ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA INFORMÁTICA

INGENIERÍA INFORMÁTICA

SSH SOBRE FEDERACIÓN DE IDENTIDAD

Realizado por
Daniel García Moreno

Dirigido por
Manuel Valencia y Diego R. López

Departamento
Tecnología Electrónica

Sevilla, Junio de 2008

Resumen

Cada vez está tomando más importancia la web y actualmente están apareciendo multitud de aplicaciones, ya sea por parte de empresas o de instituciones como universidades. Estas aplicaciones normalmente requieren autenticación y la mayoría de ellas basan dicha autenticación en bases de datos locales.

Con el crecimiento de las aplicaciones crece el número de usuarios de estas, y por parte del usuario, crece el número de cuentas creadas para diferentes aplicaciones. Para paliar este problema nace el Single Sing On(SSO), que proporciona una única cuenta y un único punto de autenticación para diferentes aplicaciones web de una misma entidad.

El siguiente paso natural es el uso de aplicaciones de otras entidades, y aquí es donde radica la importancia de la **federación de identidad**.

La federación de identidad consiste en que una serie de entidades **confían** en otras para la autenticación de los usuarios. Es decir, que una aplicación de una entidad acepta usuarios de otra entidad. Además estos usuarios se autenticarán en su entidad, por lo que la gestión de usuarios, contraseñas, y atributos queda delegada a cada entidad. Por tanto el usuario final tiene acceso a todas las aplicaciones federadas de todas las entidades que conforman la federación de identidad.

Esto facilita enormemente la gestión de usuarios por parte de las entidades, puesto que tan solo tienen que gestionar sus propios usuarios y prestan servicio a usuarios de otras entidades gracias a que la autenticación es delegada.

La idea principal de este proyecto es llevar las facilidades que proporciona la federación de identidad fuera del ámbito de la Web, más concretamente al ámbito del **SSH** (Secure SHell).

Para el caso del SSH, si un usuario tiene acceso a diferentes máquinas tendrá diferentes cuentas y diferentes contraseñas que recordar, almacenar

y gestionar con la problemática que eso conlleva. Además sus contraseñas estarán en máquinas que no tiene por qué controlar, por lo que si se compromete alguna de estas máquinas, estará comprometida su contraseña.

Utilizando SSH sobre la federación de identidad se pueden eliminar estos problemas e incrementar la comodidad, tanto por parte de los usuarios como por parte de los administradores. Para poder acceder por SSH un usuario tendría que autenticarse en la federación y, una vez autenticado, podrá entrar en todas las máquinas que ofrezcan el servicio de SSH federado sin necesidad de poner contraseña, basándose en el mecanismo de clave pública y clave privada y estando el servidor en cualquier entidad de la federación.

Así pues, por parte del usuario se tiene acceso a diferentes máquinas necesitando recordar y gestionar una sola contraseña. Además esta contraseña nunca se entrega a un servidor extraño, sólo a la entidad de la cual procedes (a través de una páginas web segura) y en la cual confías puesto que es la encargada de gestionar tu identidad.

Y por parte del administrador se puede delegar la gestión de usuarios confiando en la federación. Automatizando la creación y destrucción de cuentas y sin necesidad de proporcionar ninguna contraseña.

Índice general

Capítulo 1

Introducción

1.1. Objetivos

Las facilidades que ofrece la federación de identidad son más que evidentes y podría ser interesante en muchos casos tener estas facilidades para otros servicios que requieran autenticación.

El principal objetivo de este proyecto es llevar las facilidades de la gestión de identidad del ámbito de la Web a otros servicios, como por ejemplo el SSH. Este proyecto se centra en integrar la federación de identidad con el acceso por SSH y puede servir como prueba de concepto a la hora de llevar la autenticación por federación de identidad a servicios diferentes de la web.

Las características más importantes de la federación de identidad que serán útiles en el SSH son:

1. **Acceso a recursos de otras entidades**: La base de la federación de identidad es poder acceder a recursos de otra entidad con la misma cuenta con la que accedes a los recursos o servicios de tu propia entidad.
2. **Gestión de identidad distribuida**: Al encargarse cada entidad de la federación de sus propios usuarios y basándonos en las relaciones de confianza de la federación, se puede dar servicio a un mayor número de usuarios gestionando tan solo una pequeña cantidad de ellos. Esto

puede crear algún tipo de duda puesto que se pierde el control sobre los usuarios, pero no hay que olvidar que la federación es una red de confianza donde cada entidad debe confiar en las demás y para ello hay mecanismos seguros, como por ejemplo los certificados.

3. **Unicidad de contraseña**: La federación de identidad brinda la posibilidad de acceder a diferentes servicios, que requieren autenticación, con la misma cuenta y la misma contraseña y sin necesidad de replicar esta en los diferentes servicios sino estando en la propia entidad del usuario, incrementando así la seguridad de la misma y la comodidad a la hora de cambiar de contraseña, o de nombre de usuario. Esto tiene un inconveniente, y es que si las contraseñas son vulnerables hay un mayor número de recursos a disposición de un posible intruso. Por lo tanto es recomendable tener contraseñas más fuertes cuando se trabaja con contraseñas únicas.

4. **Login único**: También se busca implementar el Single Sing On(SSO) para el SSH sobre federación de tal forma que un usuario sólo tenga que autenticarse una vez, y a partir de ahí, tener acceso sin necesidad de introducir ningún tipo de contraseña a todos los servidores SSH disponibles.

Por otra parte se ha elegido llevar la federación al servicio SSH porque es ampliamente utilizado, además de ofrecer una gran potencia y versatilidad, abriendo así la puerta a la utilización de otros servicios de forma fácil.

Por ejemplo, en el ambiente académico, puede ser interesante dar acceso a un servidor SSH a todos los alumnos de Informática, bien sea para que utilicen un supercomputador o para que tengan una cuenta donde hacer las practicas. Dentro del objetivo de este proyecto entraría delegar la gestión de estos usuarios a la federación, facilitando así el proceso, tanto por parte del alumno como por parte del administrador de las máquinas.

1.2. Caso de uso

En el siguiente caso de uso se muestra el funcionamiento básico del sistema así como una serie de detalles que serán explicados en la sección "Implementación y despliegue" [4].

- **Descripción**:

  La federación está pensada para aplicaciones web pero sería interesante poder utilizar estos mecanismos para aplicaciones que autentican de otra manera diferente.

  En el caso del SSH federado se intenta llevar el concepto de hacer login una sola vez, y en la entidad de origen del usuario, al acceso por SSH. Buscando poder acceder por SSH a diferentes máquinas sin tener que escribir usuario y contraseña, una vez se haya autenticado el usuario. A través de SSH se pueden hacer muchas más cosas, como por ejemplo túneles ssh, port-forwarding, etc.

- **Proceso de Autenticación**:

  1. Se accede a una página especifica, protegida tras un SP.
  2. El usuario se autentica en la federación, y puede ver la página.
  3. Esta aplicación web intentará conseguir la clave RSA publica del usuario a través de los datos que manda la federación.
  4. Una vez autenticado en esa aplicación web, el usuario puede acceder a las cuentas ssh federadas de las que disponga sin tener que introducir su contraseña.

  Se puede ver un esquema del funcionamiento de este proceso en la figura 1.1

- **Proceso de Autenticación alternativo**: La clave publica que recibe la aplicación a través de la federación será la de la máquina habitual del usuario. En caso de estar utilizando otra máquina es posible utilizar otra clave temporalmente.

  1. Se accede a una página especifica, protegida tras un SP.

2. El usuario se autentica en la federación, y puede ver la página.
3. En la aplicación web se introduce la clave publica RSA temporal, para esta sesión.
4. Una vez autenticado en esa aplicación web, el usuario puede acceder a las cuentas SSH federadas de las que disponga sin tener que introducir su contraseña.

- **Requisitos**:

  Para poder acceder a cualquier máquina remota por SSH en el servidor se debe poner el servidor SSH parcheado. Además se debe crear una cuenta de usuario. Es recomendable el deshabilitar la posibilidad de cambiar la contraseña, puesto que si se puede cambiar es posible acceder a esta sin pasar por la federación. También es conveniente no permitir la creación, o el borrado de los ficheros dentro del directorio .ssh del usuario, por la misma razón que lo anterior.

  También será necesario definir un esquema para la federación, que añada el atributo ssh_rsa_public_key, si queremos que el acceso sea lo más automatizado posible.

Para una mayor compresión de la figura 1.1 vamos a explicar cada paso de manera un poco más exhaustiva:

- Paso 1: Antes de poder acceder a cualquier servicio de SSH federado el usuario tendrá que autenticarse. Para ello se utilizan los mecanismos que ofrece la federación de identidad. Por tanto el usuario intentará acceder a una aplicación web protegida tras un Service Provider (SP) de la federación de identidad.
- Paso 2: Según el funcionamiento de la federación si el usuario no está aún autenticado en la federación de identidad el SP redireccionará al usuario hacia una aplicación WAYF (where are you from) dónde seleccionará su entidad de origen, y tras esto será redirigido al Proveedor de Identidad (IdP) de su entidad. En esta aplicación web será donde el usuario proporcione sus credenciales, siendo esta una operación segura puesto que el IdP está gestionado por nuestra propia entidad, y por lo tanto, no estamos proporcionando nuestra contraseña a ninguna otra entidad desconocida.

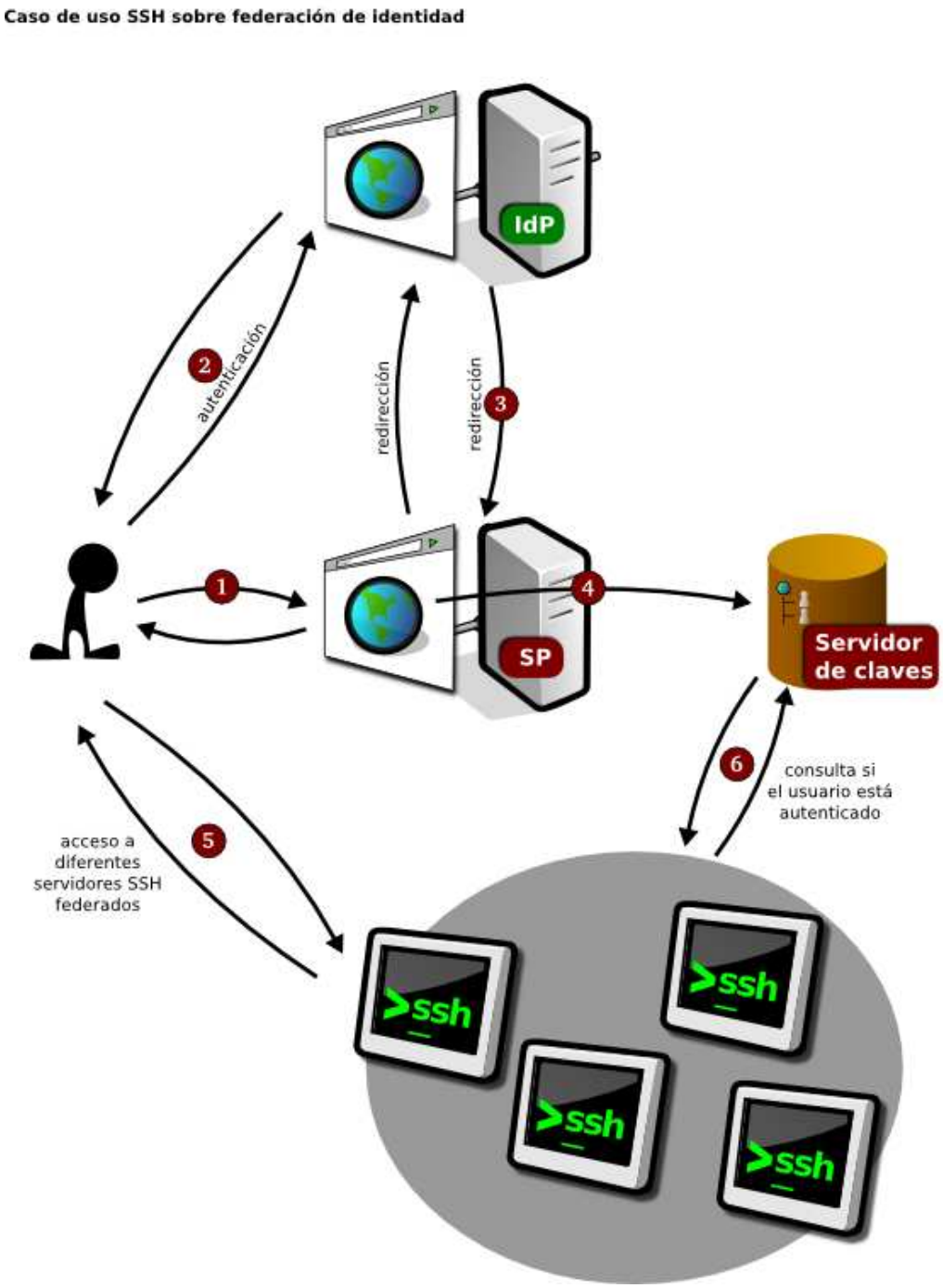


Figura 1.1: Caso de Uso

- Paso 3: Una vez autenticado en la federación de identidad el sistema redirigirá al usuario al SP al cuál quería entrar en un principio, pasándole a este los datos necesarios para asegurar la identidad del usuario. Aunque este proceso pueda parecer largo, para el usuario son un simple par de clicks puesto que todo el trabajo se realiza automáticamente por la federación.

- Paso 4: La aplicación principal, tras el SP, recibirá entonces los datos necesarios del usuario que le serán proporcionados por el IdP de la entidad de este. Y con estos datos creará una entrada en el servidor de claves, el cuál será consultado por los servidores SSH para verificar que el usuario esté autenticado en la federación. En este paso es donde se comienza a sacar la federación de identidad del ámbito Web teniendo un lugar dónde se almacenarían los usuario autenticados con un sistema diferente al de "cookies".

- Paso 5: El usuario ya está autenticado y ahora puede entrar en uno o varios servidores SSH federados sin necesidad de escribir su contraseña. Por supuesto tendrá un tiempo límite, la sesión expirará dado un cierto tiempo.

