

CS520 Web Programming

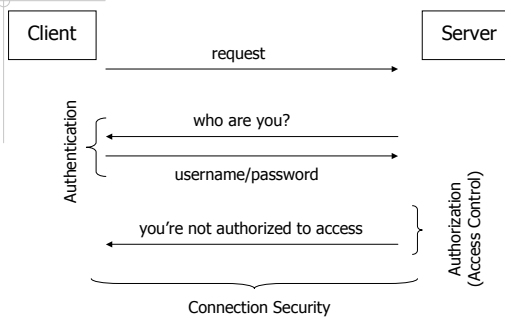
Declarative Security

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-6.0-doc/ssl-howto.html>

Programmatic Security

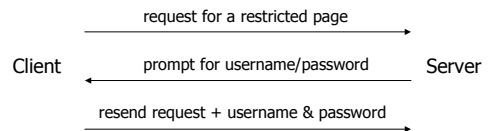
- ◆ Security is implemented in the application code
- ◆ Example:
 - `Login.jsp`
 - `Members.jsp`
- ◆ **Pros?? Cons??**

Security by J2EE 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>
```

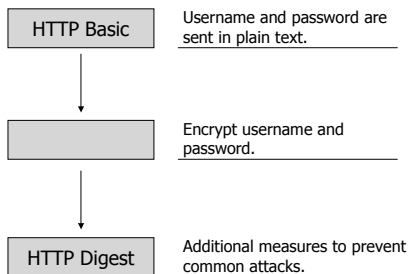
HTTP Basic – Request Again

```
GET /restricted/index.html HTTP/1.0
Host: sun.calstatela.edu
Accept: */*
Authorization: Basic Y3lzdW46YVJjZAo=
```

↑
Base64 Encoding of "cysun:abcd"

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

Improve HTTP Basic



Cryptographic Hash Function...

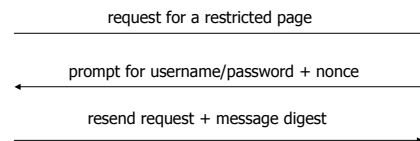
- ◆ String of arbitrary length → n bits *digest*
- ◆ Properties
 1. Given a hash value, it's virtually impossible to find a message that hashes to this value
 2. Given a message, it's virtually impossible to find another message that hashes to the same value
 3. It's virtually impossible to find two 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, sort of)
 - SHA-1 (expected to be broken soon)
 - SHA-256 and SHA-512 (recommended)

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="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  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="dcd98b7102dd2f0e8b11d0f600bfb0c093",
  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
- ◆ Username/password are passed as clear text
- ◆ Login page instead of login prompt

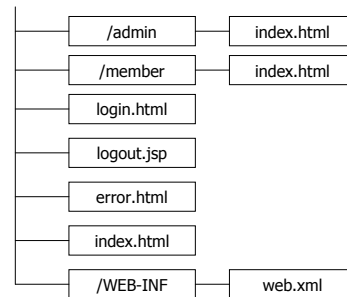
Form-base Security using Tomcat

- ◆ \$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"/>
  <role rolename="guest"/>
  <user username="cysun" password="abcd" roles="admin,member"/>
  <user username="test" password="test" roles="member"/>
  <user username="guest" password="guest" roles="guest"/>
</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
 - Domain object
 - Method invocation
 - ◆ Access to service layer, e.g. DAO
 - ◆ Access to web services

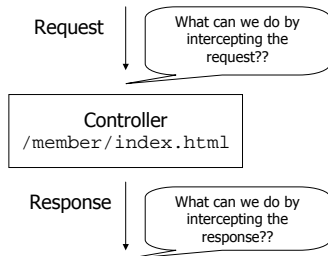
Spring Security (SS)

- ◆ A security framework for Spring-based applications
- ◆ Addresses all the security requirements of web applications
- ◆ Formerly known as Acegi Security
 - ABCDEFGHI

How Does Spring Security Work

- ◆ Intercept request and/or response
 - Servlet filters
 - Spring handler interceptors
- ◆ Intercept method calls
 - Spring method interceptors

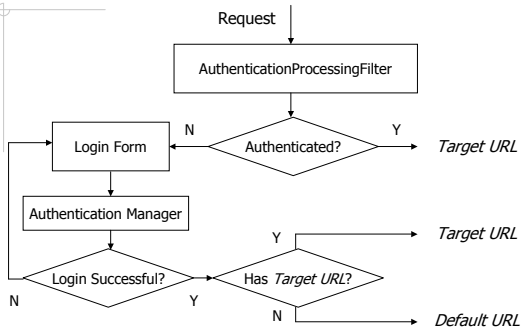
Intercept Request/Response



Intercept Method Call



Authentication Processing Filter



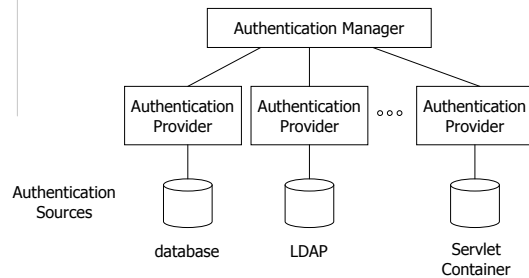
Login Form

- ◆ Action: `j_spring_security_check`
- ◆ Username: `j_username`
- ◆ Password: `j_password`

Configure Authentication Filter Beans

- ◆ `DelegatingFilterProxy` in `web.xml`
- ◆ In `spring-security.xml`
 - `springSecurityFilterChain`
 - `authenticationProcessingFilter`

Authentication Manager



Authentication Sources Supported

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

Authenticate Against a Database ...

- ◆ What SS expects your tables look like:

```
create table users (
  username string primary key,
  password string, -- encrypted
  enabled boolean
);
```

```
create table authorities (
  username string references users(username),
  authority string -- role name
);
```

... Authenticate Against a Database ...

users

username	password	enabled
'cysun'	md5('abcd')	't'
'jdoe'	md5('xyz')	'f'

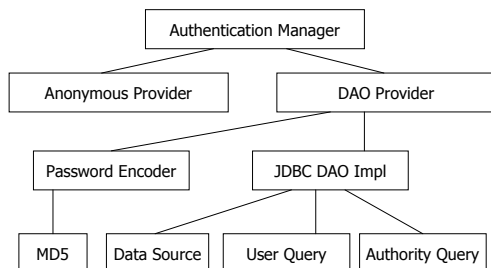
authorities

username	authority
'cysun'	'ROLE_ADMIN'
'cysun'	'ROLE_MEMBER'
'jdoe'	'ROLE_MEMBER'

... Authenticate Against a Database

- ◆ Define your own queries if your tables are different
 - usersByUsernameQuery
 - authoritiesByUsernameQuery

CSNS Example: Configure an Authentication Manager



Anonymous Authentication

- ◆ An anonymous user has their own credentials
 - AnonymousProcessingFilter
 - AnonymousAuthenticationProvider

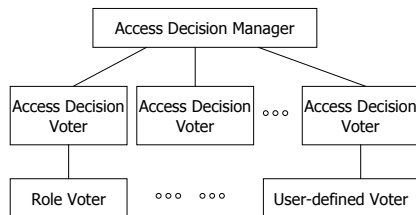
Access User Details in Application Code

- ◆ User details –
<http://static.springframework.org/spring-security/site/apidocs/org/springframework/security/userdetails/UserDetails.html>
 - Username
 - Password
 - Authorities (Roles)
- ◆ Example: `SecurityUtils` in CSNS

Authorization (Access Control)

- ◆ Secure URL access
- ◆ Secure method invocation
- ◆ Secure object access

Access Decision Manager



E.g. if a user is of Admin role, then grant access.

Types of Decision Managers

- ◆ Affirmative based
- ◆ Consensus based
- ◆ Unanimous based

How Decision Voter Works

- ◆ `AccessDecisionVoter` Interface
- ◆ Given
 - Object to be accessed
 - User information: username, roles
 - *Configuration attributes*, typically are roles names and/or access types like READ, WRITE etc.
- ◆ Return
 - `ACCESS_GRANTED`, or `ACCESS_DENIED`, or `ACCESS_ABSTAIN`

Secure URL Access

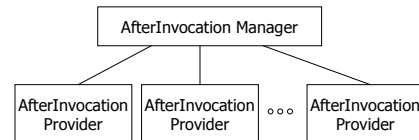
- ◆ `FilterSecurityInterceptor`
- ◆ CSNS Example:
 - Mapping from URL patterns to roles
 - `RoleVoter`

Secure Method Invocation

- ◆ `MethodSecurityInterceptor`
- ◆ CSNS Example
 - Mapping from method name patterns to roles
 - `RoleVoter`

Secure Object Access

- ◆ Implemented by checking the returned object of a method call
- ◆ Access decision is managed by `AfterInvocationManager`



Secure Object Access Example

- ◆ CSNS
 - `MethodSecurityInterceptor`
 - ◆ `AfterInvocationManager`
 - Customized `AfterInvocation` providers to provide application-specific access control
 - ◆ `SectionAccessVoter`
 - ◆ `AssignmentAccessVoter`
 - ◆ `SubmissionAccessVoter`
 - ◆ `FileAccessVoter`

Security Tag Library

- ◆ URI - <http://www.springframework.org/security/tags>
- ◆ `<authorize>`
 - `ifNotGranted`, `ifAllGranted`, `ifAnyGranted`
- ◆ `<authentication>`
 - `property`

Usage of the Security Tag Library

- ◆ CSNS Examples
 - `WEB-INF/jsp/surveys.jsp`
 - `WEB-INF/jsp/include/header.jspf`

Other Interesting Features of Spring Security

- ◆ Simplified namespace-based configuration syntax
- ◆ ACL based authorization
- ◆ Groups and hierarchical roles

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