Màster interuniversitari
de Seguretat de les tecnologies
de la informació
i de les comunicacions (MISTIC)
Proyecto de fin de Master

Implantación de un SSO (Single Sign On)

Autor: Juan Ignacio Martín (Nacho)
Director: Antoni Gonzalez Ciria
ANCERT (Agencia Notarial de Certificación)

UAB Universitat Autònoma de Barcelona

UOC Universitat Oberta de Catalunya

UIB

Dedicatorias

Con el permiso de los responsables y directores del Mistic y del trabajo de fin de máster, quiero dedicar este trabajo a mi mujer, por su comprensión y apoyo incondicional durante todo el master.

A mis hijas, Raquel y Cristina, que lo son todo en mi vida.

A mis padres, que gracias a ellos soy lo que soy.

También quiero agradecer a Antoni Gonzalez, director del presente trabajo, por las sabias aportaciones y su mirada crítica que ha sabido orientar en todo momento, el desarrollo del proyecto.

No quiero olvidarme de la UOC, en general, que ha hecho posible, con su método, didáctica y bien hacer, que yo pudiera culminar con éxito el Mistic.

Quiero también, agradecer a Jordi Martín, Gerente de Trace Logistics, S.A. por facilitar el hardware necesario para el desarrollo del proyecto.

Resumen

Este trabajo, con el que se concluye el Mistic (Master Interuniversitario de Seguridad en las TIC), se enmarca dentro de los mecanismos de identidad digital, y concretamente en los sistemas de autenticación única (Single Sign On).

En él se describen todos aquellos elementos necesarios para comprender su funcionamiento, entrando en detalle en aquellos más relevantes, especialmente en los que permiten que el usuario evite realizar nuevas identificaciones/autenticaciones en un entorno de múltiples aplicaciones J2EE.

Para ello, se centra en el mecanismo existente para el mantenimiento de las sesiones de usuario entre las diferentes solicitudes HTTP, las cookies, y las extiende para que también sean válidas entre las diferentes aplicaciones del entorno Single Sign On.

Explica el mecanismo a través del cual, el sistema Single Sign On, es capaz de gestionar los accesos a las aplicaciones J2EE, autorizando y/o denegando cuando proceda, e identificando en todo momento al usuario que accede a través de la gestión de su cookie de sesión.

En la segunda parte, y tomando como referencia el producto OpenAM (Open Access Management), se describe todo el proceso de diseño de una arquitectura base, partiendo desde cero, que permita desplegar un conjunto de aplicaciones J2EE para, acto seguido, protegerlas mediante este producto.

En esta parte, se describe la instalación y configuración de OpenAM, indicando todos los elementos necesarios, tales como elementos de red, servicios de directorio LDAP, balanceadores, proxies, DNS, servicio de aplicaciones J2EE a proteger, etc... describiendo en profundidad, en la sección de anexos, la instalación los elementos necesarios pero no relacionados directamente con el Single Sign On.

También se detalla todo lo necesario para integrar el citado servidor de aplicaciones J2EE (en nuestro caso basado en Tomcat) a través del agente de políticas, dentro del entorno Single Sign On.

Tomando como ejemplo una aplicación J2EE sencilla (proporcionada por el propio paquete OpenAM) se realiza su integración completa en el entorno Single Sign On, creando las políticas de acceso necesarias para su correcto funcionamiento.

Se repite el proceso con una aplicación no preparada para entornos Single Sign On, de forma que se puede observar las diferencias entre una y otra aplicación y la respuesta dada en entornos de este tipo.

Y para ilustrar todo el proceso de funcionamiento del sistema Single Sing On, se incluye un anexo en el que se realiza un estudio sobre las cabeceras HTTP intercambiadas entre el navegador web del usuario, el sistema Single Sign On y el servidor de aplicaciones J2EE, de forma que pueda verse la manera en la que el sistema gestiona los accesos a los recursos protegidos por el Single Sign On.

En los últimos apartados del documento, y debido a la falta de tiempo, se describen, por un lado, la base teórica necesaria para implementar un sistema de autenticación de usuarios basado en certificados x.509, y nombra los elementos de configuración que deberían trabajarse en OpenAM para implementar este mecanismo de autenticación, y por otro, el método que sigue OpenAM para gestionar el escalado de seguridad.

Y por último, se menciona la forma en la que OpenAM puede intercambiar información con la aplicación J2EE protegida sobre los usuarios que acceden, de forma que la misma pueda presentar, según convenga, esta información al usuario.

Abstract

This work , which concludes Mistic (Interuniversity Master of Security in TICs) , is part of the digital identity area, and specifically in systems Single Sign On.

In it, the elements are described to understand how it works, going into detail with the most relevant, especially those that allow the user to avoid making new Logons, in an environment with multiple J2EE applications.

To this end, it focuses on the mechanism to maintain user sessions in HTTP systems, cookies, and extends it to be valid also between different J2EE applications on Single Sign On environment.

Explain the mechanism through which the Single Sign On system is able to manage access to the J2EE applications, allowing and/or denying when necessary, and always identifying the user through its session cookie.

In the second part, using OpenAM (Open Access Management) as a reference, we describe, from the beginning, the design of the system architecture required to deploy a set of J2EE applications, then we protecting it with this product.

In this part, describes installation and configuration process of OpenAM, indicating all necessary elements, such as network , LDAP directory services, load balancers, proxies, name services, J2EE application servers, and other elements ... (all described in depth, in Annexes, which decribed how install the necessary elements but not directly related to the Single Sign On system).

Everything necessary for integrating J2EE application server based on Tomcat, through the policy agent within the Single Sign On environment are also explained.

Using the example of a simple J2EE application (provided by OpenAM) full integration is performed in the Single Sign On environment, creating access policies necessary for proper operation.

Also, an non designed for Single Sign On environment application is deployed, where we can see the differences between each application and the response in this type of environment.

And to illustrate the whole process of Single Sign On system operation, I annexed a document in which we study how work the HTTP headers exchange between the user's web browser, the Single Sign On system and the J2EE application server, so you can see how the Single Sign On system handles the accesses to protected resources.

In the final sections of this document, due to lack of time, only describes the therory of everything necessary to implement a user authentication system based on X.509 certificates, and quote all configuration items that should be changed in OpenAM to implement this authentication mechanism, and secondly, goes on to explain how the scaling security process works.

And finally, explains how you can get OpenAM send information about users to applications, so it can show that information in them.

Índice de contenido

Introducción

En un mundo tecnológicamente cada vez más complejo, la seguridad es parte primordial, y su correcta gestión obliga al diseño de entornos que faciliten su uso a un número cada vez más elevado de usuarios, sin que ello provoque una pérdida de seguridad, confidencialidad o disponibilidad (tres pilares básicos de un entorno IT seguro).

Es vital, por tanto, que la identificación de los usuarios sea adecuada, y que la autorización de acceso a cada aplicación se corresponda con el rol asignado a cada usuario.

La aparición de entornos SSO combinado con otros sistemas, como por ejemplo, los Identity Managers, etc..., facilitan la gestión de grandes volúmenes de usuarios sin pérdida de eficacia, y seguridad.

En los siguientes apartados se describe en qué consiste, conceptualmente, un sistema Single Sign On (SSO), qué pretende alcanzarse con su uso y sus ventajas e inconvenientes.

Posteriormente se realizará la descripción de un entorno clásico de aplicaciones J2EE, para, acto seguido, aplicar a este entorno, un mecanismo de identificación de usuarios basado en SSO mediante el producto OpenAM.

Justificación

En entornos productivos, en el que las aplicaciones J2EE se distribuyen a través de uno o varios servidores de aplicaciones, ha de tenerse en cuenta las premisas propias de la seguridad del negocio, esto es, la garantía de que el entorno cumple con los principios de integridad, disponibilidad y confidencialidad.

La integridad viene dada por un diseño adecuado de las aplicaciones adaptándose a los parámetros de desarrollo seguro.

La disponibilidad, en entornos de servicio de aplicaciones J2EE o de entorno web, se incrementa (o se reduce el riesgo de falta de la misma) introduciendo mecanismos de redundancia y protección de forma similar a la de cualquier servicio IT.

En cuanto a la confidencialidad, podemos destacar que el acceso a aplicaciones ha de quedar protegido por las medidas de identificación y autorización adecuadas (definidas en las políticas de seguridad de cada organización), esto es, que el acceso a la información sea realizada por el personal autorizado en cada momento.

Para ello, en entornos globalizados (Internet), los mecanismos de autenticación y autorización clásicos (usuario y contraseña, ACLs, etc...) han demostrado ser poco seguros o inapropiados, más aún cuando cada aplicación puede disponer de sus propios sistemas de autenticación, que hacen aún más compleja la tarea de gestionar las identidades y las autorizaciones, creciendo exponencialmente en dificultad según va incrementándose el número de usuarios, pudiendo llegar a ser totalmente ineficaz cualquier medida de seguridad de acceso a las aplicaciones que se tome, comprometiendo, en consecuencia, la confidencialidad de la información.

Por tanto, se desprende, de todo lo anterior, que hay que tener en cuenta en un entorno seguro de aplicaciones Web, especialmente en aquellas que deben soportar grandes volúmenes de usuarios, los siguientes aspectos:

- La información debe estar protegida mediante mecanismos redundantes, de forma que reduzcan el riesgo de caída de las aplicaciones.
- El entorno debe ser robusto y seguro, de tal forma que se reduzca la

fatiga del proceso de autenticación.
- Debe facilitar la gestión de identidades, incrementando la seguridad incluso frente a grandes volúmenes de usuarios, haciendo especialmente recomendable la utilización de sistemas de identidad unificados.

A partir de aquí, las respuestas que hemos de dar a cada aspecto son:

- Redundancia hardware y software: la disposición de varios elementos hardware que sirvan la misma aplicación permiten tolerar fallos, a esto le añadiremos que el software utilizado permite que las aplicaciones puedan distribuirse a través de estos elementos hardware, reduciendo los riesgos asociados a la falta de disponibilidad vinculados a fallos hardware/software, para ello implementaremos elementos como:
  - Clusteres de servidores de aplicación.
  - Mecanismos balanceadores de carga que equilibre el acceso a los nodos del cluster.
  - Elementos comunes de persistencia, también asegurados, que permitan que cualquiera de los nodos del cluster puede seguir teniendo acceso a la información (de usuarios y de aplicaciones).
- Sistema integrado/unificado de autenticación de usuarios: para todas las aplicaciones, se utilizará un sistema único que permita, por un lado, una identificación global de los usuarios en todas las aplicaciones del entorno, y por otro, un mecanismo unificado de gestión de identidades (altas, bajas y modificaciones) que la mejora y facilita sin perder seguridad y eficacia. Para ello emplearemos:
  - Un Sistema SSO, que al ser crítico, también deberá disponer de sistemas de redundancia (clusterización).
  - Un sistema de balanceo de carga para el sistema SSO que evitará que sobrecargas o fallos en el sistema impidan el acceso a las aplicaciones.
  - Un repositorio de identidades, por ejemplo, en un directorio LDAP, que centraliza la persistencia de las credenciales de usuarios para todas las aplicaciones en un punto único, gestionado por el propio entorno SSO.

