JAVAWUG_JSF_SECURITY_QUICKIE.ppt

Q&A

Ray Lai–rayymlai@gmail.com

Jaya Doraiswamy–jayalaxmi_d@yahoo.com

Three Approaches to Securing Your JSF-Spring-Hibernate Applications

Ray Lai
Staff Software Engineer, Intuit

Jaya Doraiswamy
Senior Engineer, ELM Resources

TS-4514