Tema 5, módulo 7: *JAAS* + *Kerberos* + *X.500*

Programación Segura
grado en ingeniería informática

Curso académico 2020-2021

Objetivos del módulo

- Presentar *JAAS* como servicio para conectar las aplicaciones *Java* con infraestructuras de autenticación.
- Realizar pruebas de autenticación y autorización empleando *Kerberos* y *keystores*.

Índice

1. Generalidades sobre JAAS

Autenticación

- Engloba cualquier proceso dirigido a acreditar la identidad de un sujeto con objeto de evitar suplantaciones.
  - Es necesario verificar la identidad de un sujeto para autorizar su acceso a recursos protegidos.
- Debe incluir algún mecanismo que obligue al sujeto a dar una evidencia que demuestre su identidad.
- Esa evidencia puede ser de diferente naturaleza:
  - Información que sólo el sujeto conoce (contraseña).
  - Información que sólo el sujeto puede producir (firma digital, huella dactilar, fondo de retina).
  - Información que sólo el sujeto puede poseer (tique *Kerberos*).

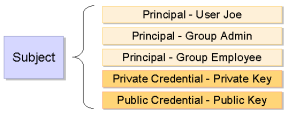
Facetas de identidad y credenciales

- Sujeto: entidad (persona/máquina/servicio) que solicita un recurso cuyo empleo requiere autorización.
  - Un sujeto puede presentar diferentes facetas de identidad por medio de diferentes principales.
- Principal: forma de identidad asociada a un sujeto.
  - Cada principal proporciona una identidad completa (diferencia a ese sujeto de cualquier otro).
  - **Ejemplo:** principales de un cierto sujeto pueden ser su identidad en el reino *Kerberos* de su entorno de trabajo y su identidad como titular de un certificado de clave pública.
- Credenciales: atributos de seguridad propios de un principal.
  - Información que permite acreditar al principal (contraseña, certificado de clave pública, tique *Kerberos*).
  - Información que permite realizar ciertas actividades al principal (firmar o cifrar datos mediante claves criptográficas).

Clase *javax.security.auth.Subject*

- Es un contenedor para mantener información sobre la identidad de una entidad (usuario/servicio) autenticada.
  - Documentación en línea:

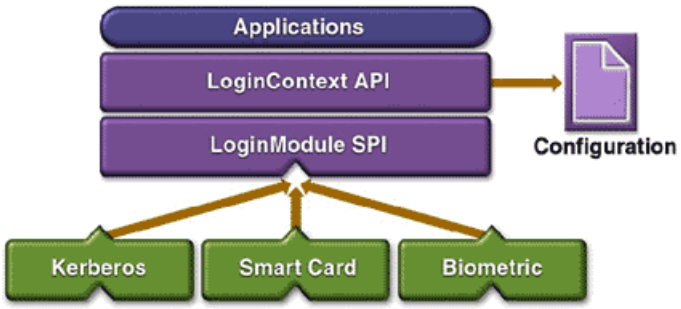
    https://docs.oracle.com/javase/8/docs/api/javax/security/auth/Subject.html
- Mantiene una relación de principales.



- Mantiene una relación de credenciales.
  - Credenciales compartidas (como certificados de clave pública) y credenciales privadas (claves criptográficas privadas) se mantienen por separado.
  - Son necesarios permisos especiales para acceder a las credenciales privadas.

JAAS (*Java Authentication and Authorization Service*)

- Es la implementación del mecanismo *PAM* (*Pluggable Authentication Module*) en forma de servicio *Java*.



- Sus componentes permiten una interacción totalmente desacoplada entre la aplicación y las infraestructuras concretas de autenticación sobre las que repose.

