

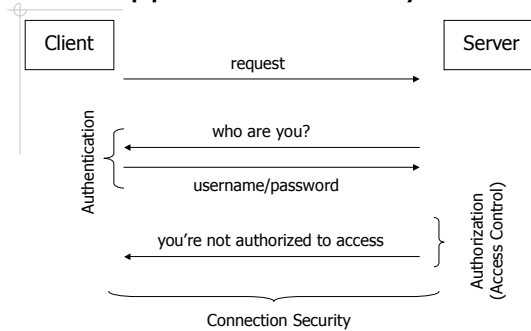
## CS520 Web Programming Declarative Security

Chengyu Sun  
California State University, Los Angeles

## Need for Security in Web Applications

- ◆ Potentially large number of users
- ◆ Multiple user types
- ◆ No operating system to rely on

## Web Application Security



## Connection Security

- ◆ Secure Socket Layer (SSL)
  - Server authentication
  - Client authentication
  - Connection encryption
- ◆ Transport Layer Security (TLS)
  - TLS 1.0 is based on SSL 3.0
  - IETF standard (RFC 2246)

## HTTPS

- ◆ HTTP over SSL
- ◆ Configure SSL in Tomcat - <http://tomcat.apache.org/tomcat-7.0-doc/ssl-howto.html>

## Programmatic Security

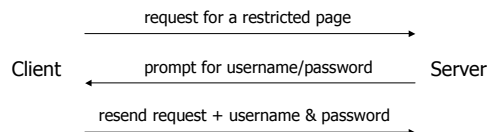
- ◆ Security is implemented in the application code
- ◆ Example:
  - `Login.jsp`
  - `Members.jsp`
- ◆ **Pros?? Cons??**

## Security by Java EE Application Server

- ◆ HTTP Basic
- ◆ HTTP Digest
- ◆ HTTPS Client
- ◆ Form-based

## HTTP Basic

- ◆ HTTP 1.0, Section 11.1-  
<http://www.w3.org/Protocols/HTTP/1.0/draft-ietf-http-spec.html>



## HTTP Basic – Configuration

```
AuthType Basic
AuthName "Basic Authentication Example"
AuthUserFile /home/cysun/etc/htpasswd
Require user cs520
```

## HTTP Basic – Request

```
GET /restricted/index.html HTTP/1.0
Host: sun.calstatela.edu
Accept: */*
```

## HTTP Basic – Server Response

```
HTTP/1.1 401 Authorization Required
Date: Tue, 24 Oct 2006 14:57:50 GMT
Server: Apache/2.2.2 (Fedora)
WWW-Authenticate: Basic realm="Restricted Access Area"
Content-Length: 484
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>401 Authorization Required</title></head>
...
</html>
```

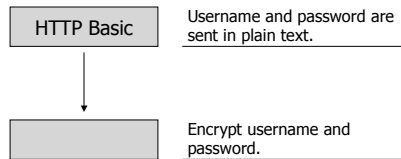
## HTTP Basic – Request Again

```
GET /restricted/index.html HTTP/1.0
Host: sun.calstatela.edu
Accept: */*
Authorization: Basic Y3lzdW46YWJjZAo=
```

↑  
Base64 Encoding of "cysun:abcd"

*An online Base64 decoder is at  
<http://www.opinionatedgeek.com/dotnet/tools/Base64Decode/>*

## Improve HTTP Basic (I)



## Cryptographic Hash Function...

- ◆ String of arbitrary length →  $n$  bits *digest*
- ◆ Properties
  1. Given a hash value, it's virtually impossible to find a message that hashes to this value
  2. Given a message, it's virtually impossible to find another message that hashes to the same value
  3. It's virtually impossible to find two messages that hash to the same value
- ◆ A.K.A.
  - *One-way hashing, message digest, digital fingerprint*

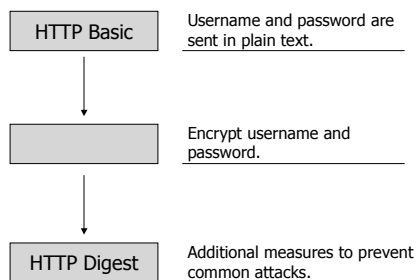
## ...Cryptographic Hash Function

- ◆ Common usage
  - Store passwords, software checksum ...
- ◆ Popular algorithms
  - MD5 (broken, partially)
  - SHA-1 (broken, sort of)
  - SHA-256 and SHA-512 (recommended)