Objetivo

En este proyecto, por tanto, se describe todo el proceso de instalación y configuración, tanto de los elementos físicos y lógicos requeridos para el desarrollo del proyecto, creando un escenario básico como punto de partida para poder, acto seguido, realizar todo el proceso de securización de las aplicaciones, para que hagan uso del sistema SSO en el proceso de autenticación y autorización de usuarios.

El objetivo del proyecto es, por tanto, disponer de un entorno de autenticación único de usuario para un entorno Web (WSSO) que sea seguro.

Antes de iniciar este proceso, es imprescindible comprender las bases del funcionamiento de un sistema Single Sign On, así como las partes de las que se compone el servicio.

A partir de aquí, habrá que diseñar la arquitectura que lo soporte e implementarla.

Punto de partida

Según el planteamiento inicial, y al carecer de ningún tipo de recurso, el punto de partida del proyecto es inexistente, es decir, se parte de cero, lo cual implica que se han de implementar todos elementos necesarios para poder realizar un despliegue de un entorno SSO basado en OpenAM (producto elegido).

Una vez realizado el despliegue hardware y software, en una segunda fase se procederá a la implementación del entorno SSO con aplicaciones de prueba, y se realizarán los test necesarios para verificar su funcionamiento.

En el caso de disponer de tiempo, sería deseable que puedan implementar configuraciones de federación con Google Apps, y así como los mecanismos de seguridad adaptativa de OpenAM.

Por último, se mencionará el acceso a las APIs de OpenAM y de cómo se pueden hacer uso de la información que proporciona el sistema a las aplicaciones del entorno SSO.

Aportación

Este proyecto aporta, por un lado, una prueba práctica del uso de sistemas SSO, con todas las implicaciones que ello supone para entornos de aplicación Web (sean o no J2EE), destacando las ventajas de su uso frente a entorno que implementan seguridad de usuario clásicas.

Por otro lado, y de cara a futuros proyectos planteados como ampliaciones del presente, aporta un entorno ya implementado de servidores y servicios virtualizados, que permiten implementar las funcionalidades que hay quedado fuera del presente y ampliar el desarrollo de funcionalidades de entornos SSO basados en OpenAM.

Enfoque

El presente trabajo, con un enfoque fundamentalmente práctico pretende, en su primera parte, ofrecer una guía de cómo debe ser la infraestructura necesaria para recibir un entorno SSO seguro en el contexto de ejecución de aplicaciones J2EE.

Y en la segunda parte, se muestra todo el proceso necesario para adaptar los mecanismos de autenticación de usuarios, entornos de seguridad y autorización, etc, para que las aplicaciones se ejecuten dentro del contexto SSO.

Y por último, se describen el procedimiento para configurar la federación de identidades entre el entorno desarrollado en el punto anterior y Google Apps.

En resumen, dado que al partir de cero, es necesario realizar el despliegue de recursos que permitan crear el entorno de ejecución SSO, el enfoque estará más orientado a la instalación y configuración de sistemas y componentes software requeridos en el entorno.

En el caso de haber podido disponer de un entorno previo con los componentes básicos ya funcionando, podría haberse establecido un enfoque más orientado a desarrollo de aplicaciones J2EE en entornos SSO, que hubiera sido muy interesante, pero no ha sido posible debido a que la instalación de todo el entorno ha requerido mucha dedicación.

Metodología

No se ha aplicado ningún tipo de metodología, más allá del desglose de tareas cotejado con el propio director del proyecto, y la planificación de las mismas.

Organización de la memoria.

La memoria está organizada en dos grandes secciones, la primera corresponde al despliegue del entorno SSO, y la última a los anexos, que contienen, en el primero anexo, información técnica de la implantación física del entorno, y el segundo información sobre los test de funcionamiento y análisis del intercambio de mensajes.

En la primera sección, a su vez, podemos dividirla en los siguientes puntos:

1. Descripción teórica de las características y funcionamiento de los entornos SSO, en general, y de OpenAM en particular.
2. Descripción detallada de la infraestructura SSO, basada en OpenAM, desde un punto de vista teórico, haciendo hincapié en aquellos aspectos más relevantes de la configuración.
3. Configuración básica requerida para la protección de una aplicación de ejemplo, siempre partiendo del entorno desplegado descrito en el Anexo I.
4. Configuración avanzada donde se describen:
    1. Los mecanismos de autenticación basados en certificados,
    2. Y el escalado de seguridad en la autenticación.
5. Otros temas de interés, donde se mencionarán, sin profundizar, las facilidades adicionales proporcionadas por OpenAM y sobre cómo inciden en las aplicaciones protegidas por el sistema SSO.

En el apartado de anexos, se han incluido dos documentos:

1. Documento de despliegue de infraestructura.
2. Documento de Test y análisis de intercambio de mensajes.

Es importante reseñar que se ha decidido separar las partes de despliegue y test del resto del documento e incluirlas como anexos debido a que, en el primer caso, se trata de la descripción de elementos necesarios para el correcto funcionamiento del entorno SSO, pero que no tienen que ver con el mismo directamente, es decir, se explican todos los elementos de infraestructura requeridos para soportar un entorno SSO sin que sea el mismo SSO.

Y en el segundo caso, porque se trata de comprobar el correcto funcionamiento del entorno SSO y de cómo interactúa el usuario con él (desde su navegador), pero no describe ningún tipo de aspecto relacionado con la arquitectura o configuración.

Planificación

El desglose detallado de tareas es:

WBS	Nombre	Inicio	Fin	Trabajo
1	Elaboración del plan de trabajo	Sep 28	Oct 7	6d 1h
2	**HITO: Entregada PAC1 (7/10/2013 - 24:00)**	**Oct 7**		
3	Despliegue de infraestructura	Oct 7	Nov 4	14d 5h
3.1	Diseño esquemático del entorno a desplegar	Oct 7	Oct 8	2h
3.2	Preparación de los sistemas anfitriones	Oct 8	Oct 8	1h
3.3	Instalación del entorno de virtualización	Oct 8	Oct 9	1h
3.4	Instalación de los sistemas operativos de las VM	Oct 9	Oct 10	2h
3.5	Configuración red de las máquinas de la LAN	Oct 10	Oct 10	1h
3.6	Configuración red de las máquinas de la DMZ	Oct 10	Oct 11	1h
3.7	Configuración de red del FW (rutas y reglas)	Oct 11	Oct 12	2h
3.8	Configuración de APPs de la LAN	Oct 12	Oct 21	6d
3.8.1	Instalación y configuración de todos los Tomcat	Oct 12	Oct 12	4h
3.8.2	Despliegue aplicaciones tomcat de prueba	Oct 12	Oct 20	5d
3.8.3	Test funcionamiento LAN	Oct 20	Oct 21	4h
3.9	Configuración DMZ	Oct 21	Oct 26	1d 4h
3.9.1	Configuración DNS	Oct 21	Oct 23	4h
3.9.2	Configuración Proxy RP a las APPs Tomcat	Oct 23	Oct 26	6h
3.9.3	Test funcionamiento WAN	Oct 26	Oct 26	2h
3.10	Despliegue infraestructura SSO	Oct 26	Nov 4	5d 7h
3.10.1	Instalación y configuración todos los tomcat	Oct 26	Oct 26	4h
3.10.2	Instalación y configuración básica OpenDJ	Oct 26	Oct 29	2d
3.10.3	Instalación y configuración OpenAM (en HA)	Oct 29	Nov 2	2d
3.10.4	Instalación y configuración HAProxy en DMZ	Nov 2	Nov 3	6h
3.10.5	Test de acceso WAN SSO en HA	Nov 3	Nov 4	5h
4	**HITO: Entrega PAC2 (4/11/2013 . 24:00)**	**Nov 4**		
5	Securización y parametrización de aplicaciones	Nov 4	Dec 2	14d 5h
5.1	Configuración de filtros de seguridad	Nov 4	Nov 14	4d 4h
5.2	Configuración autenticación/autorización aplicaciones	Nov 14	Nov 26	6d 5h
5.3	Test de acceso vía SSO LAN	Nov 26	Nov 28	3h
5.4	Test de acceso vía SSO WAN	Nov 28	Nov 30	1d 3h
5.5	Test acceso vía SSO con HA	Nov 30	Dec 2	1d 3h
6	**HITO: Entrega PAC3 (2/12/2013 - 24:00)**	**Dec 2**		
7	Memoria final TFM	Dec 2	Jan 10	19d
7.1	Montaje, maquetación y edición final	Dec 2	Dec 29	14d 2h
7.2	Montaje, maquetación presentación PPT	Dec 29	Jan 10	4d 6h
8	**HITO: Entrega memoria TFM (10/01/2014 - 24:00)**	**Jan 10**		
9	Presentación vídeo (Defensa)	Jan 10	Jan 17	3d 6h
9.1	Elaboración del guión	Jan 10	Jan 13	2d 5h
9.2	Grabación del vídeo	Jan 13	Jan 17	1d
10	**HITO: Defensa virtual del proyecto (17/01/2014 - 24:00)**	**Jan 17**		

Nota: durante el desarrollo del trabajo, parte de las tareas de configuración dela fase 5 tuvieron que rehacerse en la fase 7 debido a un error en el planteamiento de la arquitectura de proxies (fase 3), la desviación ha sido corregida a lo largo de la fase 7.

Introducción a sistemas SSO

Supongamos que nos encontramos en un aeropuerto internacional, y los usuarios pretenden acceder a los diferentes vuelos, en vez de disponer de equipo de identificación por cada compañía, todas ellas dirigen sus pasajeros a control de aduanas, que son los que se encargan de identificar a cada persona y permitir el acceso a la zona de embarque.

En el caso de aeropuertos comunitarios, solicitan identificación, si presentan DNI o lo que corresponda para cada país, se permite el acceso, pero si no disponen de este tipo de identificación, solicitarán pasaporte y, una vez identificados, muy probablemente, dependiendo del destino, el visado, etc...

Un sistema SSO[1][1] es esencialmente lo mismo, es decir, todas las aplicaciones dirigen la identificación de los usuarios al SSO, que, dependiendo del recurso al que desea el usuario acceder, exigirá un tipo de identificación más o menos fuerte, y permitirá o denegará el acceso una vez concluido el proceso de identificación del usuario.

Existe una especialización del concepto SSO para entornos Web, denominado WSSO, de uso mucho más extendido que el SSO, y yse trata de un mecanismo que permite que un usuario se autentique en un sitio web y, con la misma sesión de usuario, se le permita acceder, sin volver a autenticarse, en otros sitios web.

