Seguridad de la Información

Primer Cuatrimestre del 2012

Departamento de Computación
Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

Administrador de identidades

Integrante	LU	Correo electrónico
Edi Juan	133/08	jedi@dc.uba.ar
Festini Ruben	724/92	rfestini@dc.uba.ar
Nahabedian Leandro	250/08	leanahabedian@gmail.com

Contents

1 Requerimientos

Se describe a continuación el desarrollo de una aplicación web para ser utilizada con el fin de unificar el ABM de usuarios a distintos sistemas. La aplicación brinda credenciales de login unificada, cambio de clave, y un registro detallado de actividad accesible para auditoría. También permite modificar el criterio de complejidad y período de validez de las claves.

La comunicación con el resto de las aplicaciones puede realizarse en forma diferida. Es decir, la información llega desde el administrador al resto de las aplicaciones recién en el momento que éstas están en línea y en condiciones de operar. En caso contrario, los cambios se almacenarán en el administrador hasta que puedan ser enviados.

Era requerimiento que haya una copia master de la información guardada en una base de datos, y que cada aplicación tenga su propia copia de los datos (almacenada en el formato que se prefiera). De esta forma, nuestra aplicación haría sólo de ABM de estos datos, pero cada *aplicación cliente* estaría a cargo de la autenticación utilizando su copia. A su vez, cada aplicación debería poder utilizar un mecanismo distinto de almacenamiento de claves.

Sobre los requerimientos iniciales presentados anteriormente, se tomaron algunas decisiones adicionales, validándose en su momento.

En primer lugar, se decidió modelar un esquema de roles no jerárquico para todas las aplicaciones. Esto no significa que una aplicación no pueda implementar una jerarquía de roles por su parte, sino que esa jerarquía lógica no será mantenida en los datos por el administrador: el administrador no conocerá esta jerarquía, y los roles se asignarán aisladamente a los usuarios sin tener en cuenta su relación con otros roles.

Por otra parte, todos los cambios en los datos se realizan a través del administrador. Éste no deberá encargarse de sincronizar ningún otro tipo de dato entre las aplicaciones, y cualquier modificación en los datos fluye desde el administrador hacia las aplicaciones, nunca al revés.

Por último, cuando previamente hablamos de *login unificado* nos referimos a que se utilizarán las mismas credenciales para autenticarse a todas las aplicaciones manejadas por el administrador, pero no que se sincronizará ningún estado relacionado a las sesiones de un usuario en las aplicaciones. Tanto la autenticación como el manejo de sesión es realizada de forma completamente independiente por cada aplicación.

Junto con el administrador de identidades, se presentan dos aplicaciones *demo*, configuradas para comunicarse con el mismo. Para mostrar la independencia entre las *aplicación cliente* y el administrador, estas aplicaciones utilizan distintos *stores* para la información: una usa una base de datos distinta, mientras que la otra autentica contra un árbol *LDAP*.

2 Diseño y arquitectura

Se detalla a continuación las decisiones que se tomaron para solucionar los aspectos más importantes de diseño y la arquitectura de comunicación con el resto de las aplicaciones.

Como criterio general, se intentó siempre permitir el mayor grado de independencia posible entre el administrador y cada aplicación. Esto significa no sólo que cada aplicación pudiera escoger su propia forma de almacenar los datos, sino que también se pudiera implementar la comunicación con el administrador de identidades en una aplicación cualquiera de forma sencilla, sin imponer restricciones excesivas sobre nuevas tecnologías a integrar.

2.1 Almacenamiento de claves

Posiblemente el principal problema a resolver fue el de cómo se iban a almacenar las claves. La primera restricción surgía del requerimiento de que cada aplicación debía poder utilizar un mecanismo de autenticación propio.

Inicalmente, la solución más sencilla parecía ser que, al registrarse un usuario o cambiar su clave, se enviara la misma a través de un canal seguro en texto plano. Al recibir una nueva clave, cada aplicación cliente podría almacenarla en el formato que prefiera. De esta forma, el administrador sólo debería guardar la clave de una forma (la que se eligiera utilizar para la autenticación al mismo administrador).