## Encrypting Password is Not Enough

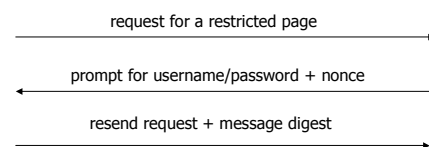
- ◆ *Why??*

## Improve HTTP Basic (II)



## HTTP Digest

- ◆ RFC 2617 (Part of HTTP 1.1) - <http://www.ietf.org/rfc/rfc2617.txt>



## HTTP Digest – Server Response

```
HTTP/1.1 401 Authorization Required
Date: Tue, 24 Oct 2006 14:57:50 GMT
Server: Apache/2.2.2 (Fedora)
WWW-Authenticate: Digest realm="Restricted Access Area",
  qop="auth,auth-int",
  nonce="dcd98b7102dd2f0e8b11d0f600bf0c093",
  algorithm="MD5",
  opaque="5ccc069c403ebaf9f0171e9517f40e41"
Content-Length: 484
Content-Type: text/html; charset=iso-8859-1

<!DOCTYPE HTML PUBLIC "-//IETF//DTD HTML 2.0//EN">
<html>
<head><title>401 Authorization Required</title></head>
... ..
</html>
```

## HTTP Digest – Request Again

```
GET /restricted/index.html HTTP/1.0
Host: sun.calstatela.edu
Accept: */*
Authorization: Digest username="cysun",
  realm="Restricted Access Area",
  nonce="dcd98b7102dd2f0e8b11d0f600bf0c093",
  uri="/restricted/index.html", qop=auth,
  nc=00000001, cnonce="0a4f113b",
  opaque="5ccc069c403ebaf9f0171e9517f40e41",
  algorithm="MD5"
  response="6629fae49393a05397450978507c4ef1"
```

Hash value of the combination of *username*, *password*, *realm*, *uri*, *nonce*, *cnonce*, *nc*, *qop*

## Form-based Security

- ◆ Unique to J2EE application servers
- ◆ Include authentication and authorization, but not connection security

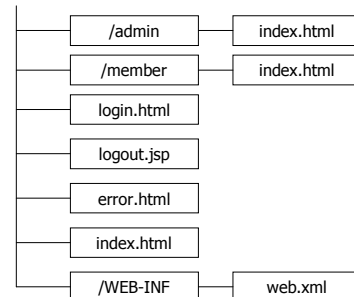
## Form-base Security using Tomcat

- ◆ \$TOMCAT/conf/tomcat-users.xml
  - Users and roles
- ◆ \$APPLICATION/WEB-INF/web.xml
  - Authentication type (FORM)
  - Login and login failure page
  - URLs to be protected

## Example – Users and Roles

```
<?xml version='1.0' encoding='utf-8'?>
<tomcat-users>
  <role rolename="admin"/>
  <role rolename="member"/>
  <user username="admin" password="1234"
    roles="admin,member"/>
  <user username="cysun" password="abcd"
    roles="member"/>
</tomcat-users>
```

## Example – Directory Layout



## Example – Login Page

```
<form action="j_security_check" method="post">
  <input type="text" name="j_username">
  <input type="password" name="j_password">
  <input type="submit" name="login" value="Login">
</form>
```

## Example – web.xml ...

```
<login-config>
  <auth-method>FORM</auth-method>
  <form-login-config>
    <form-login-page>/login.html</form-login-page>
    <form-error-page>/error.html</form-error-page>
  </form-login-config>
</login-config>
```

## ... Example – web.xml

```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>AdminArea</web-resource-name>
    <url-pattern>/admin/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
</security-constraint>
```

## Declarative Security

- ◆ Security constraints are defined *outside application code* in some metadata file(s)
- ◆ Advantages
  - Application server provides the security implementation
  - Separate security code from normal code
  - Easy to use and maintain

## Limitations of Declarative Security by App Servers

- ◆ Application server dependent
- ◆ Not flexible enough
- ◆ Servlet Specification only requires *URL access control*

## Security Requirements of Web Applications

- ◆ Authentication
- ◆ Authorization (Access Control)
  - URL
  - Method invocation
  - Domain object
  - View

