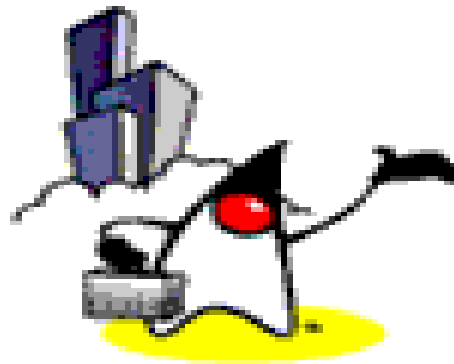


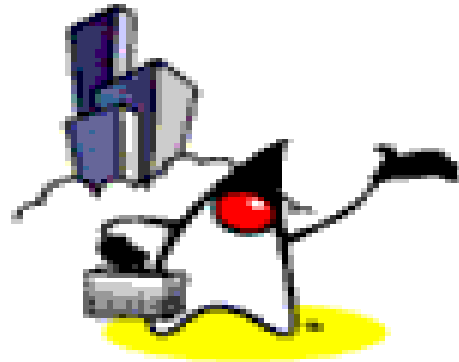


Web Application Security



Agenda

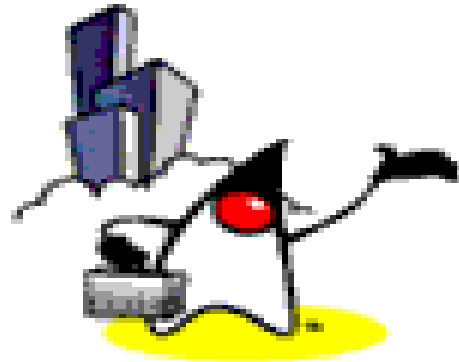
- General security issues
- Web-tier security requirements and schemes
- HTTP basic authentication based web-tier security scheme
- Form-based authentication based web-tier security scheme
- Declarative authorization
- Programmatic authorization
- Programmatic authentication



General Security Issues

General Security Issues

- Authentication for identity verification
 - Making sure a user is who he claims he is
- Authorization (Access control)
 - Making sure resources are accessible only to users who have access privilege
 - The user has to be authenticated first
- Confidentiality (Privacy)
 - Protecting the sensitive data from prying eyes while it is on the wire



Web-tier Security Requirements & Schemes

Security Requirements at Web-Tier

- Preventing unauthorized users from accessing “access controlled” web resource
 - If an unauthenticated user tries to access “access controlled” web resource, web container will **automatically** ask the user to authenticate himself first
 - Once the user authenticated, web container (and/or web components) then enforces **access control**
- Preventing attackers from changing or reading sensitive data while it is on the wire
 - Data can be protected via **SSL**

Web-Tier Security Scheme Should Address “Authentication”

1. Collecting user identity information from an end user
 - typically through browser interface
 - user identity information usually means username and password
 - this is called “logging in”
2. Transporting collected user identity information to the web server
 - unsecurely (HTTP) or securely (HTTP over SSL)

Web-Tier Security Scheme Should Address “Authentication” (Cont.)

3. Performing identity checking with backend “security database” (Realms)

- Web container checks if collected user identity matches with the one in the backend “security database”
- These backend “security database” are called Realms
- Realms maintain
 - Username, password, roles, etc.
- How these realms are organized and managed are product and operational environment dependent
 - LDAP, RDBMS, Flat-file, Solaris PAM, Windows AD

Web-Tier Security Scheme Should Address “Authentication” (Cont.)

4. Web container keep track of previously authenticated users for further HTTP operations

- Using internally maintained session state, web container knows if the caller of subsequent HTTP requests has been authenticated
- Web container also creates `HttpServletRequest` object for subsequent HTTP requests
 - `HttpServletRequest` object contains “security context” information
 - Principal, Role, Username

Web-Tier Security Scheme Should Address “Access control”

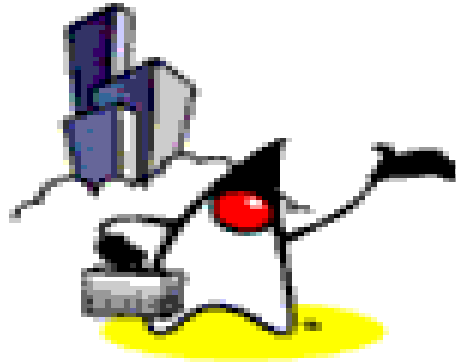
- Web application developer and/or deployer specifies access control to web resources
 - Declarative and/or Programmatic access control

Web-Tier Security Scheme Should Address “Data confidentiality”

- Providing confidentiality of the sensitive data that is being transported over the wire
 - Between browser and web server
 - Example: Credit card number
 - Using SSL

Web-tier Authentication Schemes

- HTTP basic authentication based
 - with or without SSL
- Form-based authentication based
 - with or without SSL
- Client-certificate authentication based
 - Has to use SSL
- Digest authentication based
 - Does not need to use SSL



HTTP Basic Authentication-based Web tier Security

HTTP Basic Authentication

- Web server collects user identification (user name and password) through a **browser provided dialog box**
- **Not secure** since user name and password are in “easily decode'able” form over the wire
 - Encoding scheme is Base64
 - Someone can easily decode it
 - **Not encrypted**
- Would need SSL for encrypting password

Steps for Basic Authentication-based Web-tier Security

1. Set up username, passwords, and **roles (realms)**
2. Tell web container that you are using Basic authentication
3. Specify which URLs (web resources) should be access-controlled (password-protected)
4. Specify which URLs should be available only with SSL (data integrity and confidentiality protected)

Step 1: Set up username, passwords, and roles (Realms)

- Schemes, APIs, and tools for setting up usernames, passwords, and roles (realms) are web **container and operational environment specific**
 - Flat-file based, Database, LDAP server
 - Passwords could be in either encrypted or unencrypted form
- Tomcat 4.0 can work with the following realms
 - default: file, unencrypted form
 - Relational database (via JDBCRealm)
 - LDAP server (via LDAPRealm)

Example: Tomcat's default

- <install-dir>/config/tomcat-users.xml
- Unencrypted: not secure but easy to set up and maintain

```
<?xml version='1.0'?>
```

```
<tomcat-users>
```

```
  <role rolename="manager"/>
```

```
  <role rolename="employee"/>
```

```
  <role rolename="admin"/>
```

```
  <user username="sang" password="sangPassword"  
    roles="manager,employee"/>
```

```
</tomcat-users>
```

Step 2: Tell web container that you are using Basic authentication

- In `web.xml` file of your web application

```
<web-app>
...
<security-constraint>...</security-constraint>
<login-config>
  <auth-method>BASIC</auth-method>
  <realm-name>realm name</realm-name>
</login-config>
...
</web-app>
```

Step 3: Specify which URLs should be access-controlled

<web-app>

...

<security-constraint>

<web-resource-collection>

<web-resource-name>WRCollection</web-resource-name>

<url-pattern>/loadpricelist</url-pattern>

<http-method>GET</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>BASIC</auth-method> <realm-name></realm-name>

</login-config>

...

Step 4: Specify which URLs should be available only with SSL

```
<web-app>
```

```
...
```

```
<security-constraint>
```

```
  <web-resource-collection>
```

```
    <web-resource-name>WRCollection</web-resource-name>
```

```
    <url-pattern>/loadpricelist</url-pattern>
```

```
    <http-method>GET</http-method>
```

```
  </web-resource-collection>
```

```
  <auth-constraint>
```

```
    <role-name>admin</role-name>
```

```
  </auth-constraint>
```

```
  <user-data-constraint>
```

```
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
  </user-data-constraint>
```

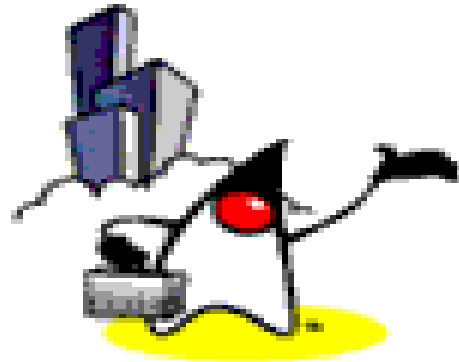
```
</security-constraint>
```

```
<login-config>
```

```
  <auth-method>BASIC</auth-method> <realm-name></realm-name>
```

```
</login-config>
```

```
...
```

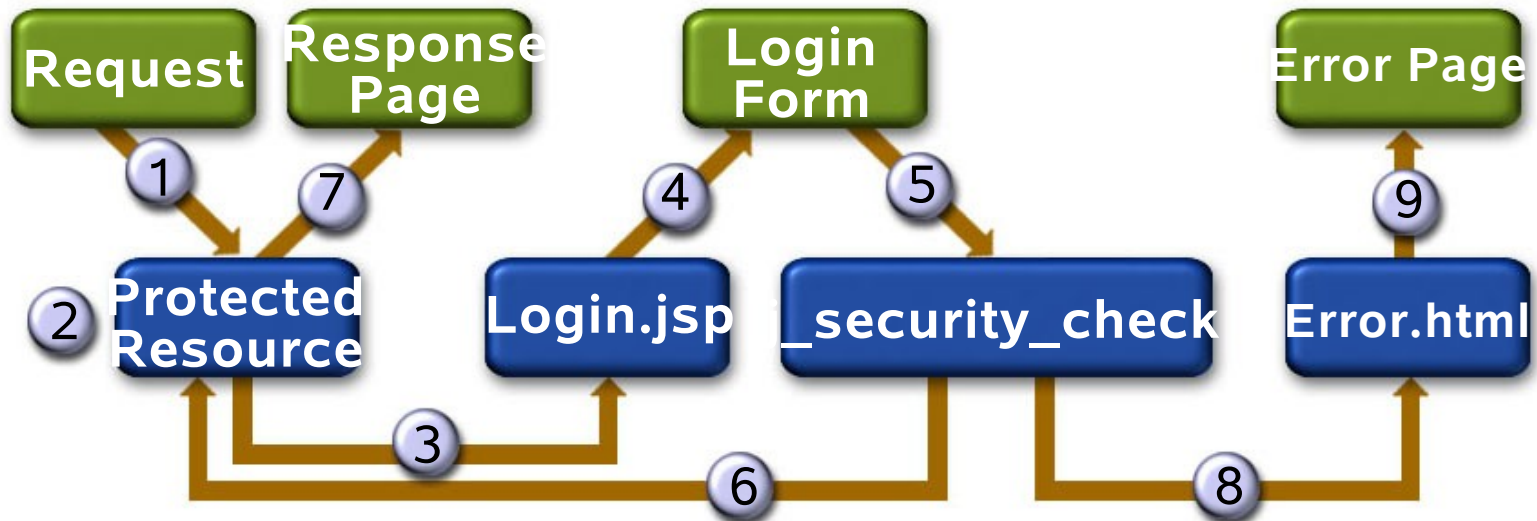


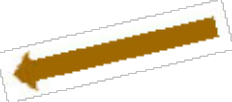
Form-based Authentication based Web-tier Security

