Seguridad en JEE

Sesión2: Autentificación-Autorización (JAAS) y Comunicación Segura (SSL)

Índice

- Autentificación-Autorización (JAAS)
- Comunicación Segura (SSL)

Índice

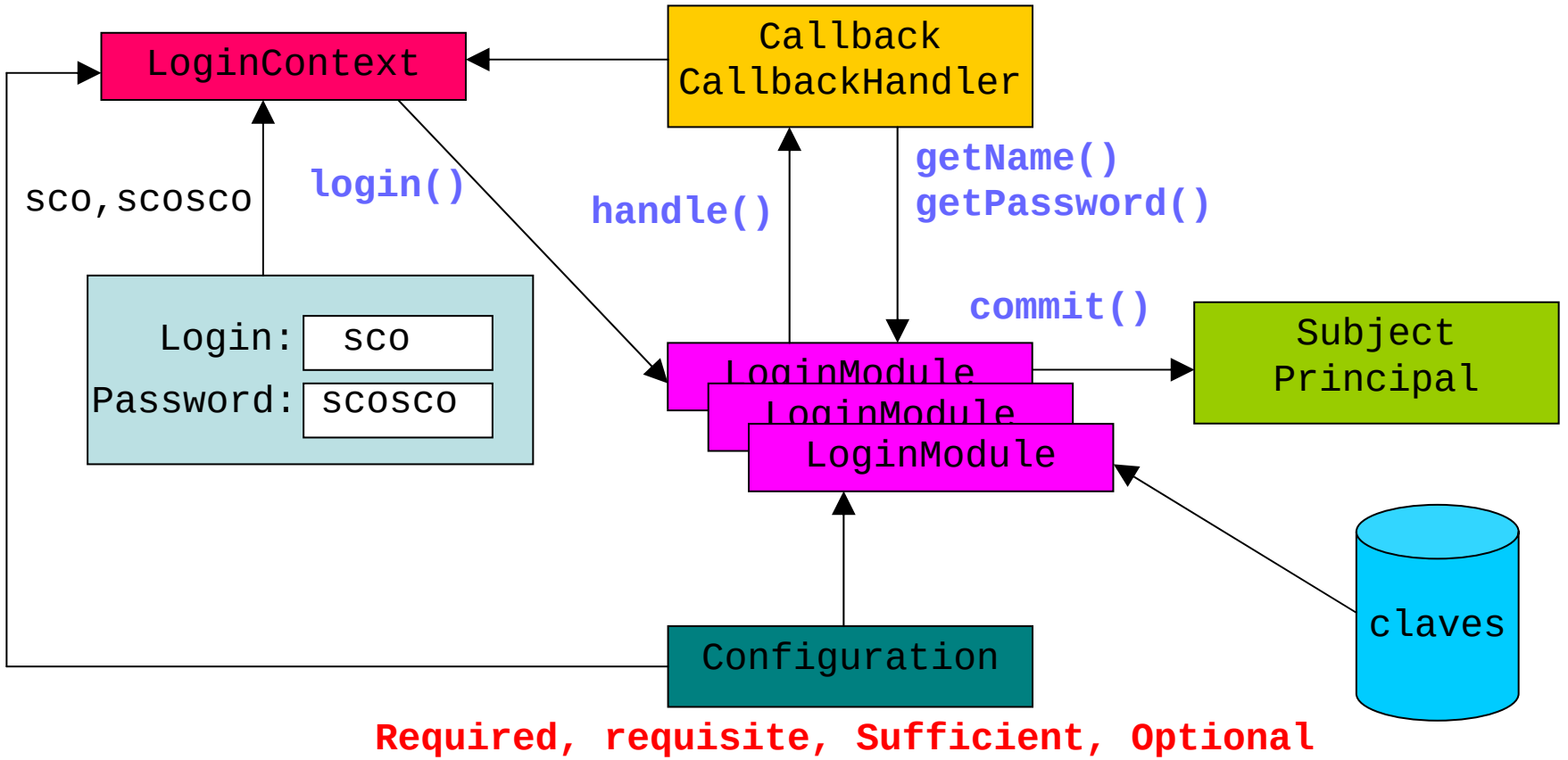
- Autentificación-Autorización (JAAS)
- Comunicación Segura (SSL)

JAAS

- Motivación:
  - API orientado a conceder permisos en función de quien ejecuta el código.
  - Basado en PAMs (Pluggable Authentication Modules)
  - Mecanismo básico: login – password
  - JAAS abarca tanto la identificación (p.e. con login-password) como la autorización a los recursos. (lo veremos más adelante en BEA WebLogic).
  - Por ejemplo, el servidor de EJBs de BEA WebLogic implementa seguridad con JAAS.