- Paso 6: El servidor SSH tiene que saber si el usuario que intenta acceder está autenticado en la federación por lo que consultará el servidor de claves para confirmar que dicho usuario está autenticado y además es quien dice ser.

1.3. Antecedentes

La federación de identidad es un concepto relativamente nuevo y por tanto hoy en día hay pocas federaciones funcionando.

La federación de identidad de universidades andaluzas está naciendo ahora mismo, por lo que se está nutriendo de la experiencia de otras federaciones con algo más de tiempo.

Una de las federaciones más activas en lo que se refiere a investigación y desarrollo de aplicaciones para la federación de identidad es la federación noruega, **feide** ([Federación Noruega]).

El proyecto de SSH sobre federación de identidad nace a partir de un documento publicado el 20 de agosto de 2007 por la federación noruega.

http://rnd.feide.no/content/feide-and-ssh-secure-shell

Este documento investiga cómo las credenciales de la federación de identidad pueden ser usadas para autenticar a diferentes servicios, como el SSH.

A partir de esta idea se proponen diferentes formas de conseguir la autenticación de SSH sobre la federación de identidad. Este proyecto se basa en las ideas propuestas por la federación noruega, dando un paso más e intentando automatizar al máximo el proceso, para dar una mayor comodidad al usuario y también al administrador.

Todos los métodos propuestos en este documento se basan en la utilización de un navegador web para obtener las credenciales de autenticación del servicio SSH. Primero el usuario entra en una página web del proveedor de servicio SSH (SP). Como no está autenticado aún, se le muestra una página por defecto de bienvenida. Luego el usuario se autentica a través del proveedor de identidad (IdP).

Como se puede observar en el caso de uso 1.2, este es el método que hemos utilizado en el proyecto de SSH sobre federación de identidad. Hemos mantenido la misma idea de autenticación propuesta por la federación noruega.

En este documento se propone colocar una página web tras un SP, por

cada servidor SSH federado que se ofrezca. En nuestro caso no hemos mantenido esta idea, puesto que complicaría el despliegue del servicio.

A continuación vamos a detallar las diferentes propuestas que se pueden encontrar en este documento, además comentaremos los pros y los contras de estas ideas.

- One-time passwords. Contraseñas de un solo uso: Este método consiste en generar una contraseña de un solo uso para el usuario. Este es un método sencillo en el cual cada vez que un usuario se autentica se generaría una cuenta con una contraseña aleatoria valida solo una vez.

  Para este caso se puede utilizar OPIE (one-time password in everything) que es una aplicación desarrollada por el laboratorio de investigación naval de los Estados Unidos de América. Se puede utilizar en el servidor web el ejecutable "opie-client" para generar las contraseñas para los usuarios.

  Con este sistema, la contraseña generada sólo será valida una vez. Después de esto no podrá ser usada otra vez para autenticarse en este servidor.

  Este método es de fácil implementación, pero tiene grandes inconvenientes. El principal inconveniente es que es muy incomodo para el usuario, ya que tiene que ir copiando y pegando la contraseña para poder acceder.

  Otro gran inconveniente es que se pierde la posibilidad de Single Sign On (SSO), puesto que por cada servidor SSH tendrás una contraseña diferente.

  En cambio solventa el problema de tener que introducir tu contraseña en un servidor extraño puesto que la contraseña que se introduce es de un solo uso.

- Credenciales de la federación: Otra posibilidad es proporcionar nuestras credenciales de la federación de identidad directamente sobre el servidor SSH, sin pasar a través del la autenticación web. Esta solución puede ser implementada usando un pequeño módulo PAM que se conecta a la federación y autentica al usuario directamente con las credenciales que este ha proporcionado.

Pero este método no es una solución viable puesto que el nombre de usuario y la contraseña nunca debería pasar a través de un proveedor de servicio.

- Clave pública: Una mejor opción es autenticar al usuario usando criptografía de clave pública. Primero el SP puede intentar obtener la clave pública del usuario a través de los atributos recibidos del IdP.

  Si la clave pública no se proporciona por parte del IdP se puede ofrecer al usuario la posibilidad de subir la suya a través de una interfaz web. La clave se almacenará en el fichero authorized_keys, permitiendo al usuario autenticarse usando su clave privada.

  Hay que tener en cuenta que si no se borra la clave del fichero authorized_keys, el usuario podrá entrar directamente sin tener que autenticarse en la federación de identidad. Esto se podría solventar con un proceso cron que limpiara los fichero cada cierto tiempo.

  Este método es en el cuál nos hemos basado de una manera más directa, puesto que ofrece al usuario una autenticación mucho más simple, y solventa la mayoría de los problemas.

  La aplicación web tras el SP que hemos desarrollado es igual a la descrita en el documento. En un principio intenta conseguir la clave pública a través de los atributos que proporciona el IdP, pero también ofrece la posibilidad de introducir una manualmente.

  Sin embargo, aunque solventa el problema de tener que recordar una clave y otros problemas relacionados con el uso de contraseñas no es una solución completa. Al estar ligada a un solo servidor SSH no es posible entrar en diferentes servidores con una sola autenticación sino que habría que autenticarse en cada uno de ellos. Esa parte es la que aporta nuestro proyecto.

- Java SSH aplet: Otra opción propuesta en el documento de la federación noruega es el uso de un servicio web, tras un SP, que lanza un applet SSH en Java. Se puede ofrecer un login automático.

  En este caso se realiza la operación inversa a lo que se busca en el proyecto, pero con similar resultado. En un principio se quieren llevar los beneficios de la federación de identidad fuera del ámbito web, pero buscando este objetivo se hace lo opuesto, y es llevar otro ámbito, como es el del SSH a la Web, mediante el uso de un applet de Java.

Lo bueno de este método es que tiene acceso a cuentas SSH de manera más simple aún puesto que funcionaría de igual manera que cualquier página web federada. Además sigue estando la posibilidad de entrar en diferentes servicios SSH, habiendo realizado la autenticación una sola vez.

Lo malo es que no proporciona al usuario un acceso SSH completo, sino que ofrece una shell, pero siempre a través del navegador, con las limitaciones que esto conlleva. Además de que no aporta nada a la idea de llevar la federación de identidad fuera de la web.

En conclusión podemos decir que hay algunas investigaciones previas sobre el concepto de SSH sobre federación de identidad y que la idea de realizar este proyecto nace, en gran medida, a partir del documento publicado por la federación noruega.

Por supuesto nuestro proyecto no se queda en una prueba de concepto, como es el caso del documento comentado, sino que a partir de estas pruebas de concepto se implementa un sistema más completo y funcional el cuál utiliza y mejora las ideas propuestas, dando así un nuevo punto de vista a la idea principal.

1.3.1. Otros antecedentes a destacar

Como se verá en el capitulo 3.3, para la implementación se ha optado por crear un parche para el servidor SSH openssh, para realizar la autenticación por clave pública a través de un servidor externo, además de por los ficheros authorized_keys.

Por ello es interesante comentar en esta sección un proyecto que realiza algo similar a nuestra implementación.

http://dev.inversepath.com/trac/openssh-lpk [Openssh-lpk]

Este proyecto consiste en un parche para el servidor SSH openssh, que nos da la posibilidad de autenticar a los usuarios a través del sistema de clave pública, almacenando las claves en un servidor LDAP.

"The lpk patch allows you to lookup ssh public keys over LDAP helping central authentication of multiple servers. This patch is an alternative to other authentication system working in a similar way (Kerberos, SecurID, etc...), except the fact that it's based on OpenSSH and its public key code."

Para este proyecto se buscaba algo parecido, pero en un principio no nos queríamos restringir al uso de un servido LDAP. Aunque posteriormente se ha basado en este protocolo para proporcionar el servidor de claves.

Sin embargo no nos hemos decantado por usar este parche, por su complejidad, y hemos decidido implementar uno nuevo, con las mínimas variaciones posibles, y que cubriera nuestro caso específico.

El parche implementado, es fácilmente modificable para que permita diferentes servidores de claves, no solo servidores LDAP.

Capítulo 2

Análisis del problema

2.1. Necesidad de federación de identidad

2.1.1. ¿Qué es la federación de identidad?

La gestión de identidad centralizada fue creada para ayudar con el trato de usuarios y seguridad de datos cuando el usuario accede desde dentro de la misma red, o al menos desde dentro del mismo dominio de control. Cada vez más, los usuarios están accediendo a sistemas externos que están fuera de su dominio de control y usuarios externos están accediendo a sistemas internos.

La federación de identidad es un concepto nuevo, nacido a partir del uso masivo de la web así como de la necesidad de interoperabilidad entre diferentes entidades que gestionan diferentes grupos de usuarios de maneras totalmente diferente.

Mediante federación de identidad los usuarios pueden emplear las mismas credenciales (típicamente usuario y contraseña) para identificarse en redes de diferentes universidades, empresas o entidades. De este modo las entidades comparten información sin compartir el almacenamiento, seguridad y autenticación de los usuarios como requieren otras soluciones (metadirectorio, Single Sign On, etc.). Para su funcionamiento es necesaria la utilización de estándares que definan mecanismos que permiten a las diferentes organi-

zaciones compartir información entre dominios. El modelo es aplicable a un grupo de organizaciones o a una gran organización con numerosas delegaciones y se basa en el çírculo de confianza"de estas, un concepto que identifica que un determinado usuario es conocido en una comunidad determinada y tiene acceso a servicios específicos.

La federación de identidad, describe las tecnologías, estándares y casos de uso los cuales sirven para habilitar la portabilidad de información de identidad entre diferentes dominios de seguridad autónomos. El objetivo final de la federación de identidad es dar acceso seguro a datos, sistemas u otros dominios, a los usuarios de un dominio, y sin necesidad de redundancia de administración de usuarios.

El uso de la federación de identidad puede reducir costes eliminando la necesidad de escalar el sistema de gestión de identidad. La federación de identidad puede incrementar la seguridad y reducir los riesgos habilitando a una organización a identificar y autenticar un usuario una vez, y luego usar esta información de identidad entre diferentes sistemas, incluso entre páginas de organizaciones externas. Puede mejorar la privacidad permitiendo al usuario gestionar qué información es compartida, o limitando la cantidad de información compartida. Además puede mejorar drásticamente la experiencia del usuario final eliminando la necesidad de registrar una cuenta nueva, o la necesidad de tener que hacer login de manera redundante.

2.1.2. Partes en la federación de identidad

Un sistema de federación de identidad está formado por diferentes partes o sistemas necesarios para dar el servicio. A continuación se detallan las partes más importantes de este tipo de sistema junto con algunos ejemplos de posibles tecnologías que pueden desempeñar el papel.

- **Single Sign On (SSO)**

  Un sistema Single Sign On es un sistema que permite simplificar el acceso por parte del usuario, y de las aplicaciones, a diferentes aplicaciones que comparten los mismos usuarios y por tanto el mismo sistema de autenticación.

Este sistema nace de la necesidad de autenticar y autorizar al mismo grupo de usuarios, pero en diferentes aplicaciones. Por lo tanto se propone un sistema de autenticación único de tal modo que todas las aplicaciones utilicen este mismo sistema, y mejorando así la experiencia del usuario final ya que siempre introducirá sus credenciales en un sistema homogéneo independientemente de la aplicación, y además la sesión permanece entre diferentes aplicaciones. Por parte de las aplicaciones que utilizan este sistema, evita la necesidad de tratar con la identidad de estos usuarios ya que el sistema de Single Sign On es el que los autentica.

SSO es un método de control de acceso que permite autenticar al usuario una vez, y conseguir acceso a diferentes recursos en diferentes sistemas software.

En una infraestructura homogénea, o al menos en una en la que exista un único esquema de autenticación o donde existe una base de datos centralizada de usuarios, el sistema SSO es una gran ventaja.

En la federación de identidad el Sigle Sign On no es algo necesario, pero sí recomendable. No es un requisito indispensable para montar un sistema de gestión de identidad federada, pero sí que facilita mucho el acceso por parte del usuario.

Por lo tanto en la mayoría de los sistemas de federación de identidad se hace uso de SSO.

Algunos ejemplos de software que puede servir como SSO:

- **OpenID** http://openid.net/
- **PAPI** http://papi.rediris.es
- **Sun Access Manager** http://www.sun.com/software/products/access_mgr/index.jsp
- **OpenSSO** https://opensso.dev.java.net/

- **Proveedor de identidad (IdP)**

  Un proveedor de identidad es un sistema que gestiona la identidad de los usuarios de una organización y ofrece un servicio de autenticación a otras aplicaciones.

  En la práctica un Proveedor de Identidad es una aplicación web a la cual serán redirigidos todos los usuarios de una organización que quieran acceder a una aplicación federada y donde se autenticarán, y este

IdP ofrecerá los datos de autenticación a las diferentes aplicaciones federadas.

Es necesario que haya un IdP por cada organización participante en la federación de identidad puesto que es la vía de autenticación de sus usuarios internos.

Es aquí donde se hace importante el sistema SSO del que hemos hablado antes ya que se puede enlazar el IdP con el sistema de SSO para ofrecer un sistema de autenticación único para todas las aplicaciones federadas, ya sean internas a la organización o externas, por lo que el usuario final tendrá un acceso homogéneo a todo el sistema federado.

Algunos ejemplos de software que pueden dar el servicio de Proveedor de Identidad:

- **PAPI** http://papi.rediris.es
- **Shibboleth IdP** http://shibboleth.internet2.edu/
- **SimpleSAMLPHP** http://code.google.com/p/simplesamlphp/

- **Proveedor de servicio (SP)**

  Un proveedor de servicio es la otra parte de la federación. Todas las aplicaciones web federadas, deberán estar protegidas por un proveedor de servicio.

  El SP permite impedir el acceso a las aplicaciones redirigiendo a los usuarios a sus correspondientes IdP para que se autentiquen, y en ese caso permite el acceso comunicándose con el IdP que le proporcionará la información necesaria para la autorización y el correcto funcionamiento de las aplicaciones.

  Un SP está estrechamente relacionado con un servidor web y se comunica con los diferentes IdPs de la federación. Esta es la forma que tiene el sistema de pasar los datos de identidad de una organización a otra de forma segura.

  Cada aplicación web federada debería estar protegida por un SP, así pues puede haber más de un SP por cada organización.