Form-based Authentication

- Web application collects user identification (user name, password, and other information) through a **custom login page**
- **Not secure** since user name and password are in “easily decode'able” form over the wire
 - Encoding scheme is Base64
 - Someone can easily decode it
 - **Not encrypted**
- Would need SSL for encrypting password

Form-Based Auth. Control Flow



1. Request made by client
2. Is client authenticated? 
3. Unauthenticated client redirected
4. Login form returned to client
5. Client submits login form
6. **Authentication** Login succeeded, redirected to resource
7. **Authorization** Permission tested, result returned
8. Login failed, redirect to error page
9. Error page returned to client

Steps for Form-based Authentication based Web-tier Security

1. Set up username, passwords, and roles (realms)
2. Tell web container that you are using Form-based authentication
3. Create custom “Login page”
4. Create custom “Login failure error page”
5. Specify which URLs (web resources) should be access-controlled (password-protected)
6. Specify which URLs should be available only with SSL (data integrity and confidentiality protected)

Step 1: Set up username, passwords, and roles (Realms)

- Same as in Basic-authentication

Step 2: Tell web container that you are using Form-based authentication

- In `web.xml` file of your web application

```
<web-app>
```

```
...
```

```
<security-constraint>...</security-constraint>
```

```
<login-config>
```

```
  <auth-method>FORM</auth-method>
```

```
  <realm-name>realm name</realm-name>
```

```
</login-config>
```

```
...
```

```
</web-app>
```

Step 3: Create custom “Login Page”

- Can be HTML or JSP page
- Contains HTML form like following

```
<FORM ACTION="j_security_check"  
  METHOD="POST">  
  ...  
  <INPUT TYPE="TEXT" NAME="j_username">  
  ...  
  <INPUT TYPE="PASSWORD" NAME="j_password">  
  ...  
</FORM>
```

Step 4: Create Login Failure page

- Can be HTML or JSP page
- No specific content is mandated

Step 5: Specify which URLs should be access-controlled (Same as Basic Auth)

<web-app>

...

<security-constraint>

<web-resource-collection>

<web-resource-name>WRCollection</web-resource-name>

<url-pattern>/loadpricelist</url-pattern>

<http-method>GET</http-method>

</web-resource-collection>

<auth-constraint>

<role-name>admin</role-name>

<role-name>executive</role-name>

</auth-constraint>

<user-data-constraint>

<transport-guarantee>CONFIDENTIAL</transport-guarantee>

</user-data-constraint>

</security-constraint>

<login-config>

<auth-method>FORM</auth-method> <realm-name></realm-name>

</login-config>

Step 6: Specify which URLs should be available only with SSL (Same as Basic Auth)

```
<web-app>
...
<security-constraint>
  <web-resource-collection>
    <web-resource-name>WRCollection</web-resource-name>
    <url-pattern>/loadpricelist</url-pattern>
    <http-method>GET</http-method>
  </web-resource-collection>
  <auth-constraint>
    <role-name>admin</role-name>
  </auth-constraint>
  <user-data-constraint>
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
  </user-data-constraint>
</security-constraint>
<login-config>
  <auth-method>FORM</auth-method> <realm-name></realm-name>
</login-config>
...
</web-app>
```

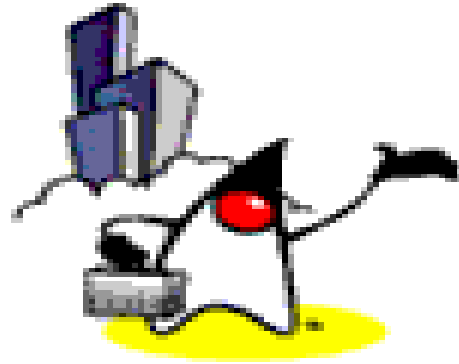
Basic vs. Form-based Authentication

Basic

- Uses “browser provided dialog box” to get username and password
- Only username and password can be collected
- Might result in different look and feel
- HTTP Authentication header is used to convey username and password
- No good way to enter

Form-based

- Uses “web application provided login page” to get username and password
- Custom data can be collected
- Can enforce consistent look and feel
- Form data is used to convey username and password
- Can enter a new user name via login page



Realm Management

Realm Management

- Management of user identity information
 - username, password, roles, etc.
 - encrypted or unencrypted
- Container and operational environment dependent
 - Tomcat
 - flat file based, RDBMS, LDAP
 - Sun ONE App server

Security Roles

- Web Application developer or assembler use **security roles** for “access control” (declarative & programmatic)
 - These are **abstract roles** that have nothing to do with usernames, passwords, groups of operating system
 - At application deployment time, these abstract security roles need to be mapped usernames, passwords, groups of operating system
- In production environment, an external security realm (LDAP) can be used both by web application and OS

Example: Tomcat's default

- <install-dir>/config/tomcat-users.xml
- Unencrypted: not secure but easy to set up and maintain

```
<?xml version='1.0'?>
```

```
<tomcat-users>
```

```
  <role rolename="manager"/>
```

```
  <role rolename="employee"/>
```

```
  <role rolename="admin"/>
```

```
  <user username="sang" password="sangPassword"  
    roles="manager,employee"/>
```

```
</tomcat-users>
```

Tomcat's default

- Flat file based realm is maintained in
 - <install-dir>/config/tomcat-users.xml
- You can change it in one of two ways
 - manually
 - admintool

Tomcat's admintool

The screenshot shows the Tomcat Web Server Administration Tool interface within a Netscape browser window. The browser's address bar displays `http://localhost:8080/admin/index.jsp`. The page title is "Tomcat Server Administration - Netscape". The main header area contains the text "TOMCAT WEB SERVER ADMINISTRATION TOOL" and two buttons: "Commit Changes" and "Log Out". On the left side, there is a navigation tree with the following items: "Tomcat Server", "Service (Internal Services)", "Service (Java Web Services Developer Pack)", "Resources", "Data Sources", "Environment Entries", "User Databases", "User Definition", "Users", "Groups", and "Roles". The "Users" item is selected, and the main content area displays a "Users List" table. The table has two columns: "User Name" and "Full Name". It lists two users: "daniel" (Daniel Shin) and "sang" (Sang Shin). The "User Name" column contains hyperlinks for each user. Below the table, there is a status bar that reads "Document: Done (0.42 secs)".

Tomcat Server Administration - Netscape

File Edit View Go Bookmarks Tools Window Help

http://localhost:8080/admin/index.jsp Search

Mail Home Radio My Netscape Search Bookmarks

Tomcat Server Administration

**TOMCAT WEB SERVER
ADMINISTRATION TOOL**

Commit Changes Log Out

Tomcat Server

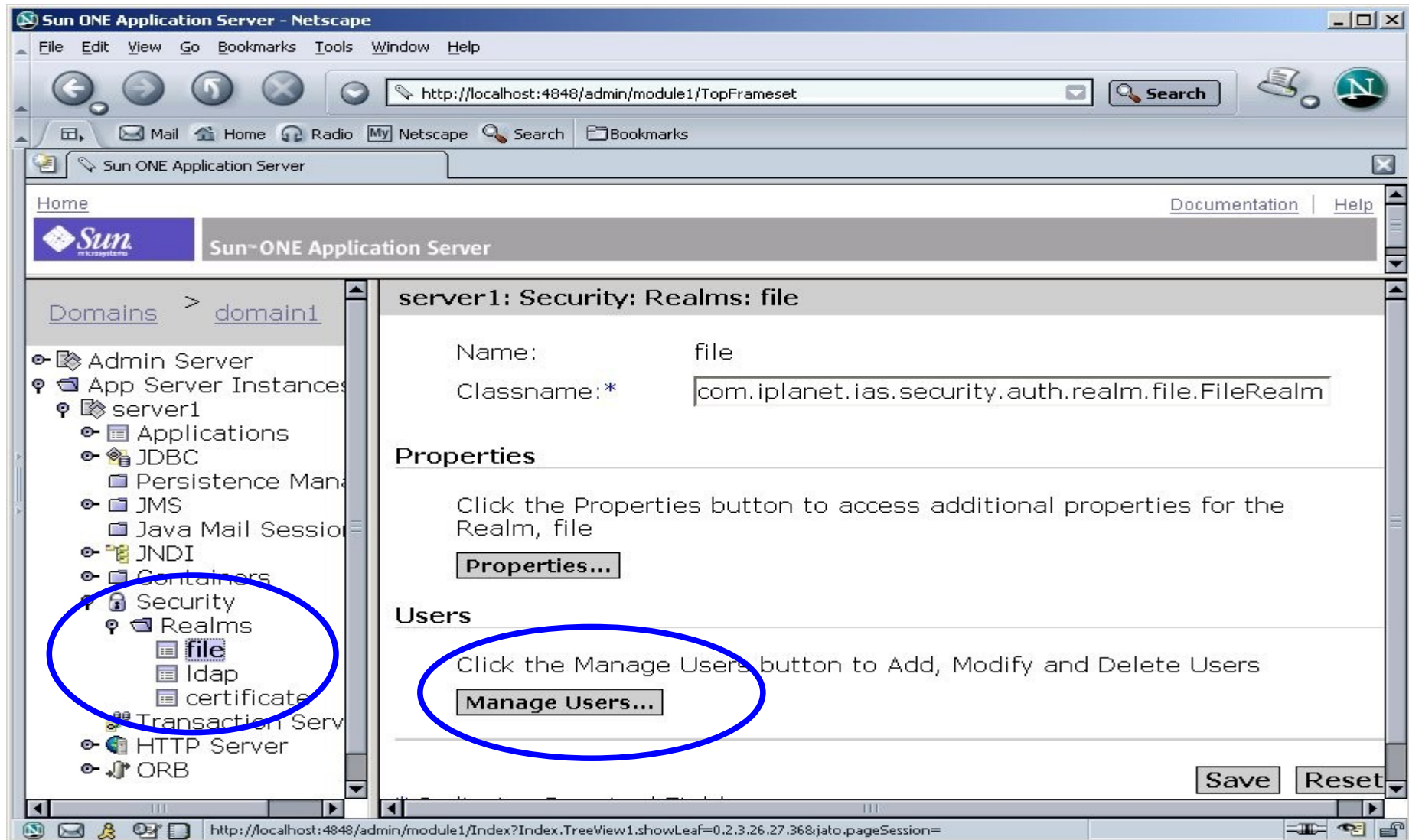
- Service (Internal Services)
- Service (Java Web Services Developer Pack)
- Resources
 - Data Sources
 - Environment Entries
 - User Databases
- User Definition
 - Users**
 - Groups
 - Roles

