CS 416
Web Programming

Security

Dr. Williams
Central Connecticut State University

Programming and Security

- **Programming Securely** To develop code in a secure manner so that the code itself is not a vulnerability that can be exploited by an attacker.
- **Programming Security** To develop code for security-specific functions such as encryption, digital signatures, firewalls, etc.

- In this lecture, we look at both sides:
  - continuing programming securely: some web application security issues and some Java guidelines.
  - programming security: overview of Java security APIs and trust models.

Overview

- **Web security issues**
- Java Security: Coding and Models
- Language futures for security

Web security: server-side threats

- **Access control**: should prevent certain files being served.
- Complex or malicious URLs
- Denial of service attacks
- Remote authoring and administration tools
- Buggy servers, with attendant security risks
- Server-side scripting languages: C or shell CGI, PHP, ASP, JSP, Python, Ruby, all have serious security implications in configuration and execution. File systems and permissions have to be carefully designed. *That's before any implemented web application is even considered. . .*

Web programming: application security

Many issues

- **Input validation**: to prevent SQL injection, command injection, other confidentiality attacks.
- **Ajax**: beware client-side validation! Understand metacharacters at every point. Use labels/indexes for hidden values, not values themselves.
- **Output filtering**: Beware passing informative error messages.
- **Careful cryptography**: encryption/hashing to protect server state in client, use of appropriate authentication mechanisms for web accounts

Overview

- Web security issues
- XSS and SQL injection
- **Java Security: Coding and Models**
- Language futures for security

Cross site scripting (XSS)

- Inserting code to be run on target server or pages returned by target server
- Common way unprotected database inserts
- Steal cookies, key logging, passwords, credit card, phishing , etc

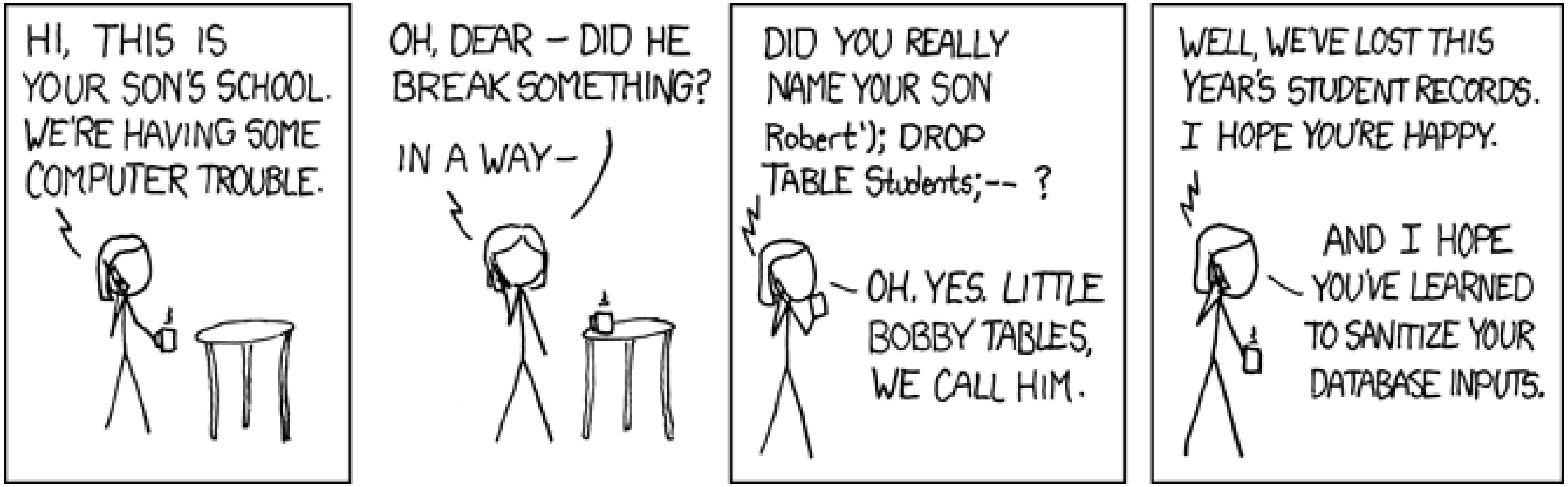
- See code demo

SQL injection

- Inserting SQL to be run in existing SQL calls to database
- Common way unprotected database selects, inserts, updates, deletes
- Insert/update/delete records, potentially drop tables
- Return information you shouldn’t be able to access

- See code demo

XKCD


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Java Secure Coding Guidelines

- Using modifiers
  - Reduce scope of methods and fields
  - Beware non-final public static (global) variables
  - Avoid public fields
  - Add security checks to public accessors.

Java security extensions

Java Cryptography Extension (JCE)

- A Java framework for cryptographic functionality, including message digests, encryption, signing, and X.509 certificates.

Java Secure Socket Extension (JSSE).

Java Authentication and Authorization Service (JAAS)

- Used for “reliable and secure” authentication of users, to determine who is currently executing Java code; and for authorization of users to ensure they have the permissions necessary for desired actions.

Java GSS-API.

- Bindings for Generic Security Service API (RFC2853). Used for securely exchanging messages between communicating applications, using various underlying mechanisms (e.g., Kerberos).

Java Cryptography Extension (JCE)

- Crypto framework.
  - A provider plug-in architecture allows multiple simultaneous implementations.
- Has algorithm independence
  - clients don’t need to understand algorithms; abstract “engine” classes provide different services.
- Service provider interfaces (SPIs)
  - added statically or dynamically; clients query installed providers to find out supported services. JVM and clients specify preference orders.
- Key management is through a “keystore” database.
  - Different providers may have different formats.
  - SUN provider implements common formats and proprietary keystore type JKS.
- See: javax.crypto, javax.crypto.interfaces, javax.crypto.spec.

JCE cryptography services

- A cryptography service is associated with a particular algorithm or type, and manipulates or generates data, keys, algorithm parameters, keystores, or certificates.
- Engine classes include:
  - **MessageDigest** generate message digests (MDCs)
  - **Signature** sign data and verify digital signatures.
  - **KeyPairGenerator** generate public-private key-pair.
  - **CertificateFactory** create certificates and CRLs.
  - **KeyStore** create and manage key databases.
  - **AlgorithmParameters** manage parameters for an algorithm.
  - **SecureRandom** random or pseudo-random numbers.
- Factory methods in engine classes are used to return instances of the class, e.g. Signature.getInstance("SHA1withDSA").

Java Secure Socket Extension (JSSE)

- The JSSE is also based on a provider plug-in architecture.
- Has a simple structure. Main use is with SSL client sockets, SSL server sockets, and SSL session handles. Sample classes:
  - **SSLSocket** socket for SSL/TLS/WTLS protocols
  - **SSLSocketFactory** factory for SSLSocket objects
  - **SSLServerSocket** sever socket for SSL/TLS/WTLS
  - **. . . Factory** factory for SSLServerSockets
  - **SSLSession** encapsulation of SSL session
- Creating SSL client or server sockets is as easy as creating ordinary Java TCP/IP sockets: each SSL class extends the corresponding ordinary TCP socket class, and provides a few extra hooks for setting security parameters.
- See javax.net.ssl, also javax.net and javax.security.cert.

Authentication and Authorization (JAAS)

- JAAS has a pluggable architecture; applications independent of underlying authentication methods. Implementation is decided at runtime, in a **login configuration file**.
- A **Subject** may have multiple identities; each is a **Principal** (name). Subjects own public and private **credentials** (e.g., key material).
- To authenticate, a **LoginContext** object is created, which then consults a configuration to load the required **LoginModules**. To authenticate a subject the login method is invoked for each module.
- **Authorization** happens when a subject is associated with a thread’s **AccessControlContext** using the **doAs** methods for performing actions (java.security.PrivilegedAction.run). Then principal-based entries in the current security policy are used.

JAAS Key elements

- Authentication framework
- Assertion of identity
- Enhanced authorization
- Low level of binding between authentication and authorization

Authentication framework

- Authentication framework
  - Policy-based
  - Generic and abstract
  - Pluggable, stackable
- Key abstractions
  - Subject - any user of computing
    - Collection of Principals, credentials
  - Principal (java.security)
    - Has a name

Identity and Authorization

Assertion of identity

- Avoids incompatible behavior
- Lexically scopes identity
- Logically associates Subject with current Thread
- static Object Subject.doAs(Subject, action)

Enhanced Authorization

- Augmentation of current Permission specification
- Principal-based
- Any authentication with any Permission

JAAS Summary

- JAAS extends base security model to accommodate concept of "users"
- Pluggable, extensible
- Provides a powerful and easy to use extension of application security

Detailed look at JAAS

- Java Authentication and Authorization Service
  - Framework for simplifying and standardizing access to system
  - Setting up users
  - Setting up security groups
  - Setting up security *realms*
- *Security realms* are crux of JAAS

Security realms cont.

THE biggest production risk is poorly implemented security

- Goal eliminate custom programming that leads to inadvertent security holes

Realms framework

- Store username, password, security groups
- Applications don’t need to implement security
- Security is configuration, can be shared across applications