  Dada la estrecha relación entre el IdP y el SP, el software necesario para dar este servicio se puede encontrar en las mismas páginas. PAPI, Shibboleth y simplesamlphp ofrecen tanto el servicio de IdP, como el de SP:

- **PAPI** http://papi.rediris.es
- **Shibboleth IdP** http://shibboleth.internet2.edu/
- **SimpleSAMLPHP** http://code.google.com/p/simplesamlphp/

En teoría es independiente el software que se utilice como IdP o como SP por las diferentes organizaciones ya que todos deberían compartir el mismo protocolo.

- **Where are you from (WAYF)**

  El objetivo del servicio "Where are you from" (WAYF) es guiar al usuario a su propio Proveedor de Identidad (IdP). A veces es llamado "Identity Provider Discovery".

  Básicamente lo que hace es presentar al usuario una lista de Proveedores de Identidad y redirigir al navegador del usuario al IdP seleccionado.

  Algunos sistemas WAYF implementan otras funcionalidades adicionales, que mejoran la experiencia del usuario, y la facilidad de uso. Por ejemplo existen varios métodos para recordar o adivinar el IdP del usuario.

  Un ejemplo de software que implementa este servicio se puede encontrar:

  - **Switch** http://www.switch.ch/aai/support/tools/wayf.html

  El WAYF es una parte importante en el sistema de la federación puesto que es el elemento que conecta los Proveedores de Servicio (SP) con los Proveedores de Identidad (IdP). Cuando un usuario intenta acceder a una aplicación protegida tras un SP, este será redirigido al WAYF donde el usuario seleccionará cuál es su organización y en consecuencia el sistema WAYF redirigirá al usuario al Proveedor de Identidad pertinente.

- **Servicio de directorio (LDAP)**

  "Un servicio de directorio (SD) es una aplicación o un conjunto de aplicaciones que almacena y organiza la información sobre los usuarios de una red de ordenadores, sobre recursos de red, y permite a los

> administradores gestionar el acceso de usuarios a los recursos sobre dicha red. Además, los servicios de directorio actúan como una capa de abstracción entre los usuarios y los recursos compartidos."
>
> Realmente no es imprescindible el servicio de directorio para desplegar una arquitectura de identidad federada pero normalmente siempre va ligada a este tipo de aplicaciones. Esta estrecha relación se debe a que últimamente se está imponiendo el uso de esta tecnología para la gestión de identidad de manera local. Por lo tanto la gran mayoría de las organizaciones ya tendrán un servicio de directorio para gestionar sus usuarios.
>
> La mayoría de sistemas y aplicaciones para desplegar una federación de identidad están desarrollados pensando en este hecho por lo tanto facilita la tarea.
>
> Sin embargo es posible utilizar bases de datos y otros tipos de almacenamiento o gestión de usuarios aunque la configuración será más compleja y tal vez no sea tan óptima.

2.2. Uso de la federación fuera de la Web

Como hemos visto, el sistema de federación de identidad surge a partir de un problema con el uso de aplicaciones externas por parte de usuarios internos o viceversa. Normalmente estas aplicaciones son aplicaciones web dada la proliferación de este tipo de aplicaciones. Es por esta razón por la que, por diseño, el sistema de identidad federada está pensado para aplicaciones web, es decir, para utilizarlo desde un navegador.

Así pues los componentes de un sistema de federación de identidad, tales como IdPs o SPs, son aplicaciones web y están fuertemente ligados a servidores web tales como apache o tomcat.

El funcionamiento se basa en el paso de cookies al navegador y variables de servidor, por lo que claramente es necesario el uso de navegadores modernos y lenguajes en la parte del servidor preparados para esta tarea.

Intentar federar aplicaciones que no son web es un trabajo difícil, dado el terreno en el que nos movemos. Puesto que estas aplicaciones tendrán que co-

municarse de alguna forma con las aplicaciones que conforman la federación, por ejemplo con el IdP para la autenticación del usuario.

Por lo tanto para poder llevar el sistema de federación de identidad a otro tipo de aplicaciones será necesario incluir interfaces y sistemas intermedios que hagan de puente o conexión entre el sistema web de la federación y la aplicación en cuestión. Es necesario idear algún mecanismo para saber si un usuario está autenticado en la federación o no.

El problema reside en que el sistema de federación de identidad conoce quién es el usuario y si está autenticado por una cookie que muestra el navegador, pero en aplicaciones que no utilicen el navegador esta cookie no existe, por lo tanto hay que considerar algún otro sistema alternativo de tal forma que el usuario pueda mostrar al servidor quién es de manera inequívoca y sin posibilidad de suplantación de identidad.

Otro problema es la autenticación, y es que todo usuario sólo proporciona sus credenciales al IdP conocido, el de su organización, por lo tanto siempre habrá una confianza en que esas credenciales no están siendo dadas a una organización externa. Esto funciona a través de redirecciones del navegador desde la aplicación protegida tras un SP a la que quiere acceder, hasta el IdP, pasando por el sistema WAYF. En el caso de otro tipo de aplicaciones crear un sistema de autenticación diferente sería muy costoso ya que habría que implementar todos estos componentes pero para la nueva aplicación.

2.3. La importancia del SSH

SSH (Secure SHell) es un protocolo que sirve para acceder a máquinas remotas a través de la red. Permite el manejo de una máquina como si se estuviera delante de ella dando acceso a una línea de comandos. Así se pueden utilizar recursos de diferentes máquinas desde un mismo lugar, utilizando este protocolo, además de administrar máquinas remotas.

Además de la línea de comandos, el protocolo SSH da otras posibilidades como la copia de ficheros entre máquinas o la ejecución de aplicaciones gráficas importando el protocolo de comunicación gráfico.

Con SSH se trabaja de forma similar a como se haría con el comando telnet junto con la posibilidad de copiar ficheros con el protocolo ftp. Sin embargo, SSH es algo más que eso, y como su propio nombre indica, es un protocolo seguro. Por lo tanto utiliza técnicas de cifrado que hacen que la información, que viaja por el medio de comunicación, vaya cifrada y así proporcionar algo más de seguridad a las operaciones realizadas en la red.

Hoy en día el protocolo SSH es muy utilizado en la administración de máquinas en redes locales así como a través de internet, puesto que ofrece una manera sencilla y segura de conseguir acceso a estas máquinas remotas.

Además en el ambiente académico es utilizado para compartir un recurso limitado. Por ejemplo un supercomputador puede ser utilizado por diferentes estudiantes, profesores o investigadores utilizando este protocolo, y consiguiendo así hacer uso de un hardware no disponible físicamente pero que a través de la red se presenta ante nosotros como un recurso más.

Existe una tecnología libre que implementa este protocolo, con diferentes funcionalidades, se trata de openssh ([OpenSSH]).

Además de lo comentado anteriormente, openssh permite cosas muy interesantes, como túneles seguros, diferentes métodos de autenticación, y soporte para todas las versiones del protocolo SSH.

Además del método de acceso común, usuario y contraseña, openssh ofrece la posibilidad de autenticar a través de un sistema de clave pública y clave privada. Esta característica es muy interesante para nuestro caso ya que queremos evitar que el usuario escriba su contraseña, así pues, más adelante hablaremos sobre cómo utilizar este mecanismo de clave pública y privada para la implementación de la solución.

Por todas estas razones, SSH es una opción interesante en la cual utilizar federación de identidad. Puesto que es muy utilizado y en todos los casos se necesita comprobar la identidad del usuario.

Capítulo 3

Solución propuesta

En este capítulo comentaremos la solución que hemos propuesto y que hemos llevado a cabo para conseguir una autenticación a través de la federación de identidad para SSH.

3.1. Posibles soluciones barajadas

La parte más importante de este proyecto es la conexión entre la federación de identidad, basada en aplicaciones web, y los servidores SSH.

Para conseguir esta conexión es casi imprescindible una pequeña aplicación web que esté tras un proveedor de servicio y la cual pueda acceder a los atributos del usuario una vez este se haya autenticado en el IdP de su organización.

Esta aplicación web será la que diga si un usuario está autenticado en la federación o no y su funcionamiento variará ligeramente según la implementación concreta de la solución.

Servidor ssh:

Como primera opción se barajó la idea de utilizar un modulo PAM (Plugable Authentication Method) ya que el servidor SSH permite este tipo de

autenticación. Los módulos PAM son un método que permite al administrador controlar cómo se realiza el proceso de autenticación de los usuario para aplicaciones a través de unas bibliotecas.

La opción de los módulos PAM permitiría el acceso sin tener que tocar el servidor SSH, simplemente añadiendo un módulo al sistema, y cambiando una línea en un fichero de configuración.

Pero como inconveniente tiene que estos módulos autentican con usuario y contraseña, por lo que o se le pasa el usuario y contraseña al servidor SSH (cosa que no queremos) o se crea una cuenta de un solo uso.

Otra posible solución es utilizar el método de autenticación por clave pública que proporciona el servidor openssh. Este método permite autenticar a un usuario sin que éste tenga que introducir su contraseña y sería lo más automático posible.

Éste método funciona buscando la clave pública del usuario en un fichero del sistema local. Pero nosotros no tenemos un fichero en cada servidor SSH con las claves de los usuarios autenticados. Además ésta lista de usuarios será dinámica.

La solución que hemos elegido para solventar el problema es la segunda, pero es necesario que además de mirar en los ficheros locales pregunte a un servidor intermedio para saber qué usuarios están autenticados en la federación. Para esto es necesario tocar un poco el código del servidor SSH sin cambiar su funcionamiento normal pero añadiendo la opción de pedir claves públicas a un servidor externo.

Como hemos optado por la opción de pedir la clave pública del usuario desde el servidor SSH a un servido externo, se nos presenta un abanico de posibilidades para guardar las claves.

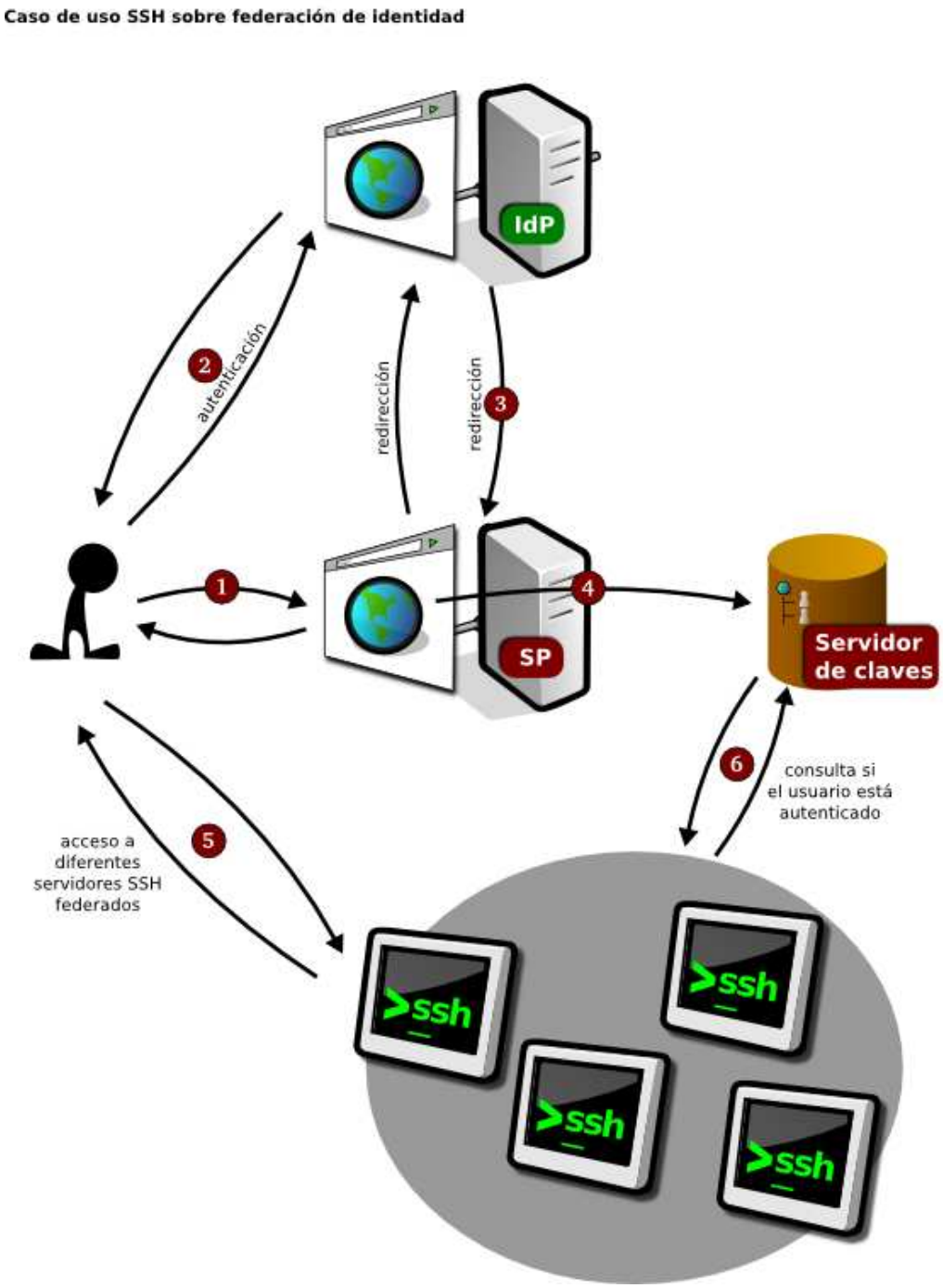
- Fichero
- Base de datos
- Directorio

En nuestro caso hemos optado por utilizar un servicio de directorio pues-

to que la federación de identidad está muy ligada al uso de esta tecnología para almacenamiento de usuarios y lo que vamos a almacenar es un conjunto de usuarios y sus claves públicas para que el servidor SSH pueda acceder a ellas.