Users List —Available Actions—

User Name	Full Name
daniel	Daniel Shin
sang	Sang Shin

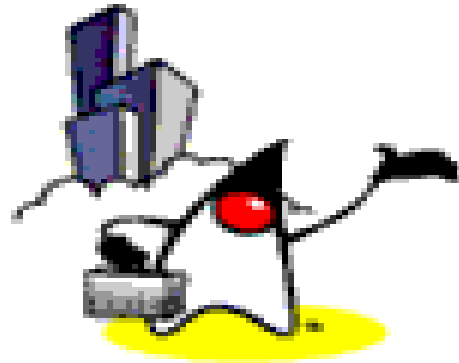
Document: Done (0.42 secs)

Sun App Server Admin Console



How to Create Custom Realms?

- For Tomcat
 - jakarta.apache.org/tomcat/tomcat-4.0-doc/realm-howto.html
- For Sun Java System App Server
 - **Security document that comes with Sun Java System App Server**



How to Secure Passwords on the wire for Basic and Form- based Authentications

Confidentiality of Passwords

- For Basic and Form-based authentication, unless explicitly specified, the password gets transported in unencrypted form (Base64)
- Same confidentiality declaration as regular data
 - If you select **CONFIDENTIAL** or INTEGRAL on a security constraint, that type of security constraint applies to all requests that match the URL patterns in the Web resource collection, not just to the login dialog
 - Uses SSL underneath

SSL-enabled Confidentiality applies to All traffic Including Password

```
<web-app>
```

```
...
```

```
<security-constraint>
```

```
  <web-resource-collection>
```

```
    <web-resource-name>WRCollection</web-resource-name>
```

```
    <url-pattern>/loadpricelist</url-pattern>
```

```
    <http-method>GET</http-method>
```

```
  </web-resource-collection>
```

```
  <auth-constraint>
```

```
    <role-name>admin</role-name>
```

```
  </auth-constraint>
```

```
  <user-data-constraint>
```

```
    <transport-guarantee>CONFIDENTIAL</transport-guarantee>
```

```
  </user-data-constraint>
```

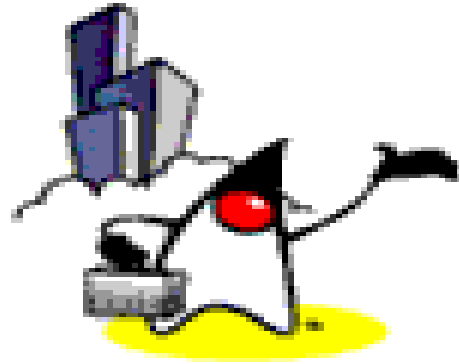
```
</security-constraint>
```

```
<login-config>
```

```
  <auth-method>FORM</auth-method> <realm-name></realm-name>
```

```
</login-config>
```

```
...
```



Web-tier Security Implementation Guidelines

Switching between SSL and non-SSL protected Web resources

- Once you switch to SSL, do not accept any further requests for that session that are non-SSL
 - Because session ID itself is not in encrypted form, impostor might perform a transaction that might involve sensitive data (i.e. credit card)
 - Use Servlet filter to reject any non-SSL requests

SSL is Expensive

- Use SSL only for web resources that need security protection

Ant Operations

- Some of you have experienced the following
 - “ant build” works OK
 - “ant install” or “ant deploy” failure with HTTP 401 error condition
- Why?
 - because “ant install” and “ant deploy” is accessing password-protected resource and you were not providing correct userid/password pair
 - maybe due to non-existence of build.properties file

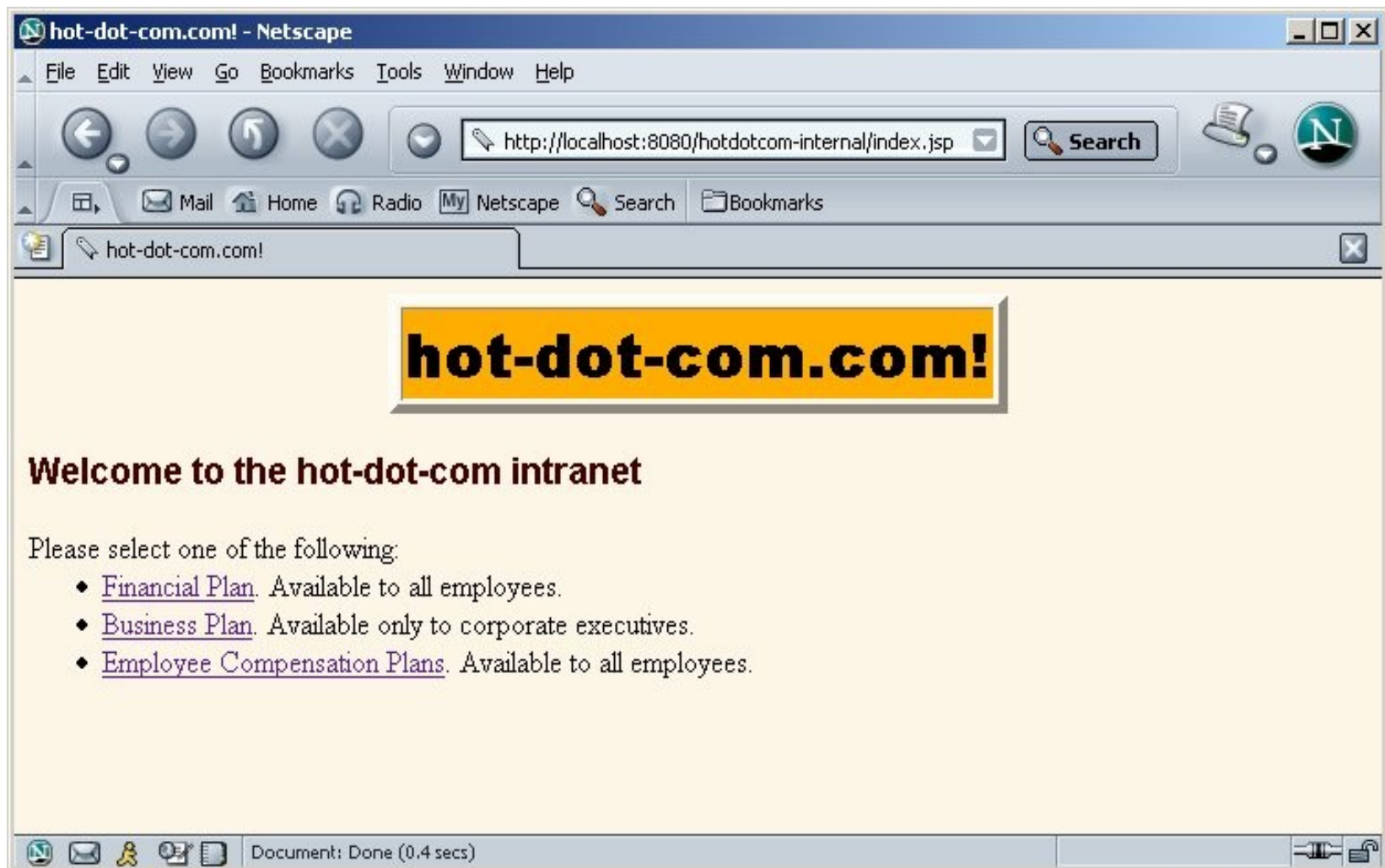
Marty Hall's Sample code Demo

- Download the sample binary and source code from
 - <http://www.moreservlets.com> (select “Source code archive”)
- Unzip into <install-dir>
- Copy “*.war” files under “<install-dir>/Security-Code” into “<jwsdp-home>/webapps” directory
- Add new usernames, roles (the ones used by the code) appropriately to your Tomcat environment (tomcat-users.xml)
- Restart Tomcat

Basic Authentication Demo

- Marty Hall's Sample code (hotdotcom-internal.war, <http://www.coreservlets.com/>)
 - Financial plan page: available to all employees
 - Business plan page: available only to executives
 - Employee compensation plan: available to all employees
- Try to access “access controlled” page
- Enter bogus username & password
- Enter valid username & password but who does not have access right (does not belong to a proper role)