Security realms

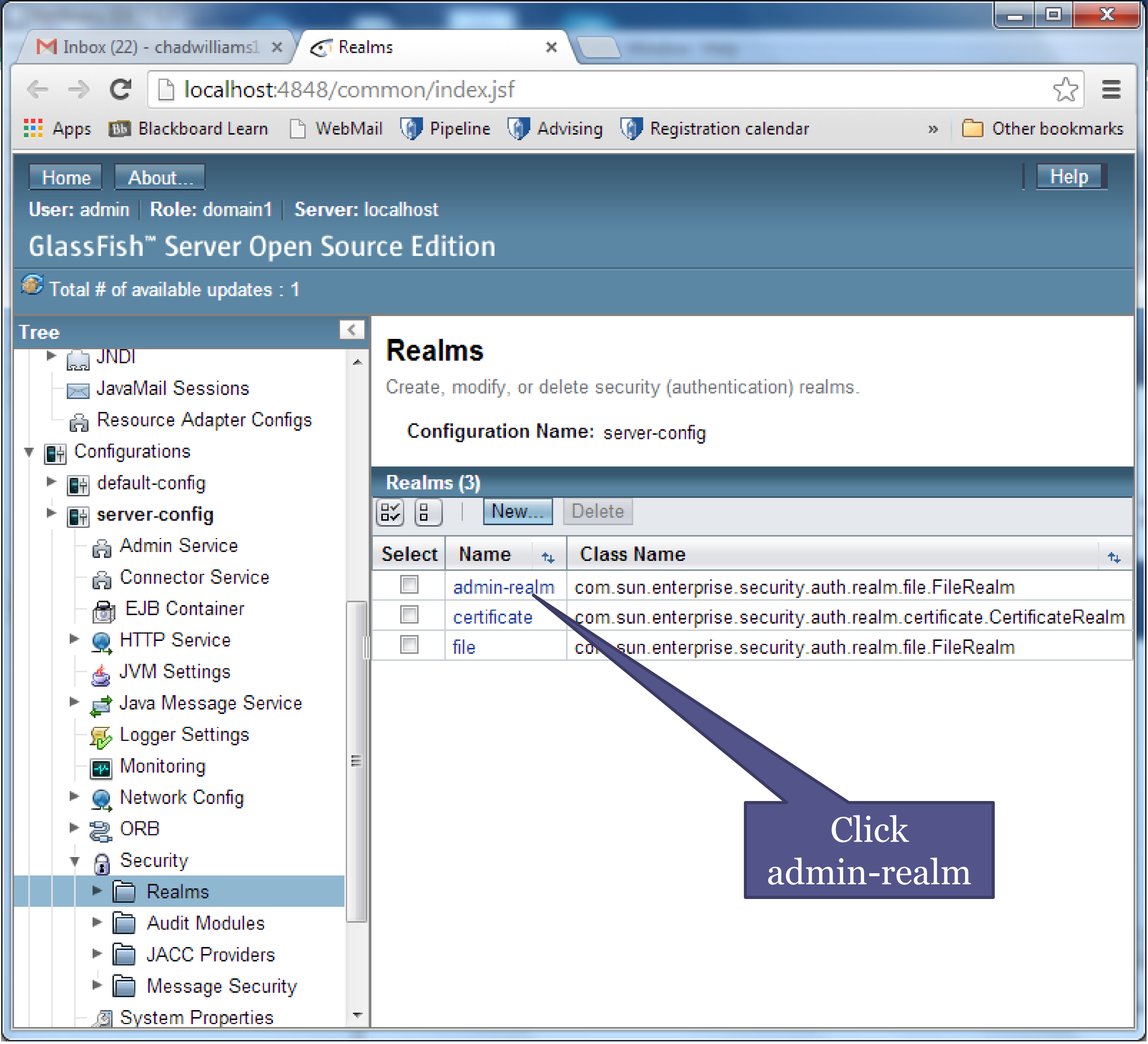
- Collections of users and related security groups
- User can belong to more than one group
- Groups define what actions the system will allow the user to do
    - Unauthenticated access to certain pages
    - Authenticated users an additional set of pages
    - Admin ability to perform actions like adding users, or ability to log in to administer application server

JAAS in practice

- **Goal eliminate custom programming** – remember we could control access with filters, but required custom solution and a lot of work to make it flexible enough for all situations. All of these add potential for security holes
- **Instead implement using JAAS**
    - Admin realm
    - File realm
    - Certificate realm
    - LDAP realm
    - JDBC realm
    - Custom realms

Glassfish predefined realms

- Admin-realm
  - Access to Glassfish web console
- File realm
  - Default realm created for controlling access to just authenticated users
- Certificate realm
  - Client side certificate authorization
    - Need for strong guarantees of users or server connecting



User: admin | Role: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Total # of available updates : 1

Realms

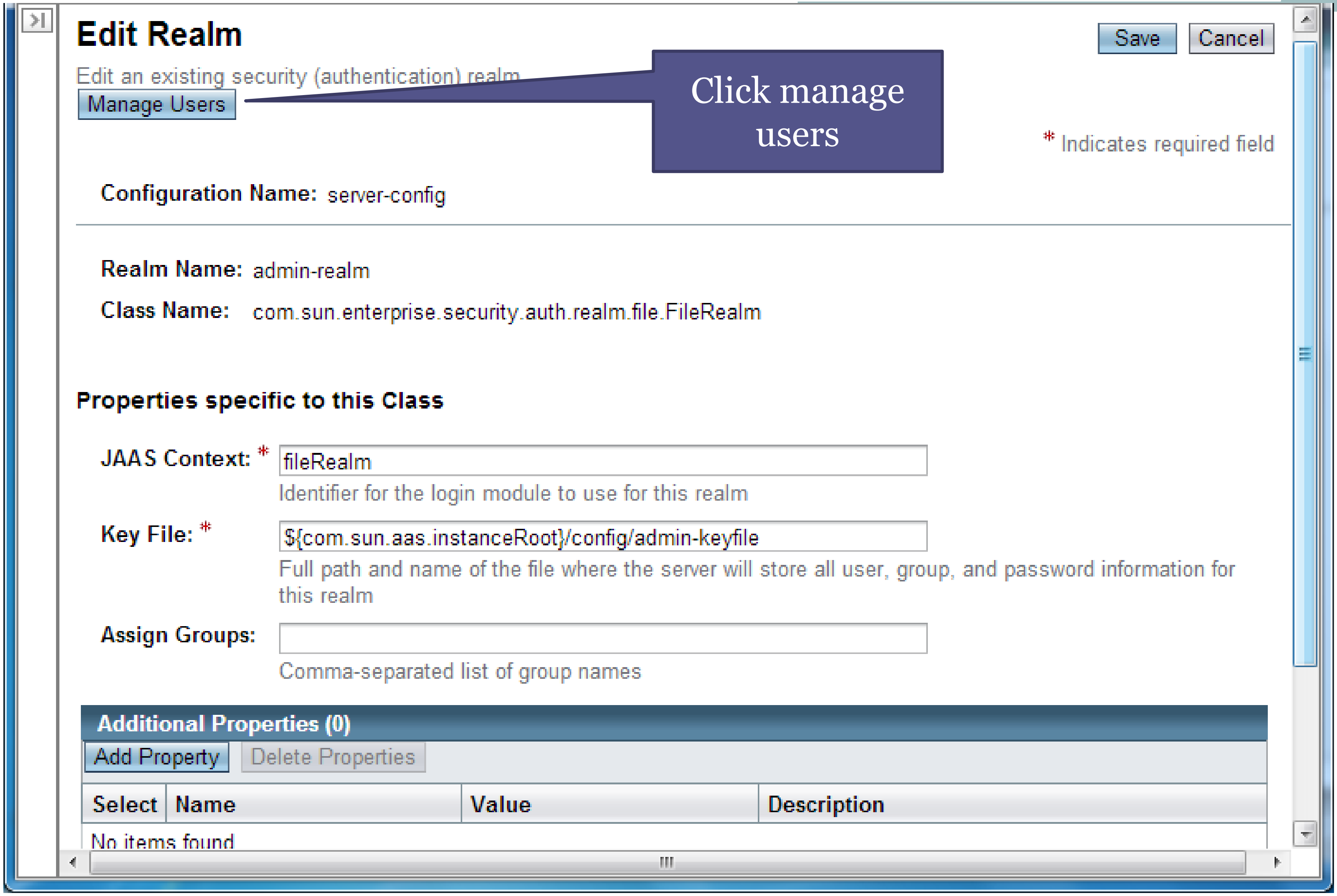
Create, modify, or delete security (authentication) realms.

Configuration Name: server-config

Realms (3)

Select	Name	Class Name
☐	admin-realm	com.sun.enterprise.security.auth.realm.file.FileRealm
☐	certificate	com.sun.enterprise.security.auth.realm.certificate.CertificateRealm
☐	file	com.sun.enterprise.security.auth.realm.file.FileRealm

Click admin-realm



Edit Realm

Edit an existing security (authentication) realm

Manage Users

Click manage users

Save Cancel

* Indicates required field

Configuration Name: server-config

Realm Name: admin-realm

Class Name: com.sun.enterprise.security.auth.realm.file.FileRealm

Properties specific to this Class

JAAS Context: * fileRealm
Identifier for the login module to use for this realm

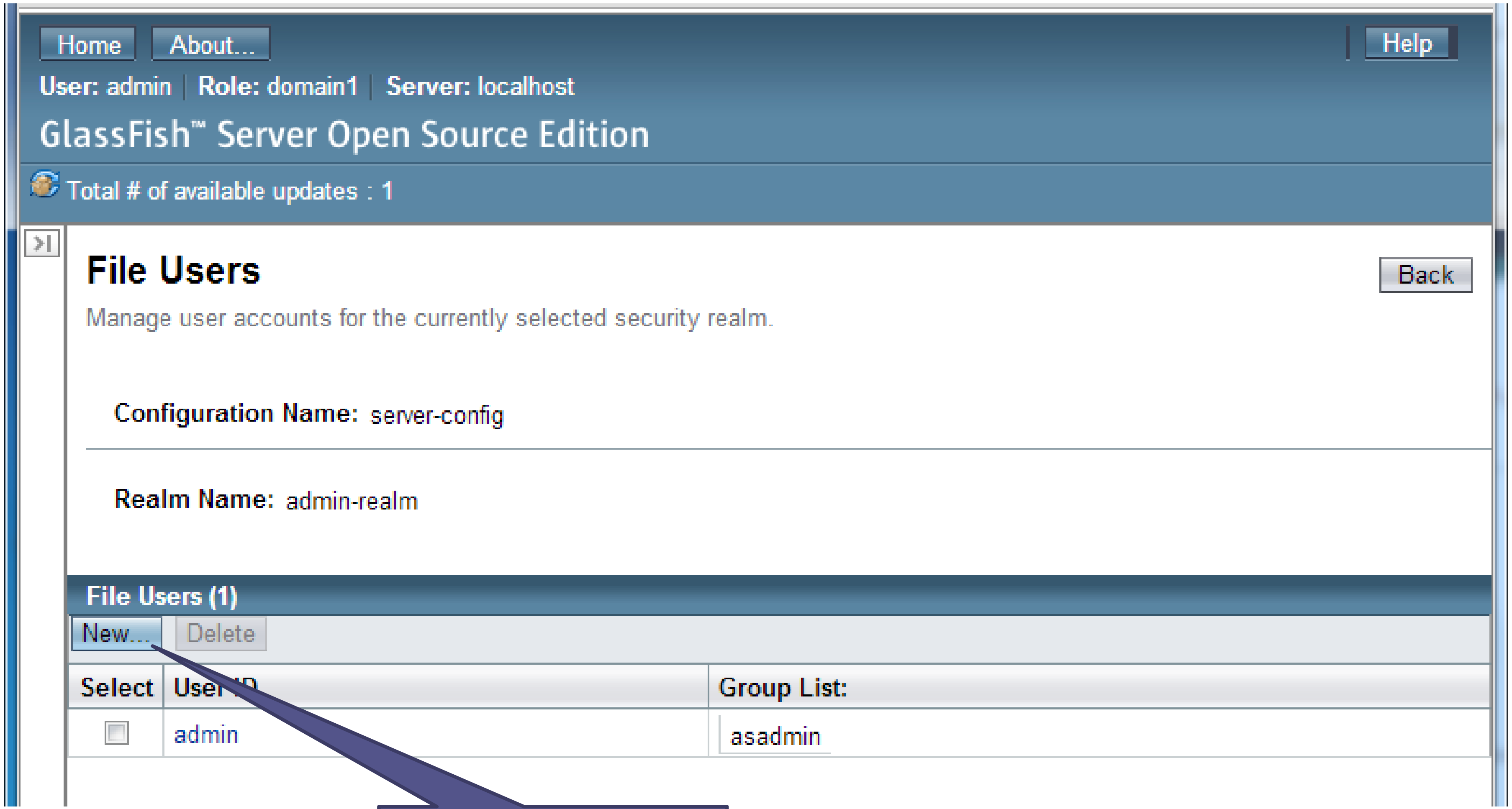
Key File: * ${com.sun.aas.instanceRoot}/config/admin-keyfile
Full path and name of the file where the server will store all user, group, and password information for this realm

Assign Groups:
Comma-separated list of group names

Additional Properties (0)

Add Property Delete Properties

Select	Name	Value	Description
No items found			



User: admin | Role: domain1 | Server: localhost

GlassFish™ Server Open Source Edition

Total # of available updates : 1

File Users

Manage user accounts for the currently selected security realm.