3.2. Solución elegida

La solución propuesta consta de varias partes, como se puede apreciar en la figura:


Caso de uso SSH sobre federación de identidad
IdP
2
autenticación
redirección
redirección
3
1
4
SP
Servidor
de claves
6
consulta si
el usuario está
autenticado
acceso a
diferentes
servidores SSH
federados
5
>ssh
>ssh
>ssh
>ssh

Las partes que incumben a este proyecto son:

- La aplicación web protegida tras el SP global de la federación. Esta aplicación será la encargada de verificar que el usuario se ha autenticado en la federación y hará las operaciones pertinentes para que se pueda acceder a los datos necesarios del usuario a través de un medio que no sea un navegador web, de tal forma que el servidor SSH pueda autenticar.
- El servidor de claves. Será un servidor que se encargará de almacenar los datos de los usuarios autenticados en la federación para facilitar el acceso a los mismos por parte del servidor SSH. Este servidor será el punto intermedio que enlace la federación (web) con el servicio de SSH.
- Los servidores SSH. Todos los servidores que presten el servicio de SSH sobre federación de identidad, deberán autenticar a los usuarios en la federación de alguna otra forma. Para conseguir esto está el servidor de claves, que facilitará la comunicación entre la federación y los servidores.

3.3. Parche para openssh (proceso de autenticación de sshd)

Para implementar la solución elegida es necesario tocar algo del código del servidor de SSH de openssh, sshd.

Una de las grandes ventajas del software libre es que si una aplicación no cumple los requisitos que necesitamos pero está cerca de conseguirlo, siempre se puede coger el código del proyecto, estudiarlo, y si es viable, mejorarlo o adaptarlo para que pueda cumplir nuestras necesidades.

En nuestro caso el servidor SSH hace casi todo lo que queremos que haga. Pero no autentica frente a una federación. Una posible solución sería implementar un servidor SSH que sí hiciera la autenticación de esta manera, pero sería tremendamente costoso. Por esta razón es mucho más simple el

basarse en un proyecto maduro, como es openssh, y modificarlo de tal forma que se adapte a nuestras necesidades.

Gracias a la licencia libre del proyecto openssh, BSD License, es posible acceder a todo el código, se puede estudiar, modificar, y distribuir de forma libre, por lo que nuestro proyecto es viable.

Por lo tanto nos hemos basado en este proyecto cogiendo su código y desarrollando un parche que permita la autenticación a través de certificado de clave pública, pero situado este certificado en un servidor de claves en lugar de en un fichero local del sistema.

La autenticación del servidor SSH funciona de la siguiente manera:

1. Intenta la autenticación por certificado de clave pública. Busca en el fichero $HOME/authorized_keys y prueba que el usuario que está intentando autenticarse tiene la clave privada de alguna de las claves que aparecen en este fichero.
2. Si no consigue autenticar y está activa la autenticación por módulos PAM, el servidor pedirá una contraseña al usuario y se ejecutará la pila de módulos.
3. En otro caso autenticará con los usuarios del sistema.

En este esquema de funcionamiento tenemos que encajar nuestro sistema de autenticación de identidad federada. Buscamos la mayor comodidad posible para el usuario, así pues, queremos que el usuario no tenga que introducir ninguna contraseña en el servidor SSH.

Si llega al paso dos de la autenticación del SSH el servidor pedirá automáticamente una contraseña al usuario, y dado que queremos evitar esta incomodidad, no es viable el crear un módulo PAM que autentique en la federación de identidad.

Por esta razón es necesario tocar el código del servidor SSH. Más concretamente la parte de autenticación por clave pública. Ya que utilizaremos este sistema para identificar unívocamente al usuario que quiere autenticarse. Puesto que no se va a introducir ninguna contraseña será necesario que el

usuario tenga algo que lo identifique. En el caso de la federación de identidad por páginas web se utiliza una cookie que guarda el navegador. Para nuestro caso, la clave privada del usuario servirá para identificarlo, teniendo acceso el servidor SSH a la clave pública una vez que el usuario esté autenticado en la federación.

Por lo tanto la autenticación del servidor SSH parcheado quedaría así:

1. Intenta la autenticación por certificado de clave pública. Busca en el fichero $HOME/authorized_keys, y prueba que el usuario que está intentando autenticarse tiene la clave privada de alguna de las claves que aparecen en este fichero.
2. **Si no consigue autenticar pregunta al servidor de claves de la federación por el usuario. Si el usuario está autenticado el servidor de claves le devolverá la clave pública del mismo. Con esta clave pública intentará autenticar al usuario que sólo tendrá acceso si tiene la clave privada**
3. Si no consigue autenticar y está activa la autenticación por módulos PAM, el servidor pedirá una contraseña al usuario y se ejecutará la pila de módulos.
4. En otro caso autenticará con los usuarios del sistema.

Se puede ver el funcionamiento en el siguiente esquema:

3.3.1. Posibles soluciones al SSO para SSH

Durante la realización de este proyecto han surgido algunos detalles y funcionalidades que se podrían implementar para el servidor SSH y que se podrían utilizar de forma independiente al proyecto aquí descrito.

Una de estas posibilidades, y quizás la de mayor interés y que tendría entidad propia fuera de este proyecto, es el Single Sing-On para SSH.

Para realizar la tarea de SSO es necesario que el cliente autenticado tenga un "token" o algo que lo identifique unívocamente, en nuestro caso sería la clave privada de SSH. Y el servidor debería conocer quién está autenticado, en nuestro caso preguntando al servidor de claves.

Para el caso de SSO únicamente, sin autenticación por federación, sería necesario retocar el parche del servidor SSH para que al acceder a un servidor a través de usuario y contraseña éste servidor comunicara al servidor de claves que el usuario se ha autenticado. Así cuando el mismo usuario vaya a acceder al mismo servidor, o incluso a otro dentro del SSO, tendría acceso directo.

Este concepto lleva consigo un par de pequeños problemas y es que se ha de poder acceder a alguno de los servidores introduciendo la clave lo que implica que el usuario ha de tener al menos la clave de uno de los servidores de SSH o por lo menos debe conocer la clave pública de este usuario de

manera permanente.

El otro problema es que el servidor donde se autentique el usuario por primera vez ha de conocer la clave pública de este usuario para informar al servidor de claves de quién se ha autenticado. Otra alternativa sería que el servidor de claves tuviera a todos los usuarios en un principio registrados, pero no activos, por lo que decir que alguien se ha autenticado tan solo sería poner a este usuario como activo.

3.4. Servidor de claves

En principio la idea del servidor de claves era que fuera un servidor independiente con un protocolo simple que solo permitiera preguntar si un usuario está autenticado, y en caso afirmativo, se recibiría la clave pública del mismo.

La primera versión del protocolo simple sería:

- Peticiones al servidor:

```
USR:nombre
XIT
```

  La orden USR es la petición de la clave de un usuario y la orden XIT indica desconexión.

- Respuestas del servidor:

  El servidor responderá con la clave pública del usuario o con la cadena "NOT FOUND" si no lo ha encontrado.

Esta implementación facilitaría enormemente la parte del parche al servidor SSH puesto que tan solo se necesita hacer una conexión por socket normal, sin necesidad de utilizar ninguna librería complementaria.

Por otra parte, con esta implementación de un protocolo simple sería muy fácil cambiar el almacenamiento de los usuarios, tan solo creando una interfaz que convirtiera las consultas al sistema de almacenamiento elegido a este protocolo. Así pues sería muy fácil almacenar los usuarios con ficheros de texto plano, con un sistema de base de datos relacional, con un servicio de directorio, etc.

Sin embargo, dificulta la parte del servidor de claves que ya tiene que ser una implementación propia, que utilice un almacenamiento de usuarios y claves determinado.

Como esta fue la primera aproximación, se ha implementado un servidor de claves de ejemplo en python. En principio almacenaba los usuarios y las claves en texto plano. Posteriormente se extendió para realizar el almacenamiento en una base de datos MySQL. Y en realidad incorporar nuevos sistemas de almacenamiento es algo tan simple como implementar una clase con un metodo *get_rsa(nombre_de_usuario)* que devuelva la clave pública RSA del usuario si está autenticado, y en otro caso que devuelva una cadena vacía.

Tratando de buscar el mínimo impacto en el código del servidor SSH las peticiones al servidor de claves se realizan en una función que está implementada en un fichero aparte, por lo que en el código del servidor tan solo hay que incluir la llamada a esta función y se puede variar el modo de funcionamiento de la petición. De esta forma si se varía el protocolo de comunicación entre el servidor SSH y el servidor de claves no haría falta tocar el código real del servidor SSH, solamente la función que es llamada desde el mismo y que es de implementación propia por lo que no afectaría al funcionamiento normal del mismo.

Por otra parte, si el desarrollo habitual del servidor varía alguna parte de la autenticación el parche es fácilmente adaptable a futuras versiones puesto que del código original toca lo menos posible.

Tras esta primera implementación de prueba y tras ver que todo era correcto, y que funcionaba bien, se decidió incrementar la seguridad entre las comunicaciones.

En un principio se optó por incrementar la seguridad del servidor de cla-

ves utilizando sockets ssl, pero tras observar que esto complicaría la solución de manera drástica se optó por utilizar algo más simple.

Así pues se pensó en utilizar como servidor de claves un servicio de directorio y utilizar como protocolo de comunicaciones el propio protocolo LDAP que tiene una versión segura.

La opción de elegir un servicio de directorio no es algo casual sino que se eligió este sistema porque cumple los requisitos necesarios para el servidor de claves, tal y como está definido, y además cuando se habla de federación de identidad es casi imposible no hablar de servicios de directorio. Hoy en día la federación de identidad está muy ligada a este tipo de almacenamiento de usuarios por lo tanto es una buena opción.

Al utilizar un servicio de directorio como servidor de claves públicas se simplifica la parte del servidor de claves, puesto que ya está hecho, y es muy fácil desplegar uno. Pero por otra parte se complica un poco la modificación al servidor SSH.

Puesto que estamos usando el servidor SSH que viene con la implementación openssh, y este está en C, nuestro parche deberá hacer peticiones a un servidor de claves a través del protocolo de comunicación LDAP, desde el lenguaje C.

Por suerte las consultas al servicio de directorio desde C no son nada complicadas e implementando una simple función es posible modificar el parche para que el servidor SSH ahora se comunique con un servidor de claves a través de LDAP.

3.4.1. Esquemas LDAP

Los sistemas de directorio utilizan diferentes atributos para guardar los datos pertenecientes a los diferentes objetos. Cada posible atributo a utilizar está definido en un "Esquema LDAP", y se podrá utilizar siempre que el objeto tenga el "objectclass" que define ese atributo.

Existen muchos estándares de esquemas LDAP, que suelen venir por defecto en la mayoría de los servicios de directorio. Y cada esquema tiene sus

atributos destinados cada uno a almacenar un tipo de dato concreto.

Es posible utilizar un atributo cualquiera, siempre que sea del mismo tipo, para guardar cualquier información, pero esto no es recomendable, puesto que complica la compresión de futuros usuarios y administradores, dado que cada atributo tiene una definición de para qué se usa.

Por ejemplo se puede usar el atributo "mail" para guardar el dni del usuario. Pero no es correcto, ya que el atributo mail está especificado para almacenar la dirección de correo.

Así pues, hay que definir o buscar unos esquemas que definan los atributos que necesitamos para almacenar los usuarios junto a sus claves en el servidor de claves.

En principio los datos que ha de almacenar el servidor de claves para este proyecto son:

- **uid** del usuario autenticado en la federación.
- **clave pública** del usuario autenticado en la federación.
- **timeout** tiempo de vencimiento de la sesión del usuario autenticado en la federación.

El primer atributo, **uid**, está cubierto en los esquemas que vienen por defecto en el esquema **person**.

Para el segundo atributo en principio se pensó en utilizar usercertificate, pero no es el más adecuado ya que está pensado para almacenar un certificado personal. Así pues buscando alternativas se ha decidido utilizar el esquema *openssh-lpk* `http://dev.inversepath.com/openssh-lpk` que ofrece un atributo sshPublicKey, destinado para este propósito.

Y por último, para cubrir las necesidades del segundo atributo se ha optado por utilizar el esquema *schac* `http://www.terena.org/activities/tf-emc2/schac.html`, y utilizar el atributo schacUserStatus para guardar el timeout con el siguiente formato: `schac:userStatus:us.es:timeout:1205274586`, donde el número representa la fecha en segundos de expiración.

Por supuesto se pueden utilizar otros atributos para almacenar los datos de usuario y tan solo habría que cambiar la configuración del servidor sshd. No sería necesario recompilar el parche.

3.5. Aplicación de login

La aplicación de login es también una parte importante de este proyecto, ya que será lo que conecte la federación con el servidor de claves al que consultarán los servidores SSH. Por lo tanto todo usuario que quiera utilizar el servicio de SSH federado deberá entrar antes en esta aplicación federada para que la propia aplicación pueda tener acceso a los datos que le facilite el IdP del usuario en concreto y así indicar al servidor de claves que el usuario está autenticado, y cuál es su clave pública.

Esta aplicación, como se ha comentado anteriormente, deberá ir protegida tras un SP, no necesariamente perteneciente a una organización, puesto que la aplicación será única para todos los usuarios de todas las organizaciones implicadas en la federación.

Se ha buscado que la interfaz de la aplicación sea lo más simple y clara posible. Al estar protegida tras un SP todo usuario que intente acceder a esta página será redirigido al WAYF de la federación si no está autenticado aún. En el WAYF seleccionará su organización de origen y será redirigido a la página de login de su IdP.

Una vez que el usuario se autentique en su IdP será redirigido a la aplicación de login, y en este momento, la aplicación tendrá acceso a los datos necesarios del usuario.

La aplicación mirará si el atributo especificado en la federación para almacenar la clave pública del usuario está asignado y en caso afirmativo cogerá la clave pública que viene dada en ese atributo y la introducirá en el servidor de claves. Se muestra al usuario una pantalla en la cual se especifica el nombre de usuario para acceder por SSH, así como la fecha y hora de caducidad de esa sesión.