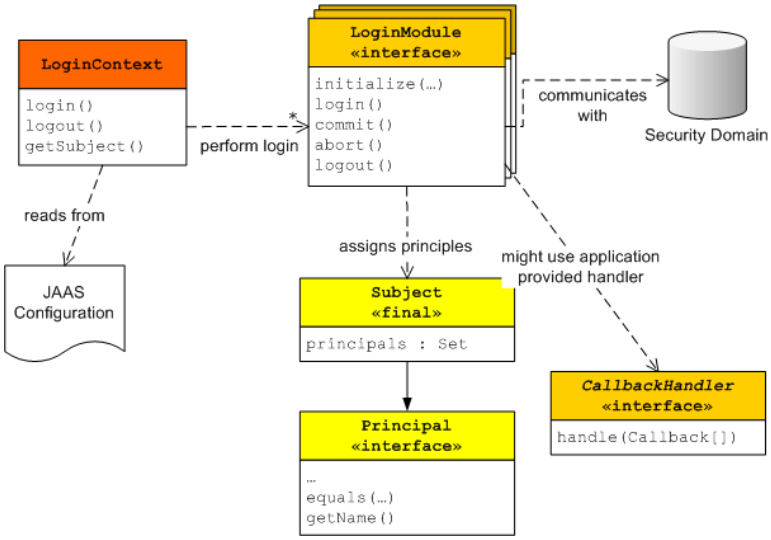
Funcionalidades ofrecidas por *JAAS*

- Autenticación de sujetos: con objeto de determinar de forma fiable y segura quién está ejecutando cada componente de código *Java*.
- Autorización de acciones de sujetos: asegurando que tienen los permisos requeridos para llevar a término las acciones iniciadas.

2. Autenticación JAAS

2.1. Sesión autenticada

Componentes que intervienen en autenticación



Componentes que intervienen en autenticación

- *LoginContext*: intermediario con el que se consigue que el proceso de autenticación realizado por la aplicación quede desacoplado de la tecnología concreta de autenticación aplicada.
- Configuración: información externa a la aplicación con la que se especifica la tecnología de autenticación a emplear.
- *LoginModule*: interfaz que conecta la aplicación con la tecnología de autenticación sobre la que se base la identificación de sujetos.
  - Autenticación basada en *username/password*
  - Autenticación basada en sistema de clave pública
  - Autenticación basada en tarjetas inteligentes
  - Autenticación basada en dispositivos biométricos

Ejemplo de sesión autenticada

```
// Let the LoginContext instantiate a new Subject
final LoginContext loginContext =
        new LoginContext("Login1a", new TextCallBackHandler());
try {

  // Authenticate the Subject
  loginContext.login();
  System.out.println("Authentication successful");

  // Get the authenticated Subject
  final Subject subject = loginContext.getSubject();
  .
  .
  .

  // All finished - logout
  loginContext.logout();

} catch (final LoginException ex) {
  System.err.println("Authentication unsuccessful");
  return;
}
```

Pasos de una sesión autenticada

1. Para identificar a un sujeto se instancia un objeto de la clase *javax.security.auth.login.LoginContext*.
   - Documentación en línea:

     https://docs.oracle.com/javase/8/docs/api/javax/security/auth/login/LoginContext.html

2. Durante la construcción del objeto *LoginContext* se consulta la configuración de *login* de la aplicación.
   - Se van cargando todos los modulos que corresponden a las diferentes formas de identificación previstas en el fichero de configuración.
3. Una vez construido el contexto de *login*, la aplicación puede iniciar el proceso de identificación invocando el método `login()` del objeto *LoginContext*.
4. El método `login()` considera cada uno de los objetos *LoginModule* cargados por el objeto *LoginContext*.
   - Cada módulo intenta autenticar al sujeto.
5. El objeto *LoginContext* retorna el estado de la autenticación a la aplicación.
   - Si la autenticación resulta fallida, lanza una excepción de clase *LoginException*.
   - Si la autenticación resulta positiva, cada módulo con identificación positiva añade un objeto *Principal* y sus credenciales asociadas al objeto *Subject* que representa al sujeto identificado.
6. Cuando la autenticación ha sido positiva, la aplicación puede recuperar al sujeto autenticado mediante el método `getSubject()` aplicado al objeto *LoginContext*.
7. Para dar por finalizada la sesión de un sujeto y eliminar las credenciales de sus diferentes principales se invoca el método `logout()`.

Interfaz *javax.security.auth.callback.CallbackHandler*

- Un módulo *LoginModule* sigue la interfaz *CallbackHandler* siempre que deba interactuar con un sujeto para obtener información sobre la que basar la autenticación.
  - Documentación en línea:

    https://docs.oracle.com/javase/8/docs/api/javax/security/auth/callback/CallbackHandler.html

- Desde el contexto de *login* de la aplicación se deben pasar objetos que implementen esa interfaz.
- Esos objetos son directamente trasladados al módulo *LoginModule* subyacente.
- Con esto se consigue que los módulos se mantengan independientes de la forma concreta de interacción con el sujeto que se haya decidido emplear.
- **Ejemplo:**

```
final LoginContext lc = new LoginContext("Login1a",
                                         new TextCallBackHandler());
```