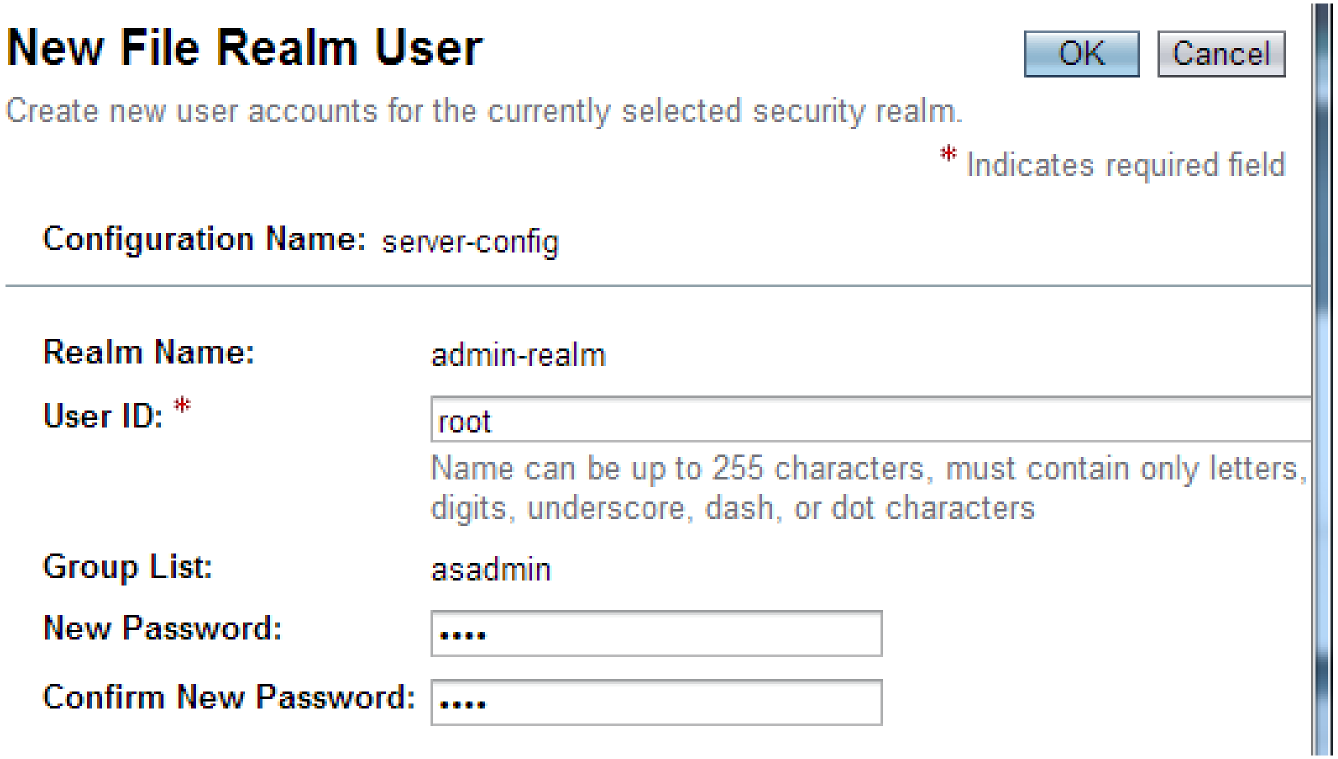
Configuration Name: server-config

Realm Name: admin-realm

File Users (1)

Select	User ID	Group List:
☐	admin	asadmin

To create a new admin user



New File Realm User

Create new user accounts for the currently selected security realm.

* Indicates required field

Configuration Name: server-config

Realm Name: admin-realm

User ID: * root

Name can be up to 255 characters, must contain only letters, digits, underscore, dash, or dot characters

Group List: asadmin

New Password: ••••

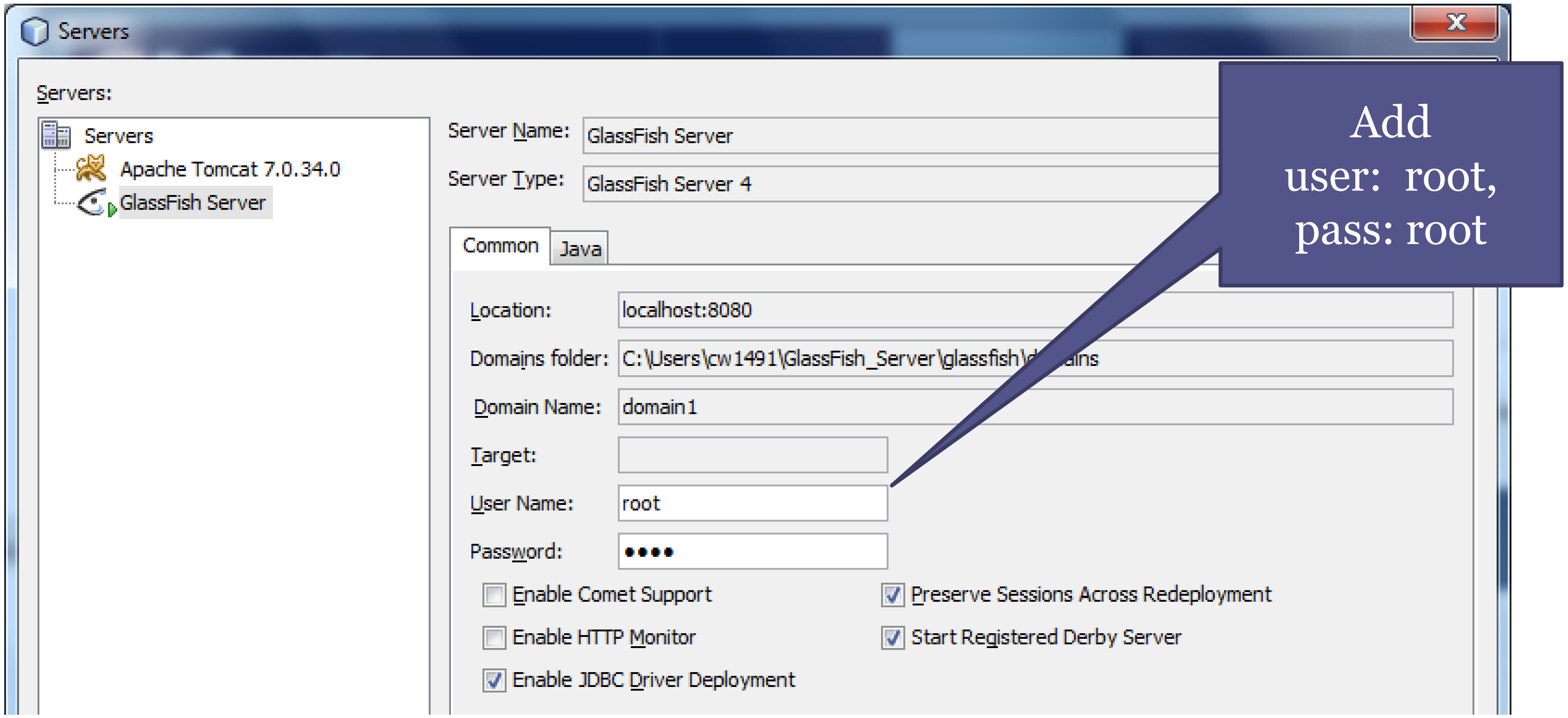
Confirm New Password: ••••

Add user id: root, pass: root

Note after you do this you **must** configure your server to start up with this user/password

Update Glassfish server properties

- Back in NetBeans, right click Glassfish server and select properties

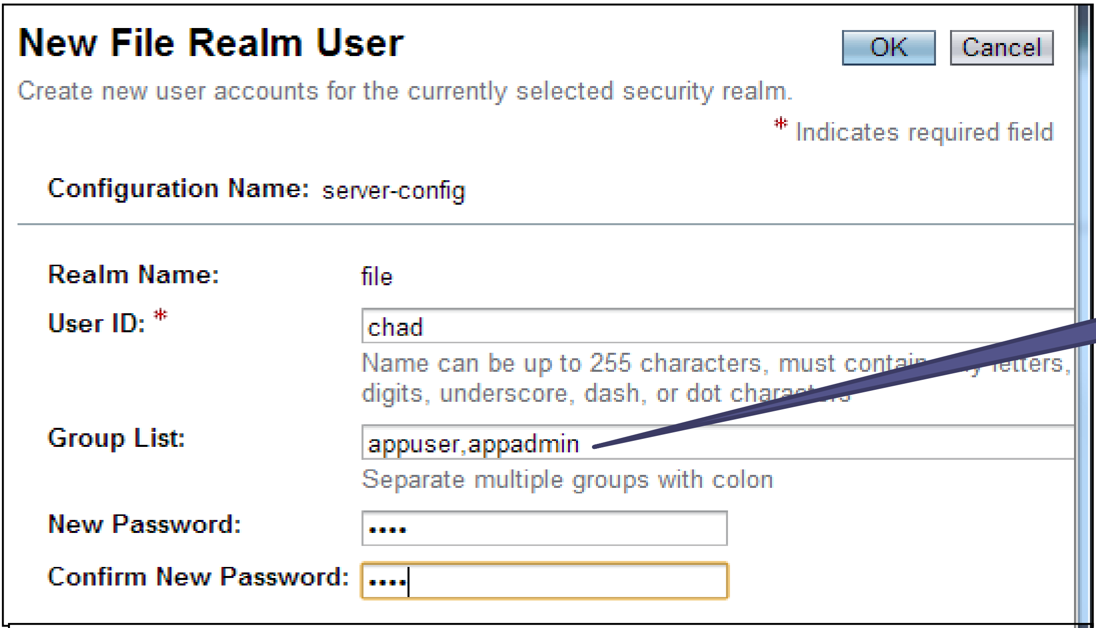


Server Name: GlassFish Server
Server Type: GlassFish Server 4
Location: localhost:8080
Domains folder: C:\Users\cw1491\GlassFish_Server\glassfish\d ains
Domain Name: domain1
Target:
User Name: root
Password: ••••

[ ] Enable Comet Support
[✓] Preserve Sessions Across Redeployment
[ ] Enable HTTP Monitor
[✓] Start Registered Derby Server
[✓] Enable JDBC Driver Deployment