Siempre y cuando se pudiese confiar en el canal de transmisión, esta alternativa parecía bastane acertada: brindaba independencia total entre el administrador y la forma de almacenar las claves de cada aplicación. Sin embargo, se tiene el problema de que al agregar una nueva aplicación al administrador no habría forma de informarle las credenciales de los usuarios previamente existentes. Dado que en esta situación no es viable forzar a todos los usuarios a cambiar sus claves, se descartó esta posibilidad.

Una modificación ingenua sobre lo anterior hubiese sido seguir enviando las claves a las otras aplicaciones en texto plano, pero esta vez guardándolas también en el administrador utilizando algún tipo de cifrado reversible. De esta forma, se podría obtener las claves de los usuarios en cualquier momento con el fin de enviarlos a otras aplicaciones. Aparte de las consideraciones adicionales de implementación que representaría, la principal desventaja de esta opción es que en caso de comprometerse la clave de cifrado se estarían conociendo inmediatamente las claves de todos los usuarios.

Finalmente se terminó optando por almacenar utilizando varios mecanismos de hash distintos. En primer lugar, esto requiere definir previamente cuáles son los algortimos soportados. También necesariamente el administrador de identidades debera conocer qué mecanismo de almacenamiento de claves utiliza cada aplicación. Si bien esto es una pérdia de independencia con respecto a la implementación de las aplicaciones, soluciona el problema anterior: siempre se cuenta con la clave en el *formato*.

Observamos de todos modos dos desventajas en la alternativa elegida:

- En cierto modo, nuestra forma de almacenar las claves será tan débil como el mínimo que se ofrezca al resto de las aplicaciones. De todos modos, si bien es cierto que se impone un techo sobre la seguridad del mecanismo de almacenamiento, consideramos razonable asumir que el sistema se implementaría en organizaciones donde se determina un nivel de seguridad mínimo para el tratamiento de datos sensibles, por lo cuál este techo del que hablábamos antes no debería ser muy bajo: no se deberían ofrecer opciones que se consideren inseguras.
- La opción inicial tenía la ventaja de que el administrador no necesitaba conocer cómo las aplicaciones almacenaban sus claves. Esto permitía que no hiciera falta hacer ningún cambio en caso de de agregarse una aplicación con un mecanismo de almacenamiento no conocido hasta el momento. Con la alternativa elegida, no es trivial agregar un nuevo algoritmo (requerirá necesariamente resetear las claves de todos los usuarios).

2.2 Comunicación con el resto de las aplicaciones

Otro tema a definir era cómo se iba a implementar la comunicación entre el administrador y las distintas aplicaciones. Por un lado se tenía el requerimiento de que una aplicación podía no encontrarse en línea al momento de efectuarse un cambio en el administrador, lo que imponía persistir en algún punto los datos a transmitir. Aparte de esto, había que definir cómo se notificaba a las aplicaciones la presencia de cambios en los datos. Cualquier decisión debía tomarse teniendo en cuenta que las aplicaciones posiblemente se encontrarían en lugares distintos, comunicándose seguramente a través de Internet.

Se consideró la posibilidad de que sean las mismas aplicaciones las encargadas de consultar periódicamente al administrador por cambios que le sean relevantes, pero se descartó rápidamente. En primer lugar, sería deseable que la misma infraestructura garantize la inmediatez de los cambios: si una aplicación no está configurada para consultar los cambios muy frecuentemente, se podría dar que pase un tiempo demasiado prolongado entre que se modifica un dato en el administrador y el mismo se ve impactado en una aplicación cliente.

Más allá de esto, la mayor desventaja de este esquema era que requería implementar dentro de nuestra aplicación el estado de los cambios ya efectuados por cada aplicación (con la información asociada a os mismo), y ofrecer una interfaz común hacia el exterior de consulta.