JAAS

- Esquema (elementos)



JAAS

- Secuencia:
  1. Aplicación: pide login-password y espera contestación.
  2. `LoginContext:` Contexto al que pasamos un manejador (`CallbackHandler`) parametrizado por un nombre de usuario y un password.
  3. `CallbackHandler:` Recibe datos del log-in.
  4. `Configuration:` Especificación de uso del LoginContext y del módulo de login (LoginModule).
  5. `LoginModule:` Comprueba credenciales del sujeto y prosigue o aborta dependiendo del resultado.
  6. Aplicación: Recibe los Subject y sus Principal

login: LoginContext

- EjemploJAAS.java
- LoginContext:

```
String usuario = args[0];
char[] password = args[1].toCharArray();
LoginContext loginContext =
    new LoginContext("Ejemplo", new
UserPasswordCallbackHandler(usuario, password));
```

login: Configuration

- `jaas.config:` **`PasswordLoginModule.java`**

```
Ejemplo { PasswordLoginModule required; };
```

  - `Required:` Debe tener éxito para que el log-in completo tenga éxito. Incluso si falla se consulta a otros módulos.
  - `Requisite:` Debe tener éxito. Si falla, el proceso de log-in es cortocircuitado y no se llama a ningún otro módulo de login.
  - `Sufficient:` Si tiene éxito y ningún otro modulo “required” o “requisite” falla, el log-in completo tiene éxito.
  - `Optional:` El éxito no influye en el resto del proceso. Si ningún módulo de los tres tipos anteriores falla el log-in tiene éxito independientemente de que otro “optional” tenga éxito.

callback: CallbackHandler

- `UserPasswordCallbackHandler.java`
- Implementa la interfaz: `CallbackHandler`
  - Constructor recibe los datos del log-in (nombre de usuario y password).
  - `handle()`: Recibe del `LoginModule` un array de objetos `Callback` que pueden ser de dos tipos: `NameCallback` o bien `PasswordCallback` , e inicializa sus valores con los datos del log-in
  - Dicha inicialización se hace a través de los métodos `setName()` y `setPassword()`.

callback: CallbackHandler, Callback

```
    public UserPasswordCallbackHandler(String usuario, char[]
    password) {
        mUsuario = usuario; mPassword = password;
    }
```

```
public void handle(Callback[] callbacks) throws UnsupportedCallbackException {
for(int i=0;i<callbacks.length;i++) {
     Callback callback = callbacks[i];
if (callback instanceof NameCallback) {
       NameCallback nameCallback = (NameCallback)callback;
       nameCallback.setName(mUsuario);
     } else if (callback instanceof PasswordCallback) {
       PasswordCallback passwordCallback = (PasswordCallback)callback;
       passwordCallback.setPassword(mPassword);
     } else {
       throw new UnsupportedCallbackException(callback, "Tipo de callback no
soportado"); } }
```

spi.LoginModule

- `PasswordLoginModule.java`
- Interfaz: `LoginModule`
  - `initialize()`: Inicializa el módulo para un intento.
  - `login()`: Comprueba credenciales del usuario (acceso a BD, leer fichero encriptado de passwords,….).
  - `commit()`: Se invoca solo si login() tiene éxito. Añade las identidades y credenciales del sujeto, se limpia el estado y se añade el sujeto al contexto actual.
  - `abort()`: Si el login() fallá se invoca este método y se limpia el estado del login.
  - `logout()`: Borra identidades y credenciales necesarias.

spi.LoginModule

```
public boolean login() throws LoginException {

 if (mCallbackHandler == null) {
   throw new LoginException("CallbackHandler no definido");
 }

Callback[] callbacks = new Callback[2];
callbacks[0] = new NameCallback("Usuario");
callbacks[1] = new PasswordCallback("Password", false);

try {

  mCallbackHandler.handle(callbacks);
  mUsuario = ((NameCallback)callbacks[0]).getName();
  char[] tempPassword = ((PasswordCallback)callbacks[1]).getPassword();
  mPassword = new char[tempPassword.length];
  System.arraycopy(tempPassword, 0, mPassword, 0, tempPassword.length);
  ((PasswordCallback)callbacks[1]).clearPassword();
} catch (IOException ioe) {
    throw new LoginException(ioe.toString());
} catch (UnsupportedCallbackException uce) {
    throw new LoginException(uce.toString());}
```

spi.LoginModule

```
if (
        "sco".equals(mUsuario) &&
        mPassword.length == 6 &&
        mPassword[0] == 's' &&
        mPassword[1] == 'c' &&
        mPassword[2] == 'o' &&
        mPassword[3] == 's' &&
        mPassword[4] == 'c' &&
        mPassword[5] == 'o'
      ) {

          mLoginExito = true;
          return true;
      } else {
          mLoginExito = false;
          mUsuario = null;
          clearPassword();
          throw new FailedLoginException("Password Incorrecto");
      }
  }
```

Subject

- `Subject:`
  - `Subject:` Entidad o “sujeto” que puede tener diferentes identidades o `java.security.Principal`. Dichas identidades las devuelve `getPrincipals()`en un `Set.`
  - `Credentials:`Lista de “credenciales” (passwords, certificados): `getPrivate{Public}Credentials().` Las clases `Publickey` y `Certificate` pueden usarse como credenciales.
  - Los sujetos representan quien está ejecutando el código. El “sujeto activo” se obtiene de `getSubject().`Es JAAS quien asigna dicho “sujeto activo”.