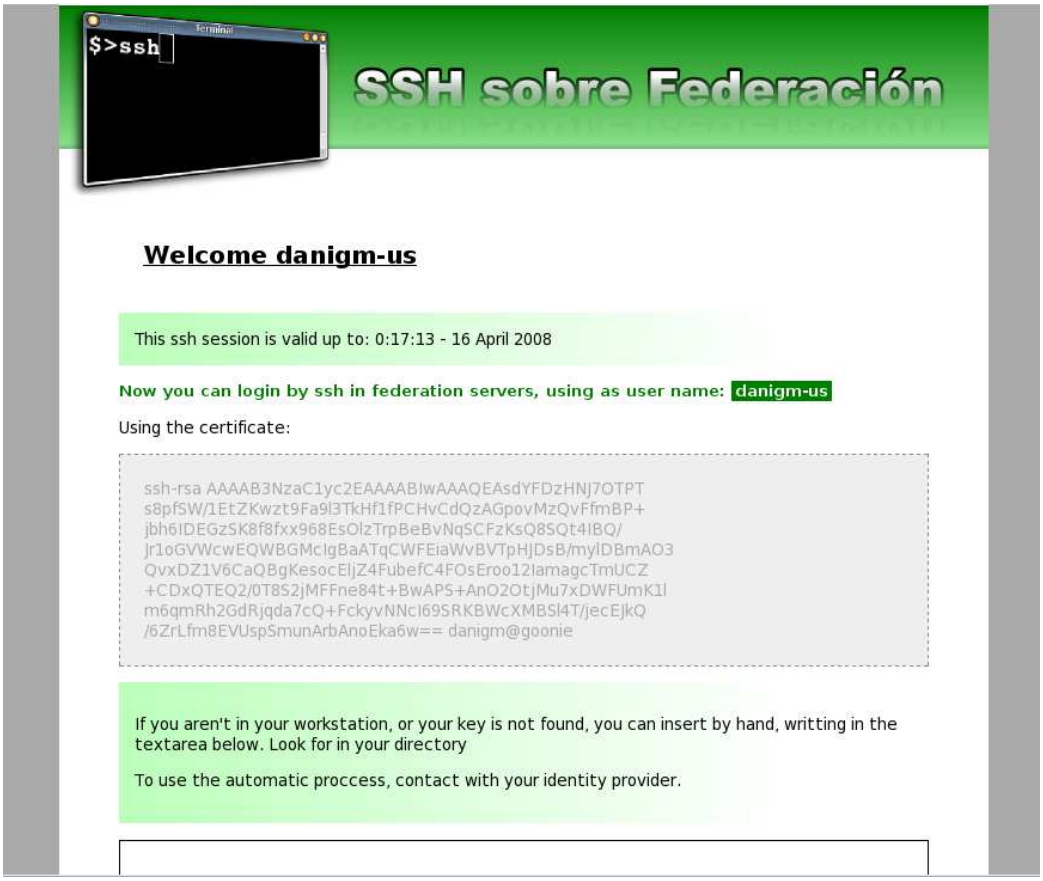
En el caso en el que no se encuentre la clave pública del usuario se

ofrece la posibilidad de introducirla manualmente en un cuadro de texto. Este atributo de texto también es de utilidad cuando un usuario no está en su puesto de trabajo habitual y quiere acceder utilizando un par de claves temporal.

Cuando se introduce de manera manual la aplicación verifica que es una clave pública correcta y la introduce en el servidor de claves actualizando el timeout.

Cada vez que se refresque esta página la aplicación actualiza el timeout como si el usuario acabara de acceder por primera vez.

Esta es la apariencia de la aplicación de login:



Por facilidad de desarrollo e integración con distintos servidores web se ha elegido utilizar la tecnología PHP para el desarrollo de esta aplicación.

La aplicación está separada en un frontend y un backend. El frontend se encarga de mostrar los datos y en el backend están definidas las funciones tanto de búsqueda y petición de datos como las de almacenamiento en el servidor de claves.

La implementación actual de esta aplicación está ligeramente ligada al SP de shibboleth 1.3. Pero es fácilmente modificable para que utilice otro tipo de SP solo cambiando unas variables en el código, de tal forma que consulte otro tipo de cabeceras.

Puesto que hemos decidido utilizar un servidor de claves basado en un servicio de directorio se han tenido que utilizar las bibliotecas pertinentes del lenguaje PHP para LDAP.

Toda la información sobre atributos a consultar y atributos necesarios para guardar la información, está especificada en forma de variables para que su modificación sea fácil.

Para aumentar el grado de seguridad del sistema cabe la posibilidad de extender la aplicación de manera sencilla de tal forma que antes de almacenar los datos del usuario en el servidor de claves informara al usuario sobre los datos que se van a almacenar y pidiera confirmación.

3.6. Ejemplo aplicación creación de cuentas

Un problema que hay que tener en cuenta en el despliegue de este proyecto es la necesidad de crear las cuentas de los usuarios de forma dinámica en los servidores parcheados.

En principio y según la implementación actual del proyecto, el servidor SSH parcheado supone que la cuenta de usuario existe y por tanto cuando pida la clave pública al servidor de claves generará un fichero temporal con esa clave en el directorio HOME de ese usuario y utilizará ese fichero para hacer la autenticación por clave pública.

Dado que esta implementación delega el problema de la creación de cuentas al administrador del sistema es necesario que se creen y se destruyan las cuentas de manera dinámica en el servidor.

La delegación de la creación de cuentas en el administrador del sistema permite un mayor control sobre quién va a poder acceder a este servicio, permitiendo crear políticas de creación de cuentas y no permitiendo entrar al servidor a cualquier usuario autenticado en la federación.

Para facilitar el trabajo de administración de estos servidores y además ofrecer al usuario un punto central de acceso a servidores SSH, se ha desarrollado un ejemplo sencillo de aplicación de creación de cuentas.

La idea consiste en hacer una aplicación con un aspecto visual similar a la de autenticación y que ofrezca una lista de los servidores de SSH disponibles junto con una corta descripción del servicio. Además da la posibilidad de solicitar las cuentas a los diferentes servidores para así tener una creación de cuentas automática y que no haya que hacer una creación de cuentas manual.

La aplicación iría protegida tras un SP. Cuando el usuario acceda a la aplicación, si no está autenticado ya en la federación, será redirigido al WAYF. En el WAYF seleccionará su institución de origen y será redirigido al IdP correspondiente, donde se autenticará. Una vez autenticado tendrá acceso a la aplicación y la misma aplicación a partir del uid del usuario consulta si el usuario tiene cuenta en el servidor y permite la creación de la cuenta con un enlace.

Para crear las cuentas desde el servidor donde esté instalada esta aplicación web hay que permitir el acceso por SSH al comando useradd, puesto que se va a utilizar una llamada al sistema para lanzar el comando a través de este protocolo, y así crear la cuenta solicitada. Para ello se ha introducido la clave pública de acceso SSH del servidor web en el fichero authorized_keys de cada servidor SSH.

Este despliegue tiene fallos de seguridad puesto que cualquiera que tenga acceso al servidor web tendrá acceso a la creación de cuentas de todos los servidores SSH.

Una mejora de esta aplicación, y el siguiente paso lógico si esto se quiere implantar en un sistema federado en producción, sería utilizar servicios web

o llamadas a otro tipo de procesos remotos que verificaran el origen de la petición con algún sistema criptográficamente seguro y tan sólo permitiendo la creación de la cuenta, es decir, un proceso en cada servidor que recibiera peticiones del servidor web que aloja esta aplicación y que creara las cuentas.

El desarrollo de este sistema permitiría también definir políticas de creación de cuentas, dando la posibilidad de que cada administrador de cada servidor SSH federado definiera su política. Así por ejemplo para un servidor general, de poca trascendencia, se podría dejar que se crearan las cuentas de forma automática e inmediata, y en otro tipo de servidores, con recursos más limitados, se mandará un correo al administrador para que este se ponga en contacto con el solicitante y según el solicitante le permitiera la creación de la cuenta o no.

Esta sería la aplicación de creación de cuentas:

Otra posibilidad menos flexible pero más integrada sería incluir en el parche la posibilidad de que se crearan y destruyeran las cuentas de manera dinámica. Así por ejemplo, se crearía la cuenta justo antes de intentar

autenticar al usuario.

Esta segunda forma es mucho más invasiva en el sistema ya que para implementarla el servidor SSH debe comunicarse con el sistema en cuestión y hacer llamadas al mismo para crear las cuentas.

Independientemente del sistema utilizado para la creación de las cuentas la aplicación de ejemplo de creación de cuentas puede servir como listado de todos los servicios SSH sobre federación de identidad que se ofrecen. Así estaría accesible en un punto centralizado y cualquier usuario de cualquier organización tendría acceso al listado y podría saber a qué recursos tiene acceso.

Capítulo 4

Implementación y despliegue

4.1. Cambios realizados sobre openssh

El código del parche para el servidor SSH debe tocar lo mínimo posible para tener seguridad a la hora de aplicar actualizaciones de openssh. Así pues la idea principal y que ha guiado el desarrollo de esta modificación ha sido esta y por lo tanto se ha creado un parche lo más simple posible, con las menores dependencias de librerías externas y fácilmente ampliable pensando en casos futuros.

Lo primero que hay que hacer para hacer una mejora o una adaptación de un proyecto software libre, es descargarse el código y empezar a estudiarlo para ver en qué partes del mismo hay que encajar la nueva funcionalidad.

Para descargar el código se ha usado el sistema de control de versiones que utilizan para este proyecto, CVS (`http://www.openssh.com/portable.html`), bajando la versión para linux, puesto que es el sistema sobre el cuál se ha desarrollado todo el proyecto.

```
export CVSROOT=anoncvs@anoncvs.mindrot.org:/cvs
export CVS_RSH=/usr/bin/ssh
cvs get openssh
```

Una vez conseguido el código hubo un proceso de estudio del mismo para entender minimamente el funcionamiento de la aplicación, antes de empezar a tocar nada.

Tras estudiar el código y realizar varias pruebas se localizó la zona de código dónde se realiza la autenticación del usuario y se comienza a modificar este código para añadirle la funcionalidad que estamos buscando.

El fichero fuente, dentro de openssh, que se encarga de la autenticación SSH es el `auth2-pubkey.c`. Concretamente la autenticación por clave pública se realiza en la función `int user_key_allowed(struct passwd *pw, Key *key)`, por lo tanto esta es la función que hay que modificar.

Concretamente este trozo de código es el encargado de comprobar que en el servidor existe un fichero que contiene las claves públicas que tienen acceso.

```
1
2     file = authorized_keys_file(pw);
3     success = user_key_allowed2(pw, key, file);
4     xfree(file);
5     if (success)
6         return success;
7
8     try suffix "2" for backward compat, too */
9     file = authorized_keys_file2(pw);
10    success = user_key_allowed2(pw, key, file);
11    xfree(file);
12
13    return success;
```

La variable *file* tiene la ruta hacia el fichero dónde están las claves públicas para acceder, así pues añadir a este código la idea de que si un usuario no puede acceder por el método estandar se compruebe si está en la federación, es tan simple como:

```
1
2     file = authorized_keys_file(pw);
3     success = user_key_allowed2(pw, key, file);
4     xfree(file);
5     if (success)
```

```
6         return success;
7
8     try suffix "2" for backward compat, too */
9     file = authorized_keys_file2(pw);
10    success = user_key_allowed2(pw, key, file);
11    xfree(file);
12
13    if (success)
14        return success;
15
16    // try external file fed+ssh <danigm>
17    if(options.usefed == 1){
18        get_rsa_key_ldap(options.fedserver, options.fedport,
19                pw->pw_name, rsa_key);
20        debug("RSA_EXTERNAL_KEY: trying this -> %s\n",rsa_key);
21
22        if(strcmp(rsa_key,"") != 0){
23            strcat(rsa_key, "\n");
24            fwrite(rsa_key, strlen(rsa_key), sizeof(char), tmp_file);
25            fclose(tmp_file);
26            success
27                =
28                user_key_allowed2(pw, key, file2);
29            unlink(file2);
30        }
31    }
32
33    return success;
```

De esta manera si no se ha podido acceder de la manera convencional y si está puesta la opción de usar el acceso federado, se hace una llamada a la función `get_rsa_key_ldap` que devolverá una cadena con la clave pública de este usuario si está autenticado y en tal caso se creará un fichero temporal que se usará para intentar autenticar.

Este simple cambio, y alguno más para definir las posibles opciones configurables con el fichero de configuración, es lo que se cambiaría de verdad sobre el código del servidor ssh real por lo que el riesgo de introducir fallos adicionales al código se reduce considerablemente.

Por tanto se delega la responsabilidad de saber si el usuario está autenticado o no a la función `get_rsa_key_ldap`. Esta función se puede modificar y cambiar por cualquier otra que devuelva una cadena con la clave pública del usuario que se le está pasando como argumento. Por esto es fácil implementar la comunicación con otros sistemas que no sean un servicio de directorio, o

por alguna función que haga alguna operación más.

Ya que la complejidad de la autenticación se ha delegado en esta función, se ha implementado en un fichero aparte que se introduce en las herramientas de compilación, `automake`, y se enlaza para que sea accesible desde el código anteriormente comentado.

Lo que hace la función básicamente es realizar una consulta al servidor LDAP buscando al usuario en cuestión. Si existe se extraen los datos de clave pública y timeout. Inmediatamente después se comprueba que el timeout no esté cumplido y en ese caso se devuelve la clave pública del usuario y se termina. En caso de que el timeout esté cumplido o que no se encuentre al usuario se devuelve una cadena vacía.

En realidad la función es bastante sencilla y simple, pues así lo hemos requerido y este era nuestro objetivo. Donde radica la mayor complejidad es en el uso de las llamadas a LDAP en el lenguaje de programación C, puesto que el acceso no es trivial, y durante el desarrollo del proyecto en la infraestructura de pruebas se ha modificado la librería de acceso openldap cambiando la forma de acceso y poniendo las funciones utilizadas en la primera versión como `deprecated`, por lo tanto se ha tenido que reescribir el código para utilizar las nuevas funciones.