La alternativa obvia era que sea el administrador quien envíe a las aplicaciones los cambios apenas son efectuados. Sin embargo, una primera implementación de esta opción tampoco escapa de la necesidad de incluir dentro de nuestra aplicación el estado de *actualización* de cada aplicación, ya que en caso de que una aplicación no esté en condiciones de aplicar los cambios el dato deberá almacenarse hasta que pueda ser consumido.

Se optó finalmente por resolver la comunicación utilizando un broker de mensajes. Al agregarse una aplicación al administrador se definiría su set de colas de mensajes asoci-

adas (una por cada tipo de actualización), que pueden pensarse como casillas donde se almacenan los datos que deberá procesar. Gran parte de los brokers disponibles brindan la posibilidad de establecer una conexión persistence para comunicar a un suscriptor de una cola la presencia de nuevos mensajes, con lo que sólo debería definirse en el cliente un suscriptor para cada cola, con la lógica propia de cada aplicación para procesar los distintos tipos de mensajes.

Aparte de esto, los principales brokers brindan facilidades tanto para la configuración de permisos por usuario para leer distintas colas (con lo cuál se podrían aislar fácilmente las aplicaciones entre sí) como para la comunicación vía SSL a través de internet.

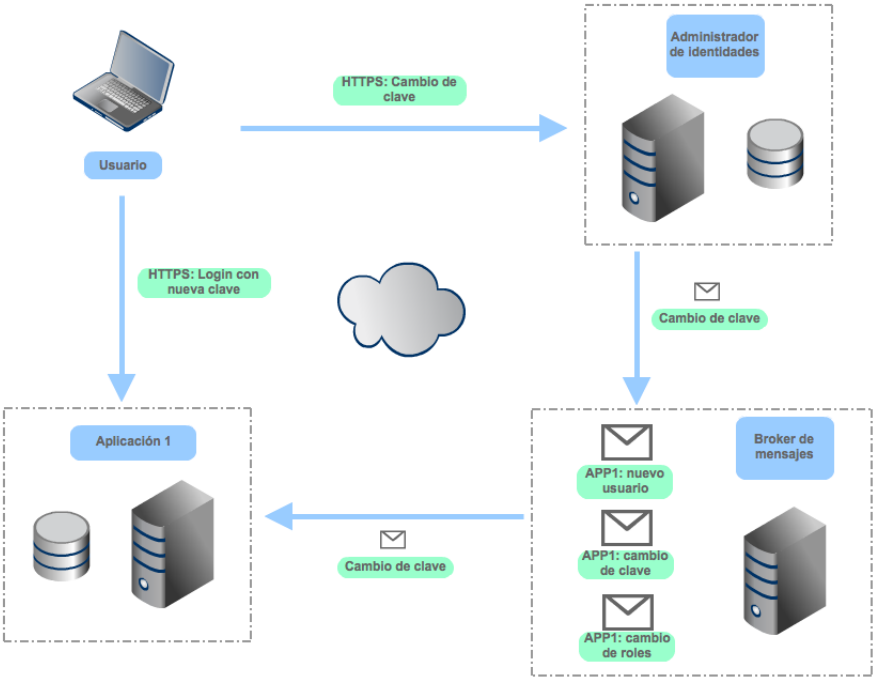

Figure 1: Arquitectura de comunicación con el resto de las aplicaciones.

3 Herramientas Utilizadas

A continuación se presentan las principales herramientas utilizadas, el criterio de elección estuvo basado en la utilidad para el proyecto, el conocimiento de los integrantes del grupo y además se privilegió a las herramientas open source.

3.1 Play! Framework

Como framework web elegimos Play! Framework, una herramienta Java (el lenguaje mayormente manejado por el grupo), que busca proveer agilidad en el desarrollo (a diferencia de la mayoría de los frameworks de esta plataforma).

Play es un framework open source escrito en Java basado en el patrón de diseño model view controller y similar a Ruby on Rails o Django. Entre sus principales características se encuentran:

- Stateless
- Poca configuración, basado más en convención que en configuración.
- Integrado a JUnit
- Se integra fácilmente con los principales IDE (Eclipse, Netbeans o IntelliJ).