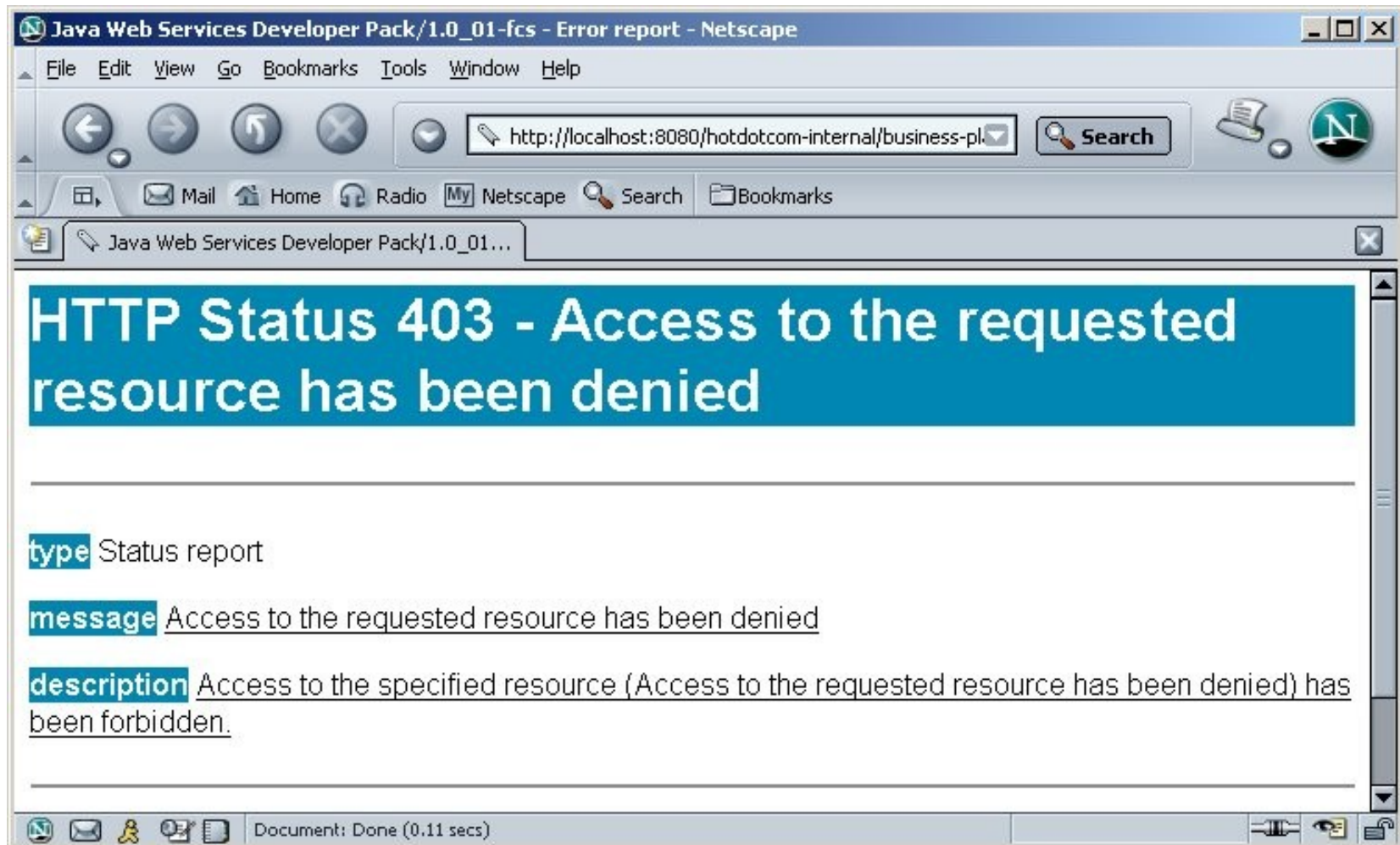
Basic Authentication Demo

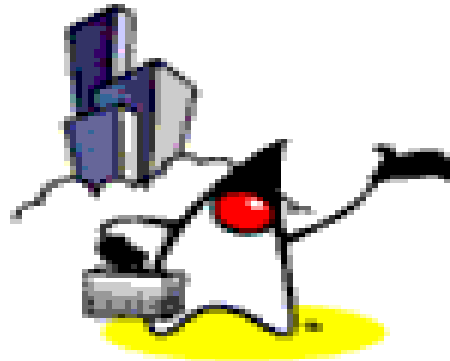


Access “access controlled” page with bogus username



Access “access controlled” page with valid username





Demo: Form-based Authentication

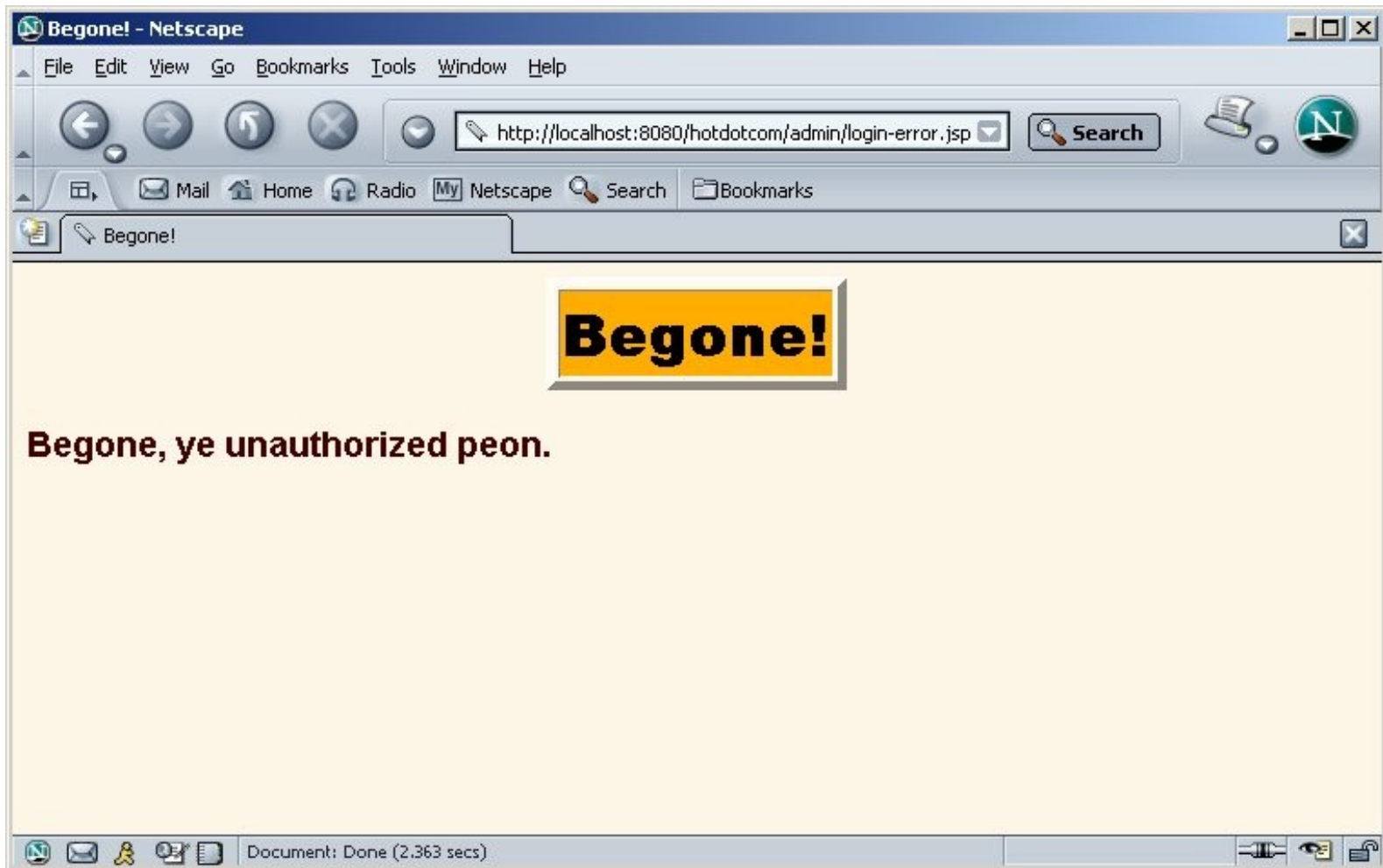
Form-based Authentication

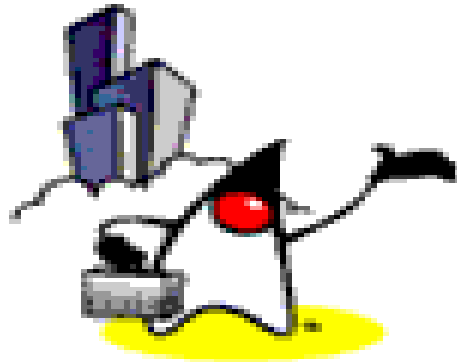


Custom login page



Custom error page





Client Certificate-based Authentication

Why Certificate-based Authentication?