Aquí está el código realmente relevante de esta función:

```
1
2     LDAPMessage *entry=ldap_first_entry(ld, msg);
3     for( attr = ldap_first_attribute(ld, entry, &ber);
4             attr != NULL; attr = ldap_next_attribute(ld, entry, ber))
5     {
6         vals = ldap_get_values_len(ld, entry, attr);
7         if (vals != NULL)  {
8             for(i = 0; vals[i] != NULL; i++) {
9                 val = vals[i];
10                /* process the current value */
11                if (strcmp(attr, timeattr) == 0){
12                        strcpy(timeout, val->bv_val);
13                    }
14                if (strcmp(attr, attribute) == 0){
15                        strcpy(rsa_key2, val->bv_val);
16                        debug("1 %s:%s\n", %attr, %rsa_key2);
17                    }
18            }
19            if (check_timeout(timeout)) {
```

```
20                      strcpy(rsa_key, rsa_key2);
21                      debug("2 %s:%s\n", %attr, %rsa_key);
22                  }else debug("\nTIMEOUT CUMPLIDO\n");
23          }
24          ldap_value_free_len(vals);
25      }
```

4.2. Aplicaciones federadas: ssh, useradd

Como se vió en 3.5, la aplicación de login tiene un cometido específico y una funcionalidad claramente definida. Por lo tanto la implementación de la misma ha sido el objetivo más fácil dentro de este proyecto.

Se ha elegido la tecnología PHP por su rápidez para el desarrollo web además de por su simple integración con todo tipo de servidores web, y en nuestro caso con apache2. Además, dado que se ha decidido utilizar un servidor de claves basado en un servicio de directorio se ha buscado una tecnología que ofrezca simplicidad a la hora de realizar estos accesos, y PHP cumple todos los requisitos.

Se ha descompuesto la aplicación en dos partes diferenciables para seguir un modelo vista controlador. De tal forma que las operaciones con datos y la lógica de la aplicación esté centralizada en `ssh_backend.php` y la parte de muestra de información y formateo de los datos con html esté en `ssh.php`.

Se explica a continuación paso a paso las funciones que se pueden encontrar en `ssh_backend.php`.

- En primer lugar hay una serie de variables que definen las opciones del entorno dónde queramos desplegar la aplicación.

```
1
2       $base_dn ='o=People,dc=us,dc=es';
3       $servidor_ldap = "goonie.us.es";
4       $puerto_ldap = 389;
5       $bn = 'cn=admin,dc=us,dc=es';
6       $pw = 'xxxxx';
7       $minutes_timeout = 30;
8       $shib_header = "HTTP_USERCERTIFICATE";
9       $rsa_server_key_attr = 'sshpublickey';
```

```
10      $rsa_server_timeout = 'schacuserstatus';
```

Están los datos para el acceso al servidor de claves, el tiempo valido de sesión, el atributo de shibboleth dónde vendrá el certificado desde el IdP, y los atributos en el servidor de claves, dónde se almacenarán los datos.

- función `modify`

```
1
2       function modify($ds, $uid, $pubkey){
3           global $base_dn;
4           global $minutes_timeout;
5           global $rsa_server_key_attr;
6           global $rsa_server_timeout;
7           // preparar los datos
8           $timeout = $minutes_timeout * 60; //5 minutos
9           $hoy = getdate();
10          $timeout = $hoy[0]+$timeout;
11          $dn = "uid=".  $uid .",".  $base_dn;
12          $info[$rsa_server_key_attr][0] = $pubkey;
13          $info[$rsa_server_timeout][0] =
14              "schac:userStatus:us.es:timeout:" . $timeout;
15
16          // anadir la informacion al directorio
17          $r=ldap_modify($ds, $dn, $info);
18          $h = getdate($timeout);
19
20          echo '<p class="info">'._('Esta sesion de ssh es valida
21          hasta: ').$h["hours"].':'.$h["minutes"].':'.$h["seconds"].
22          ' - '.$h["mday"].' '.$h["month"].' '.$h["year"].'</p>';
23
24          return $r;
25      }
```

Esta función modifica una entrada ya existente en el directorio cambiando la clave pública y poniendo nuevamente el timeout.

- función `add`

```
1
2       function add($ds, $uid, $sn, $cn, $pubkey){
3               global $base_dn;
4               global $minutes_timeout;
5           global $rsa_server_key_attr;
6           global $rsa_server_timeout;
7               // preparar los datos
8               $timeout = $minutes_timeout * 60; //5 minutos
9               $hoy = getdate();
```

```
10              $timeout = $hoy[0]+$timeout;
11              $dn = "uid=". $uid .",". $base_dn;
12              $info["objectClass"][0] = "person";
13              $info["objectClass"][1] = "ldapPublicKey";
14              $info["objectClass"][2] = "schacUserEntitlements";
15              $info["uid"] = $uid;
16          if($sn == '')
17              $sn = $uid;
18          if($cn == '')
19              $cn = $uid;
20              $info["sn"] = $sn;
21              $info["cn"] = $cn;
22
23              $info[$rsa_server_key_attr] = $pubkey;
24              $info[$rsa_server_timeout]
25               = "schac:userStatus:us.es:timeout:" . $timeout;
26
27              // anadir la informacion al directorio
28              $r=ldap_add($ds, $dn, $info);
29          $h = getdate($timeout);
30          echo '<p class="info">'._('Esta sesion de ssh es valida
31          hasta: ').$h["hours"].':'.$h["minutes"].':'.$h["seconds"].
32          ' - '.$h["mday"].' '.$h["month"].' '.$h["year"].'</p>';
33              return $r;
34      }
```

Añade una nueva entrada en el servidor de claves.

- función `get_remote_user`

```
1
2       function get_remote_user(){
3               $cad = $_SERVER["REMOTE_USER"];
4
5               //$cad = $_SERVER["REMOTE_USER"];
6               //partiendo el nombre, para crear danigm-us
7               //a partir de danigm@us.es
8               $cad = $_SESSION['user'];
9               $nado = explode('@', $cad);
10              $name = $nado[0];
11              $dominio = substr($nado[1], 0, stripos($nado[1], '.'));
12              $name = $name.'-'.$dominio;
13              return $name;
14      }
```

Devuelve el uid del usuario que se utilizará como nombre de usuario para el acceso por SSH. Como cada uid no tiene por qué ser único, pues to que en diferentes organizaciones pueden existir usuarios con el mismo id, se concatena el dominio del mismo para poder diferenciar a usuarios

con el mismo uid, pero pertenecientes a diferentes organizaciones. Se ha optado por utilizar el caracter '-' como separador, puesto que si se utiliza la '@', complicaría un poco el acceso por ssh.

- función `get_certificate`

```
1
2     function get_certificate(){
3         global $shib_header;
4             $certificate = "";
5         if (isset($_POST['key'])) {
6             // Public key was not received from IdP.
7             // First check if user is posting its public key
8             $certificate = $_POST['key'];
9             }
10            // Check if userCertificate attribute is
11            //set in SAML response
12            else if (isset($_SERVER[$shib_header])) {
13                    $certificate = $_SERVER[$shib_header];
14                    $certificate = base64_decode($certificate);
15            }
16            // Trim certificate string
17            $certificate = trim($certificate);
18            $certificate = str_replace("\r","",$certificate);
19            $certificate = str_replace("\n","",$certificate);
20
21            return $certificate;
22    }
```

Intenta conseguir la clave pública del usuario a través de las cabeceras que introduce el SP de shibboleth a través de los datos que le manda el IdP.

- función `check_certificate`

```
1
2     function check_certificate($certificate){
3         // Check if certificate syntax is correct
4         if (substr($certificate, 0, 7) == "ssh-rsa" ||
5            substr($certificate, 0, 7) == "ssh-dss")
6                 return true;
7         else
8                 return false;
9     }
```

Comprueba si un certificado es valido.

- función `get_certificate_used`

```
1
2     function get_certificate_used($uid){
3         global $rsa_server_key_attr;
4         $timestamp = get_attr($uid, $rsa_server_timeout);
5         $timestamp = split(":", $timestamp);
6         $timestamp = $timestamp[count($timestamp)-1];
7         $now = getdate();
8         if ($now > $timestamp)
9         return get_attr($uid, $rsa_server_key_attr);
10        else
11        return "";
12    }
```

Mira en el servicio de directorio para un usuario ya existente cuál es la última clave pública que se utilizó.

- función `get_attr`

```
1
2     function get_attr($uid, $attr){
3             global $base_dn;
4             global $servidor_ldap;
5             global $puerto_ldap;
6             global $bn, $pw;
7
8             //Conectando con el ldap
9             $ds=ldap_connect($servidor_ldap, $puerto_ldap)
10                or die("No ha sido posible conectarse al
11                  servidor ".$servidor_ldap."");
12            //Version del protocolo que vamos a usar
13            ldap_set_option($ds, LDAP_OPT_PROTOCOL_VERSION, 3);
14            //Bind como usuario, vamos a buscar, para ver si ya esta
15            ldap_bind($ds, $bn, $pw) or
16                die("No ha sido posible enlazar con el servidor".
17                $servidor_ldap." con el usuario ".$bn."");
18
19            //se guardan por uid, por lo que filtramos por este campo
20            $filter = '(uid='.$uid.')';
21            $resource = ldap_search($ds, $base_dn, $filter);
22            $info = ldap_get_entries($ds, $resource);
23        ldap_unbind($ds);
24        if ($info["count"] == 0){
25            return null;
26        }
27        else{
28            return $info[0][$attr][0];
29        }
30
31    }
```

Busca en el servicio de directorio para un usuario existente un atributo.

- función `doit`

```
1
2     /**
3      * Esta funcion mira en el servidor de claves,
4      * si este usuario esta ya
5      * si esta modifica la clave, y el timeout
6      * si no esta lo anade
7     **/
8     function doit($uid, $pubkey){
9             global $base_dn;
10            global $servidor_ldap;
11            global $puerto_ldap;
12            global $bn, $pw;
13
14            //Conectando con el ldap
15            $ds=ldap_connect($servidor_ldap, $puerto_ldap)
16             or die("No ha sido posible conectarse al servidor
17             ".$servidor_ldap."");
18
19            //Version del protocolo que vamos a usar
20            ldap_set_option($ds, LDAP_OPT_PROTOCOL_VERSION, 3);
21            //Bind como usuario, vamos a buscar, para ver si ya esta
22            ldap_bind($ds, $bn, $pw)
23            or die("No ha sido posible enlazar con el servidor
24             ".$servidor_ldap." con el usuario ".$bn."");
25
26            //se guardan por uid, por lo que filtramos por este campo
27            $filter = '(uid='.$uid.')';
28            $resource = ldap_search($ds, $base_dn, $filter);
29            $info = ldap_get_entries($ds, $resource);
30
31            if ($info["count"] > 0){
32                    //solo hay que anadir un pubkey, si es distinto
33                    //y modificar el timeout.
34                    $response = modify($ds, $uid, $pubkey);
35            }else {
36                    //nueva entrada
37                    $cn = $_SERVER["HTTP_SHIB_PERSON_COMMONNAME"];
38                    $sn = $_SERVER["HTTP_SHIB_PERSON_SURNAME"];
39                    $response = add($ds, $uid, $sn, $cn, $pubkey);
40            }
41
42            ldap_unbind($ds);
43            return $response;
44    }
```

Mira en el servidor de claves si este usuario está ya. Si está modifica la

clave y el timeout. Si no está lo añade.

- función `anadir_usuario`

```
1
2     /**
3      * Codigos de error:
4      * -1 certificado no valido
5      * -2 No se ha podido completar la operacion
6      * -3 El certificado esta en blanco, no lo ha establecido el idp
7      *  1 Todo bien
8     **/
9     function anadir_usuario(){
10            // If $certificate is set, public key has been
11            // successfully received
12            $certificate = get_certificate();
13            if ($certificate != "") {
14                    // Check if certificate syntax is correct
15                    if(check_certificate($certificate)) {
16                            $name = get_remote_user();
17                            $response = doit($name, $certificate);
18                            // Check if command executed successfully
19                            if($response)
20                                    return 1;
21                            else
22                                    return -2;
23                    } else {
24                            return -1;
25                    }
26            }
27            else return -3;
```

Esta función intenta conseguir el certificado pasado por el formulario de la página, si no lo consigue intenta conseguirlo mirando los atributos pasados por el IdP.

Una vez que tiene el certificado comprueba su validez y, si es correcto, añade al usuario en el servidor de claves.

Por otro lado está la parte que muestra la información, `ssh.php`, cuyo funcionamiento es muy simple y sólo hace llamadas al `ssh_backend.php`.

La aplicación de ejemplo de creación de cuentas es una simple aplicación en php que está basada en la anterior, por lo tanto comparte la mayor parte del estilo y algo del código.

Sin embargo, dado que es una aplicación de ejemplo y su funcionamiento en teoría es muy simple, no se ha separado en varios ficheros, sino que toda la aplicación está sobre un único fichero `useradd.php`.

Esta aplicación hace uso de llamadas al sistema para realizar comandos remotos en los servidores SSH registrados. Para que esto funcione es necesario que el servidor donde se ejecute esta aplicación tenga las claves públicas de los demás servidores, puesto que se van a hacer llamadas a comandos remotos a través de SSH y esto debería hacerse de manera automática.

Veamos con más detalle cada una de las funciones que componen la pequeña aplicación de ejemplo.

- función `check_user`

```
1
2 function check_user($uid, $servidor){
3         if ($servidor == '')
4                 $servidor = "federacion21.us.es";
5
6         exec("sudo ssh root@".$servidor." id ".$uid, $array, $retval);
7         return $retval == 0;
8 }
```

  Comprueba si un usuario tiene una cuenta ya creada en un servidor determinado. Para ello utiliza una llamada al sistema con el comando `ssh` hacia el servidor remoto, para ejecutar el comando `id`.

- función `useradd`

```
1
2 function useradd($uid, $servidor){
3         if ($servidor == '')
4                 $servidor = "federacion21.us.es";
5
6         $val = system("sudo ssh root@".$servidor."
7       useradd -m -s /bin/bash -p xx -d /home/".$uid." ".$uid, $retval);
8         return $retval;
9 }
```

  Crea una cuenta para un usuario en un servidor remoto. Para ello, al igual que la función anterior utiliza el comando `ssh` para ejecutar el comando `useradd` en el servidor remoto.

El resto de la aplicación es trivial, y no tiene importancia comentarlo.

4.2.1. Internacionalización

Puesto que estas aplicaciones son la parte que va a ver el usuario se ha utilizado un sistema de internacionalización, `gettext`, para facilitar la traducción de las mismas. Así pues sería muy simple traducir estas aplicaciones y ofrecerlas en diferentes idiomas, según el idioma en el que esté configurado el navegador del usuario.