Definición

En la documentación de la asignatura de la UOC "Identidad Digital"[2], se define un sistema SSO como:

> *Un sistema de autenticación único, en la que el usuario sólo se identifica una sola vez en alguno de los sistemas asociados, y el sistema SSO, a partir de aquí, le permite el acceso al resto de los sistemas asociados sin requerir un nuevo proceso de autenticación.*

En la Wikipedia[2][3], un sistema SSO viene definido como:

> *Es un sistema de control de acceso (de usuario), a múltiples sistemas relacionados pero independientes. Con este sistema, el usuario se identifica (logon) una única vez y gana acceso a todos los sistemas/aplicaciones que integran en el sistema Single Sign-On*

Como ventajas inherentes a este mecanismo podemos encontrar:

- Descenso del número de procesos de logon en los sistemas asociados, disminuyendo la "fatiga" del propio proceso de autenticación (el uso de varias combinaciones de usuario/password para una misma identidad).
- Aceleración del proceso de acceso (autenticado) a los sistemas, evitando tener que volver a introducir las credenciales de usuario.
- Gestión centralizada de usuarios y autorizaciones, evitando que sean las propias aplicaciones las que tengan que implementar estos mecanismos (se delega al propio SSO) agilizando el proceso de aprovisionamiento de credenciales de usuarios (altas) para las diferentes aplicaciones, y automatizando los procesos de actualización y baja. reduciendo considerablemente la probabilidad de error.

1 http://blogs.forgerock.org/petermajor/2013/02/single-sign-on-the-basic-concepts/
2 https://en.wikipedia.org/wiki/Single_sign-on

- Sistema multiplataforma, esto es, permite la integración de diferentes sistemas, procedentes de distintos fabricantes en un único mecanismo de autenticación de usuario.

Y como contrapartida tenemos que:

- Un compromiso del proceso de acceso al sistema SSO permite que un intruso pueda acceder a todos los sistemas amparados bajo el sistema SSO. Este inconveniente se suele mitigar haciendo que el proceso de autenticación sea mucho más estricto que en los procesos habituales.

Atendiendo a la arquitectura[3][4] de un sistema SSO, se pueden clasificar de la siguiente manera:

- Simple: en el que el sistema SSO es único (esté o no clusterizado), y otorga acceso a los usuarios de un único dominio de seguridad.
- Complejo: es una arquitectura propia de sistemas federados, en los que existe más de un sistema de autenticación (SSO), y entre los que existe algún mecanismo de interrelación o confianza. Generalmente este tipo de sistemas están compuestos por varios dominios de seguridad, habiendo un mecanismo SSO en cada uno de ellos.
    - Autenticación centralizada: en este tipo de arquitectura, el usuario es identificado a través de un elemento (Token o certificado) que es el que intercambia con las entidades de autenticación.
        - Basada en Tokens: La entidad de autenticación inicial es la que proporciona un token de sesión al usuario (token de kerberos o una cookie, etc...), y es el que, de forma transparente usa el cliente para acceder al resto de recursos, de la primera entidad o del resto de la federación. El resto de entidades validarán el token contra la primera.
        - Basada en PKI: en este caso no se asigna token al usuario, sino que, este, usa un certificado PKI que es validado contra una CA de confianza para todos los SSO federados.
    - Autenticación Múltiple: en este tipo de mecanismos, las credenciales son cacheadas, ya sea en el lado del cliente, o ya sea en el servidor, y son independientes para cada autoridad de autenticación. Se trata de sistemas más pesados porque requieren mecanismos muy seguros para las cachés de credenciales, así como software adicional para la gestión y sincronización de credenciales en la parte cliente o servidora.

      En esta arquitectura, realmente se produce la autenticación en las diferentes entidades de autenticación (múltiple autenticación), ya que el cliente utiliza la información de sus credenciales que tiene cacheadas (ya sean en el servidor o en el propio cliente) en cada recurso al que desea acceder.

Y atendiendo al alcance del ámbito de operación del sistema SSO, podemos diferenciar dos tipos:

- ESSO: Enterprise Single Sign-On, es un sistema SSO completamente heterogéneo, que es capaz de gestionar los procesos de autenticación de los usuarios en cualquier sistema de un entorno IT (que esté integrado en el SSO).
- Microsoft introduce una variante, denominada *Integrated Windows Authentication*, en la que proporciona todos los mecanismos de autenticación de usuario basado en Directorio Activo a cualquier

3 http://studies.ac.upc.edu/FIB/CASO/seminaris/1q0304/T7.ppt

sistema Microsoft, y hace extensión de este mecanismo a entornos Linux, Unix y Macs.

- WSSO: Web Sigle Sign-On[4][5][6]. Es una especialización de los sistemas ESSO, que centran la gestión de autenticación en sistemas Web, generalmente, de acceso público.

En definitiva, como podemos intuir, todo el trabajo de un sistema Single Sign On se centra en el establecimiento y posterior mantenimiento de la sesión de usuario (propio o de un entorno federado) a lo largo de todo el entorno SSO, así como la identificación y autorización (local o a través de terceros) de dicha sesión en cada uno de los accesos (exitosos o no) del usuario a los recursos gestionados por el entorno SSO.

Antecedentes

El protocolo HTTP, en el que se basa la comunicación entre el navegador de usuario y los sitios web, carece de mecanismos de control y gestión de sesión, lo cual implica que no es posible, sin ningún otro elemento de apoyo, relacionar los diferentes mensajes HTTP que puedan enviarse desde el navegador de un usuario cuando este esté, por ejemplo, actualizando su perfil en un sitio web de Internet.

Para solucionar este problema, se emplean las *cookies* (creadas por Lou Montull en 1994).

- Cookie: se trata de un conjunto de datos enviados por el servidor web al navegador del usuario en el momento que este accede por primera vez al mismo (aplicación web). El cliente almacenará estos datos en local y serán reenviados al servidor cada vez que el usuario, de nuevo, vuelva a acceder al sitio web.

  Este conjunto de datos almacena información de estado que debe mantenerse a lo largo de la actividad de navegación del usuario. Además permite almacenar otros elementos que pueden ser usados por los sitios web para recabar información de la actividad web del usuario para tareas de customización, etc...

- RFC 6265: Norma que especifica cómo se han de implementar las cookies para establecer mecanismos de gestión de estado el uso del protocolo HTTP.

  Esta norma establece que los elementos que debe almacenar una cookie para gestión del estado son:

  - Domain: Dominio donde podrá ser transmitida la cookie, y de no existir, se utilizará como cookie basada en el host. Es decir, si la cookie tiene el dominio "tfmuoc.edu", estará disponible para cualquier subdominio de este dominio (es decir, por ejemplo, estará disponible para apps1.tfmuoc.edu y apps2.tfmuoc.edu). El dominio ha de ser válido en modo FQDN, cualquier otra opción, como "localhost" o la dirección IP será rechazada (la cookie).
  - Max-Age: indica el tiempo de validez máximo de la cookie, y la cookie será valida al menos durante este tiempo o hasta que el navegador del usuario se cierre.
  - Path: contiene la URL de aplicación de la cookie (dónde es

4 Microsoft denomina a los WSSO como Extranet SSO, y otros fabricantes usan acepciones como Web Access Management (WAM):
- http://msdn.microsoft.com/en-us/library/aa745042%28v=bts.10%29.aspx
- http://www.courion.com/products/enterprise-single-sign-on.html

válida).

- Secure: si se usa este parámetro, la cookie sólo puede ser transmitida por HTTP Seguro (HTTPS/443), en caso contrario, esta cookie no podrá ser incluida en las transmisión.
- HttpOnly: si se usa, indica que no se podrá usar desde JavaScripts, lo cual ofrece una protección contra ataques de tipo XSS.

Los sistemas de Single Sign On, por tanto, requieren una buena gestión de cookies HTTP de sesión de usuario, a través de las cuales poder hacer un seguimiento para mantener las sesiones en todas las aplicaciones que se encuentren bajo el control del SSO.

Sticky Sessions

Cuando, en un entorno empresarial, los mecanismos de seguridad y redundancia, obligan al uso de agrupaciones de servidores (clusters, farms, etc...) que incrementen la tolerancia a fallos, aparecen problemas adicionales sobre la cuestión de la gestión de la sesión de usuario en dichos entornos web, ya que, puede haber varios servidores que, ejecutando la misma aplicación (con carácter redundante), puedan atender a una misma sesión de usuario.

En este punto es donde aparece el concepto *Sticky Session*, el cual está asociado a sistemas que disponen de algún tipo de balanceo de carga, y se refiere a la forma de asignar las peticiones de conexiones de usuarios a un determinado servidor físico que forma la agrupación (cluster) destinataria del balanceo de carga.

En otras palabras, sabemos que el protocolo HTTP no está orientado a la sesión (en el contexto HTTP), lo cual implica que cuando una solicitud enviada por un usuario es respondida por el servidor, no hay manera de vincularla con la siguiente en términos de transacción. Es fácil imaginar la necesidad de que todas las solicitudes HTTP de un usuario que está modificando datos de una base de datos deban estar agrupadas lógicamente en sesiones y sean tratadas siempre por el mismo servidor.

Cuando son varios los servidores HTTP que responden en el mismo contexto de aplicación Web, es necesario disponer de un balanceador que reparta las peticiones, y este balanceador debe, necesariamente enviar todas las peticiones de una misma sesión de usuario al mismo servidor, y para ello hace uso de un identificador en la cookie de sesión del usuario.

De este modo, el balanceador enviará al mismo servidor web todas las peticiones de la misma sesión (misma cookie).

Por tanto, la sticky session[5][7], es la característica de vincular una sesión de usuario a un servidor web (de un cluster) para que sea este quien atienda todas las solicitudes de la misma sesión.

En otras palabras, marca la afinidad de la sesión de usuario a un servidor.

Esto, obviamente, implica que el balanceador de carga debe ser capaz de gestionar sesiones de usuario (sticky sessions).

User Session Failover

Ahora ya disponemos de un mecanismo que nos permite mantener la sesión de

5 http://web.archive.org/web/20090302072037/http://daveonsoftware.blogspot.com/2007/08/load-balancer-persistence-sticky.html

usuario a través de los diferentes mensajes HTTP que emite desde su navegador cuando interactúa con una aplicación web: las *cookies*.

Por otro lado, hemos visto que podemos establecer una afinidad de esa sesión con un servidor dentro de un entorno altamente redundado a través del soporte de las *Sticky Sessions* de los mecanismos de balanceo de carga.

La problemática aparece cuando el servidor con el que se ha creado una afinidad desaparece debido a algún tipo de problema, ¿Quién continúa la sesión? ¿Los servidores de la granja pueden hacerlo? ¿La sesión se pierde? ¿Y si se trata de una transacción bancaria?