## Spring Security (SS)

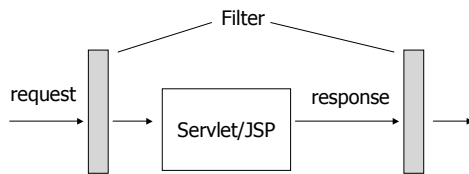
- ◆ A security framework for Spring-based applications
- ◆ Addresses all the security requirements of web applications

## How Does Spring Security Work

- ◆ Intercept requests and/or responses
  - Servlet filters
  - Spring *handler interceptors*
- ◆ Intercept method calls
  - Spring *method interceptors*
- ◆ Modify views
  - *Spring Security Tag Library*

## Servlet Filter

- ◆ Intercept, examine, and/or modify request and response

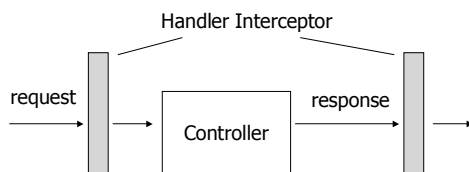


## Servlet Filter Example

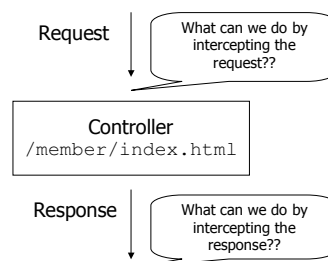
- ◆ web.xml
  - `<filter>` and `<filter-mapping>`
- ◆ Modify request
- ◆ Modify response

## Spring Handler Interceptor

- ◆ Serve the same purpose as servlet filter
- ◆ Configured as Spring beans, i.e. support dependency injection



## Intercept Request/Response



## Intercept Method Call



## Adding Spring Security to a Web Application ...

- ◆ Dependencies
  - spring-security-config
  - spring-security-taglibs
  - cglib

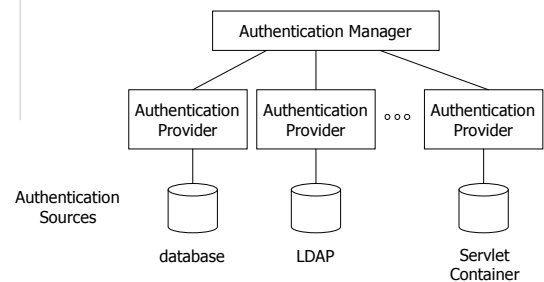
## ... Adding Spring Security to a Web Application

### ◆ web.xml

```
<filter>
  <filter-name>springSecurityFilterChain</filter-name>
  <filter-class>
    org.springframework.web.filter.DelegatingFilterProxy
  </filter-class>
</filter>

<filter-mapping>
  <filter-name>springSecurityFilterChain</filter-name>
  <url-pattern>/*</url-pattern>
</filter-mapping>
```

## Authentication



## Authentication Sources Supported

- ◆ Database
  - ◆ LDAP
  - ◆ JAAS
  - ◆ CAS
  - ◆ OpenID
  - ◆ SiteMinder
  - ◆ X.509
  - ◆ Windows NTLM
- ◆ Container-based
    - JBoss
    - Jetty
    - Resin
    - Tomcat

## Authenticate Against a Database – Configuration

### ◆ applicationContext.xml

```
<authentication-manager>
  <authentication-provider>
    <jdbc-user-service
      data-source-ref="dataSource" />
  </authentication-provider>
</authentication-manager>
```

## Authenticate Against a Database – Default Schema

```
create table users (  
  username string primary key,  
  password string,  
  enabled boolean  
);  
  
create table authorities (  
  username string references users(username),  
  authority string -- role name  
);
```

## Authenticate Against a Database – Customization

- ◆ `<jdbc-user-service>`
  - `users-by-username-query`
  - `authorities-by-username-query`
- ◆ `<authentication-provider>`
  - `<password-encoder>`
  - `user-service-ref`

## Implement Your Own UserDetailsService

- ◆ Implement a `loadUserByUsername` method
- ◆ Return an object that implements the `UserDetails` interface
- ◆ Example: CSNS2

## Authentication – Login Form and More

```
<http auto-config="true" />
```



```
<http>  
  <form-login />  
  <http-basic />  
  <logout />  
</http>
```

## Customize `<form-login>`

- ◆ `login-page`
- ◆ `authentication-failure-url`
- ◆ More at <http://static.springsource.org/spring-security/site/docs/3.1.x/reference/appendix-namespaces.html#nsa-form-login>