Cabe destacar que Play tiene dos ramas, conocidas como Play1 y Play2. Esta última provee soporte para funciones más avanzadas (entre las que se encuentra procesamiento asincrónico de pedidos y un robusto modelo de programación paralela), pero al momento de elección todavía no estaba muy probado, y tampoco se requerían estas funcionalidades.

3.2 RabbitMQ

Al momento de la elección del broker de mensajes nos decidimos por RabbitMQ que es un broker de mensajes open source que implementa el estándard AMQP (Advanced Message Queuing Protocol), se integra perfectamente con Play y es multi plataforma. Otra de sus ventajas para nuestro proyecto es la fácil configuración de canales SSL para la transmisión de mensajes que era vital para tener una comunicación segura con las distintas aplicaciones.

Entre sus características principales se encuentran:

- Gateways para los protocolos HTTP, XMPP y STOMP.
- Servidores y Clientes en varios sistemas operativos y lenguajes.
- Fácil de usar.

3.3 Motor de base de datos

Durante el ciclo de desarrollo se utilizó como motor de base de datos MySQL, pero una de las ventajas de play es que se adapta a cualquier motor de base de datos que posea un driver JDBC, por lo tanto sims puede ser utilizado con múltiples motores.

3.4 OpenLDAP

OpenLDAP es una implementación libre y open source del protocolo LDAP (Lightweight Directory Access Protocol), elegimos esta implementación por ser considerada un estandard.

4 Descripción de la Solución

La función principal del sistema de administración de identidades es la administración centralizada de usuarios y roles para diversas aplicaciones. La información generada es enviada a las aplicaciones cliente a través del broker de mensajes.

Por cada acción realizada el sistema registra logs de auditoría que posteriormente pueden ser consultados. Para el acceso a estas funcionalidades el sistema tiene definido un esquema de roles, practicamente un rol por funcionalidad, que es detallado a continuación.

El sistema cuenta en un comienzo con un usuario administrador (admin), el mismo tiene asociado todos los roles.

Respecto a la seguridad la aplicación de administración chequea su clave utilizando como hash SHA512, pero al igual que el resto de las aplicaciones, la clave está persistida en la base en todos los hash definidos.

Al expirar una clave se notifica a todas las aplicaciones para que éstas dejen de permitir el acceso a la aplicación para el usuario. Sin embargo, se sigue permitiendo acceder al administrador para que el usuario pueda cambiar su clave.

Adicionalmente, se notifica al usuario a los 15 y 7 días de que caduque su clave, asi como también se muestra una alerta visual en el administrador durante los 15 días antes del vencimiento, con el objetivo de recordar al usuario que renueve su clave con tiempo.

4.1 Esquema de roles

En el sistema de administración de identidades están definidos los siguientes roles:

- Base: Permite a cualquier usuario cambiar su clave, si bien no es un rol persistido cualquier usuario creado tiene permiso de cambiar su clave
- Creación de usuarios
- Creación de aplicaciones: Permite el alta de la aplicación, asignándole un usuario encargado de su configuración y administración.
- Administración de aplicación: Este rol de otorga *sobre aplicaciones* al momento de crearlas, y define qué usuario tiene permiso para otorgar/revocar acceso y roles específicos a otros usuarios sobre esa aplicación. Además, el usuario con este permiso será el encargado de realizar la configuración inicial de la aplicación (credenciales para el broker de mensajes, configurar esquema de roles de la aplicación, etc.).
- Politica de Claves: Permite dar de alta o modificar políticas de claves
- Auditoria: Permite consultar los eventos registrados para la auditoría

4.2 Creación de Usuarios

Para el alta de usuarios se ingresa usuario, email, nombre y apellido. El sistema envía un mail al usuario con los datos ingresados más una clave generada aleatoriamente. El rol asociado al usuario para el sistema administrador es el base.

4.3 Alta de aplicaciones

Al dar de alta una nueva aplicación es necesario el alta de la misma en el administrador, la creación de un nuevo usuario y queues en el broker de mensajes y la configuración en las aplicaciones cliente.