Para ello definiremos el concepto de ***User Session Failover***[6][8], y para ello partiremos de un escenario con un único servidor SSO, es bastante fácil imaginar el problema que supone para la sesión de usuario que el servidor SSO falle, simplemente se pierden las sesiones y el usuario deja de poder validarse.

Si pasamos a un escenario con dos o más servidores SSO, con la idea de que el usuario siempre disponga de un servidor con el que validarse, pero no proporcionamos ningún servicio adicional, cuando el servidor en el que el usuario se ha validado cae, la sesión se pierde, o si se prefiere, deja de tener validez porque el resto de servidores no la conocen, y el usuario al cambiar de servidor SSO, debe volver a introducir sus credenciales.

La solución a este problema pasa por el establecimiento de un mecanismo mediante el cual los servidores del agrupamiento (los SSO en este caso) intercambien, en tiempo real, la información de todas las sesiones activas y válidas que existen en este momento en todos los servidores que forman la agrupación, de tal manera que, cuando uno de ellos desaparece (fallo, etc...), cualquiera del resto de servidores puede hacerse cargo, de forma transparente, de gestionar la sesión del usuario y mantenerla activa, evitando que este vuelva a realizar el proceso de logon.

Esto permite, a su vez, que el usuario mantenga las sesiones con las aplicaciones con las que está trabajando y evite potenciales pérdidas de información, especialmente en procesos transaccionales (bancarios, etc...).

En cambio, si proporcionamos un mecanismo de intercambio de credenciales, que en este caso será un servicio especial en alta disponibilidad, que podríamos denominar Core Token Service (CTS), que almacena y comparte la información de las sesiones de usuario activas, cuando el servidor SSO que atiene la sesión de usuario cae, el resto de servidores son capaces de leer las sesiones almacenadas en el CTS y continuar dando validez a las sesiones de usuario, sin que estos tengan que volver a validarse.

Si el servidor caído vuelve a estar disponible, podrá volver a leer las sesiones del CTS y continuar dando servicio a los usuarios sin requerir un nuevo Logon.

OpenAM

Se trata de un producto[7][9] para la implementación de entornos SSO, ofrecido bajo licencia Open Source. Nacido fruto de un "fork" del producto OpenSSO.

Este último, que también apareció en el mercado con licencia Open Source, era propiedad de Sun Microsystems, y fue adquirido posteriormente, a través de la compra de Sun, por Oracle Corp, momento en el cual dejó de ofrecerse con licencia Open Source.

6 http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/install-guide/index/chap-session-failover.html
7 https://en.wikipedia.org/wiki/OpenAM

A partir de ese momento, la última versión de OpenSSO con licencia Open Source fue tomada por la empresa ForgeRock[8][10] para su continuación, cambiando la denominación a OpenAM (Open Access Management), y manteniendo el licenciamiento Open Source.Entre las características que ofrece el producto, tenemos:

- Autenticación de usuarios, proporcionando múltiples métodos de autenticación a través de módulos.
- Autorización de usuarios, mediante la creación y aplicación de políticas de acceso a recursos.
- Autenticación mediante el cálculo de riesgo adaptativo, es decir, que permite identificar las circunstancias en las que se produce la autenticación y modificar el módulo o módulos de autenticación en función del riesgo calculado.
- Servicios Federación, para permitir la autenticación y autorización de usuarios de otros entornos SSO (ej, con usuarios de entornos Google, con Google Apps, Gmail, etc...).
- Single Sign-On, creando un entorno de validación único dentro de un entorno de ejecución Web (J2EE, web, etc...)
- Mecanismos de alta disponibilidad a través de la implementación de mecanismos para gestionar la "User Session Failover".
- Api de desarrollo, que permite a las aplicaciones ejecutadas en el entorno SSO, acceder a recursos de OpenAM para obtener información de los usuarios autenticados, perfiles, etc... mediante el uso de estructuras como Principals, propias de las librerías JAAS de Java.

Elementos de OpenAM

En la configuración de OpenAM, como en la de cualquier sistema SSO, hemos de destacar los dos principales servicios, los de autenticación, y los de autorización.

En la fase de acceso de los usuarios a las aplicaciones, el primer servicio SSO que se encontrarán es el de Identificación, es decir, aquel servicio que se encargará de, mediante el mecanismo que sea, decir quien es este usuario.

Para ello, OpenAM debe, en primera instancia, de disponer de algún tipo de repositorio donde se almacenen las credenciales de usuario, que puede ser local (integrado mediante OpenDJ) o remoto, por ejemplo, en un servicio LDAP o un directorio activo, etc...

Adicionalmente, OpenAM dispone de todo un repositorio de módulos que permiten diferentes sistemas de autenticación de usuario, que permite implementar qué método usará para identificar a los usuarios para acceder a determinados recursos.

Incluso, en determinados casos, podría establecer una combinación de métodos.

Autenticación

En definitiva OpenAM utiliza los módulos de identificación como mecanismo para verificar las credenciales de los usuarios.

8 http://www.internetnews.com/dev-news/article.php/3881681/ForgeRock+Extending+Suns+OpenSSO+Platform.htm

Proceso de autenticación:

Los módulos de autenticación pueden ser configurados como:

- Requeridos (required): cuando un módulo está configurado como requerido implica que si durante el proceso de autenticación falla (no es capaz de identificar el usuario como válido), el proceso completo de autenticación falla, aunque completa su ejecución (resto de módulos).
- Opcionales (optional): en este caso, si el módulo falla, continúa el proceso de autenticación, y si el resto del proceso es válido, la autenticación tiene éxito.
- Suficiente (sufficient): cuando este módulo finaliza correctamente, se da por concluido el proceso de identificación y no ejecuta el resto del proceso, finalizando con éxito.
- Requisito (requisite): cuando se configura en esta modalidad, implica que es obligatorio que no falle si se desea continuar con el proceso de autenticación, es decir, si falla, el proceso de autenticación se detiene.

Adicionalmente, todos los módulos pueden configurarse con niveles de autenticación[9], siendo los valores más bajos los asociados a recursos menos restrictivos, y los más altos a los más restrictivos.

Principales métodos de autenticación

Descripción de los principales módulos, pero no los únicos, ya que OpenAM proporciona más, y permite agregar nuevos.

- Active Directory Athentication Module: En realidad se trata de un módulo muy parecido al de autenticación por LDAP, pero que OpenAM ofrece por separado para facilitar su identificación y gestión. Este módulo establece una conexión LDAP con un servicio de directorio activo para realizar la autenticación a través de dicho protocolo.
- Certificate Authentication Module: Permite el uso de certificados X.509 para la identificación del usuario, no siendo necesario el uso de identificado de usuario y contraseña u otro tipo de credenciales.

  Este módulo debe poder acceder a las claves públicas de los usuarios almacenadas en el propio OpenAM y las almacenadas en algún servicio directorio (LDAP). Este mismo módulo verifica la validez del certificado, la cadena de certificación, y si los CA utilizados son de confianza.

  Asimismo hace uso de las CRL (Certificate Revocation Lists), por lo que deberá poder acceder a la misma.
- HTTP Basic Authentication Module: Permite una autenticación básica HTTP (usuario/password) contra el Backend de OpenAM.
- JDBC Authentications Module: Permite el uso de una base de datos relacional (MySQL, Oracle, etc...) como repositorio de usuarios, y por tanto, se trata de un módulo que usa este tipo de repositorios para el proceso de autenticación.
- LDAP Authentication Module: Se trata de un módulo muy parecido al de autenticación en directorio activo, en el que se indica contra qué servidor/es LDAP hay que validar las credenciales, cuales son los DN y atributos de búsqueda.

  Y pueden definirse algunos valores de gestión de políticas de usuarios/contraseñas.

9 Heredado de la especificación de niveles de autenticación definidos en la librería Java JAAS.

- Radius Authentication Module: Este módulo hace uso de un servicio Radius (Remote Authentication Dial-In User Service) para realizar el proceso de autenticación de usuarios.
- Anonymous Authentication Module: Este módulo en si mismo no requiere autenticación, pero permite hacer un seguimiento de los usuarios que acceden a recursos anónimamente para, en el momento que accedan a algún recurso que requiera más protección, forzar la autenticación del usuario.
- Authentication Chains: Una vez configurados los módulos de autenticación e instanciados, es posible configurar cadenas de autenticación, que no es otra cosa que una secuencia (workflow) de uso de módulos de autenticación alternativas al proceso principal.

Realm

Es el conjunto de elementos necesarios para poder llevar a cabo el proceso de autenticación un dominio de autenticación concreto, y que incluyen, al menos:

- Un repositorio de credenciales (ej, un fichero de texto tipo /etc/passwd, una base de datos, un LDAP, etc.)
- Un proceso de autenticación mediante el cual, se indica cómo acceder a las credenciales almacenadas en el repositorio para el cotejo de la información de identificación entregada por el usuario.

Plugins post autenticación

Son aquellos elementos que se ejecutarán justo tras una autenticación que finaliza de forma exitosa, es decir, las tareas que debe realizar OpenAM una vez el usuario ha conseguido autenticarse correctamente.

Estas tareas pueden incluir acciones como la del envío de cookies de sesión al cliente, o la instanciación de variables de sesión (Principal). Estas acciones incluyen la comunicación con los Policy Agent para informar de la información generada en el proceso (cookies, etc...).

El bloqueo de las cuentas de usuario

El bloqueo de una cuenta de usuario se produce cuando el usuario realiza varios intentos de autenticación fallidos, siendo el número máximo de intentos uno de los parámetros que se puede configurar en OpenAM.

El bloqueo se puede producir:

- En memoria, en el que se produce de forma temporal y no se produce registro del bloqueo de la cuenta físicamente. Este bloqueo desaparece transcurrido un tiempo (Delay) definido previamente, o cuando OpenAM es reiniciado.

  Es posible programar características de bloqueo como, por ejemplo, que el retardo entre bloqueos dentro de un intervalo de tiempo concreto sea cada vez mayor.

- Permanente, se produce cuando el estado de la cuenta del usuario pasa a ser “inactivo”, y se registra permanentemente en el perfil del usuario (LDAP o el repositorio que proceda).

Autorización (políticas de autorización)

Mediante la autorización se determina si un usuario está o no autorizado a acceder a un determinado recurso. **La política de autorización** es el procedimiento que permite determinar si un usuario dispone de autorización para acceder a un determinado recurso.

Proceso de autorización en OpenAM

El proceso de autorización siempre es consecuencia del acceso de un usuario a un recurso protegido por OpenAM.