De momento y para este proyecto se ha realizado la versión en castellano y una traducción al inglés. Realizar cualquier otra traducción sería tan simple como modificar un fichero de texto plano donde aparecen las cadenas en el idioma original y habría que introducir las cadenas traducidas.

4.3. Necesidades para montar la plataforma

Este proyecto se ha realizado con un software base determinado puesto que depende de muchos componentes. En este apartado explicaremos cómo montar el proyecto desde cero y tener una versión funcional.

Como este proyecto depende de una federación de identidad supondremos que pertenecemos a una organización de una federación, y por lo tanto tendremos diferentes SPs montados.

Todo el código, el parche y las dos aplicaciones en php, se pueden conseguir en la forja de rediris, en el proyecto AUPAAI [Forja].

Para conseguir los fuentes del proyecto se debe utilizar el sistema de control de versiones subversion [Subversion]. Con el sistema de control de versiones se conseguiría una copia del código fuente más reciente. Para poder hacer uso de estos comandos es necesario tener instalado "Subversion".

Se utiliza este sistema de control de versiones porque es el que ofrece la forja de software que ofrece RedIRIS para alojar proyectos.

Este proyecto está englobado dentro del proyecto aupaai que contiene más software para federación, por lo tanto sólo nos interesa en este caso el directorio fed+ssh dentro del repositorio.

```
svn checkout https://forja.rediris.es/svn/aupaai
cd aupaai/fed+ssh/
```

4.3.1. Instalando el servidor SSH parcheado

Para poder aplicar el parche, es necesario bajarse el código fuente de openssh, `http://www.openssh.org/portable.html`. Utiliza CVS como sistema de control de versiones, por lo tanto, si queremos descargar la última versión de desarrollo tendremos que tener instalado esta aplicación y se descargaría de la siguierte manera:

```
export CVSROOT=anoncvs@anoncvs.mindrot.org:/cvs
export CVS_RSH=/usr/bin/ssh
cvs get openssh
```

Una vez hecho esto el siguiente paso sería copiar el fichero .patch dentro del directio donde vayamos a compilar el servidor SSH y aplicar el parche.

Para esto se utiliza el comando patch al que se le pasa un fichero de diferencias y modifica los fuentes con los cambios necesarios.

```
cd openssh
cp ../aupaai/fed+ssh/parche-openssh.patch .
patch -p0 < parche-openssh.patch
```

Por otra parte, para simplificar la instalación y dado que la licencia BSD lo permite, está disponible el código parcheado junto con las aplicaciones

en la forja de rediris `http://forja.rediris.es/frs/download.php/775/openssh-federado.tar.gz` de donde se puede descargar de manera más simple el código del servidor ya parcheado y listo para su compilación.

Si todo ha ido bien, es hora de compilar. Antes de nada es necesario tener instalado el compilador y todas las dependencias de openssh además de las librerías openldap, ya que el parche hace uso de las mismas para comunicarse con el servidor de claves. Así pues se deben ejecutar los siguientes comandos.

```
./configure --prefix=/ruta/de/instalacion
make
make install
```

Openssh utiliza las herramientas de compilación autotools, que son un conjuto de herramientas que facilitan la compilación de aplicaciones en diferentes arquitecturas y sistemas. Por tanto es necesario tener instaladas estas herramientas, tanto `configure` como `make` son herramientas que facilitan la compilación.

Una vez compilado es hora de configurarlo para que funcione con nuestro sistema, cambiando el fichero `sshd_config`

```
# External RSA key
fedserver host.servidorclaves.com
fedport 389
usefed yes
fedserver_root_dn "cn=admin,dc=us,dc=es"
fedserver_root_pw password
fedserver_base "o=People,dc=us,dc=es"
fedserver_attr sshPublicKey
fedserver_timeattr schacUserStatus
```

Y con esto ya estaría el servidor SSH parcheado listo para funcionar de manera federada. Ahora sólo sería necesario crear las cuentas de los usuarios, que se puede hacer de forma dinámica o estática y sólo tendrán acceso cuando se autentiquen frente a la federación.

4.3.2. Cómo montar el SP y la aplicación web

La aplicación web de autenticación está hecha pensando en el SP de shibboleth 1.3. Por lo tanto será necesario tener un SP de shibboleth sobre un servidor web apache.

Para funcionar la aplicación requiere que estén instaladas las librerias de acceso a LDAP para php, además por supuesto del modulo de php para el servidor web correspondiente, puesto que se va a comunicar con el servidor de claves que en este caso será un servicio de directorio.

Sobre un directorio protegido por el proveedor de servicio se despliega el directorio `ssh`, y ya estaría la aplicación de autenticación instalada en la federación.

Cualquier usuario que quisiera acceder a algún servidor SSH federado tendrá que entrar en esta página, de tal forma que escriba el registro correspondiente en el servidor de claves y así el usuario tenga acceso a todos los servidores SSH federados.

La aplicación es configurable así que habrá que cambiar unas variables en el fichero `ssh_backend.php`

```
$base_dn ='o=People,dc=us,dc=es';
$servidor_ldap = "goonie.us.es";
$puerto_ldap = 389;
$bn = 'cn=admin,dc=us,dc=es';
$pw = 'xxxx';
$minutes_timeout = 30;
$shib_header = "HTTP_USERCERTIFICATE";
$rsa_server_key_attr = 'sshpublickey';
```

```
$rsa_server_timeout = 'schacuserstatus';
```

Con estas opciones se pueden configurar el servidor de claves, el tiempo en minutos que va a durar una sesión de un usuario autenticado, la cabecera de shibboleth que mirará la aplicación para intentar conseguir la clave pública y los atributos en el servidor de claves para almacenar la clave pública y el tiempo de sesión.

4.3.3. Cómo instalar el servidor de claves (openldap)

La última parte que hace falta para tener montado todo el proyecto es el servidor de claves.

Puesto que nos hemos decidido por un servicio de directorio aquí es muy fácil la instalación, es más, hay diferentes opciones comerciales y no comerciales. Y dado que los atributos a utilizar para almacenar los datos son variables tanto en el servidor SSH parcheado como en la aplicación de autenticación, no es obligatorio instalar ningún esquema, aunque sí recomendable.

Explicaremos aquí cómo instalar un servidor de claves en una distribución tipo Debian y aplicarle los esquemas usados.

Lo primero es instalar el servicio de directorio.

```
apt-get install slapd
```

La configuración está en el directorio /etc/ldap/ y los esquemas para openldap hay que meterlos en /etc/ldap/schemas/.

Descarga de los esquemas necesarios:

Esquema schac: `http://www.rediris.es/ldap/esquemas/schac/schac-20061017-1.3.0b2.schema.txt`

Esquema openssh-lpk: `http://dev.inversepath.com/openssh-lpk/openssh-lpk_openldap.schema`

Se ha de mover los esquemas al directorio /etc/ldap/schemas/ y posteriormente modificar el fichero `/etc/ldap/slapd.conf` añadiendole las lineas

```
include         /etc/ldap/schema/openssh-lpk_openldap.schema
include         /etc/ldap/schema/schac-20061017-1.3.0b2.schema
```

4.3.4. Todo en conjunto

Una vez que ya están todas las partes instaladas, hay que configurar tanto los servidores ssh parcheados como la aplicación de autenticación para que se comuniquen con el servidor de claves públicas.

El despliegue de este proyecto requiere de diferentes partes que tienen sus problemas propios de despliegue. Así pues es importante no introducir demasiados requisitos para el despliegue de nuestras herramientas.

En realidad el servicio es un conjunto de pequeñas herramientas que hacen bien su trabajo y que se comunican entre sí para dar un servicio conjunto. Esto requiere configurar cada parte del proyecto para que se comunique correctamente con todas las demás partes.

También influye enormemente el que sea un sistema distribuido, es decir, que cada servicio pueda estar en una máquina diferente y en una subred diferente. Por lo tanto existe otro factor de complejidad que es introducido por la complejidad de los sistemas distribuidos en red.

Tanto las aplicaciones web en PHP como los servidores SSH parcheados forman parte de la federación de identidad, puesto que comparten información de identidad a través de diferentes servidores de diferentes organizaciones

basándose en un sistema de confianza.

Capítulo 5

Despliegue de una maqueta en CONFIA

Para la prueba del proyecto, se ha utilizado la federación CONFIA ([CONFIA]) montada en fase de pruebas. CONFIA es la federación de identidad de las universidades andaluzas. Es un proyecto pionero en España que está comenzando ahora. Su objetivo principal es montar una infraestructura de identidad federada a nivel andaluz con todas las universidades andaluzas.

Esta federación no está aún en producción y utiliza usuarios de prueba, pero es prácticamente funcional. Actualmente utiliza el protocolo Shibboleth 1.3, y hay varios IdPs (Universidad de Sevilla, Universidad de Málaga, Universidad de Córdoba, etc), y varios SPs protegiendo diferentes aplicaciones en diferentes universidades.

Además existe una infraestructura de metadatos y un WAYF global que están en el CICA.

Así pues, el proyecto se puede desplegar en este entorno, y en este entorno es en el que se ha desarrollado, sobretodo en la parte de la Universidad de Sevilla.

Por parte de la Universidad de Sevilla, hay un IdP de Shibboleth sobre un servidor Tomcat que utiliza como sistema de SSO el SUN Access Manager.

En la máquina `federacion21.us.es` hay un servidor Apache2 protegido tras un SP de shibboleth.

5.1. Máquinas involucradas

Hay varios servidores SSH parcheados funcionando dentro de la federación a modo de ejemplo. Los servidores SSH parcheados se han colocado en el puerto 2222 para no interferir con los servidores SSH por defecto de cada máquina.

- `federacion21.us.es`, sistema operativo Ubuntu.
- `federacion22.us.es`, sistema operativo Ubuntu.
- `goonie.us.es`, sistema operativo debian. Máquina en la cuál se ha desarrollado todo el proyecto.

Tanto federacion21 como federacion22 son máquinas virtuales que corren sobre un entorno de virtualización XEN. Así pues el despliegue de pruebas no pone en peligro ninguna infraestructura delicada puesto que estos servidores no contienen información sensible ni prestan ningún servicio de cara al usuario final.

Las aplicaciones web necesarias para la autenticación y para la creación de cuentas están desplegadas en el servidor `federacion21.us.es`, tras el servidor web apache2, y protegidas con un SP.

El servidor de claves es un servidor openldap, instalado en `goonie.us.es`, que almacena las claves públicas, los nombres de usuarios y el tiempo de sesión que luego consultan los servidores SSH parcheados.

Dada esta infraestructura de pruebas implementada hoy en día es posible acceder a estos servidores SSH parcheados con cualquier usuario de la Universidad de Málaga, que se autentique en la federación, o de la Universidad de Córdoba, etc.

5.2. Pruebas

Se han realizado pruebas de acceso a diferentes servidores SSH (tanto a federacion21 como a federacion22) con usuarios de la federación.

En principio las pruebas se han realizado con usuarios de la misma Universidad de Sevilla. Puesto que el proveedor de identidad de la Universidad trabaja con una maqueta que aprovisiona los usuarios no se trata de usuarios reales, en producción, sino que son ficticios aunque basados en datos reales.

Por otra parte, una vez el proyecto se consideró lo suficientemente maduro se publicó el sistema a toda la federación y diferentes universidades desde sus propias instalaciones han probado y verificado el correcto funcionamiento del proyecto.

Así pues se puede decir que se han realizado varias pruebas, con usuarios ficticios, y reales, aunque aún quedarían una serie de pruebas más estrictas con mayor número de usuarios y un nivel de concurrencia alto para comprobar qué carga soporta y comprobar que no ocurre ningún fallo no previsto.

5.3. Futuros despliegues

- El proyecto se ha desarrollado en paralelo con la federación de identidad de las universidades andaluzas y en colaboración con CONFIA, por lo que es posible que cuando la federación se implante alguna Universidad quiera dar este servicio. Por esa parte el proyecto tiene futuro dentro de las universidades andaluzas.
- El proyecto está teniendo difusión a nivel internacional, ha sido presentado en foros europeos donde se ha mostrado interés en este proyecto por parte de diferentes federaciones. Por lo tanto es posible que en un futuro alguna otra federación se anime a implantar este proyecto en su sistema.
- El supercomputador más potente de España, llamado "MareNostrum", está situado en Barcelona. Actualmente se está dando acceso a diferentes universidades a partes de este supercomputador, para aprovechar

su capacidad de cálculo en la investigación universitaria. Este caso es para el que se ha desarrollado este proyecto.

- Además de para federaciones, este proyecto se puede utilizar en organizaciones o empresas que no pertenezcan a ninguna federación. Sería tan fácil como modificar el código de la aplicación php, y hacer que consiguiera los datos de otro lugar, ya sea una base de datos, un servicio de directorio, etc. Por esto podría ser interesante para cualquier empresa que tenga que lidiar con una gran cantidad de servidores, el no tener que cambiar contraseñas cada poco tiempo, ni tener que crear usuarios nuevos en cada máquina.

Capítulo 6

Conclusiones

Este proyecto ha supuesto para mi una experiencia nueva puesto que no se trata de un desarrollo propio sino que extender algo ya existente para adaptarlo a unas necesidades nuevas. También este proyecto consiste en la unión de diferentes herramientas para que funcionen de tal manera que ofrezcan un servicio conjunto.

Además he presenciado cómo ha nacido y cómo se está creando una federación de identidad desde dentro, a todos los niveles, desde el nivel técnico al nivel institucional.

Al principio puede parecer un proyecto muy complejo, puesto que no se conoce el código del servidor SSH, hay muchas herramientas involucradas, la federación de identidad es todavía una tecnología poco documentada y realizar una adaptación de algo tan grande puede resultar abrumador.

Con respecto al servidor SSH he de decir que gracias a la claridad del código, y a la separación del mismo, he conseguido realizar la modificación sin demasiadas complicaciones. Es en este momento cuando se comprende realmente la importancia de la modularidad y la claridad a la hora de escribir buen código fuente.

Puedo concluir de esta experiencia que las licencias libres ayudan a la innovación y aprendizaje ya que te puedes basar en el código y conocimiento de otras personas para mejorar y realizar tus proyectos, además de poder

distribuir tu trabajo sin problemas legales de ningún tipo.