Add user: root, pass: root

Add 2 users to file realm



New File Realm User

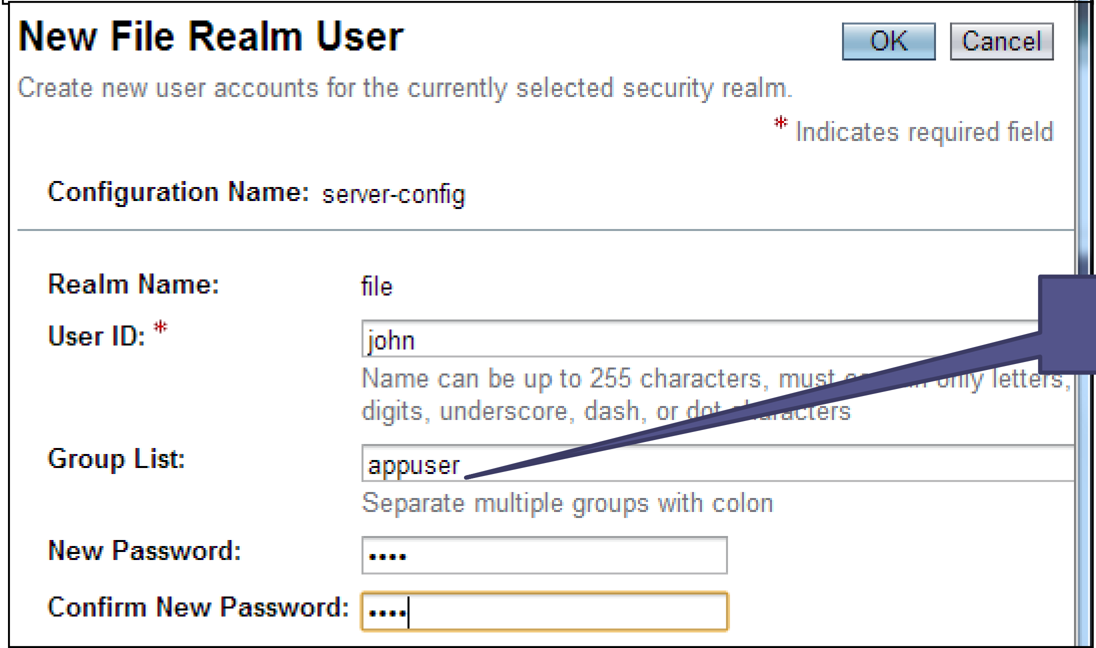
OK | Cancel

Create new user accounts for the currently selected security realm.

* Indicates required field

Configuration Name: server-config

Realm Name:	file
User ID: *	chad
	Name can be up to 255 characters, must contain only letters, digits, underscore, dash, or dot characters
Group List:	appuser,appadmin
	Separate multiple groups with colon
New Password:	••••
Confirm New Password:	••••



New File Realm User

OK | Cancel

Create new user accounts for the currently selected security realm.

* Indicates required field

Configuration Name: server-config

Realm Name:	file
User ID: *	john
	Name can be up to 255 characters, must contain only letters, digits, underscore, dash, or dot characters
Group List:	appuser
	Separate multiple groups with colon
New Password:	••••
Confirm New Password:	••••

appuser,appadmin

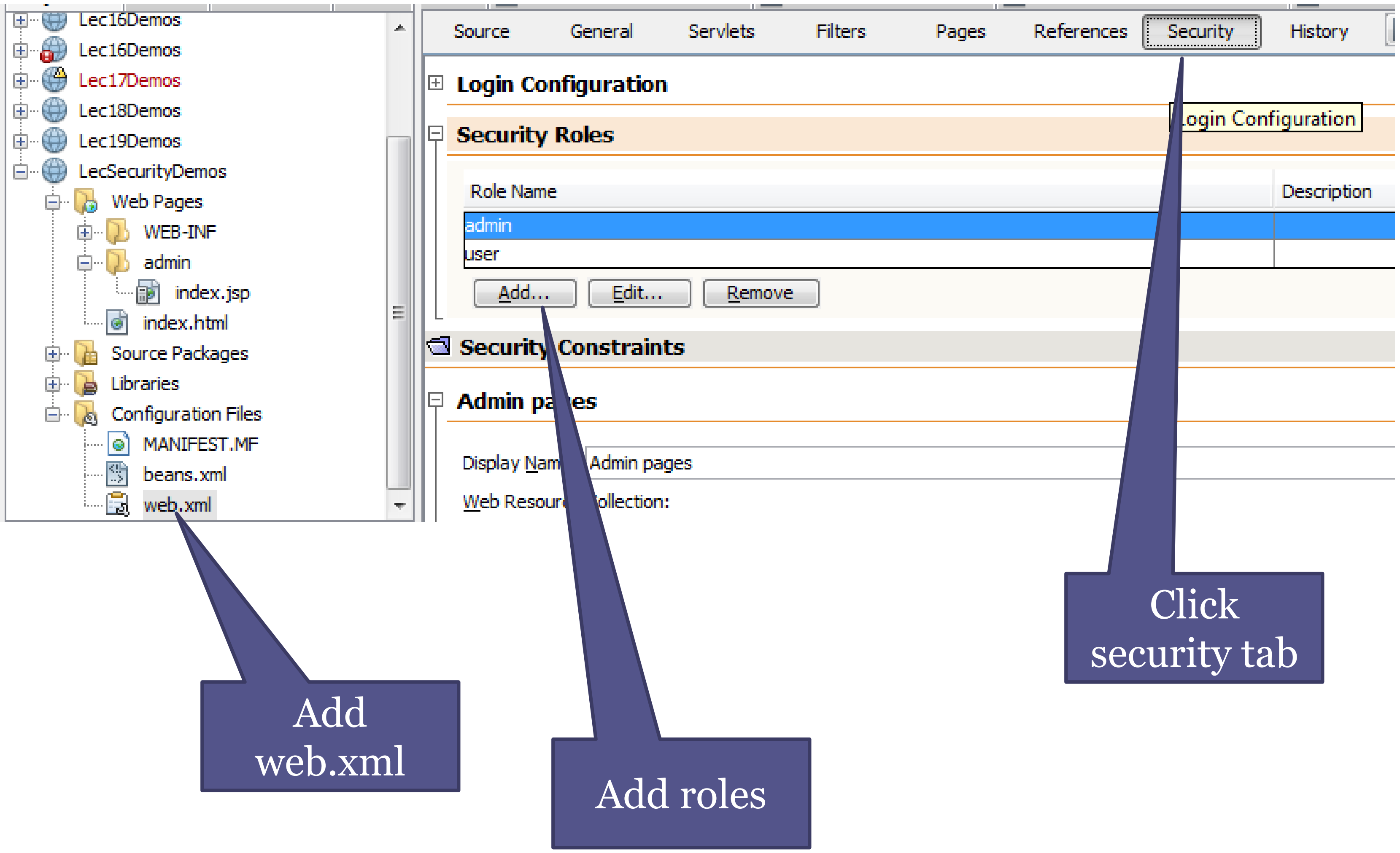
appuser

Adding basic authentication

- Configure web.xml

```
<security-constraint>
    <display-name>Admin pages</display-name>
    <web-resource-collection>
      <web-resource-name>Admin pages</web-resource-name>
      <description/>
      <url-pattern>/admin/*</url-pattern>
    </web-resource-collection>
    <auth-constraint>
      <description/>
      <role-name>admin</role-name>
    </auth-constraint>
  </security-constraint>
```

Making use of NetBeans tools



Lec16Demos
Lec16Demos
Lec17Demos
Lec18Demos
Lec19Demos
LecSecurityDemos
Web Pages
WEB-INF
admin
index.jsp
index.html
Source Packages
Libraries
Configuration Files
MANIFEST.MF
beans.xml
web.xml

Login Configuration

Security Roles

Role Name	Description
admin	
user	

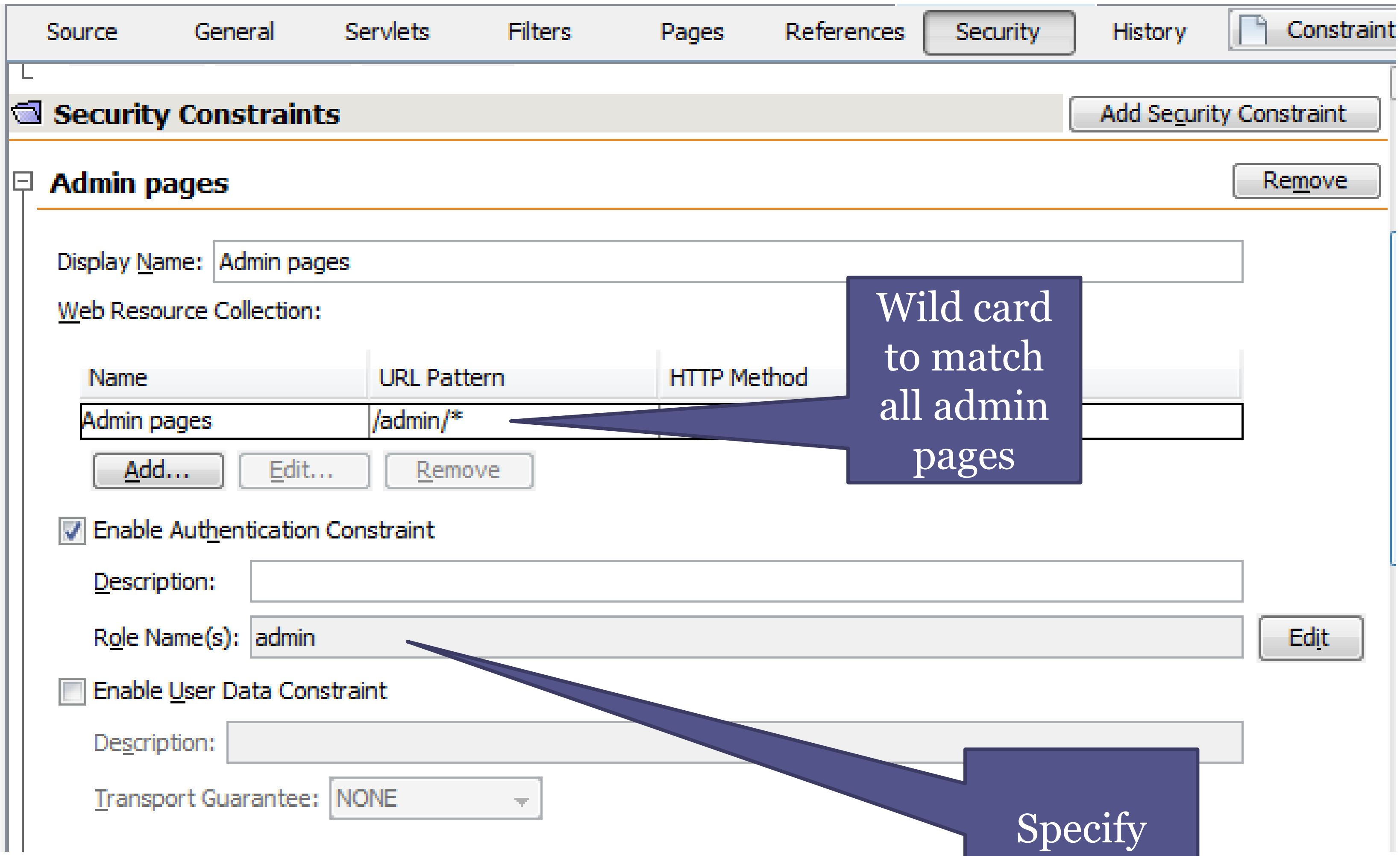
Security Constraints

Admin pages

Display Name: Admin pages

Web Resource Collection:

NetBeans tools cont.
Add security constraint for admin



Security Constraints

Admin pages

Display Name: Admin pages

Web Resource Collection:

Name	URL Pattern	HTTP Method
Admin pages	/admin/*	

☑ Enable Authentication Constraint

Description:

Role Name(s): admin

☐ Enable User Data Constraint

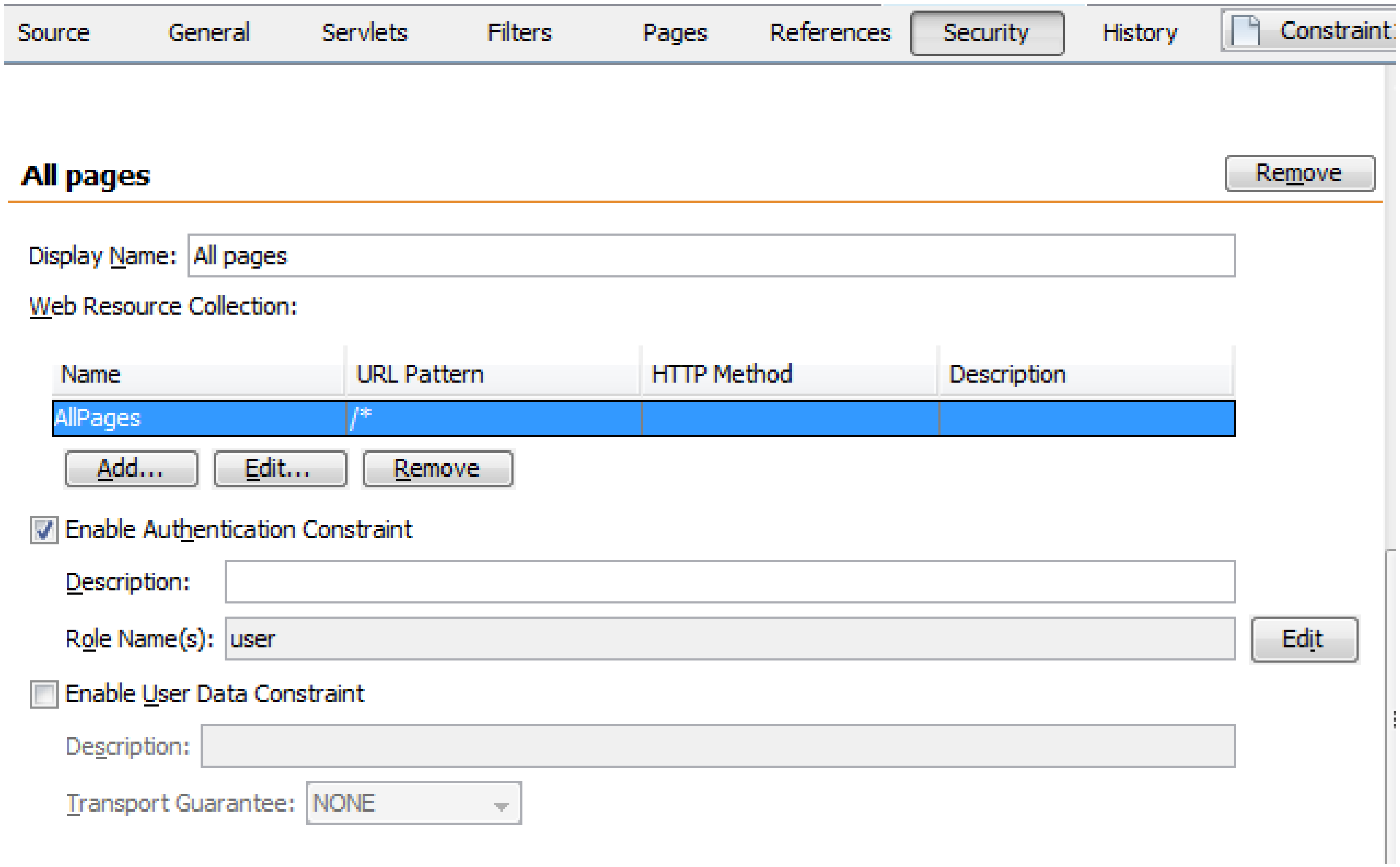
Description:

Transport Guarantee: NONE

Wild card to match all admin pages

Specify role names

NetBeans tools cont.
Add security constraint for all pages



All pages

Display Name: All pages

Web Resource Collection:

Name	URL Pattern	HTTP Method	Description
AllPages	/*		

[✓] Enable Authentication Constraint

Description:

Role Name(s): user

[ ] Enable User Data Constraint

Description:

Transport Guarantee: NONE

Authentication methods

Auth-method options are BASIC, DIGEST, FORM, and CLIENT-CERT

- **BASIC** – built in pop up authentication – password is sent Base64 encoded…unless this is through HTTPS this isn't much better than sending password in the clear
- **DIGEST** – built in pop up authentication…password is sent as MD5 digest, means the password itself cannot be recovered
- Both have **very serious drawback** logout/session invalidation not effective as browser caches login information. If you return to a page where you must be authenticated it will log you in again.

Authentication methods cont.

Auth-method options are BASIC, DIGEST, FORM, and CLIENT-CERT

- **FORM** – Specify page (HTML, JSP, XHTML) with form that collects and sends user name, and password. Values in form will be automatically validated against specified security realm. Password sent in the clear unless HTTPS used
- **CLIENT-CERT** – uses client side certificates to authenticate user (more on this later)

Specifying authentication method

For BASIC and DIGEST authentication:

```
 <login-config>
      <auth-method>DIGEST</auth-method>
      <realm-name>file</realm-name>
   </login-config>
```

Adding form based authentication

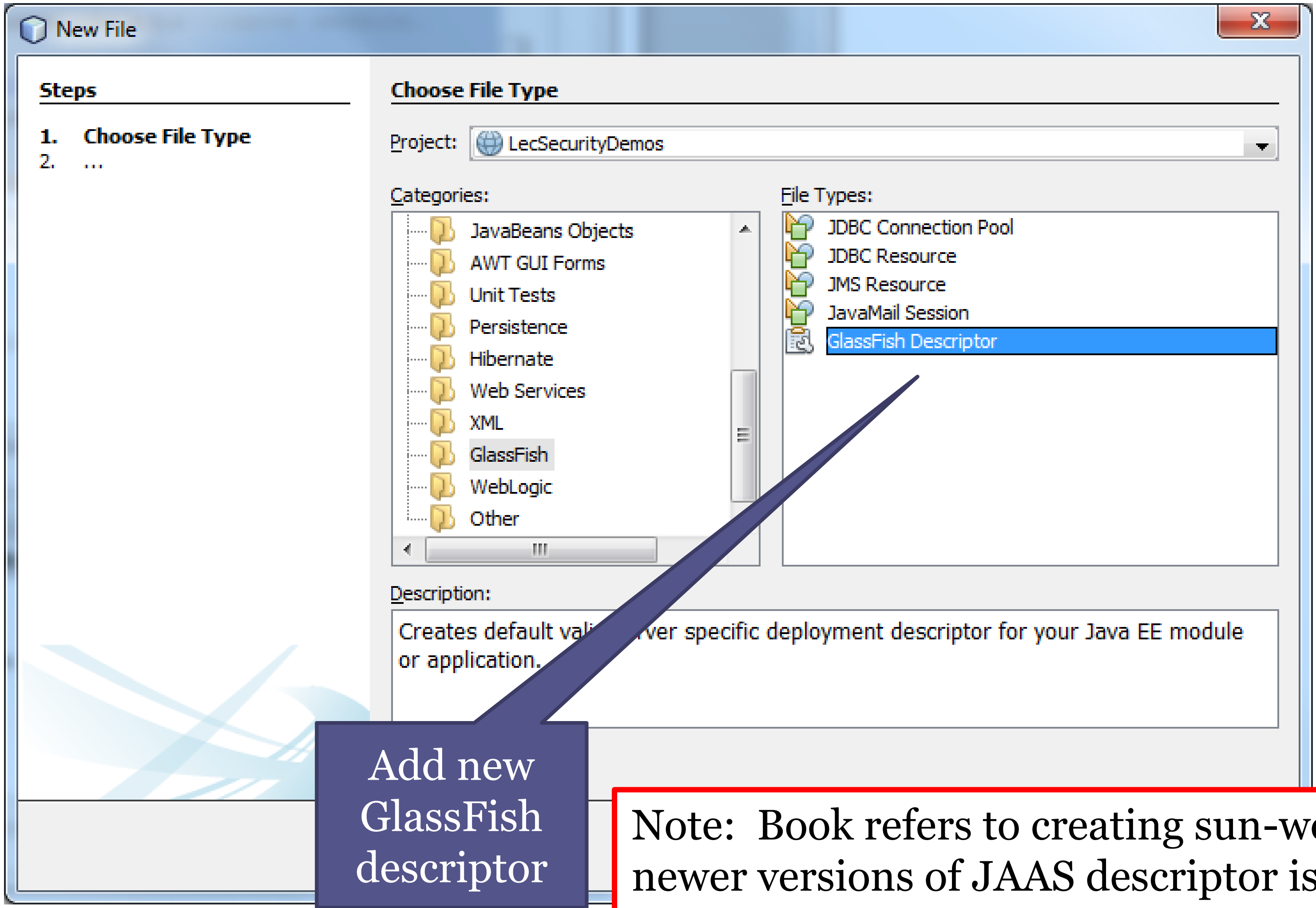
```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>file</realm-name>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/loginError.jsp</form-error-page>
    </form-login-config>
 </login-config>

<form action="j_security_check" method="POST">
      User<input type="text" name="j_username"/><br/>
      Pass<input type="pass" name="j_password"/><br/>
      <input type="submit" value="Login"/>
</form>
```

Implementing logout

- To force user to reauthenticate you invalidate the session (shown in servlet):

```
if (request.getSession(false) != null) {
  request.getSession(false).invalidate();//remove session.
}
if (request.getSession() != null) {
  request.getSession().invalidate();//remove session.
}
response.sendRedirect("index.html");
```