- Username/password authentication cannot be used between program to program authentication
 - Certificates may identify end-users, business organizations, servers, software entities
- Username/password pair might not provide enough credentials
 - Certificate can contain much more than username and password

Certificate-based Authentication

- Client authentication
 - server verifies client's identity
- Server authentication
 - client verifies server's identity
 - occurs “transparently” in SSL-based browser and web server communication
- Mutual authentication
 - both server and client verify each other's identity

Certificate Formats

- Standard Certificate format is X.509
- X.509 does specify the format of the certificate but does not specify how certificates are exchanged
- SSL specifies how certificates are exchanged

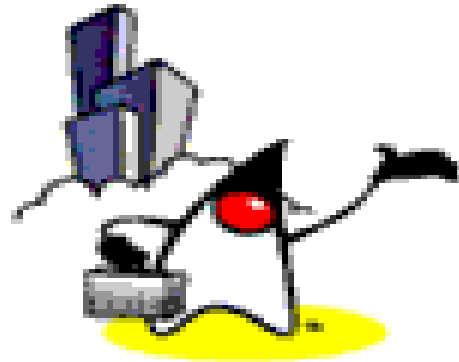
Client Certificate-based Authentication

- Client gets authenticated by sending **Client certificate** to Web server
 - When the server gets authenticated to the client as well, we call it mutual authentication
- Client certificate has to be created beforehand for each browser based client
 - Not as popular as Basic or Form-based authentication because of this reason
- Uses SSL over HTTP

Tell web container that you are using Client-Cert authentication

- In `web.xml` file of your web application

```
<web-app>
...
<security-constraint>...</security-constraint>
<login-config>
    <auth-method>CLIENT-CERT</auth-method>
    <realm-name>realm name</realm-name>
</login-config>
...
</web-app>
```



Digest Authentication

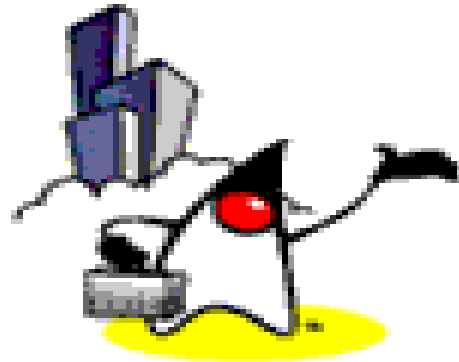
Digest Authentication

- User password is transformed into “digested form” before being sent to the server
 - You cannot get the original password from the digested password (property of message digesting)
 - Even a single bit of change of original password results in different digested password value
 - No cleartext (or uuencoded) user password on the wire even without on a non-SSL connection
- Server compares the passed digested value with its own, if matched, then authentication succeeds

Tell web container that you are using Digest authentication

- In `web.xml` file of your web application

```
<web-app>
...
<security-constraint>...</security-constraint>
<login-config>
    <auth-method>DIGEST</auth-method>
    <realm-name>realm name</realm-name>
</login-config>
...
</web-app>
```

Authorization (Access Control)

4 Types of Authorization (Access Control) over J2EE

- Web-tier vs. EJB-tier
 - Can be used together
- Declarative vs. Programmatic
- 4 possible types
 - Declarative access control at Web-tier
 - Programmatic access control at Web-tier
 - Declarative access control at EJB-tier
 - Programmatic access control at EJB-tier

Web-tier vs. EJB-tier

Web-tier

- (D) Access control to Web resources
- (D) Declared in web.xml
- (D) Enforced by web container
- (P) Coded in servlet or JSP

EJB-tier

- (D) Access control to bean methods
- (D) Declared in EJB deployment descriptor
- (D) Enforced by EJB container
- (P) Coded in EJB bean

(D): Declarative (P): Programmatic access control

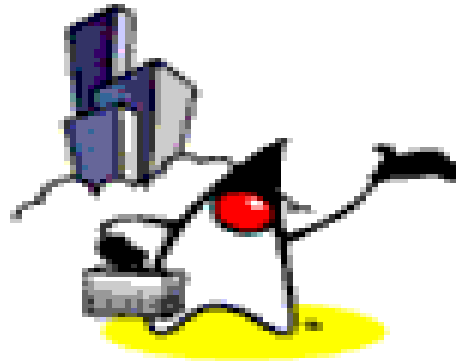
Declarative vs. Programmatic

Declarative

- Access control is declared in deployment descriptor
- Container handles access control
- Does not handle fine-grained access control, it is all or nothing

Programmatic

- Access control is coded in your program
- Your code handles access control
- Can handle fine-grained access control, i.e. instance-based or business logic based access

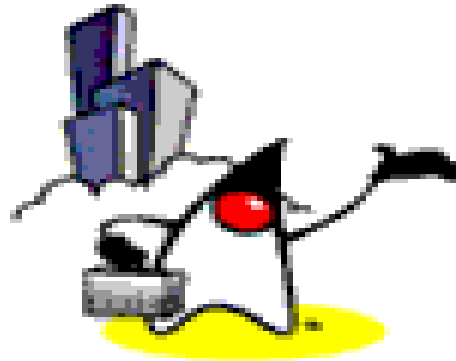


Declarative Authorization at Web-tier

Steps for Declarative Authorization at Web-tier

1. Deployer maps **actual user identities** to security roles using vendor specific tools
2. Deployer declares **security roles** in the deployment descriptor
3. Deployer declares **URL permissions** in the deployment descriptor for each security role

This is already covered above under “Web-tier security schemes” segment!



Programmatic Authorization at Web-tier

Declarative and Programmatic Authorization (Access Control)

- They are usually used together
 - Declarative access control for role-based access control
 - Programmatic access control for user instance-based and business logic based access control
 - User instance
 - Time of the day
 - Parameters of the request
 - Internal state of the web component

Steps for Programmatic Authorization at Web-tier

1. Set up username, passwords, and roles (realms)
2. Servlet programmer writes **programmatic authorization logic** inside the Servlet code using abstract security roles
3. Deployer maps abstract security roles to **actual roles** (for a particular operational environment) in the web.xml

Step 2: Servlet Programmer writes programmatic authorization logic

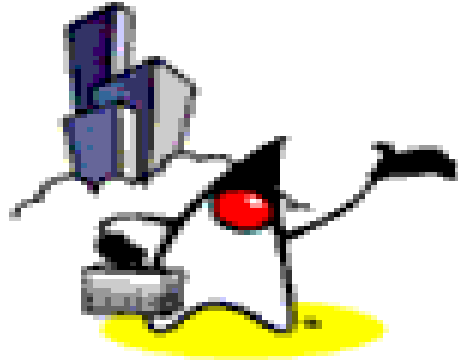
```
public interface javax.servlet.http.HttpServletRequest{  
    ...  
    // Find out who is accessing your web resource  
    public java.security.Principal getUserPrincipal();  
    public String getRemoteUser();  
  
    // Is the caller in a particular role?  
    public boolean isUserInRole(String role);  
    ...  
}
```

Example: “Employees” can only access their own Salary Information

```
public double getSalary(String employeeId) {  
    java.security.Principal userPrincipal =  
        request.getUserPrincipal();  
    String callerId = userPrincipal.getName();  
  
    // "manager" role can read employee salary information  
    // employee can read only his/her own salary information  
    if ( (request.isUserInRole("manager")) ||  
        ((request.isUserInRole("employee")) &&  
         (callerId == employeeId)) ) {  
        // return Salary information for the employee  
        getSalaryInformationSomehow(employeeId);  
    } else {  
        throw new SecurityException("access denied");  
    }  
}
```

Step 3: Deployer maps abstract security roles to actual roles

```
<web-app>
...
<servlet>
  <servlet-name>...</servlet-name>
  <servlet-class>...</servlet-class>
  <!-- Servlet programmer declared abstract security roles -->
  <security-role-ref>
    <role-name>manager</role-name>           <!-- preexisting -->
    <role-link>managerOfAcme</role-link>      <!-- alias -->
  </security-role-ref>
</servlet>
...
</web-app>
```



Programmatic Authentication at Web-tier

Programmatic Authentication at Web-tier

- Web application itself performs the authentication
 - More customized control is possible (but this is rather marginal benefit)
- Rarely used in practice

Steps for Programmatic Authentication at Web-tier

- Check if there is authorization header
- Decode the Base64 encoded username and password
- Check username/password pair against security realm
 - If successful match, performs access control
 - if access control succeeds, return page
 - otherwise, display appropriate page
 - Otherwise (authentication fails), send back ask for new username & password

Check if there is authentication Header

```
public void doGet(...)... {  
  
    // Check if authentication header is present in  
    // HttpServletRequest. If not, ask for it.  
    String authorization =  
        request.getHeader("Authorization");  
    if (authorization == null) {  
        askForPassword(response);  
    } else {  
        ...  
    }  
}
```

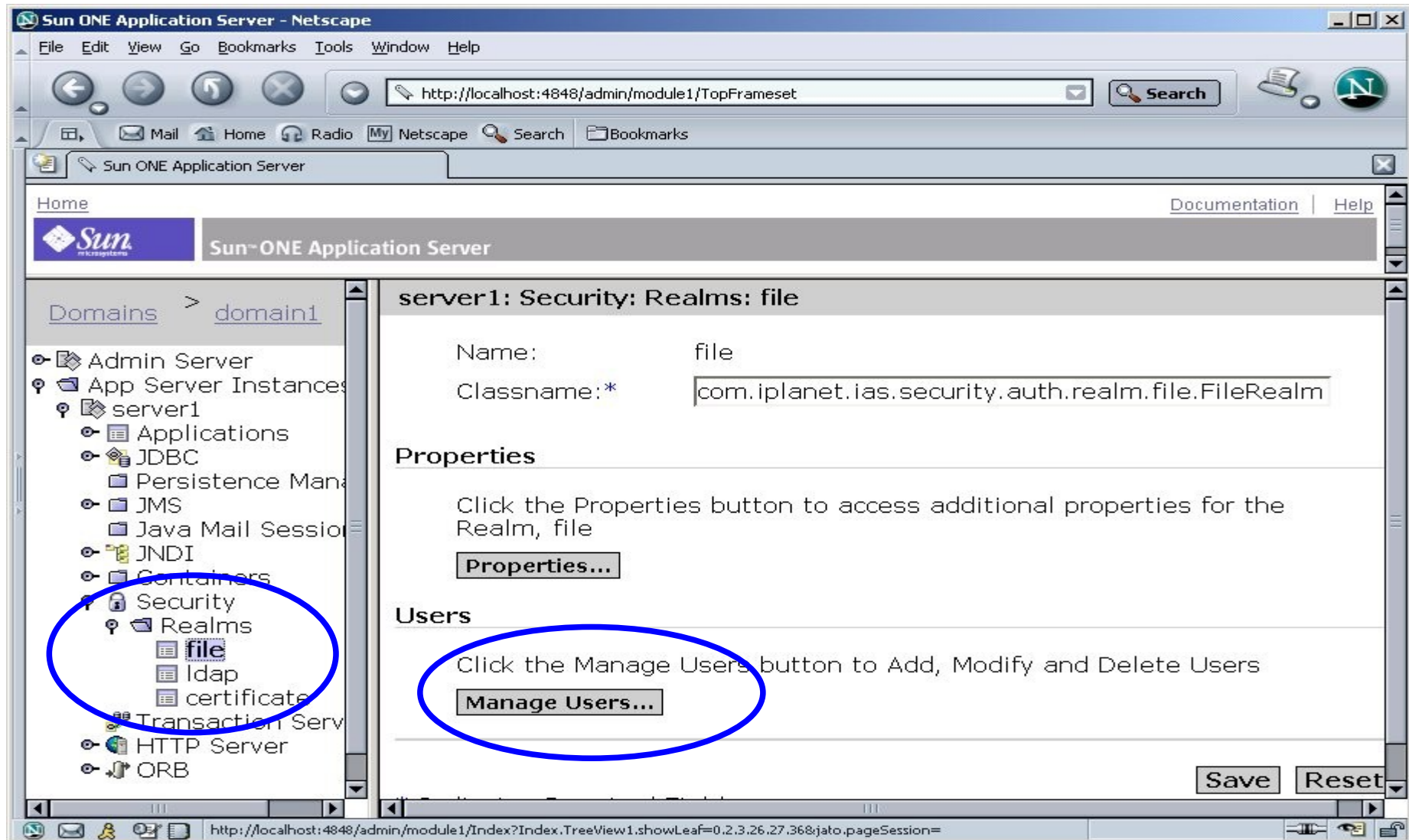

Decode Username and Password

```
if (authorization == null) {
    askForPassword(response);
} else {
    String userInfo =
        authorization.substring(6).trim();
    BASE64Decoder decoder = new BASE64Decoder();
    String nameAndPassword =
        new
String(decoder.decodeBuffer(userInfo));
    int index = nameAndPassword.indexOf(":");
    String user =
        nameAndPassword.substring(0, index);
    String password =
        nameAndPassword.substring(index+1);
    ...
}
```