## Authorization Examples

- ◆ Users must log in to see the user list
- ◆ A user can only view/edit their own account
- ◆ An administrator can view/edit all accounts
- ◆ Only administrators can create new accounts
- ◆ Operations not available to a user should be hidden from the user



## Example: URL Security

- ◆ Users must log in to see the user list



ROLE\_USER is required to access  
/user/viewUsers.html

## URL Security

- ◆ applicationContext.xml

```
<http auto-config="true" use-expressions="true">  
  <intercept-url pattern="/user/viewUsers.html"  
    access="hasRole('ROLE_USER')" />  
</http>
```

## Pattern for <intercept-url>

- ◆ Default to ANT path pattern, e.g.

- /user/viewUsers.html
- /user/\*
- /user/\*\*
- /user/\*/\*.html
- /\*\*/\*.html

- Case-insensitive

## Spring Expression Language (SpEL)

- ◆ <http://static.springsource.org/spring/docs/current/spring-framework-reference/html/expressions.html>

## Security-Related SpEL Methods and Properties

- |                  |                      |
|------------------|----------------------|
| ◆ hasIpAddress() | ◆ anonymous          |
| ◆ hasRole()      | ◆ authenticated      |
| ◆ hasAnyRole()   | ◆ rememberMe         |
| ◆ permitAll      | ◆ fullyAuthenticated |
| ◆ denyAll        |                      |

<http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/org/springframework/security/web/access/expression/WebSecurityExpressionRoot.html>

## Example: Method Security

- ◆ A user can only edit their own account



A user may only invoke `userDao.saveUser()` if the `user` object to be saved has the same id.

## Enable Method Security

- ◆ applicationContext.xml

```
<global-method-security
  pre-post-annotations="enabled" />
```

## @PreAuthorize("SpEL expr")

- ◆ Allow method invocation if the SpEL expression evaluates to `true`
- ◆ Throw an `AccessDeniedException` if the expression evaluates to `false`

## More Security-Related SpEL Properties

- ◆ `authentication`
- ◆ `principal`
- ◆ Method parameter: `#<param_name>`
- ◆ Method return value: `returnObject`

## About authentication and principal

- ◆ The `Authentication` interface - <http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/org/springframework/security/core/Authentication.html>
- ◆ Usually `principal` is an object that implements the `UserDetails` interface - <http://static.springsource.org/spring-security/site/docs/3.1.x/apidocs/org/springframework/security/core/userdetails/UserDetails.html>

## Method Security

```
@PreAuthorize ("principal.username == #user.username")
public User saveUser( User user )
```

- ◆ Exercise: implement the following security constraints
  - An administrator can edit all accounts
  - Only administrators can create new accounts

## Example: Object Security

- ◆ A user can only view their own account



The `user` object returned by `userDao.getUserById()` must have the same id as the user invoked the method

## Object Security

```
@PostAuthorize ("principal.username == returnObject.username")  
public User getUserById( Integer id )
```

- ◆ Exercise: implement the following security constraints
  - An administrator can view all accounts

## Example: View Security

- ◆ Operations not available to a user should be hidden from the user

| ID | Name  | Operations                                     |
|----|-------|--|
| 1  | admin | <a href="#">Details</a>   <a href="#">Edit</a> |
| 2  | cysun | <a href="#">Details</a>   <a href="#">Edit</a> |
| 3  | jdoe  | <a href="#">Details</a>   <a href="#">Edit</a> |

## Security Tag Library

- ◆ <http://static.springsource.org/spring-security/site/docs/3.1.x/reference/taglibs.html>
- ◆ <authorize>
  - access
- ◆ <authentication>
  - property

## View Security

```
<security:authorize access="hasRole('ROLE_ADMIN')  
or principal.username == '${user.username}'">  
  <a href="viewUser.html?id=${user.id}">Details</a> |  
  <a href="editUser.html?id=${user.id}">Edit</a>  
</security:authorize>
```

## Implement Security Measures in Controller

- ◆ Don't do it, but if you have to ...

```
SecurityContextHolder  
    .getContext ()  
    .getAuthentication ()  
    ... ..
```

See SecurityUtils in CSNS2

## Conclusion

- ◆ Declarative security vs. Programmatic security
- ◆ Spring Security provides the best of both worlds
  - Declarative security framework
  - Portability and flexibility
  - Separate security code from regular code