En este punto, el Policy Agent intercepta la solicitud HTTP al recurso, que podrá o no contener una cookie de sesión (dependiendo de si el usuario se ha autenticado o no) y procede de la siguiente manera:

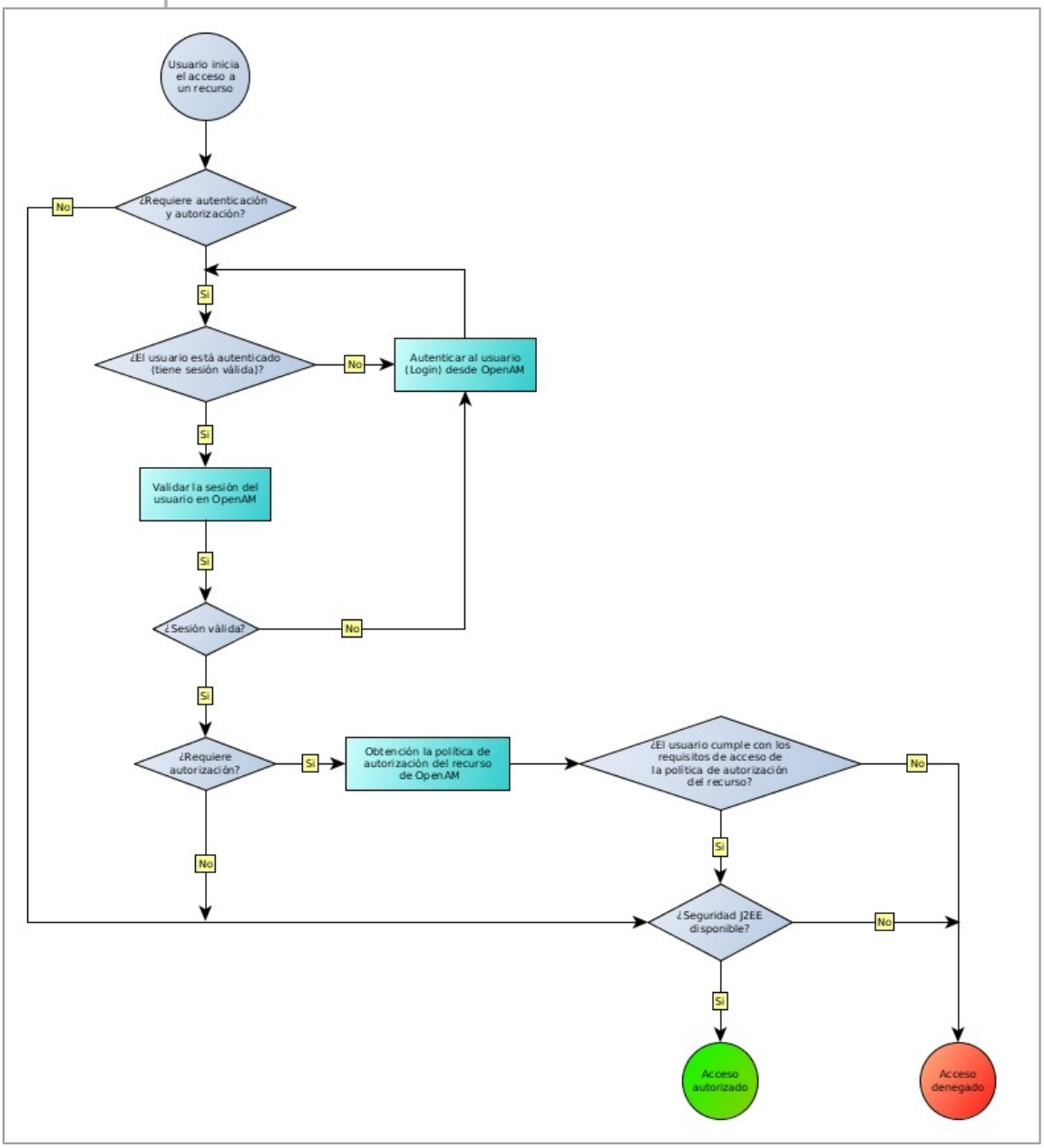

figura 1: Algoritmo de autenticación/autorización

Todo este algoritmo se orquesta a través de varios "elementos" independientes, uno de ellos, que es quien redirige al usuario (vía HTTP) hacia OpenAM para su autenticación, es el agente de políticas, que se integra dentro del servidor de aplicación y es quien "intercepta" las peticiones de acceso del usuario.

Una vez autenticado el usuario, conceder o no la autorización dependen de las reglas de acceso definidas para el entorno corporativo en el que se ejecutan las aplicaciones, y que están creadas dentro del servidor OpenAM.

Las políticas de autorización, en su forma más básica, establecen una relación entre los usuarios y los recursos (de la aplicación) a los que pretenden acceder, y una regla de autorización o denegación de acceso. Adicionalmente pueden incluir mecanismos para determinar las condiciones en las que se realiza el acceso (entorno más o menos seguro, condiciones temporales o de calendario, etc...) que pueden hacer variar la regla de autorización.

En este caso el agente de políticas, interroga al OpenAM sobre su posible autorización, y al recibir respuesta del servidor SSO, el agente hace cumplir la autorización (o no) de acceso a la aplicación o recurso.

A partir de aquí, y siguiendo el algoritmo anterior, el flujo de mensajes entre todos los participantes será, por tanto, el siguiente:

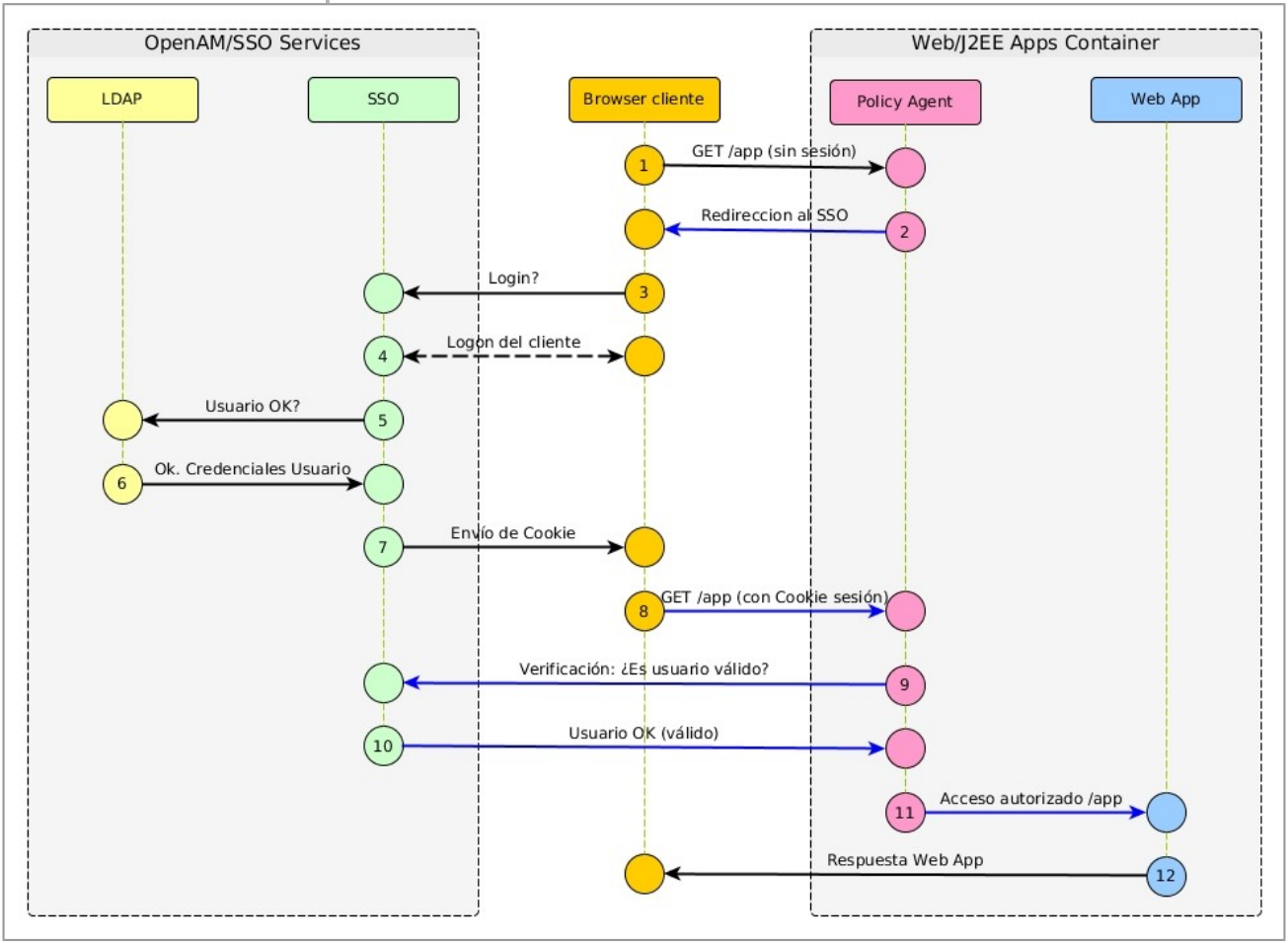

figura 2: Interacción usuario-aplicación con SSO

Dado que la responsabilidad de todo el control de acceso a los recursos recae en el agente de políticas instalado en el servidor de aplicación y en OpenAM, los desarrolladores de las aplicaciones no han de preocuparse de administrar el proceso de Login, ni de gestionar quién puede o no acceder a una página o recurso.

OpenAM ofrece los elementos necesarios, a través de un API, a los desarrolladores, para que puedan consultar información de credenciales de los usuarios, con el fin de poderlos usar a título informativo en las aplicaciones.

El despliegue de una nueva aplicación que quede amparada bajo OpenAM implica la creación de todas las políticas de acceso, así como el establecimiento de estas con el repositorio de credenciales de usuarios que se utilice en cada caso.

Por otro lado, esta independencia conlleva la ventaja de que es posible cambiar el mecanismo de autenticación de una aplicación, o las políticas de acceso, sin realizar ningún tipo de modificación en la aplicación accedida.

En definitiva, las políticas de autorización de OpenAM definen quien puede acceder a que y bajo qué condiciones, y las respuestas que puede proporcionar al agente de políticas de la aplicación pueden ser algo tan simple como un "Allow" o un "Deny", o puede ir acompañada de información sobre las credenciales de usuario que pueden ser utilizadas en la aplicación.

También es posible que una respuesta de OpenAM produzca una nueva invocación del agente de políticas a OpenAM solicitando acciones adicionales de refuerzo (que incrementen la seguridad).

Configuración de los Realm

Como ya se indicó en el capítulo anterior, los Realms son "*el conjunto de elementos necesarios para poder llevar a cabo el proceso de autenticación y autorización en un dominio de autenticación concreto*", y en el caso de OpenAM, permite la asociación de un (o más) repositorio de identidades, por ejemplo un LDAP y un proceso de autenticación.

OpenAM también asocia al Realm las políticas de autorización y los *Entitlements* para usuarios (que es un mecanismo similar al Realm, pero que permite un nivel de detalle más elevado en el proceso de autorización).