4.3.1 Configuración en el administrador

En primer lugar le damos el alta ingresando nombre, usuario a cargo y el método de encriptación de clave. Esto da de alta la aplicación dentro del administrador, y realiza la configuración del broker de mensajes (creación de queues, usuario, configuración de permisos, etc). Desde este momento, cualquier mensaje que sea generado para la aplicación será enviado al broker y quedará almacenado a la espera de ser consumido.

En esta instancia el usuario asignado como responsable de la aplicación debe configurar la misma, eligiendo la clave con la que se conectará al broker de mensajes y cargando el esquema de roles de la aplicación.

En caso de que sea conveniente, se pueden setear roles por defecto, los cuáles son sugeridos automáticamente cuando se asignan nuevos usuarios a la aplicación.

4.3.2 Configuración de aplicación cliente

Cada aplicación debe ahora conectarse con el broker de mensajes, utilizando como nombre de usuario el nombre de la aplicación tal como se cargó en el administrador, y como contraseña la definida al momento de la configuración inicial.

El broker debería estar configurado para recibir conexiones seguras en el puerto estándar 5671. Depende de la implemetación de cada cliente de RabbitMQ cómo se definen los certificados en que se confían. Para aplicaciones Java, se provee junto con la distribución de la aplicación una biblioteca para facilitar la conexión con el administrador. En el caso de usar la misma, sólo hace falta especificar en un archivo de propiedades la ruta a un *trustore jks* que tenga la información del certificado del servidor. Para más detalles ver la configuración de las aplicaciones demo provistas.

Por convención, los nombres de las queues son del tipo: `NOMBRE_APLICACION/TIPO_MENSAJE`. Los distintos tipos de mensaje se pueden ver en la especificación del protocolo de mensajes, en las siguientes secciones. Todas las conexiones deben realizarse al virtual host *sims*.

4.4 Política de Claves

La política de claves nos permite definir

- longitud de la clave
- tipo de caracteres de los cuales debe tener al menos una ocurrencia
  - minúsculas (a-z)
  - mayúsculas (A-Z)
  - números (0-9)
  - caracteres especiales (@#$%)
- duración (en cantidad de días)
- cantidad de claves diferentes sin repetición

Esta política es validada en la creación de usuarios y en los cambios de clave.

4.5 Auditoría

Hay determinados eventos que registran log para la posterior auditoría, en un principio el sistema registra

- Cambio de clave
- Clave expirada
- Creación de un usuario
- Creación de una aplicación
- Asignación o revocación de acceso de un usuario a una aplicación
- Asignación o revocación de un rol a un usuario
- Nueva política de complejidad de clave
- Cambio de la política de complejidad de claves

Para la consulta de estos eventos es necesario ser un usuario con el rol auditor. Se puede buscar por tipo de evento, rango de fechas, y aplicaciones y usuarios relacionados con el evento.

5 Protocolo de comunicación con aplicaciones

La comunicación con las aplicaciones cliente se realiza a través del broker de mensajes. El sistema envía mensajes por cada evento relacionado a una aplicación. La aplicación cliente, que escucha por medio de un listener, debe interpretar el mensaje e impactar los cambios como corresponda. Los mensajes se serializan es strings encodeado en UTF-8 formado por una lista de campos separados por comas. A su vez, hay campos que son listados entre corchetes de elementos separados por comas. Las fechas se escriben en formato ISO (yyyy-MM-dd). Se detalla a continuación la composición de los distintos tipos de mensajes enviados a las aplicaciones, y el nombre de las queues en donde se envían:

5.1 Nuevo Usuario

Queue asociada `NOMBRE_APP/NEW_USER`

Descripción del evento se otorgó acceso a un usuario a la aplicación

Campos

username nombre de usuario

firstName primer nombre

lastName apellido

hashType tipo de hash utilizado para la clave (a modo de control, es el mismo de la aplicación)

password clave hasheada

passwordExpiration fecha de expiración de la clave

roles listado de roles del usuario para la aplicación

Ejemplo `mtinelli,Marcelo,Tinelli,SHA1,0DPiKuNIrrVmD8IUCuw1hQxNqZc=,2012-11-01,[ROL1]`