Subject

- `Subject:` (continuación)
  - El método `commit()` de la interfaz `LoginModule` llama a una implementación de la interfaz `Principal` (ver código `ImplPrincipal.java`) crear una identidad y añadirla al objeto `Subject.`
  - En ese momento podemos añadir credenciales al `Subject`.
  - Tras el `commit()`desde la aplicación podemos acceder al contexto del sujeto activo:

```
// 3. getSubject() e imprimir
Subject subject = loginContext.getSubject();
System.out.println(subject);
```

Subject

```
public boolean commit() throws LoginException {
  if (mLoginExito == false) {
       return false;
  }
  // Login con éxito: crear Principal y añadirlo al Subject
  mPrincipal = new ImplPrincipal(mUsuario);
  if (!(mSujeto.getPrincipals().contains(mPrincipal))) {
      mSujeto.getPrincipals().add(mPrincipal);
  }
     // Si queremos que el Subject contenga credenciales
     // este es el momento para añadirlas.

     // Borrar usuario y password.
     mUsuario = null;
     clearPassword();
     mCommitExito = true;
     return true; }
```

Probando el ejemplo

- Elementos:
  - `EjemploJAAS.java`
  - `jaas.config`
  - `UserPasswordCallbackHandler.java`
  - `PasswordLoginModule.java`
  - `ImplPrincipal.java`
- Probar:
  - `java –Djava.security.auth.login = jaas.config EjemploJAAS sco scosco`

Autorización programática

- Motivación y ejemplo:
  - Cuando interesa determinar si quien está ejecutando el código (p.e. el “sujeto activo”) está autorizado o no.
  - `EjemploJAAS2.java`. Independientemente de que el login tenga éxito llamamos al método `doAs()` de la clase `Subject`:

```
sujeto.doAs(sujeto, new ActionEjemplo);
```

  - Donde `AccionEjemplo` es una clase que implementa la interfaz `java.security.PrivilegedAction`

Autorización programática

- Ejemplo:
  - La interfaz `PrivilegedAction` contiene únicamente el método `run()`. Supongamos que dicho método contiene una llamada a un método nuestro llamando `getSecretText()`.

```
class AccionEjemplo  implements PrivilegedAction {
  public AccionEjemplo() {}

  public Object run() {
      System.out.println("Texto secreto: " +
EjemploJAAS2.getSecretText());
      return null;
  }
}
```

Autorización programática

- Ejemplo:
  - El método `getSecretText()` está diseñando para dar una respuesta dependiendo de si el sujeto está autorizado o no (ver transparecia siguiente).
  - Para saberlo hemos definido previamente una instancia de `java.security.AccessControlContext` que obtenemos llamando al método `getContext()` de `java.security.AccessController` y una vez obtenida pasársela al `getSubject()` de

```
AccessControlContext contexto = AccessController.getContext();
Subject sujeto = Subject.getSubject(contexto);
```

Autorización programática

```
if (sujeto == null) {
   System.out.println("Sujeto null");
   return TEXTO_GENERICO;
}

// Obtener todos los principales: instancias de ImplPrincipal.
// Devolver el texto secreto si el usuario "sco"
Set principales = sujeto.getPrincipals();
Iterator iterador = principales.iterator();
while (iterador.hasNext()) {
   ImplPrincipal principal = (ImplPrincipal)iterador.next();
   if (principal.getName().equals("sco")) {
       return TEXTO_PARTICULAR;
   }
}
return TEXTO_GENERICO;
```

Índice

- Autentificación-Autorización (JAAS)
- Comunicación Segura (SSL)

Ejercicios…

- Ejemplos JAAS y SSL en WebLogic:
  - Probad los ejemplos del directorio de seguridad en Weblogic (JAAS y SSL)
- Usar un ldapbrowser con WebLogic:
  - Usar un ldapbrowser para ver el contenido del LDAP de WebLogic.
  - Crear usuarios nuevos y probad el ejemplo de JAAS relativa a un cierto usuario creado nuevo.