Hoy en día cualquier organización maneja una gran cantidad de máquinas y la mayoría se manejarán de manera remota. Para el acceso remoto lo más rápido y sencillo es utilizar el protocolo SSH ya que proporciona seguridad y flexibilidad.

Por otra parte, cada vez más se está imponiendo la federación de identidad, tanto en el ámbito universitario como en el empresarial, por lo tanto la federación de identidad es una tecnología a tener en cuenta para el futuro próximo.

Este proyecto hace que la gestión de máquinas de manera remota se simplifique al utilizar los conceptos de la federación de identidad en el acceso por SSH, por lo tanto simplifica la gestión de multitud de máquinas de igual manera que la federación de identidad simplifica la gestión de los usuarios a las aplicaciones web.

Con esta idea de gestión de identidad para el acceso SSH el permitir o prohibir el acceso a un servicio SSH es tan simple como visitar una página web.

Además este proyecto abre la puerta para la federación de identidad en aplicaciones que no sean web. Este proyecto es una prueba de que es posible utilizar los mecanismos de federación de identidad para autenticar a usuarios en una aplicación que utiliza un navegador como medio.

El resultado de este proyecto puede resultar una aportación relevante al uso de la federación de identidad que a día de hoy se está popularizando cada vez más, puesto que cada vez hay más usuarios y la federación de identidad simplifica la gestión de identidad de los mismos.

Capítulo 7

Anexos

7.1. Descripción enviada a un grupo de trabajo

Tratando de que el proyecto se conozca fuera de la federación andaluza se ha redactado una pequeña descripción del proyecto en inglés para enviarla a un grupo de trabajo que trata sobre estos temas.

> Using Federation credentials to SSH login
>
> This project is based on the Feide paper available at: `http://rnd.feide.no/content/feide-and-ssh-secure-shell`
>
> Problem:
>
> When we have lots of SSH accounts, we need to remember lots of passwords, and these passwords are stored by each individual SSH server.
>
> Because of that, whenever we want to change passwords, it is required to change it in all servers. Furthermore, if one of those SSH servers is hacked our password at many others may be at risk.
>
> Other problem related to password management in SSH deployments arises when you want to give SSH service to many users,

complicating the management of that many users accounts.

Identity federation can solve those problems, but it is currently designed for web-only scenarios. This document describes how we use identity federation credentials to solve those problems in SSH services.

Objective:

The objective of this project is to enable the use of identity federation credentials to authenticate in SSH. We have tried to make it as easy as possible for both the user and the admin of the service.

Solution:

To achieve this, we take the advantage of the possibility of using a public key system with SSH servers. By means of the use of trusted public keys there is not needed to write passwords.

The SSH server needs to get the user public key from the federation system, that works only for web applications. Therefore we put a common LDAP server. In that LDAP server we add an entry whenever the user logins to the federation, so the SSH server can verify whether the user is authenticated.

Our solution has two parts:

1.- SP application

One is a SP application made in PHP. This is an application global for the federation; only one instance is needed. The user enters to that web application, being redirected to their own IdP. Once the user is authenticated is redirected to that application.

The application expects to receive an uid and a ssh public key from the IdP, and writes an entry in the global LDAP server.

Furthermore, the application allows the user to write their ssh public key manually, covering the case of the IdP not having it or of the user not using the machine where their private key lives.

The application makes an entry in the LDAP server, containing the uid concatenated with the IdP name, something like username-IdP (danigm-us), the ssh public key, and a timeout value.

From the moment access to that web application is granted, the user can login to all the participating SSH servers automatically.

2.- Openssh patch

The ssh server works in this way: First, it tries to authenticate the user with a ssh public key, looking for keys in $HOME/.ssh/authorized_keys. If that works, it accepts the login without asking for a password.

Second it tries to authenticate with PAM modules if PAM has been activated.

At first, we thought of a PAM module but, because of the way in which the ssh server works, that method required the user to write a password or something similar, losing the automatic login.

The way we took is to modify the openssh server, taking advantage of its condition of free software.

We wrote a patch for this ssh server. The objective of this patch was to modify the least possible number of files, making it robust with respect to future evolutions of the openssh code.

The patch medifies the authenticate() method in the server, including a subroutine right after the segment in which the server looks for the public key in the local filesystem.

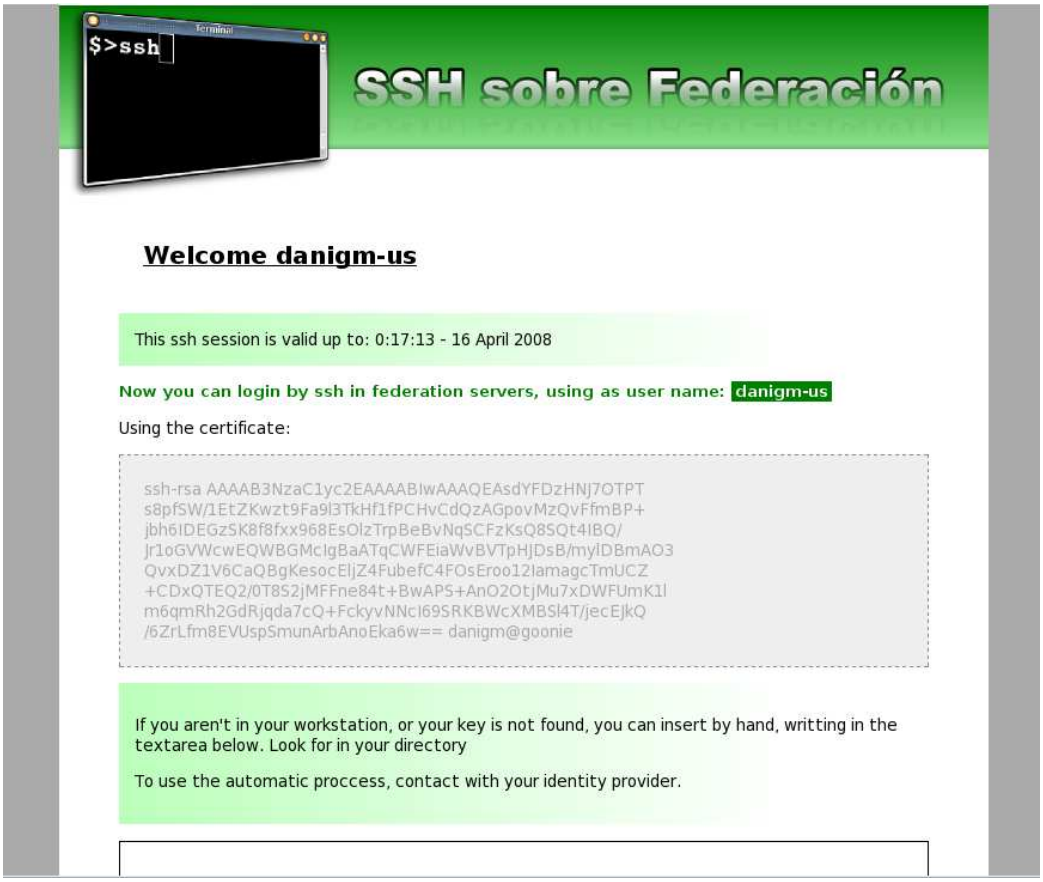
If the server does not find the user public key, that subroutine is called. It queries the user at the LDAP server that we mentioned before. If the user is authenticated the patched ssh server receives the public key and the timestamp that indicates the timeout of the login.

The subroutine checks if the time is correct and then creates a temporary file in the local filesystem with the rsa public key. After that, it checks again the user identity but now with the new temp file. If all goes well, the user is authenticated directly. Otherwise, the ssh server authentication method continues asking for the user password.

All the parameters needed by that patch could be set in the sshd.conf file.

Currently we are using and LDAP server as public key server, but the method could be able to easily support others query protocols. You only need to write a function that asks for the uid to the server and returns the ssh public key if it exists.

Images: [1] sshApp.png



[2] userAdd.png

Links: [3] `https://forja.rediris.es/projects/aupaai/`

7.2. Sobre el despliegue de la aplicación

Para una posible implantación del proyecto en un proyecto de RedIRIS, se redactó una pequeña guia que facilita el despliegue del mismo.

El despliegue completo consta de tres partes:

1.- Aplicación web php. Solo hace falta desplegarla en una máquina, que esté protegida por un SP, un SSO o algo así. Esta aplicación escribirá en el LDAP cada vez que un usuario se autentique.

Hay que configurar las siguientes variables en el fichero ssh_backend.php:

//base para la busqueda e introduccion de usuarios logueados en el LDAP $base_dn ='o=People,dc=us,dc=es'; $servidor_ldap = "goonie.us.es"; $puerto_ldap = 389; $bn = 'cn=admin,dc=us,dc=es'; $pw = 'xxxx';

//tiempo de sesion valido $minutes_timeout = 30;

//El atributo de shibboleth que mira para coger el certificado $shib_header = "HTTP_USERCERTIFICATE";

//atributo del LDAP para almacenar la clave $rsa_server_key_attr = 'sshpublickey';

//atributo del LDAP para el timeout, se guarda como un timestamp $rsa_server_timeout = 'schacuserstatus';

De momento está pensada para mirar los atributos de shibboleth, pero si no los recibe, creo que también funciona, solo que el certificado hay que meterlo a mano.

Mira el nombre de usuario de la variable $_SERVER[REMOTE_USER"] y debería ser un nombre@dominio.loquesea

2.- LDAP para las claves públicas de los usuarios que se han autenticado:

Un simple openldap bastaría. Hay que añadirle los esquemas openssh-lpk_openldap, para el campo "sshpublickeyz el schac para el campo "schacuserstatus", que lleva el timeout.

En este LDAP es dónde va a escribir la aplicación php, y de donde van a leer los diferentes servidores ssh parcheados.

3.- Servidor ssh parcheado:

Lo mando como adjunto, y solo haría falta compilarlo y cambiar la configuración del fichero sshd_config

Si se quiere compilar sobre el mismo directorio, si ya hay algún otro servidor ssh (lo más normal), se recomienda así. ./configure –prefix=$PWD && make y se lanzaría: $PWD/sshd -f sshd_config

Si se quiere compilar para el sistema: ./configure && make && make install y se lanzaría según el sistema.

Configuracion: el fichero sshd_config Para la federación se le han añadido una serie de opciones adicionales:

Opcion para activar el acceso federado usefed yes

Servidor de claves RSA, el LDAP fedserver goonie.us.es fedserver_root_dn çn=admin,dc=us,dc=es" fedserver_root_pw xxxxxx fedport 389

base de busqueda para usuarios en el LDAP fedserver_base .º=People,dc=us,dc=es"

Atributos de clave, y de timeout fedserver_attr sshPublicKey fedserver_timeattr schacUserStatus

Si hay otro servidor corriendo, también se puede cambiar el puerto con Port 2222

—

Con esto ya habría una versión funcional. Ahora hay que tener unas consideraciones previas.

1.- Creación de cuentas: Para que un usuario pueda acceder, además de autenticarse tiene que tener una cuenta creada en el sistema. Para eso hay otra aplicación, useradd.php, desde la cual podría hacerse de manera centralizada. Es decir, esta aplicación muestra una lista de los servidores ssh disponibles, y si tienes cuenta o no, además se puede solicitar una cuenta, y se crearía de manera inmediata.

La instalación sería igual que la de ssh, y habría que configurar los servidores en el index.php:

Por cada servidor una entrada en el array, direccion del servidor, y descripcion: $servers[0] = 'federacion21.us.es'; $desc[0] = "servidor1";

Para que esta aplicación funcione bien el servidor donde esté debe tener acceso por ssh a todos los servidores con cuenta de root. Para ello hay que añadir en el $HOME/.ssh/authorized_keys de cada servidor la clave pública de este servidor (ssh-keygen, $HOME/.ssh/id_rsa.pub) donde está la aplicación. Además para que tenga acceso la aplicación, si se está ejecutando con el usuario www-data, este usuario debería tener acceso a sudo para el comando ssh.

#visudo www-data ALL=NOPASSWD: /usr/bin/ssh

—-

Detalles a tener en cuenta:

Se debería bloquear la posibilidad del cambio de password de los usuarios, ya que si un usuario ejecuta passwd, podrá entrar en

un futuro sin loguearse. También sería necesario impedir que los usuarios escriban en el directorio $HOME/.ssh/ para evitar que metan alguna clave rsa, y poder tener acceso sin pasar por la federación.

Cómo solucionar esto: Para bloquear el cambio de password, basta con chmod 554 /usr/bin/passwd. Para impedir la escritura se puede poner una tara cron que elimine de cada usuario ese directorio cada cierto tiempo, o se le puede quitar el acceso al mismo con chmod 000 .ssh/ para cada usuario

Nada más. Pienso que el despliegue, aunque pueda parecer complejo no debería serlo tanto, por lo menos en debian y en ubuntu a mí no me ha costado casi nada.

Capítulo 8

Agradecimientos

8.1. Servicio SSH federado en la Universidad de Sevilla

La Universidad de Sevilla es una de las impulsoras del proyecto, siendo parte importante en él, y cobrando relevancia en las pruebas realizadas.

A este proyecto se le ha facilitado acceso a todas las máquinas necesarias, y se le ha dado permiso para implantar un entorno de prueba, permitiendo así probar la viabilidad del sistema en un entorno más o menos real, y con diferentes organizaciones implicadas.

De no ser así, habría sido imposible probar este proyecto, puesto que la infraestructura que hay detrás del proyecto implica tener muchas máquinas, en localizaciones diferentes, y con una base de usuarios de prueba.

Además de los medios materiales, me han permitido el asistir a diferentes reuniones de trabajo del proyecto CONFIA, lo que me ha permitido compartir opiniones y difundir este proyecto, más allá del ámbito puramente académico.

Bibliografía

[OpenSSH] http://openssh.org/

[Federación Noruega] http://rnd.feide.no/

[Openssh-lpk] http://dev.inversepath.com/trac/openssh-lpk

[SCHAC] http://www.terena.org/activities/tf-emc2/schac.html

[PAPI] http://papi.rediris.es

[WAYF] http://www.switch.ch/aai/support/tools/wayf.html

[Forja] https://forja.rediris.es/projects/aupaai/

[Subversion] http://subversion.tigris.org/

[CONFIA] http://confia.aupa.info/