Link user roles to defined realms -define config file

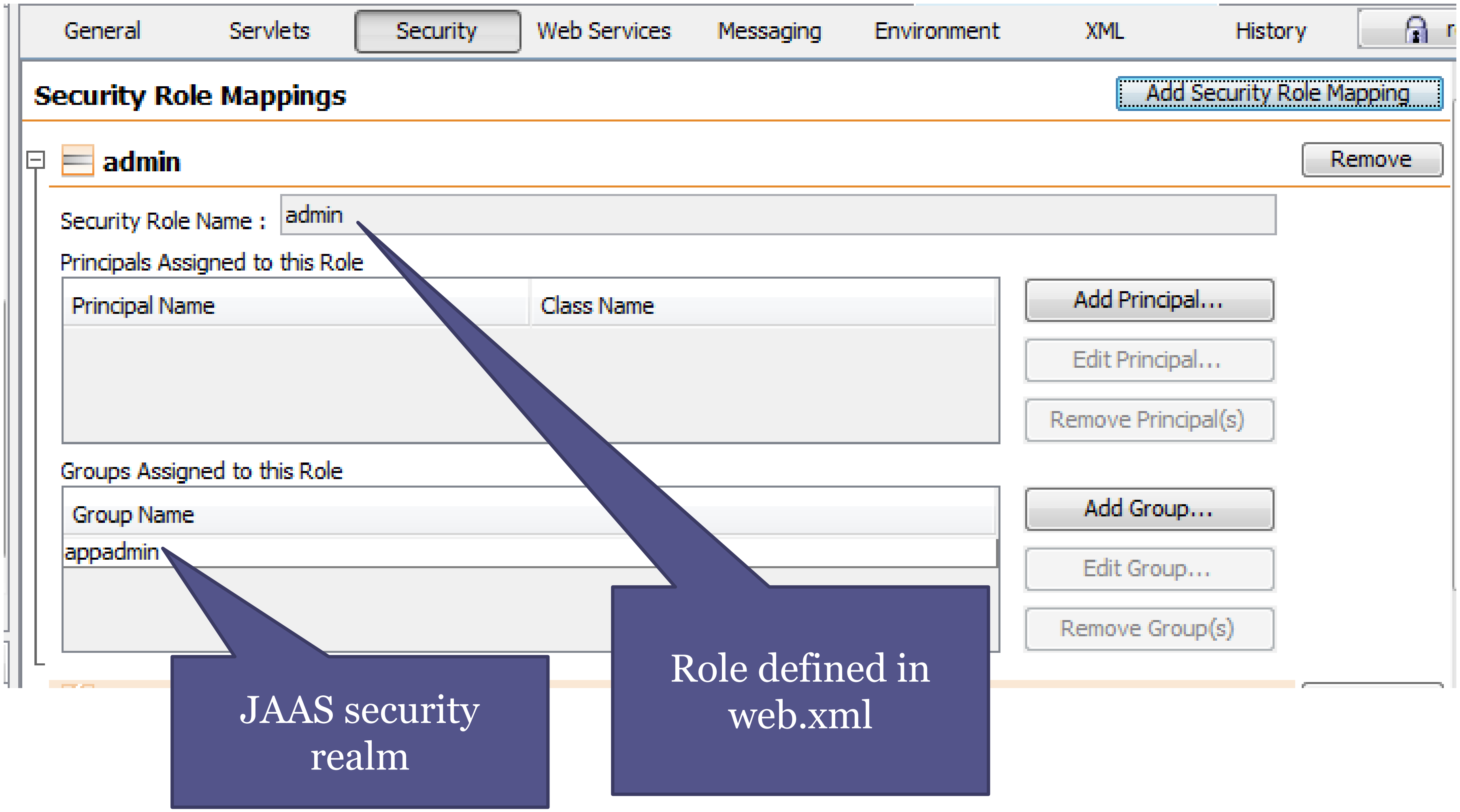


Creates default val... ver specific deployment descriptor for your Java EE module or application.

Add new GlassFish descriptor

Note: Book refers to creating sun-web.xml, with newer versions of JAAS descriptor is specific to application server (i.e. GlassFish, WebLogic, etc.)

Link user roles to defined realms



Security Role Mappings

admin

Security Role Name : admin

Principals Assigned to this Role

Principal Name	Class Name

Groups Assigned to this Role

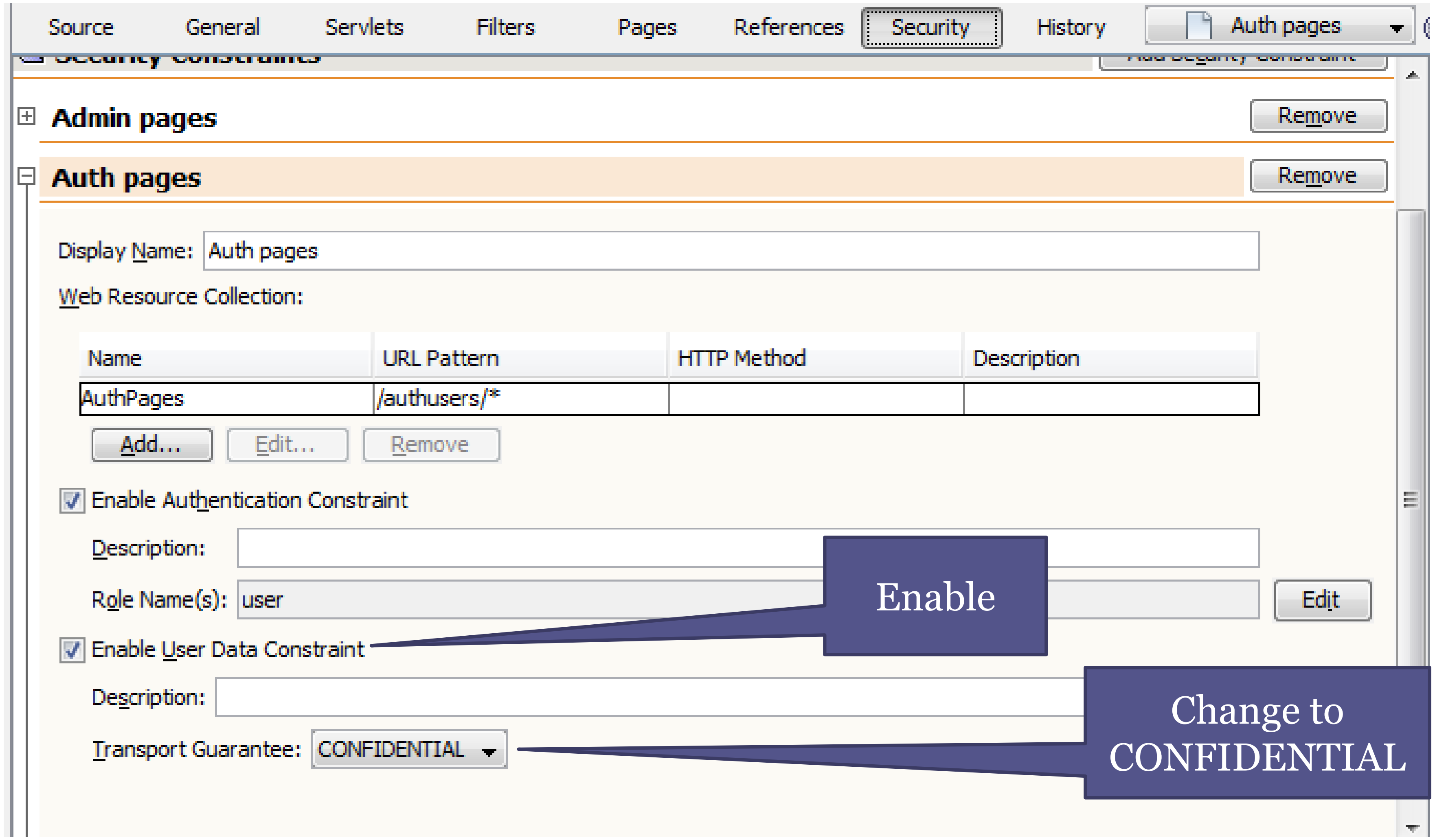
Group Name
appadmin

HTTPS

- Important security note, while input type "pass" masks input *submission is sent in the clear*
- ***Always*** have any login page through HTTPS
- Specifying HTTPS required in web.xml:

```
<security-constraint>
      <user-data-constraint>
         <transport-guarantee>CONFIDENTIAL</transport-guarantee>
      </user-data-constraint>

         ….
</security-constraint>
```

HTTPS – NetBeans tools (web.xml)



Admin pages

Auth pages

Display Name: Auth pages

Web Resource Collection:

Name	URL Pattern	HTTP Method	Description
AuthPages	/authusers/*		

✓ Enable Authentication Constraint

Description:

Role Name(s): user

✓ Enable User Data Constraint

Description:

Transport Guarantee: CONFIDENTIAL

Configuring production HTTPS

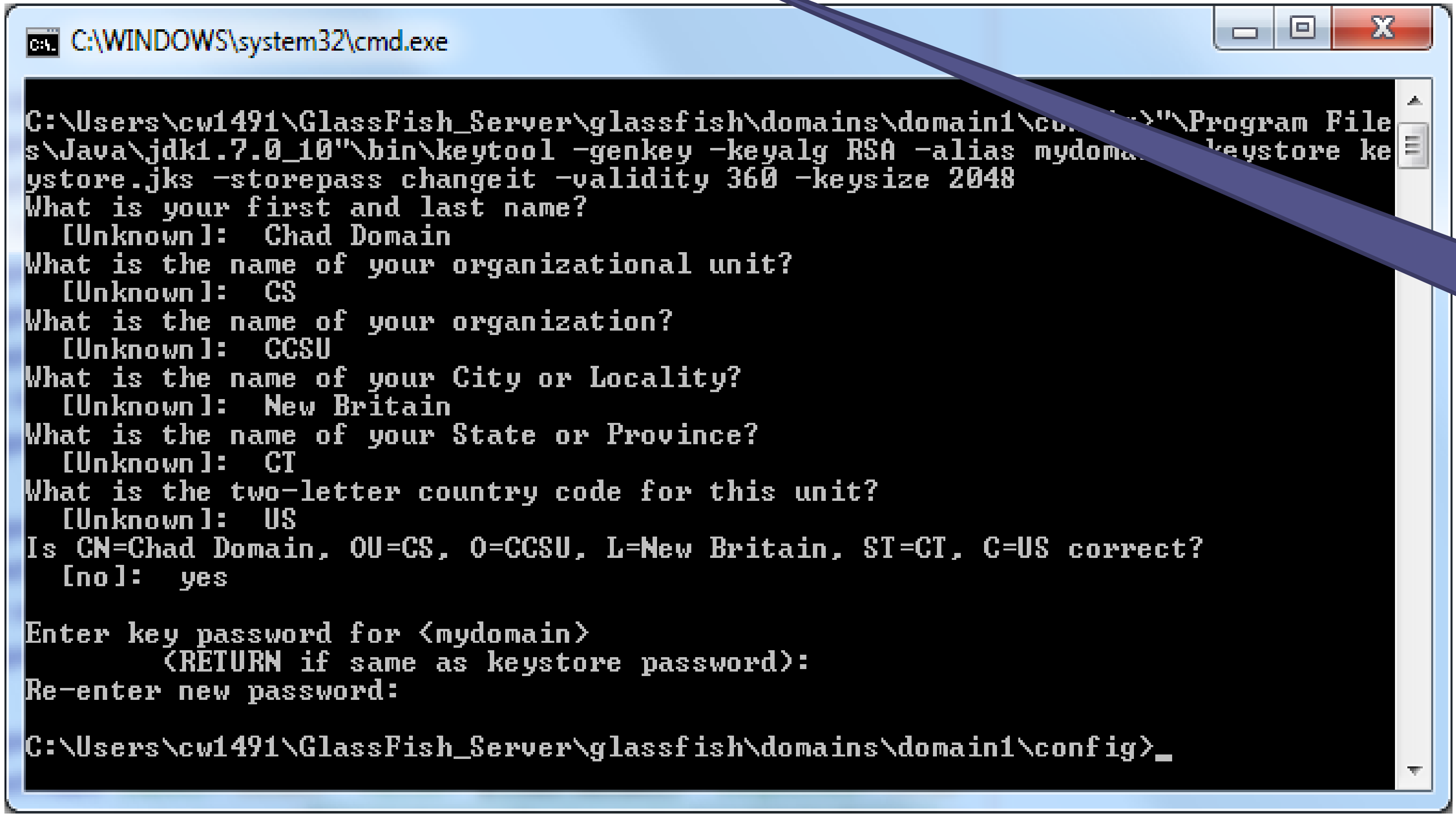
- By default GlassFish uses self-signed certificate, thus the warning you see when you access the https pages
- For a production environment you would need to get your certificate signed by a trusted certificate authority (currently minimum about $400)