JAAS en Weblogic

- Package: examples.security.jaas
- Ejemplo: `SampleClient.java`
  - Objetivo:
    - Implementar una autentificación login-password y si ésta tiene éxito llama a un EJB.
  1. Instanciar un LoginContext:
     - Con una implementación de CallbackHandler (ver código de SampleCallbackHandler.java)
  2. Intentar autentificación: LoginContext.login()
     - Se define en una implementación de la interfaz LoginModule.
     - El fichero de configuración es sample_jaas.config
     - Autenticación: weblogic.security.auth.Authenticate.authenticate

JAAS en WebLogic

- Ejemplo: `SampleClient.java`

3. Obtener datos del Subject autenticado y proceder:
   - La acción a realizar es SampleAction.java.
   - Implementa la interfaz PrivilegedAction.
   - Consiste en un método run() que hace una llamada a un EJBTrader.
   - Este método se ejecuta al llamar desde el cliente al método weblogic.security.Security.runAs().

Configuración:

- Copiar sample_jaas.config en el jre/lib/security del java de bea.
- Añadir en el jre/lib/security/java.security:

```
login.config.url.1=file:${java.home}/lib/security/sample_jaas.config
```

JAAS en WebLogic

- Ejemplo: `SampleClient.java`

  Lanzar el ejemplo:

  - Ejecutar ant run.
  - Resultado: usuario,password y resultado del EJB

  Explicación:

  - Estamos entrando con login=weblogic y password=weblogic.
  - Si modificamos build.xml en el target=run y cambiamos estos datos no podremos acceder ya que no se ha creado ningún usuario en el sistema.
  - Para crear usuarios: En el applet de la izquierda:

    Security->Realms->myRealm->Users

SSL en WebLogic

- Ejemplo: `SSLClient.java`
  - Objetivo:
    - Crear un cliente que establezca una conexión HTTP y otra HTTPS con un JSP servido por el servidor de ejemplos.
    - Usando la implementación JSSE de WebLogic en lugar de la implementación de Sun:

```
try { wlsUrl =  new URL("http",
host,Integer.valueOf(port).intValue(), query);
weblogic.net.http.HttpURLConnection
    connection = new
weblogic.net.http.HttpURLConnection(wlsUrl);
tryConnection(connection, out); }
catch ...
```

SSL en WebLogic

- Ejemplo: `SSLClient.java`
  - Lanzamos el ejemplo:
    - Package examples.security.sslclient
    - Construir ejemplo: ant run.sslclient
      - Copiar el certificado digital del servidor CertGenCA.der y su clave privada CertGenCAKey.der
      - Lo mismo certificado y clave del cliente: clientcer.der y clientkey.der. Tener en cuenta mykeystore.
    - Al cliente no se le pide autentificación (one-way).
  - Salida:
    1. Package que implementa el protocolo SSL
    2. Proveedores de seguridad definidos en java.security
    3. Resultados de la conexión HTTP (puerto 7001) y HTTPS(7001)

SSL en WebLogic

- Ejemplo: `SSLSocketClient.java`
  - Objetivo:
    - Mostrar el uso de sockets SSL para realizar conexiones seguras

```
SSLContext sslCtx = SSLContext.getInstance("https");
KeyStore ks = KeyStore.getInstance("jks");
ks.load(new FileInputStream("mykeystore"), null);
PrivateKey key = (PrivateKey)ks.getKey("mykey",
"testkey".toCharArray());
Certificate [] certChain = ks.getCertificateChain("mykey");
sslCtx.loadLocalIdentity(certChain, key);
TrustManagerJSSE tManager = new NulledTrustManager();
sslCtx.setTrustManagerJSSE(tManager);
```

SSL en WebLogic

- Ejemplo: `SSLSocketClient.java`

2. `SSLSocketFactory` y `SSLSocket`:

```
SSLSocketFactory sslSF = (SSLSocketFactory)
sslCtx.getSocketFactoryJSSE();

System.out.println(" Creating and opening new
SSLSocket with SSLSocketFactory");

// using createSocket(String hostname, int port)

SSLSocket sslSock = (SSLSocket)
sslSF.createSocket(argv[0],
      newInteger(argv[1]).intValue());
```

3. Enviar la petición.

```
String req = "GET /examplesWebApp/ShowDate.jsp
HTTP/1.0\r\n\r\n";out.write(req.getBytes());
```

SSL en WebLogic

- Ejemplo: `SSLSocketClient.java`
  - Lanzamos el ejemplo:
    - Construir ejemplo: ant run.sslsocketclient
  - Salida:
    1. La primera parte de la respuesta tiene que ver con el proceso de creación del socket, incluyendo los datos de certificado del cliente
    2. La segunda parte tiene que ver con la salida HTML generada por el JSP invocado.
- Ejemplo: `SSLClientServlet.java`
  - Servlet que envuelve a SSLClient (salida similar)

http://localhost:7001/examplesWebApp/SSLClientServlet