2.2. Configuración para autenticación

Configuración de *login*

- El proceso de autenticación mediante *JAAS* está diseñado para que la aplicación se mantenga independiente de la tecnología de autenticación aplicada.
  - La aplicación percibe el proceso de autenticación como una operación en bloque completada mediante una llamada al método `login()`.
- El administrador de la aplicación emplea un fichero de configuración para establecer los módulos que conectan con las tecnologías concretas de autenticación a aplicar en cada contexto de *login*.
- Ese fichero se selecciona al arrancar la aplicación con la opción de *VM*:

```
-Djava.security.auth.login.config=<fichero de configuración>
```

Ejemplo de fichero de configuración de *login*

```
Login1 {
  sample.SampleLoginModule requisite;
};

Login2 {
  com.foo.SmartCard required;
  sample.SampleLoginModule required;
  com.sun.security.auth.module.NTLoginModule sufficient;
  com.sun.security.auth.module.Krb5LoginModule optional;
};
```

- La descripción de los diferentes significados del *flag* de control se pueden consultar en el apéndice B del enlace:

  https://docs.oracle.com/javase/8/docs/technotes/guides/security/jaas/JAASRefGuide.html

Ejemplo: configuraciones de ejecución para *JaasAcn.java*

- Sin *SecurityManager*: ejecutar con las opciones de *VM*

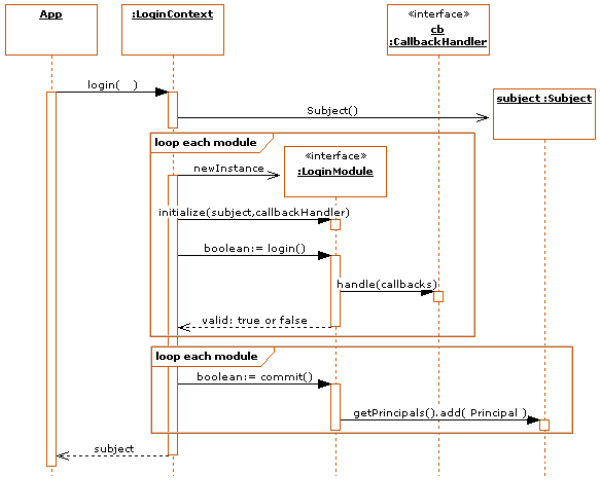
```
-Djava.security.auth.login.config=./etc/login.conf
-Djava.security.krb5.realm=LABOPROGSEGURA.UNAVARRA.ES
-Djava.security.krb5.kdc=<IP servidor kdc>
```

- Con *SecurityManager*: ejecutar con las opciones de *VM*

```
-Djava.security.manager
-Djava.security.policy=./etc/jaasacn.policy
-Djava.security.auth.login.config=./etc/login.conf
-Djava.security.krb5.realm=LABOPROGSEGURA.UNAVARRA.ES
-Djava.security.krb5.kdc=<IP servidor kdc>
```

2.3. Proceso interno

Secuencia del método `login()`



- Observése que el principal obtenido por cada módulo de *login* es finalmente incorporado al sujeto identificado.

Proceso en dos fases

Primera fase: autenticación por módulos

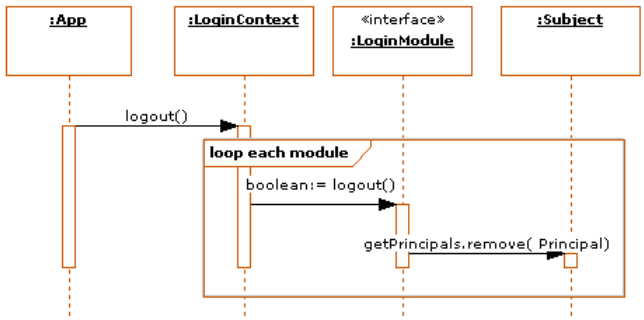
- Método `login()` del contexto *LoginContext* invoca sucesivamente el método `login()` de cada módulo especificado en la configuración de *login*.
- Método `login()` de cada módulo *LoginModule* realiza la autenticación real[1] y salva el resultado de autenticación como información privada de estado.
- Esa información de estado sirve para aplicar la lógica definida en el fichero de configuración (*requisite*, *required*, *sufficient* y *optional*) y finalizar esta fase en positivo o negativo.