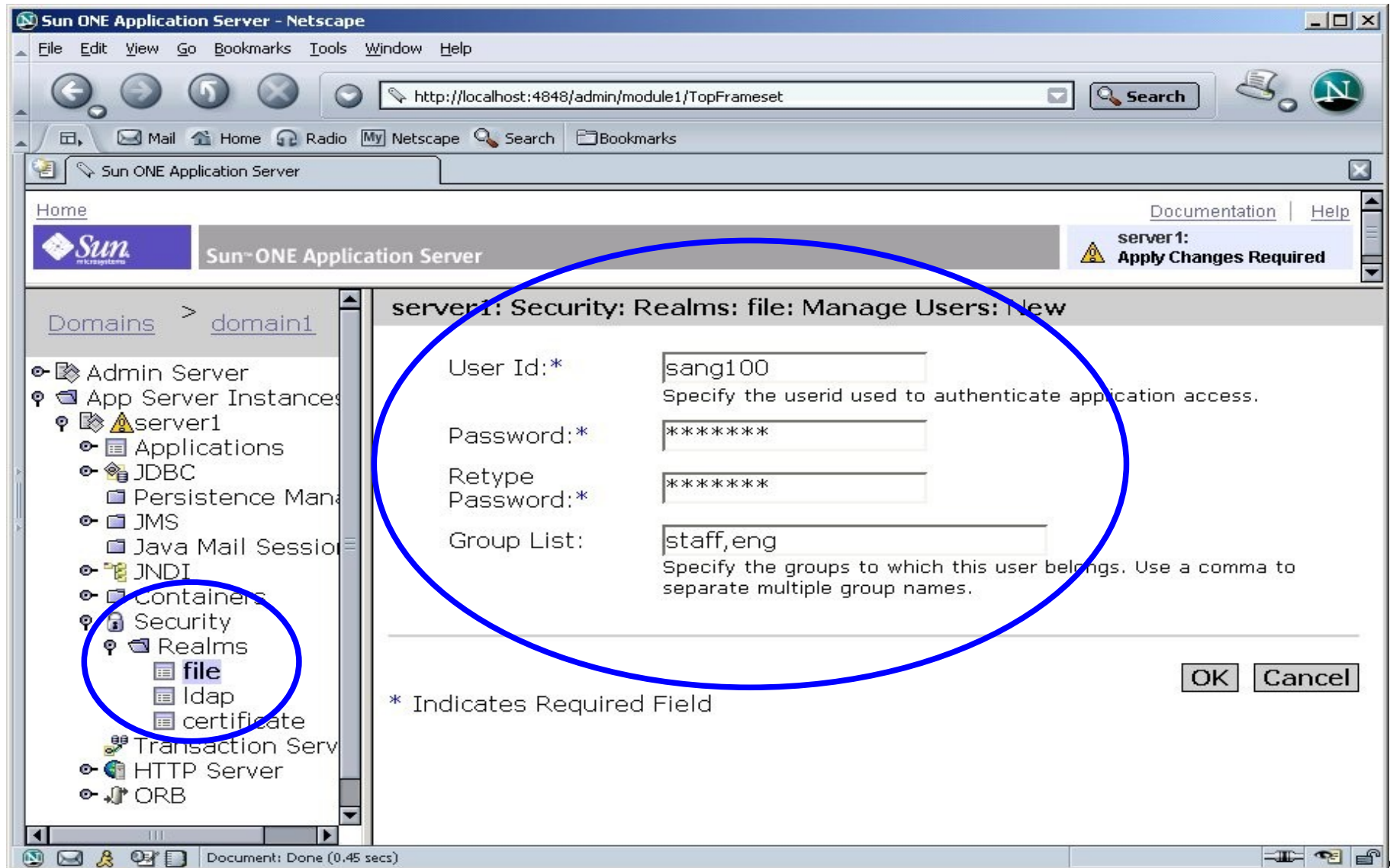
If success, return page, Otherwise ask for a new username & password

```
if (authorization == null) {
    askForPassword(response);
} else {
    ...
    // If authentication succeeds, return page.
    // Otherwise, ask for correct username & password
    if (areEqualReversed(user, password)) {
        showStock(request, response);
    } else {
        askForPassword(response);
    }
}
}
```

Open App Server Admin Console



Add Userid/Password Pair to File Realm



Restart App Server

Sun ONE Application Server - Netscape

File Edit View Go Bookmarks Tools Window Help

http://localhost:4848/admin/module1/TopFrameset

Sun ONE Application Server

Home Documentation Help

server1: Apply Changes Required

Domains > domain1

- Admin Server
- App Server Instances
 - server1
 - Applications
 - JDBC
 - Persistence Manager
 - JMS
 - Java Mail Session
 - JNDI
 - Containers
 - Security
 - Realms
 - file
 - ldap
 - certificate
 - Transaction Service
 - HTTP Server
 - ORB

server1

General JVM Settings Logging Monitoring Advanced

Start Stop Delete Apply Changes

Start in Debug Mode: ☐

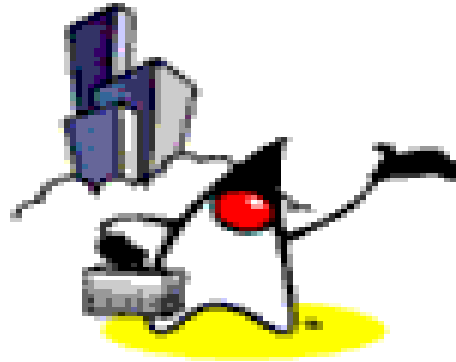
Apply Changes Required
Configuration changes have not yet been applied. "Apply Changes" to make these changes effective.

Instance Name: server1

Host Name: sang-laptop

HTTP Port(s): 80

http://localhost:4848/admin/module1/Index?Index.TreeView1.showFolder=0.2.38:jato.pageSession=



Sun Java System App Server 8 Security Features

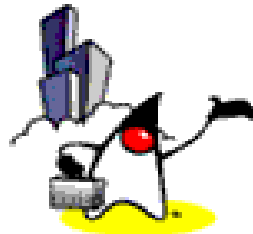
Sun Java System App Server J2EE Security Features

- Declarative and programmatic security
- Realm administration
 - file, LDAP, certificate, Solaris-based realms
- Pluggable authentication via JAAS
 - You can add custom realm
- Single sign-on (value-add)
 - Same authenticate state shared among multiple J2EE applications
- Programmatic login (value-add)



Demo

(Sun Java System App Server Security)



Demo1: Basic Authentication

1. Deploy basic-auth web application (from Sun App Server samples)
2. From a browser, access a protected web resource
 - <http://localhost:80/basic/index.jsp>
3. Web container asks browser to display a default login dialog box
4. Enter an invalid userid/password pair
 - Container keep asking browser to display dialog box
5. Using Sun App Server Admin console, add the userid/password pair to the “userid/password” file-based realm
6. Restart Sun App Server
7. Access the protected web resource again

Enter invalid userid/password



Prompt [X]

Enter username and password for "basic-file" at localhost:80

User Name:

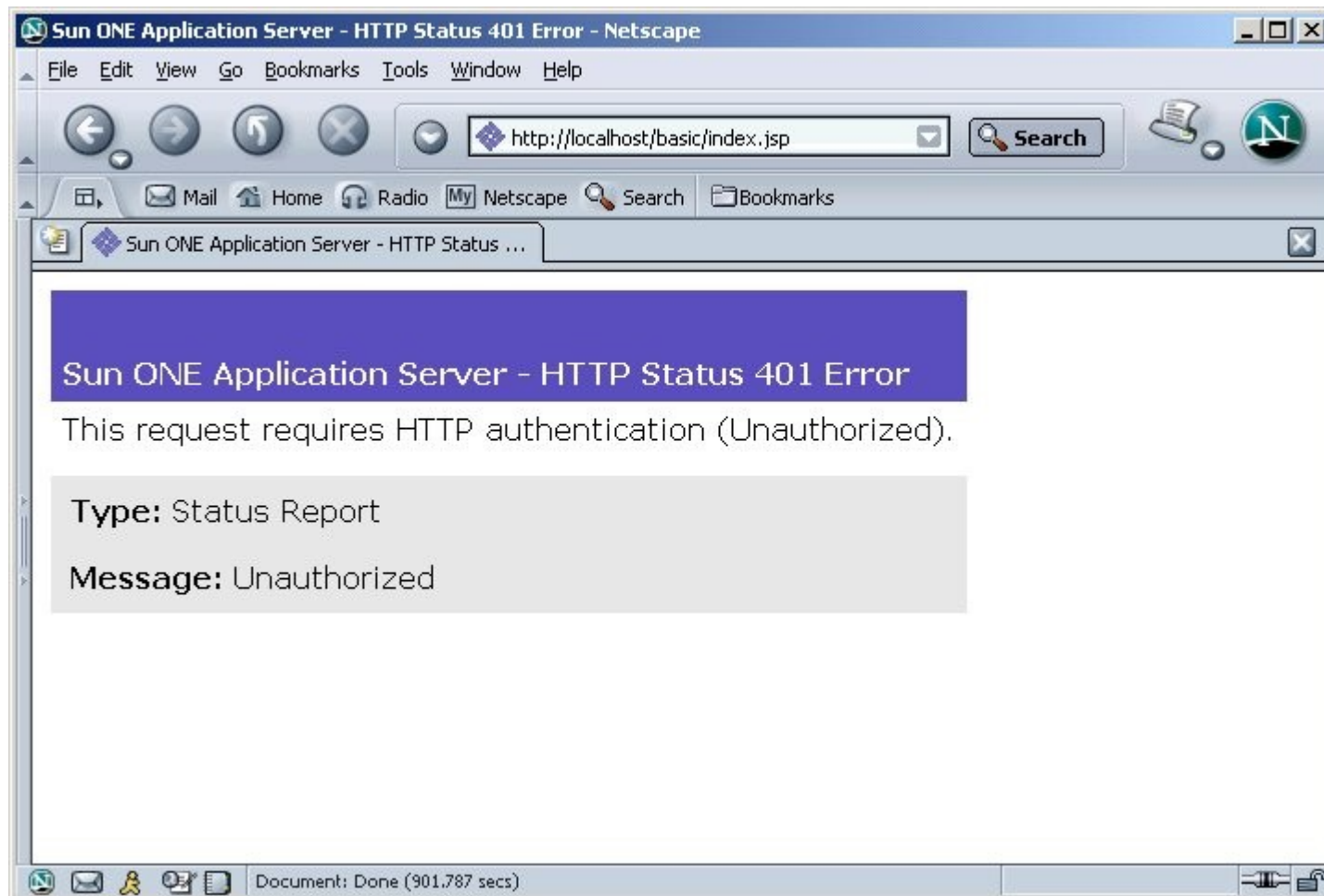
sang100

Password:

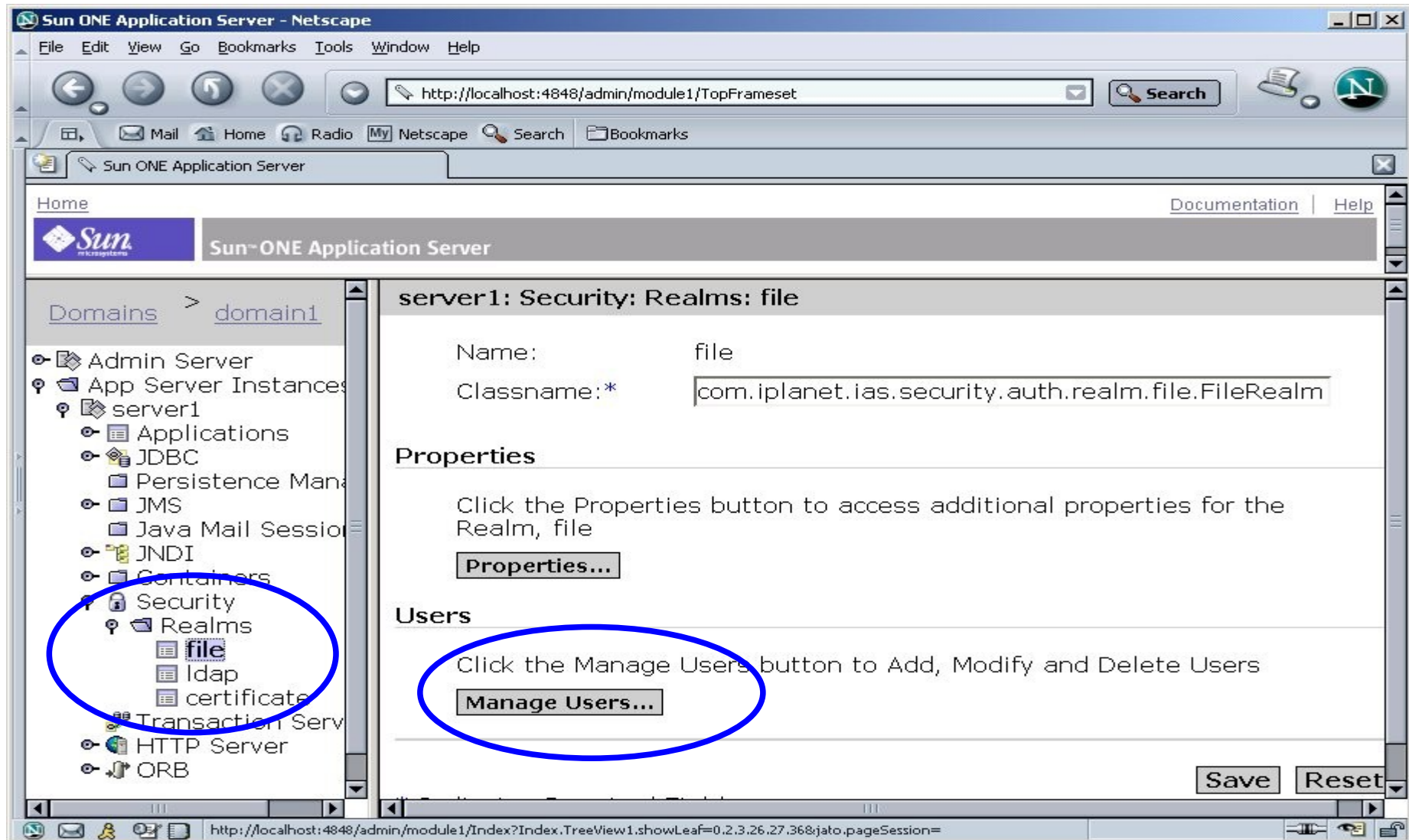
☐ Use Password Manager to remember these values.

OK Cancel

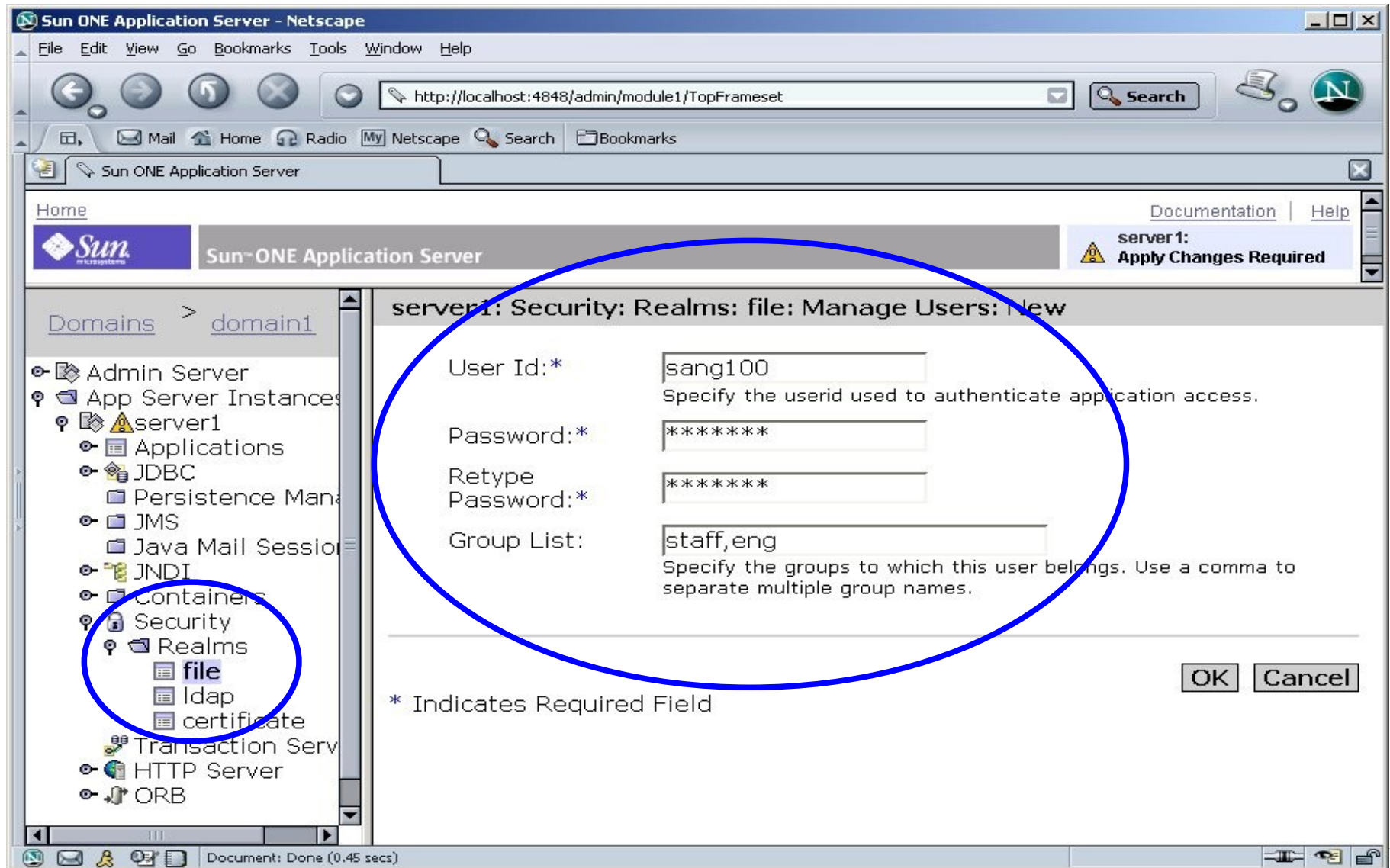
Authentication Failed



Open App Server Admin Console



Add Userid/Password Pair to File Realm



Restart App Server

Sun ONE Application Server - Netscape

File Edit View Go Bookmarks Tools Window Help

http://localhost:4848/admin/module1/TopFrameset

Sun ONE Application Server

Home Documentation Help

server1: Apply Changes Required

Domains > domain1

- Admin Server
- App Server Instances
 - server1
 - Applications
 - JDBC
 - Persistence Manager
 - JMS
 - Java Mail Session
 - JNDI
 - Containers
 - Security
 - Realms
 - file
 - ldap
 - certificate
 - Transaction Service
 - HTTP Server
 - ORB

server1

General JVM Settings Logging Monitoring Advanced

Start Stop Delete Apply Changes

Start in Debug Mode: ☐

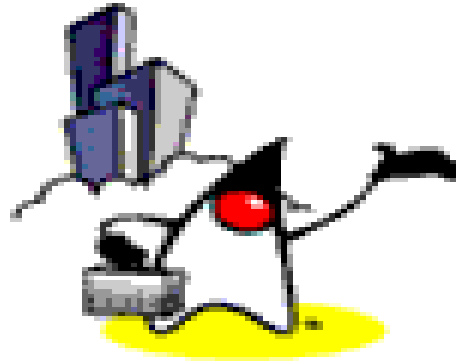
Apply Changes Required
Configuration changes have not yet been applied. "Apply Changes" to make these changes effective.