WARNING before you start messing with certificates
Go into your
C:\Users\cw1491\GlassFish_Server\glassfish\domains\domain1\config
Directory and make a backup copy of **cacerts.jks** and **keystore.jks**
Playing with keystores is very finicky and if you get one step wrong you
won't be able to start your server back up

Certificate generation

- Create the certificate in the domain config dir.

C:\Users\cw1491\GlassFish_Server\glassfish\domains\domain1\config>"\Program Files\Java\jdk1.7.0_10"\bin\keytool -genkey -keyalg RSA -alias **mydomain** -keystore keystore.jks -storepass **changeit** -validity 360 -keysize 2048



```
C:\Users\cw1491\GlassFish_Server\glassfish\domains\domain1\co   g>"\Program File
s\Java\jdk1.7.0_10"\bin\keytool -genkey -keyalg RSA -alias mydoma    keystore ke
ystore.jks -storepass changeit -validity 360 -keysize 2048
What is your first and last name?
  [Unknown]:  Chad Domain
What is the name of your organizational unit?
  [Unknown]:  CS
What is the name of your organization?
  [Unknown]:  CCSU
What is the name of your City or Locality?
  [Unknown]:  New Britain
What is the name of your State or Province?
  [Unknown]:  CT
What is the two-letter country code for this unit?
  [Unknown]:  US
Is CN=Chad Domain, OU=CS, O=CCSU, L=New Britain, ST=CT, C=US correct?
  [no]:  yes

Enter key password for <mydomain>
        (RETURN if same as keystore password):
Re-enter new password:

C:\Users\cw1491\GlassFish_Server\glassfish\domains\domain1\config>_
```

Note default password for GlassFish key store

Create certificate signing request

- **Generate a certificate signing request (CSR)**

keytool -certreq -alias mydomain -
keystore keystore.jks -file mydomain.csr

- Send to Verisign or the like and pay to have them sign it

Installing production certificate

- Install it in the keystore:

```
keytool -import -trustcacerts -alias mydomain -
file mydomain.crt -keystore keystore.jks
```

- The "mydomain" above is the nickname you will need to use to reference the certificate in GlassFish
- Open GlassFish_Server\glassfish\domains\domain1\config\domain.xml
  - Perform a global replace of "s1as" with "mydomain"
  - Restart GlassFish

Certificate realm

- Uses client-side certificates for authentication
- Not useful for general web application, but advantageous when additional security wanted
  - Server to server connections
  - Tight control over user base – corporation issues certificates to employees for accessing corporate intranet remotely
    - Eliminates risk of hacker getting in through password alone
    - Certificates signed by company, if computer with certificate gets compromised certificate is revoked

Certificate realm cont.

- To create client certificates:

```
keytool -genkey -v -alias selfsignedkey -keyalg RSA -storetype PKCS12 -
keystore client_keystore.p12 -storepass password
```

Install it in the browser

- Usually in browser's advanced tab under encryption something like "manage certificates"
- Import the generated ".p12" file

- Export to format that GlassFish can import:

```
keytool -export -alias selfsignedkey -keystore client_keystore.p12 -
storetype PKCS12 -storepass password -rfc selfsigned.cer
```

```
keytool -export -alias selfsignedkey -keystore client_keystore.p12 -
storetype PKCS12 -storepass password -file selfsigned.cer
```

- Import into GlassFish

```
keytool -import -file selfsigned.cer -keystore cacerts.jks -storepass changeit
```

Certificate realm cont.

- Previous steps will establish certificates signed by themselves as a trusted certificate authority (not something you want to do in a production environment, but convenient for development)
- Result is client-side certificates sent from the browser are trusted implicitly i.e. require no additional verification of user other than presenting their certificate

Configuring client-certificate access

In web.xml

```
<login-config>
      <auth-method>CLIENT-CERT</auth-method>
      <realm-name>certificate</realm-name>
 </login-config>
```

- This has the effect of GlassFish asking the client for a certificate for authorization when the client requests a protected page
- On the client side the way this typically works is browser will prompt you to pick a certificate you have installed which should be used for authentication

Configuring client-certificate access

- Any page that requires certificate based authentication must be accessed via HTTPS so security constraints should be set to CONFIDENTIAL as described earlier

Configuring client-certificate access

- Modify glassfish-web.xml

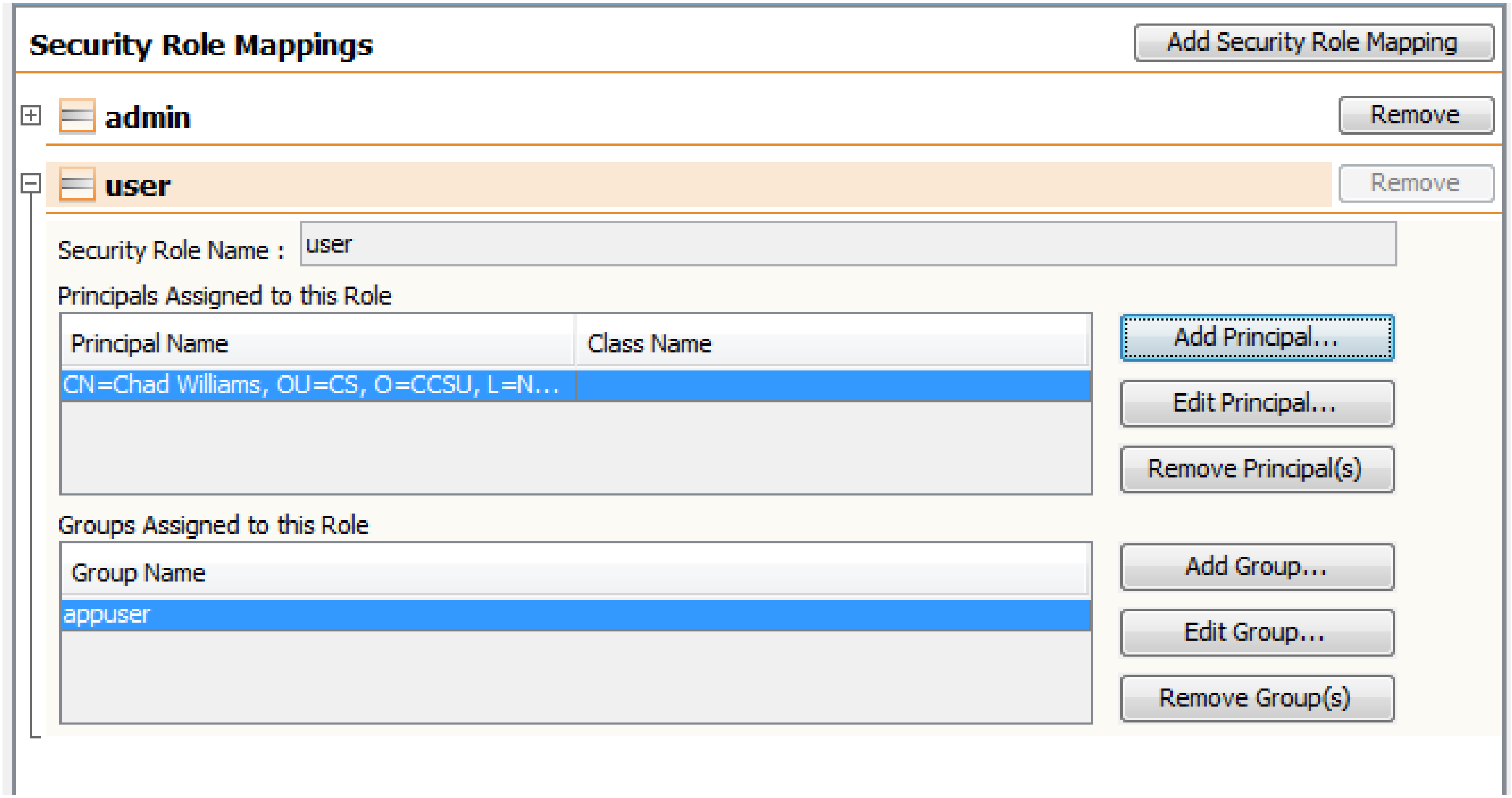
```
<context-root>/certificaterealm</context-root>
<security-role-mapping>
  <role-name>user</role-name>
  <principal-name>CN=Chad Williams, OU=CS, O=CCSU, L=New
Britain, ST=CT, C=US</principal-name>
</security-role-mapping>
```

- You can get the principal name by calling:

keytool -printcert -file selfsigned.cer

- Then copying the “issuer” that gets printed

GlassFish wizard



Security Role Mappings

Add Security Role Mapping

admin — Remove

user — Remove

Security Role Name : user

Principals Assigned to this Role

Principal Name	Class Name
CN=Chad Williams, OU=CS, O=CCSU, L=N...	