[1] Por ejemplo, en el caso de un módulo *Kerberos*, pidiendo nombre de usuario y contraseña, y verificando ésta.

Segunda fase: poblado de principales

- Si la primera fase finalizó en positivo:
  - Se invoca sucesivamente el método *commit()* de cada módulo.
  - Ese método chequea el estado privado para conocer cómo resultó la autenticación del propio módulo.
  - Si su propia autenticación tuvo éxito, añade el objeto *Principal* correspondiente y sus credenciales asociadas al sujeto que resulta de la autenticación.
- Si la primera fase finalizó en negativo:
  - Se invoca sucesivamente el método *abort()* de cada módulo.
  - Ese método elimina cualquier estado de autenticación previamente salvado.
  - Finalmente se informa al contexto y el método `login()` del contexto lanza una *LoginException*.

Secuencia del método *logout()*



- Se eliminan los diferentes principales y sus credenciales asociadas que componen la identidad del sujeto autenticado por medio del objeto *LoginContext* sobre el que se aplica el método *logout()*.
  - Se invoca el método *logout()* de cada módulo.
  - Ese método elimina principal y credenciales aportadas al sujeto junto con información sobre el estado de sesión.

3. Autorización JAAS

Arquitectura de seguridad de *Java*

- Basada en la definición de políticas que explícitamente especifican los permisos que se conceden al código en ejecución.
- **Ejemplo:** cláusula que concede permiso para crear un contexto de *login* concreto al código embalado en el fichero *acn.jar* sólo si viene firmado por *ana*.

```
grant codeBase "file:${user.dir}/acn.jar", signedBy "ana" {
  permission javax.security.auth.AuthPermission
             "createLoginContext.Login1";
};
```

Permisos para principales

- Como resultado del proceso de autentificación se crea un objeto *Subject* que representa al sujeto identificado.
  - Cada sujeto está constituido por una serie de formas de identidad denominadas principales.
- En una política de seguridad es posible introducir cláusulas que se refieran a uno o más principales.
- Al ejecutar el código protegido por una cláusula, cada sujeto representado por alguno de esos principales tiene asignados los permisos especificados en la cláusula.
- Eso permite asignar selectivamente a cada principal permisos de ejecución específicos para cada componente de código que se presente embalada en un fichero *jar*.
- **Ejemplo:** cláusula que asigna al sujeto cuyo principal *Kerberos* es *aldaz* permisos especiales para ejecutar el código embalado en el fichero *actions.jar*.

```
grant codeBase "file:${user.dir}/actions.jar"
      principal javax.security.auth.kerberos.KerberosPrincipal
        "aldaz@LABOPROGSEGURA.UNAVARRA.ES" {
  permission java.util.PropertyPermission "java.home", "read";
  permission java.util.PropertyPermission "user.home", "read";
  permission java.util.PropertyPermission "user.dir", "read";
  permission java.io.FilePermission "${user.dir}${/}foo.txt", "read";
};
```

Ejecución en nombre de principal

- La ejecución de código autorizado a algún principal de un sujeto autenticado se debe realizar por medio de los métodos estáticos:
  - *Subject.doAs(sujeto, accion)*
  - *Subject.doAsPrivileged(sujeto, accion, contexto)*
- Como argumentos se pasan el sujeto autenticado y una tarea encapsulada en un objeto *PrivilegedAction* o *PrivilegedExceptionAction*.
- El método `run()` de ese objeto debe concentrar el código a ser ejecutado como sujeto autenticado.
  - Código que requiere permisos especiales de ejecución.
- La ejecución del método `run()` sólo será posible si la política de permisos autoriza el acceso a los recursos protegidos a alguno de los principales que componen la identidad del sujeto.

Ejercicio

- El código del método `run()` de la clase *SampleAction* no requiere permisos especiales para su ejecución.
- El ejercicio consiste en conseguir que ese método sólo pueda ser ejecutado por un principal específico.
- Con ese fin se puede definir un permiso *ad hoc* e incluir ese permiso en la cláusula correspondiente de la política de seguridad.
- Se pide:
  - Definir el permiso *azn.actions.ADPermission*.

- Modificar la acción *SampleAction* para que el controlador de acceso asegure que el contexto disponga del permiso asignado
- Adaptar la política de seguridad para asigne el nuevo permiso definido al principal *Kerberos* creado para realizar las prácticas.
- Verificar que es el único que puede ejecutar esa acción.