5.2 Cambio de Clave

Queue asociada `NOMBRE_APP/CHANGE_PASSWORD`

Descripción del evento cambió la clave de un usuario

Campos

username nombre de usuario

hashType tipo de hash utilizado para la clave (a modo de control, es el mismo de la aplicación)

password nueva clave hasheada

passwordExpiration fecha de expiración de la nueva clave

Ejemplo `dmaradona,SHA256,jGl25bVBBBW96Qi9Te4V37Fnqchz/Eu4qB9vKrRIqRg=,2013-02-01`

5.3 Vencimiento de clave

Queue asociada `NOMBRE_APP/PASSWORD_EXPIRED`

Descripción del evento expiró la clave de un usuario

Campos

username nombre de usuario

hashType tipo de hash utilizado para la clave (a modo de control, es el mismo de la aplicación)

password hash de la clave expirada (a modo de control, la aplicación ya lo posee)

Ejemplo `dmaradona,SHA256,jGl25bVBBBW96Qi9Te4V37Fnqchz/Eu4qB9vKrRIqRg=`

5.4 Remoción de usuario

Queue asociada `NOMBRE_APP/DELETE_USER`

Descripción del evento se revocó acceso a un usuario para la aplicación

Campos

username nombre de usuario

Ejemplo `mtinelli`

5.5 Cambio de roles

Queue asociada `NOMBRE_APP/CHANGE_ROLES`

Descripción del evento se modificaron los roles de un usuario en la aplicación

Campos

roles listado completo de roles actualizados. incluye los que mantiene y los nuevos.

Ejemplo `dmaradona,[ROL1,ROL3]`

6 Conclusión

Respecto a la elección de la tecnología creemos que fueron adecuadas las decisiones tomadas, el uso del framework play nos dio velocidad de desarrollo y nos permitió preocuparnos simplemente de la lógica del sistema, asistiéndonos en lograr una aplicación portable a distintos sistemas operativos y a distintos motores de base de datos.

Creemos que el broker de mensajes también fue una buena elección: pudimos resolver fácilmente el requerimiento de poder enviar mensajes a las aplicaciones sin necesidad de que estén disponibles. Se pudo implementar una comunicación segura entre las aplicaciones gracias al soporte SSL que brinda RabbitMQ y además queda abierta la posibilidad de integración con clientes desarrollados con otros lenguajes gracias a su gran integración.

Uno de los temas que más nos preocupó al momento del análisis fue la necesidad, dado los requerimientos, de guardar las claves sin importar el método de almacenamiento de cada aplicación. Eso nos llevo a tomar la decisión de diseño de guardar las claves de múltiples maneras. En un principio lo vimos como una debilidad, dado que la seguridad de la solución está dada por la peor complejidad de almacenamiento. De todos modos, entendemos que al ser una solución pensada para centralizar los usuarios que acceden a sistemas de una organización, se supone que la misma tiene debería tener políticas que no permitan el guardado débil de las claves.

Igualmente nos parece una mejor solución brindar la autenticación desde el sistema central en lugar de replicar los datos del usuario a cada aplicación. Una de las desventajas que vemos en la solución actual es que se aumentan los puntos de ataque dado que si un atacante logra vulnerar alguna de las aplicaciones podría obtener acceso al resto (incluso el administrador). Esto implicaría, de todas formas, un cambio importante el la lógica de autenticación de las aplicaciones.

En resumen, dentro de las oportunidades de mejora que observamos podemos destacar:

- Agregar la posibilidad de modificar y agregar roles una vez que la aplicación fue configurada.
- Desarrollar un artefacto *stand-alone* para desplegar fuera de las aplicaciones cliente, que se encargue de recibir los mensajes del broker, para minimizar los cambios sobre cada aplicación.
- Autenticación centralizada a través del administrador. Podría brindarse una biblioteca para facilitar la implementación de un protocolo de autenticación a través de la aplicación central.