Es posible crear subdominios (subrealms) para dividir las autorizaciones en apartados que se ajusten mejor a los procesos de una organización, por ejemplo, es posible que interese que las autorizaciones a las aplicaciones de recursos humanos sean tratadas de forma independiente del resto.

En una configuración estándar, el Realm de nivel superior, Root, suele contener la información de seguridad general de las aplicaciones de la organización, mientras que los subdominios (Realms) contendrán, agrupadas, todas las políticas y perfiles de agentes de cada agrupación (por ej, de un departamento), generalmente vinculada a algún proceso empresarial.

Es posible que los Subrealms dispongan de repositorio de credenciales diferenciado. Se puede delegar el proceso de gestión de políticas al subrealm.

Los elementos que forman parte del Realm, y que configuran todo el proceso de autenticación y autorización de los usuarios en todas las URLs definidas en el mismo son:

- Nombres DNS asociados al Realm, y sobre los que se aplicará el Realm, incluidos los servidores de OpenAM que realizarán el proceso de

autenticación/autorización.

- Configuración general, aplicable a todos los elementos del Realm cuando no exista configuración específica al respecto, lo que incluye:
  - *Módulo de autenticación por defecto* (incialmente es LDAP a través de OpenDJ)
  - *Página de inicio por defecto* cuando el usuario se autentique (y no tenga definida una).
  - *Módulos de autenticación disponibles* para este Realm (ver pg.11)
  - *Cadenas de autenticación disponibles*: define la lista de los módulos de autenticación concatenados que deberán usarse, y en qué condiciones (requeridos, opcionales, requisito o suficiente), así como cuales serán la página de inicio, error de logon y acciones posteriores al logon.
- Almacén de datos donde se almacenarán los Realms configurados (por defecto en el propio LDAP OpenDJ).
- Las políticas de autorización definidas, en las que se relacionan:
  - *URLs de aplicaciones a proteger*, y la acción a realizar en caso de que la autenticación tenga éxito (acción Allow o Deny)
  - *Usuarios/Grupos de usuarios* sobre los que aplica la protección, al resto simplemente no se autoriza en ningún caso.
  - *Condiciones en las que se producirá la autenticación*, es decir:
    - Si requerirá que la autenticación se produzca desde una IP o dominio determinado
    - Si el proceso de búsqueda dentro del dominio debe incluir algún elemento adicional.
    - Si se requiere que un nivel de autenticación sea igual, mayor o menor que un nivel de referencia.
    - Si se debe realizar en una franja horaria/fecha concreta.
    - Etc...
- Usuarios/grupos (Subjects) disponibles a través del módulo de autenticación del repositorio de credenciales usado por el Realm.
- Agentes de políticas (Policy Agents) remotos: que se deben instalar en cada uno de los servidores de aplicación que ejecuten las aplicaciones a proteger, y que pueden variar en función del tipo de servicio sobre el que se ejecutan: Web, J2EE, Proveedor o Cliente de Web Services, Clientes STS, Clientes Oauth 2.0, etc...
  - Interceptan las peticiones HTTP hechas al contenedor de aplicación donde se encuentran instalados.
  - La política de seguridad es enviada al agente desde el servidor OpenAM.

El agente interrogará al servidor OpenAM cada vez que requiera validaciones de usuarios nuevos o sesiones de usuario existentes.

  - La política de seguridad indica cómo actuar:
    - Si las hay, qué partes de la aplicación son accesibles sin autenticación de usuario y cuales no.
    - Qué partes requieren que el usuario proporcione una cookie válida.
    - Qué partes requieren operaciones de autenticación de mayor nivel (mayor seguridad), o métodos adicionales de

autenticación, pese a que el usuario pueda estar ya autenticado.

- Condiciones de "vida" de la sesión de usuario (duración, etc...), y condiciones para que una cuenta de usuario pueda quedar bloqueada.

- Estos agentes son los que en definitiva se encargan de ejecutar las políticas proporcionadas por el servidor OpenAM en el servidor de aplicación o web, y permitir, o no, el acceso al usuario.

  - En resumen, la estructura de un Realm definido en OpenAM puede ser representado de la siguiente manera:

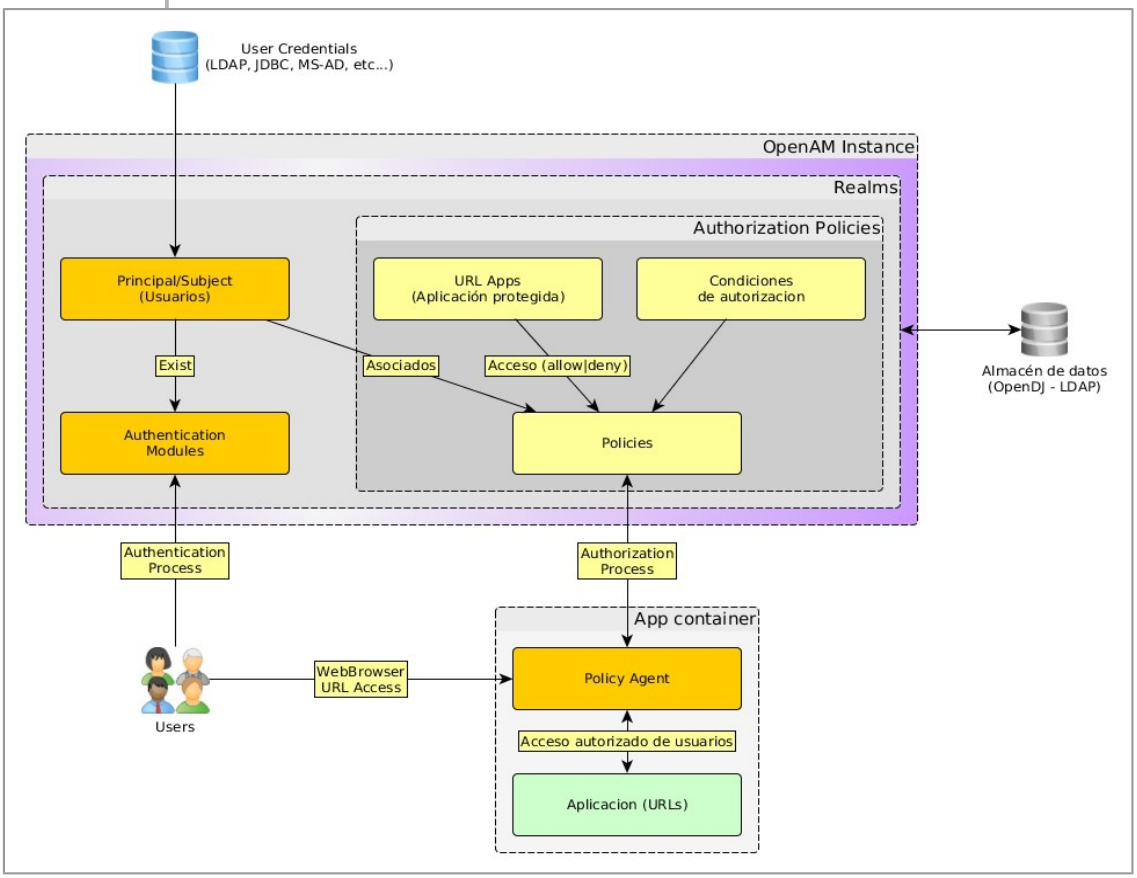

figura 3: Estructura de Realm en OpenAM

OpenAM: inspección de las políticas de seguridad

Cuando un usuario intenta acceder a un determinado recurso protegido por OpenAM es interceptado por el Policy Agent, que inspecciona qué recurso es al que se pretende acceder (URL), así como la información de las credenciales del usuario (Subject) contrastándolo con aquellas políticas con las que hace "Matching" en base esta información (recurso y el usuario).

De esas políticas, OpenAM obtiene todas las reglas en las que aparezca el recurso y para las que el usuario disponga de algún tipo de acceso, teniendo en cuenta que siempre priorizará las reglas más restrictivas, es decir, si existen dos reglas, una de acceso (Allow) y otra de prohibición (Deny), siempre primará la segunda.

OpenAM, en el momento que detecta que el resultado de la inspección de algunas de las políticas de acceso es denegado, deja de inspeccionar más reglas y deniega el acceso al recurso al usuario.

Infraestructura

Para poder comprender cual es la infraestructura necesaria para poder realizar el despliegue de un sistema SSO basado en OpenAM, debemos primero observar cómo debe funcionar la interacción entre el usuario y la aplicación protegida por OpenAM (ver figura 1).

De esta interacción se deduce que la infraestructura completa debe tener la siguiente estructura física:

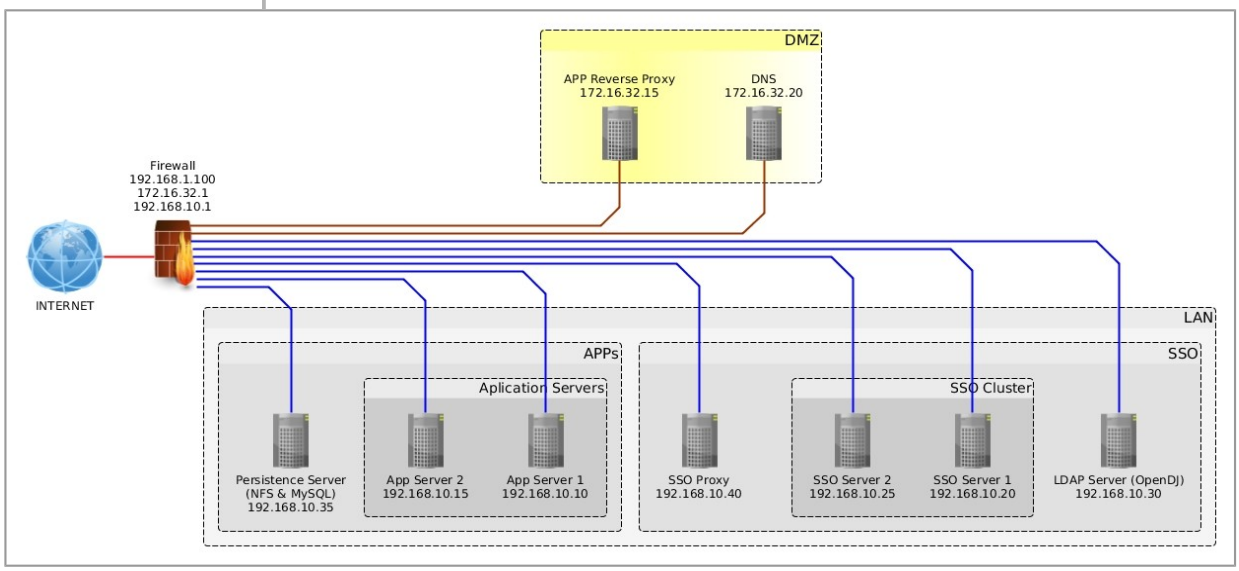

figura 4: Infraestructura SSO

Esta misma infraestructura, vista desde el punto de vista de interacción con el usuario será la siguiente:

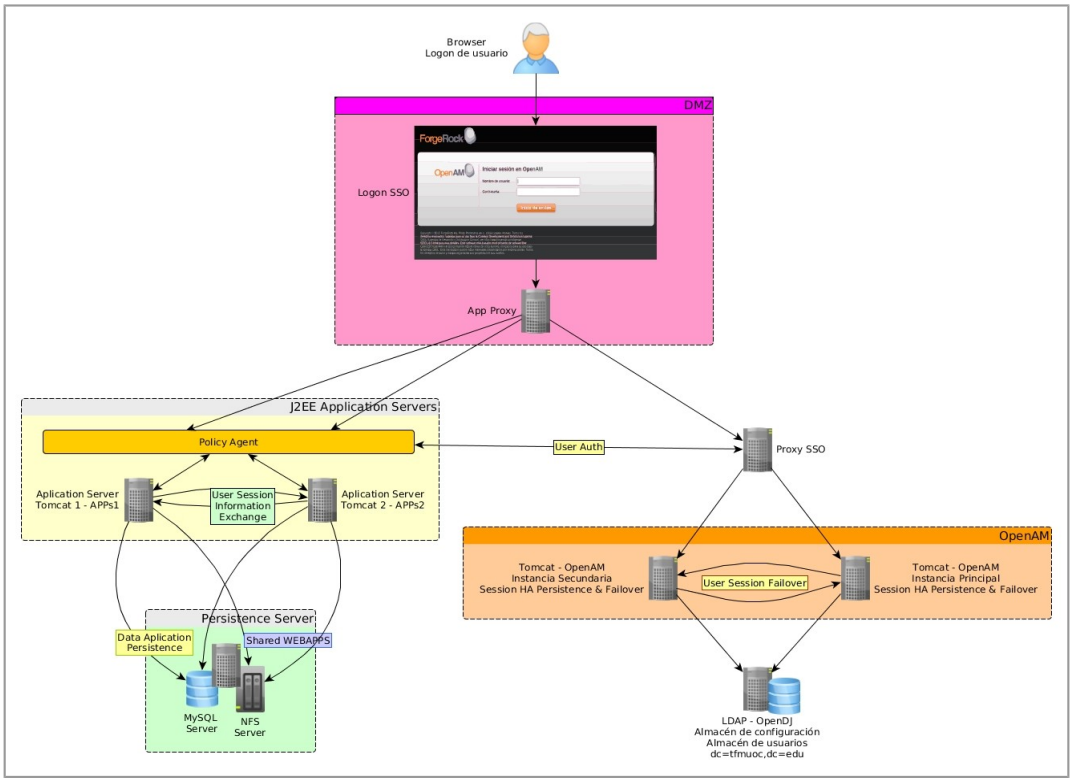

figura 5: Infraestructura SSO desde el punto de vista del usuario

Como se puede observar, cuando el usuario accede a una aplicación protegida por OpenAM, requiere, en primera instancia, que existan unos servicios publicados a través de un DNS, al que se accede mediante un Firewall que protege la infraestructura.

A su vez, este da acceso al entorno de aplicación por medio de un proxy/balanceador de carga que dirige al usuario hacia los servidores de aplicación de forma transparente.

Los servidores de aplicación, a través del propio agente de políticas de OpenAM instalado en el propio servidor, intercepta las llamadas HTTP hechas por el usuario y las reenvía hacia la infraestructura SSO para validar la sesión de usuario, si la hubiere, o para crearla.

En la infraestructura SSO, se accede a través de un balanceador que dirige las peticiones a cualquiera de los dos servidores OpenAM (que es una aplicación J2EE ejecutándose en un servidor de aplicación Tomcat), y que almacenan todas las credenciales de usuario, políticas, autorizaciones, etc... en un entorno persistente del tipo directorio LDAP.

Por tanto, para desplegar esta infraestructura necesitaremos 10 servidores, si observamos la figura 4, tendremos:

- Un Firewall. Que protege la infraestructura separándola en dos zonas:
  - La DMZ con:
    - Un DNS que resuelve nombres de los servicios y aplicaciones.
    - Un proxy balanceador publico de aplicaciones Web/J2EE
    - Opcionalmente, un proxy balanceador para el sistema SSO. (nota: este proxy puede estar integrado en el de aplicación en un único proxy, con lo que nos evitaríamos desplegarlo).
  - La LAN con:
    - Servicio de aplicaciones J2EE:
      - Dos servidores J2EE Tomcat, que ejecutarán las aplicaciones de usuario (en modo cluster).
      - Un servidor de persistencia con dos servicios, uno de ficheros NFS, que es usado por los servidores tomcat para compartir el área de despliegue de aplicaciones (/webapps),
    - Servicio de autorización/autenticación SSO con OpenAM:
      - Un balanceador para permitir el acceso al entorno SSO desde la LAN sin necesidad de pasar por la DMZ (opcional).
      - Dos servidores J2EE con Tomcat y OpenAM.
      - Un servidor de directorio para las credenciales de usuario y almacenamiento de configuración de políticas, autorizaciones, etc...

Toda la infraestructura se desplegará en un entorno virtualizado con la configuración de red de todas las máquinas virtuales independizadas por LANs privadas, simulando al máximo una situación de red real.

El software utilizado es:

- Para la implementación del proyecto
  - Servidor anfitrión (físico): Linux Mint x86_64 R.15
  - Servicio de virtualización: Oracle Virtual Box 4.2
  - Servidores virtuales: CentOS 5.9 x86_64
  - Servicio SSO: OpenAM 10.1
  - Servidores de aplicación: Tomcat 6.0 (para la aplicación Web de test y para el servicio SSO).
  - Persistencia: SQL (MySQL 5.2) y NFS (nativo CentOS).

- Soporte Java: **Oracle Java 1.6**
- Servidor de directorio: **OpenDJ 2.6**
- DNS: **Bind 9.8**
- Firewall: **Iptables 1.3** (del propio kernel del Linux) y **Firewall Builder 5.1** (GUI para Iptables).
- Proxies: **Apache 2.4 y HAProxy 1.4**

- Para el posible test del entorno y pruebas de API de OpenAM
  - Aplicación J2EE "**AgentSample**" incluida en el propio producto OpenAM 10.1 dentro del paquete "**Policy Agent**".
  - Aplicación J2EE "**AgentSample**" modificada (para no entrar en conflicto con la anterior) del producto OpenAM 11.0, dentro del paquete del "**Policy Agent**".
- Para la elaboración de la memoria
  - Documentación y presentación: **LibreOffice 4.0.2**
  - Planificación: **Planner 0.14.6**
  - Video presentación: **recordMyDesktop 0.3.8**
  - Edición gráfica y audio: **Gimp 2.8**, **Audacity 2.0.3**
  - Diagramas: **yEd 3.11.**

> **Nota**: todas las herramientas elegidas para el desarrollo del presente proyecto están amparadas por alguna de las licencias GNU, GPL, Apache o variantes, que permiten su libre uso sin necesidad de adquisición de licencias.

Configuración básica

En este punto, ilustraremos toda la teoría anteriormente descrita con la configuración básica para proteger una aplicación J2EE de ejemplo.

Introducción

Dado que partimos del escenario descrito en el "*Anexo I – Despliegue de la infraestructura necesaria*", en el que ya disponemos de un servidor Tomcat con una serie de aplicaciones instaladas, de un servicio OpenAM formado por dos instancias y accesible por un balanceador de carga, vamos a proceder a describir todos los pasos necesarios para la protección de una de las aplicaciones J2EE instaladas en el servidor Tomcat.

En resumen, los pasos a seguir son:

1. Identificar las URLs y nombres DNS de todos los elementos que intervienen.
2. Instalación del agente de políticas en el servidor Tomcat.
3. Creación de la política de seguridad (autorización):
   1. Selección de los usuarios que tienen accesos
   2. Identificación de las partes de la aplicación que se deben proteger, y las que no lo requieren
   3. Autorización o denegación del acceso según proceda.
4. Test de acceso.

Identificación de los elementos

Dado que la aplicación a proteger se encuentra ejecutándose en un servidor J2EE Apache Tomcat versión 6, necesitaremos conocer los directorios en los que se encuentra ubicado dicho servidor, que serán necesarios para la instalación del policy agent.

En nuestro caso, el servicio Tomcat se encuentra en (variable **CATALINA_HOME**):

```
/usr/share/apache-tomcat-6.0.37
```

La aplicación a proteger es una aplicación de ejemplo para el test de las políticas de OpenAM y del Policy Agent ubicado en el paquete de instalación del propio agente.

El path de instalación del policy agent, en el servidor de aplicación, es:

```
/usr/share/apache-tomcat-6.0.37/amAgent
```

Y las aplicaciones que desplegaremos en el servidor tomcat para realizar el test de políticas, situadas en (amAgent es el agente de políticas de OpenAM):

```
/usr/share/apache-tomcat-6.0.37/amAgent/sampleapp/dist/agentsample.war
http://psi-probe.googlecode.com/files/probe-2.3.3.zip
/usr/share/apache-tomcat-6.0.37/webapps/manager
```

Que copiaremos en $CATALINA_HOME/webapps para que tomcat la despliegue.

Las URLs del servidor Tomcat en si mismo y para estas aplicaciones respectivamente, son (vía proxy)

```
http://apps.tfmuoc.edu:80/agentsample
http://apps.tfmuoc.edu:80/agentsample2
http://apps.tfmuoc.edu:80/manager/html
http://apps.tfmuoc.edu:80/probe
```

La instancia OpenAM es accesible desde la URL:

```
http://apps.tfmuoc.edu:80/openam (vía proxy)
```

Y por último, la URL (que aún no existe) desde la que queremos que sea visible el policy agent, será:

```
http://apps1.tfmuoc.edu:80/agent
```

Creación del perfil del Policy Agent

Antes de iniciar el despliegue del Policy Agent, es necesario informar al servidor OpenAM de la existencia de un nuevo agente, para ello, en la consola de gestión de OpenAM, que en nuestro caso corresponde a: http://apps.tfmuoc.edu/openam/, accederemos con el usuario administrador (amAdmin) y seleccionaremos la opción de "Control de acceso":

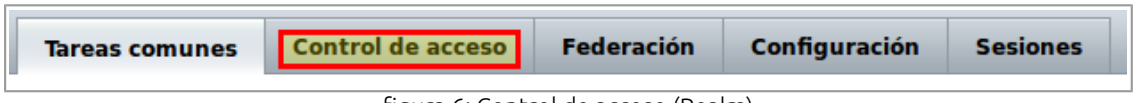

figura 6: Control de acceso (Realm)

Y seleccionamos el dominio donde queramos que opere nuestro agente.

