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

<table>
<thead>
<tr>
<th>Client</th>
<th>Server</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>request</td>
<td></td>
</tr>
</tbody>
</table>

Authentication

- who are you?
- username/password
- you're not authorized to access

Authorization (Access Control)

Connection Security

HTTP Secure (HTTPS)

- HTTP over SSL/TLS

SSL and TLS

- 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)

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

  - request for a restricted page
  - prompt for username/password
  - resend request + username & password

HTTP Basic – Configuration

AuthType Basic
AuthName "Basic Authentication Example"
AuthUserFile /home/cysun/etc/htpasswords
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.base64decode.org/
Improve HTTP Basic (I)

HTTP Basic
Username and password are sent in plain text.
Encrypt username and password.

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)

Storing Passwords

- Why encrypting stored password??
- Common attacks on encrypted passwords
  - Brute force and some variations
  - Dictionary
- Common defenses
  - Long and random passwords
  - Make cryptographic hash functions slower
  - Salt

Encrypting Password is Not Enough

- Why??

HTTP Basic
Username and password are sent in plain text.
Encrypt username and password.

Improve HTTP Basic (II)

HTTP Basic
Username and password are sent in plain text.
Encrypt username and password.

HTTP Digest
Additional measures to prevent common attacks.
HTTP Digest

- RFC 2617 (Part of HTTP 1.1) - http://www.ietf.org/rfc/rfc2617.txt
  
  - request for a restricted page
  
  - prompt for username/password + nonce
  
  - resend request + message digest

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="dcd98b7102dd2f0e8b11d0f606b0c093",
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="dcd98b7102dd2f0e8b11d0f606b0c093",
uri="/restricted/index.html", qop=auth,
nc=00000001, cnonce="0a4f113b",
opaque="5ccc069c403ebaf9f0171e9517f40e41",
algorithm="MD5"
response="6629fae49393a05397450978507c4ef1"

Hash value of the combination of 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

```
/admin
  /member
  /admin
/login.html
/logout.jsp
/error.html
/index.html
/WEB-INF/web.xml
```

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

Controller
/member/index.html

Interceptor Method Call

BeforeAdvice
User getUserById(1)

AfterAdvice

Adding Spring Security to a Web Application ...

- Dependencies
  - spring-security-config
  - spring-security-taglibs

... 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 Sources Supported

- Database
- LDAP
- JAAS
- CAS
- OpenID
- SiteMinder
- X.509
- Windows NTLM

- Container-based
  - JBoss
  - Jetty
  - Resin
  - Tomcat

Authentication

- Authentication Manager
  - Authentication Provider
  - Authentication Provider

Authentication Sources

- database
- LDAP
- Servlet Container
Authenticate Against a Database – Configuration

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

Spring Security namespace:
http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security.xsd

Authenticate Against a Database – Default Schema

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

Authenticate Against a Database – Customization

```xml
<jdbc-user-service>
  <users-by-username-query>...
  <authorities-by-username-query>...
</jdbc-user-service>
```

```java
class User {
  String username;
  String password;
  boolean enabled;
  String email;
  Set<String> roles;
  Integer id;
}
```

Customize <jdbc-user-service>

```java
users
  id
  username
  password
  enabled
  email
```

```java
authorities
  user_id
  authority
```

Implement Your Own User Service

```java
Example: CSNS2
  User
    implements UserDetails interface
    UserDetailsServiceImpl implements
    UserDetailsService interface
    Use UserDetailsServiceImpl for the authentication provider
```
Access Authentication Information in Controller

SecurityContextHolder
   .getContext()
   .getAuthentication()
   .getPrincipal();

See SecurityUtils in CSNS2 for more examples

What is Principal?

Principal is an object representing the user who’s currently logged in
Principal implements the UserDetails interface – access to username, password, authorities etc.
Principal is not your own User object unless you implement your own user service

Authentication – Login Form and More

<http auto-config="true"/>

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

Default Login URLs and Parameters

/j_spring_security_check
/j_spring_security_logout
j_username
j_password

Customize <form-login>

login-page
authentication-failure-url
More at http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#nsc-login
Example: CSNS2

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/list.html

URL Security

- applicationContext.xml

  \[
  \text{<http auto-config="true" use-expressions="true">}
  \text{<intercept-url pattern="/user/viewUsers.html"}
  \text{access="hasRole(ROLE_USER)"/>}
  \text{</http>}
\]

Pattern for <intercept-url>

- Default to ANT path pattern, e.g.
  - /user/list.html
  - /user/*
  - /user/**
  - /user/***/*.html
  - /**/*.html
- Case-insensitive

Spring Expression Language (SpEL)

- [http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlexpressions.html](http://docs.spring.io/spring/docs/current/spring-framework-reference/htmlexpressions.html)

Security-Related SpEL Methods and Properties

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


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 username.
Enable Method Security

```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`

Method Security

```java
@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

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

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

Object Security
Example: View Security

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

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Operations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>admin</td>
<td>Details</td>
</tr>
<tr>
<td>2</td>
<td>cysun</td>
<td>Details</td>
</tr>
<tr>
<td>3</td>
<td>jdoe</td>
<td>Details</td>
</tr>
</tbody>
</table>

Security Tag Library

- [http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#taglibs](http://docs.spring.io/spring-security/site/docs/current/reference/htmlsingle/#taglibs)
- `<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>
```

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