Groups Assigned to this Role

Group Name
appuser

Weaknesses of file and certificate

- Both the previous methods file realm, certificate realm provide very secure authentication suitable for a production environment, but have weaknesses
  - **File realm** – authentication against application server properties
    - Difficult to maintain
    - Limits authentication to applications that can use app server for authentication
  - **Certificate realm** – authentication requires client side certificates which isn't practical for most applications

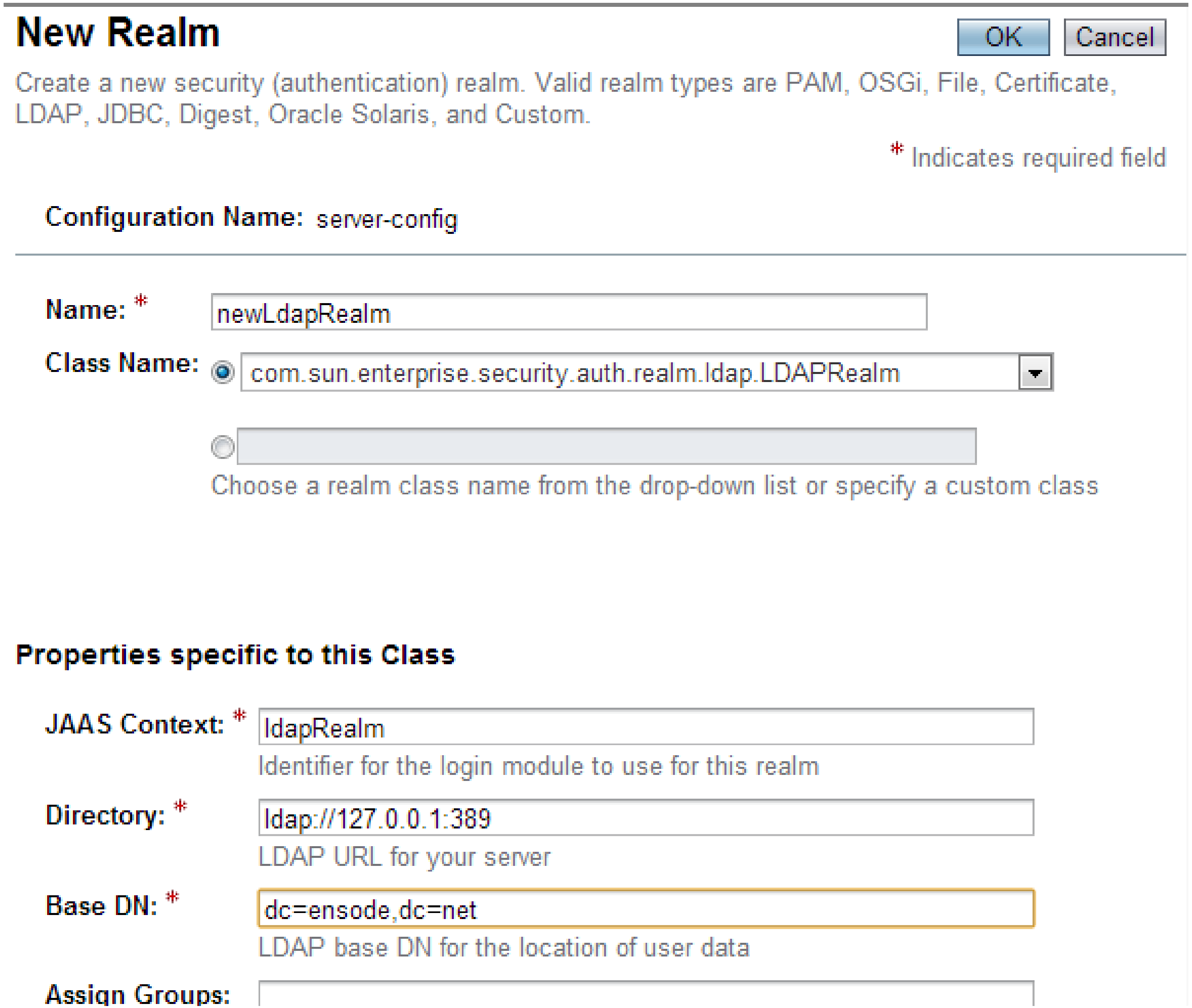
Single sign-on realms

- Most production environments have moved or are migrating from rather than having a different login for each system using a single authentication source
  - LDAP (Lightweight Directory Access Protocol)
  - Database
- Both sources are widely used and fairly easy to maintain
- LDAP has advantages when database access not needed by an application otherwise

Defining LDAP realms

- Through admin console create new realm
  - **Class name** is: com.sun.enterprise.security.auth.realm.ldap.LDAPRealm
  - **JAAS context** is how the realm will be referenced in glassfish config file
  - **Directory** is URL for directory server
  - **Base DN** is the base distinguised name (DN) to be used in searching for user data

GlassFish admin console



New Realm

Create a new security (authentication) realm. Valid realm types are PAM, OSGi, File, Certificate, LDAP, JDBC, Digest, Oracle Solaris, and Custom.

* Indicates required field

Configuration Name: server-config

Name: * newLdapRealm

Class Name: ● com.sun.enterprise.security.auth.realm.ldap.LDAPRealm

○

Choose a realm class name from the drop-down list or specify a custom class

Properties specific to this Class

JAAS Context: * ldapRealm

Identifier for the login module to use for this realm

Directory: * ldap://127.0.0.1:389

LDAP URL for your server

Base DN: * dc=ensode,dc=net

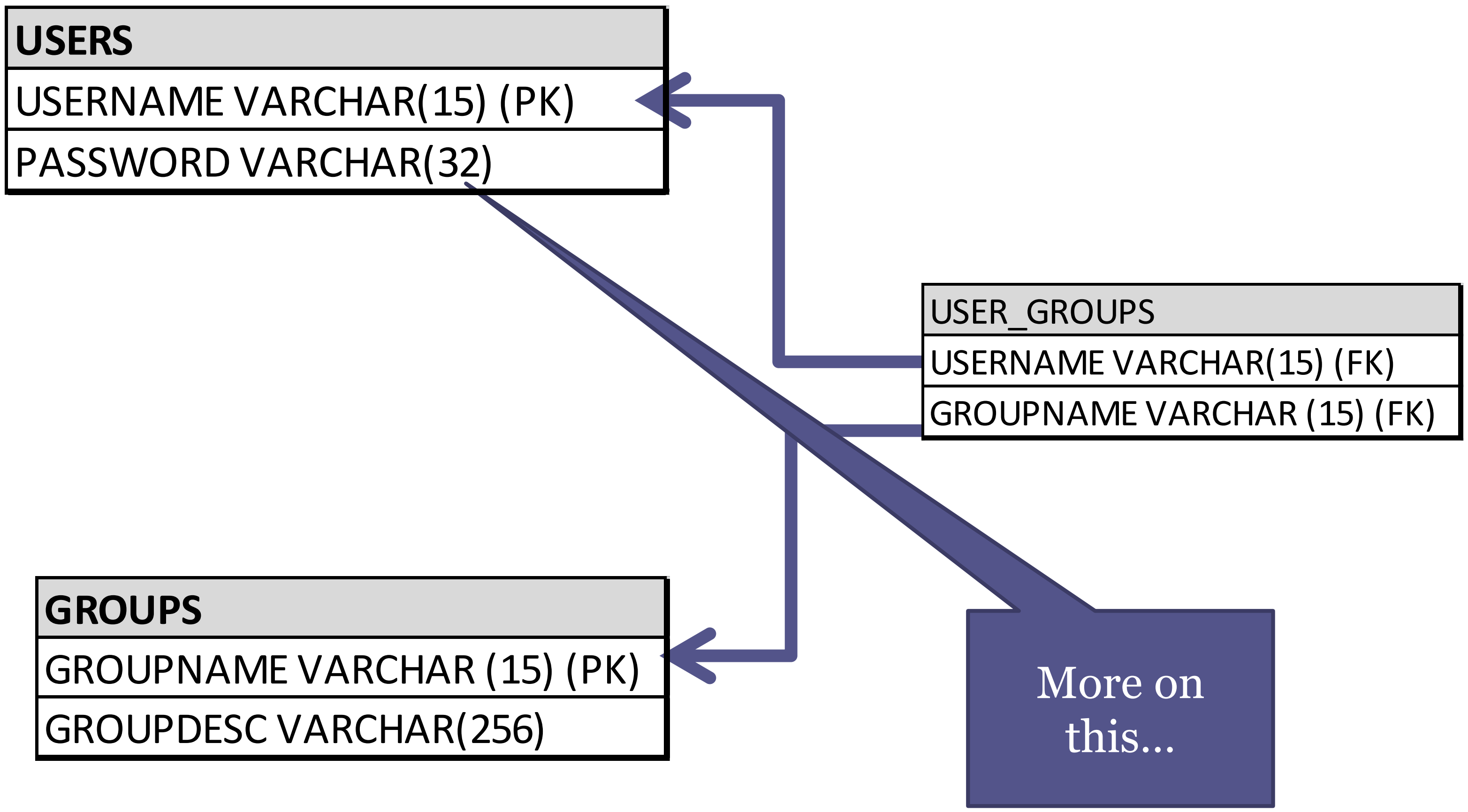
LDAP base DN for the location of user data

Assign Groups:

LDAP realms cont.

- To use new realm mapping created in glassfish-web.xml
- Users and roles in LDAP database can be mapped to principals and groups in that xml
- Authentication method must be BASIC or FORM

JDBC realm example setup



USERS
USERNAME VARCHAR(15) (PK)
PASSWORD VARCHAR(32)

USER_GROUPS
USERNAME VARCHAR(15) (FK)
GROUPNAME VARCHAR (15) (FK)

GROUPS
GROUPNAME VARCHAR (15) (PK)
GROUPDESC VARCHAR(256)

JDBC password

- You never want to store a password in the database (or elsewhere) in plain text.
  - If system is ever compromised may help attacker break into other sites or systems
- Because of this password expected as MD5 hash (which is 32 characters long)
- Result is if you want to write to field such as when you create a new user you need to hash the field first

Creating MD5 hash

```
String password = "myPass";
MessageDigest msgDigest =
MessageDigest.getInstance("MD5");
byte[] bs;
msgDigest.reset();
bs = msgDigest.digest(password.getBytes());
StringBuilder sBuilder = new StringBuilder();
for (int i=0;i<bs.length;i++){
  String hexVal = Integer.toHexString(0xFF & bs[i]);
  if (hexVal.length()==1){
    stringBuilder.append("0");
  }
  stringBuilder.append(hexVal);
}
return stringBuilder.toString();
```

Creating JDBC realm through console

- Class name:
  - com.sun.enterprise.security.auth.realm.jdbc.JDBCRealm
- JAAS context: jdbcRealm
- JNDI: jdbc/Lect8aDB
- User table: USERS
- User name: USERNAME
- Password: PASSWORD
- Group table: USER_GROUPS
- Group table user name column: USERNAME
- Group name column: GROUP_NAME
- Password encryption algorithm: MD5

Add config

- Add to web.xml

```
<login-config>
    <auth-method>FORM</auth-method>
    <realm-name>jdbcRealm</realm-name>
    <form-login-config>
      <form-login-page>/login.jsp</form-login-page>
      <form-error-page>/loginError.jsp</form-error-page>
    </form-login-config>
  </login-config>
```

- Add to glassfish xml principals and groups as before