Instance Name: server1

Host Name: sang-laptop

HTTP Port(s): 80

http://localhost:4848/admin/module1/Index?Index.TreeView1.showFolder=0.2.38:jato.pageSession=



Sun Java System App Server 8 Security Features

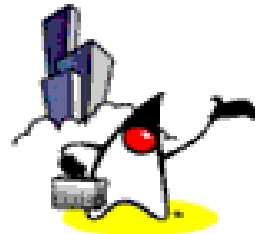
Sun Java System App Server J2EE Security Features

- Declarative and programmatic security
- Realm administration
 - file, LDAP, certificate, Solaris-based realms
- Pluggable authentication via JAAS
 - You can add custom realm
- Single sign-on (value-add)
 - Same authenticate state shared among multiple J2EE applications
- Programmatic login (value-add)

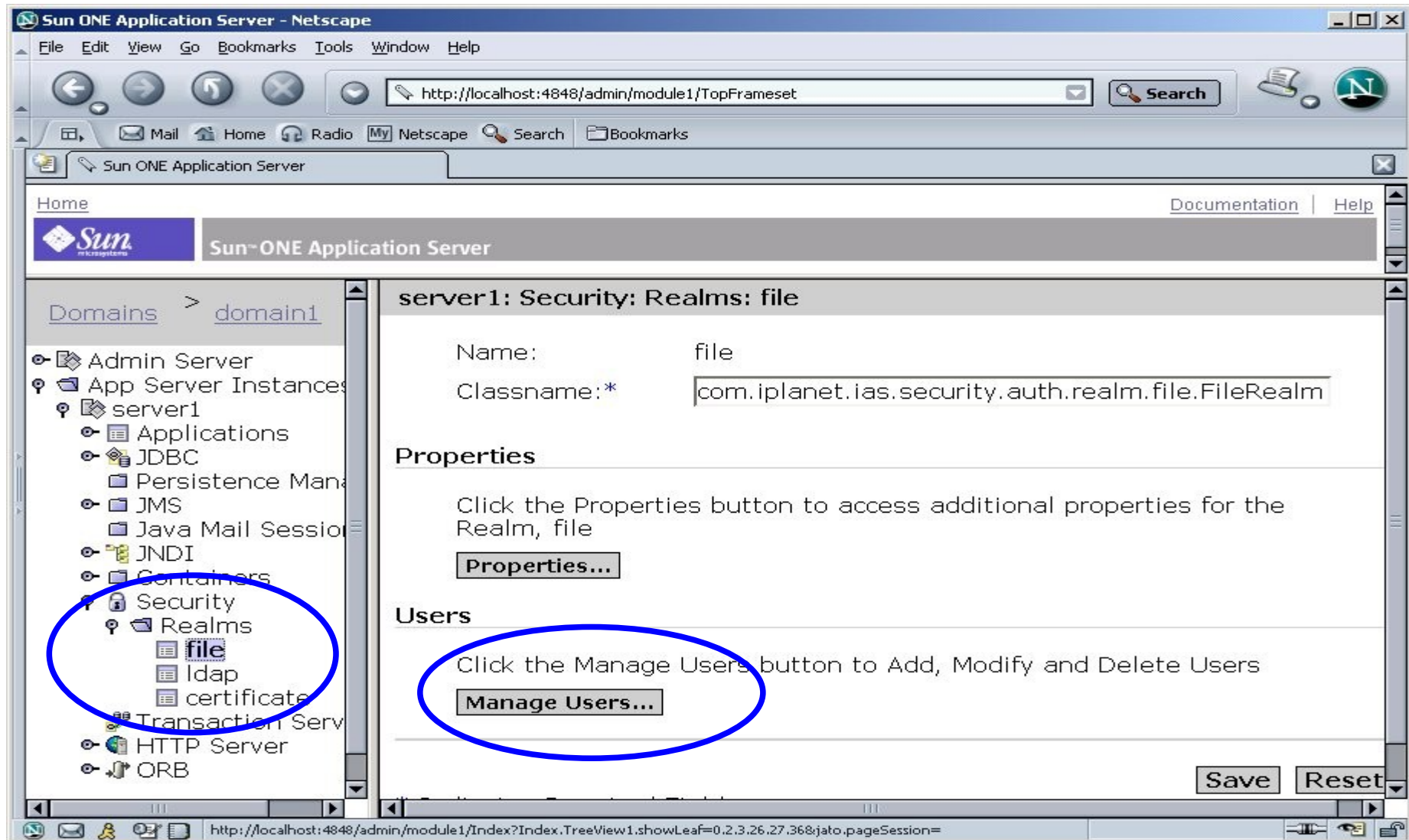


Demo

(Sun Java System App Server Security)



Open App Server Admin Console



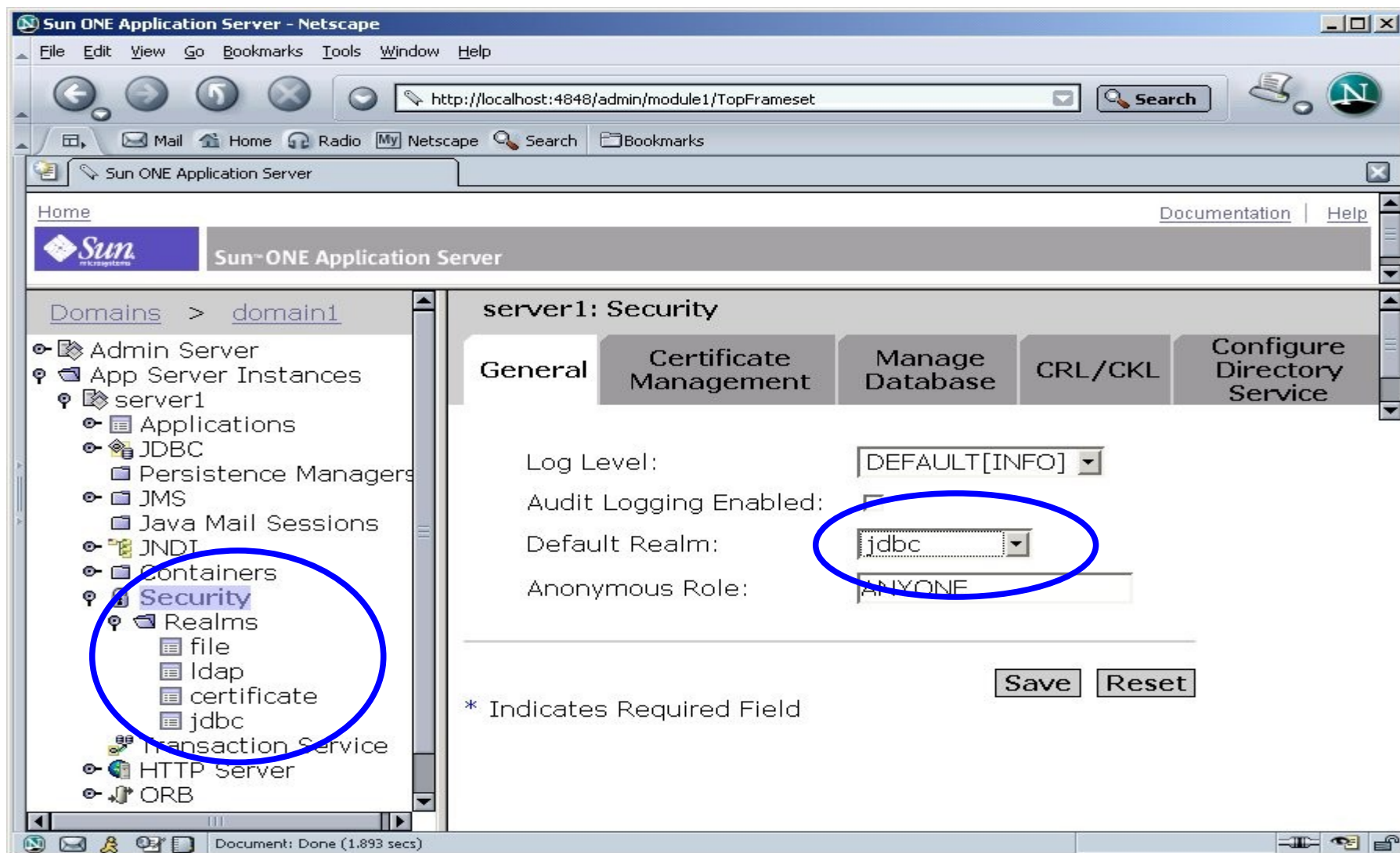
Access the Protected Page Again and Enter the same userid/password pair



Demo2: Add Custom Realm

- Add JDBC custom realm
- Use pointbase or Oracle database as storage of userid/password information (instead of file realm)

Use JDBC Realm as Default Realm





Passion!