Que en este caso será el raíz:

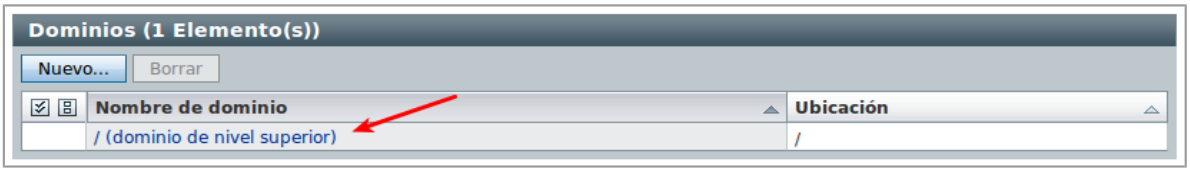

figura 7: Realm

Y seleccionaremos la pestaña de "Agentes", del tipo "J2EE" (no olvidemos que instalaremos el Policy Agent en un Tomcat), y en la lista "Agentes", seleccionaremos "Nuevo", donde nos aparecerá el formulario:

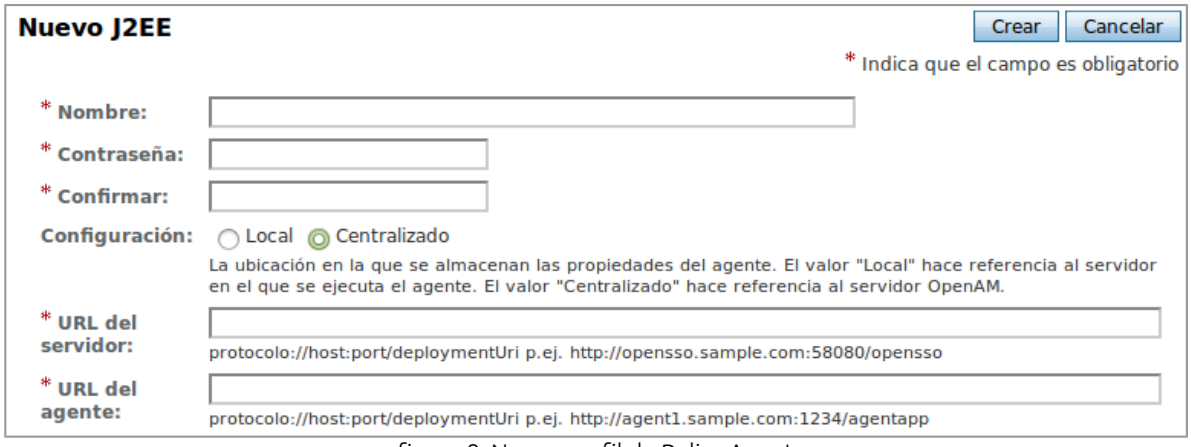

figura 8: Nuevo perfil de Policy Agent

Aquí informaremos del nombre del perfil, que en nuestro caso se llamará "apps1Agent" y la contraseña, seleccionaremos también, en qué ubicación deseamos que se almacene la configuración del agente (optamos por "Centralizado"), e introduciremos las URL del agente y del servidor OpenAM (ver apartado anterior), que serán:

```
http://apps.tfmuoc.edu:80/openam
http://apps1.tfmuoc.edu:80/agent
```

Y al pulsar el botón "Crear" nos aparecerá el perfil del agente en la lista de "Agente".

Repetiremos la operación de forma idéntica para crear el agente "apps2Agent" que será el Policy Agent del servidor apps2.tfmuoc.edu.

Instalación del Policy Agent

En primer lugar, el policy agent requiere contraseña para poder conectarse al servicio OpenAM, la forma de proporcionársela es almacenada en un fichero de texto que, por razones de seguridad, debe quedar protegido, únicamente accesible por el creador (root), este paso lo haremos así:

```
echo "passwordAgent" > /tmp/apps1Agent.pwd; chmod 400
```

Por otro lado, con el fin de que el agente pueda interceptar las peticiones HTTP que le llegan al servidor de aplicaciones, para poder aplicar las políticas de OpenAM, es necesario que se agreguen algunas de las librerías en el servidor Apache Tomcat, y se incluyan varias opciones de configuración general

(propiedades de despliegue) en los ficheros server.xml (configuración del Realm de OpenAM) y web.xml (filtros de operaciones de acceso al servidor).

En este punto, dado que es necesario que el propio agente se cargue durante la inicialización del servidor Tomcat, es necesario que este último esté parado.

Por tanto, una vez detenido el servidor de aplicación, procederemos a instalar el Policy Agent en el servidor Tomcat, para ello usaremos los datos recabados en el apartado anterior a través del diálogo ofrecido por el instalador del agente, el cual se invocará del siguiente modo:

```
/root/tomcat_v6_agent/bin/agentadmin --install
```

A partir de aquí, se introduce la información solicitada, que de forma resumida será:

```
Enter the Tomcat Server Config Directory Path
[/opt/apache-tomcat-6.0.14/conf]:
/usr/share/apache-tomcat-6.0.37/conf
OpenAM server URL: http://apps.tfmuoc.edu:80/openam
Enter the $CATALINA_HOME environment variable:
/usr/share/apache-tomcat-6.0.37
Install agent filter in global web.xml ? [true]: true
Agent URL: http://apps1.tfmuoc.edu:80/agent
Enter the Agent Profile name: apps1Agent
Enter the path to the password file: /tmp/apps1Agent.pwd
```

Al finalizar, el instalador nos muestra un resumen de la información recogida y si aceptamos, procede a la instalación del agente.

Al finalizar podemos observar los cambios[10][11] que el instalador ha realizado en la configuración y librerías del servicio Tomcat.

El primero de todo, en la configuración del servidor de aplicaciones, sustituyendo cualquier entrada de tipo "Realm" en el fichero server.xml de tomcat.

Nota: Tomcat usa, por defecto, un Realm basado en un fichero de textos denominado tomcat-users.xml, al que hace referencia con el Realm:

```
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
            resourceName="UserDatabase"/>
```

Donde UserDatabase es un recurso global tipo fichero xml definido al inicio de la configuración.

por:

```
<Realm     className="com.sun.identity.agents.tomcat.v6.AmTomcatRealm"
debug="99"/>
```

Esto implica que cualquier aplicación tomcat que utilice el entorno realm de identificación de usuarios de Tomcat, será automáticamente redirigido al de OpenAM, que es el que nosotros hemos definido en la consola de administración de OpenAM, y que incluye una o varias políticas de seguridad.

También realiza una modificación en el fichero de configuración general de Tomcat, el deployment descriptor general, para las aplicaciones desplegadas: web.xml (en el directorio "conf" de la instalación de Tomcat)

10 http://docs.oracle.com/cd/E19316-01/820-4729/ggtzd/index.html

En dicho fichero introduce lo siguiente:

```
<filter>
    <filter-name>Agent</filter-name>
    <display-name>Agent</display-name>
    <description>SJS Access Manager Tomcat Policy Agent Filter</description>
    <filter-class>com.sun.identity.agents.filter.AmAgentFilter</filter-class>
</filter>

<filter-mapping>
    <filter-name>Agent</filter-name>
    <url-pattern>/*</url-pattern>
    <dispatcher>REQUEST</dispatcher>
    <dispatcher>INCLUDE</dispatcher>
    <dispatcher>FORWARD</dispatcher>
    <dispatcher>ERROR</dispatcher>
</filter-mapping>
```

Este filtro no es específico de ninguna aplicación, y por tanto aplica a todas las aplicaciones desplegadas en el servidor, con independencia de los ficheros de deploymen específicos de cada aplicación ubicados en el directorio "WEB-INF" (Servlet 2.4 specification[11][12]).

En general, los filtros (Filters[12][13]) son indicaciones de control sobre las interacciones externas realizadas sobre las aplicaciones, y pueden ser definidos en el contexto de la aplicación J2EE o en el contexto global de la instancia de Tomcat.

Estos filtros realizan operaciones como (dependiendo del fichero web.xml donde se definan):

- Definir el juego de caracteres que se podrá usar en la aplicación (o en general).
- El tipo MIME definido en global o particular de una aplicación.
- Agregar información en determinados atributos de las solicitudes HTTP.
- Agregar protección CSRF.
- Redirecciones.
- Etc...

En nuestro caso con la cláusula <filter></filter> define la clase que usará el filtro, y con <filter-mapping></filter-mapping> indica en qué contexto se usará el filtro a través de las directivas <dispatcher>, que indican lo siguiente:

- REQUEST: el filtro se aplicará a todas las llamadas al path (url) o al servlet directamente.
- INCLUDE: el filtro aplicará a las llamadas realizadas a través de RequestDispatcher.include()
- FORWARD: aplicará en las llamadas a RequestDispatcher.forward().
- ERROR: se usará cuando se invoque el mecanismo de la página de error de tomcat.

En este caso, se han especificado todos los posibles dispatchers, y por tanto, aplica en todas las operaciones realizadas contra el servidor.

11 http://download.oracle.com/otn-pub/jcp/servlet-2.4-fr-spec-oth-JSpec/servlet-2_4-fr-spec.pdf
12 http://docs.oracle.com/cd/E14571_01/web.1111/e13712/web_xml.htm#autoId0

Creación de una política de seguridad

Por defecto, OpenAM, a través del policy agent, denegará el acceso a todas las aplicaciones del servidor J2EE en el que se encuentre desplegado (el agente), y será a través de una o varias políticas, como autorizaremos o denegaremos el acceso a dichas aplicaciones.

Para crear una política de seguridad debemos seguir el siguiente procedimiento:

1. Ingresaremos en la URL de OpenAM (**http://apps.tfmuoc.edu/openam**), y accederemos a la configuración del Realm a través de la opción "Control de acceso":

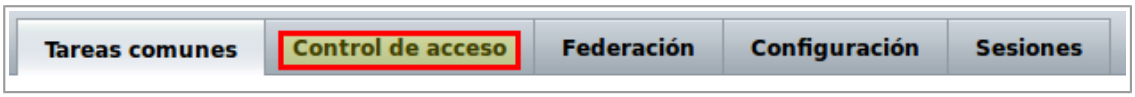

figura 9: Control de acceso (Realm)

2. Esto nos llevará a una pantalla en la que aparecerá un listado de los dominios de seguridad definidos (Realms), en el que, como mínimo, existirá el dominio de nivel superior (ROOT), que será el que seleccionemos.
3. De las pestañas que aparecen, pertenecientes a todas las opciones disponibles para ese dominio de seguridad, nos centraremos en las tres últimas, correspondientes a las "**Directivas**" (Policy), "**Asuntos**" (Users/Groups) y "**Agentes**" (Agents).
    1. En primera instancia debemos saber qué usuarios o grupos de usuarios son los que han de tener acceso a los recursos protegidos de la aplicación, que, en nuestro ejemplo, serán:

	employee	manager	everyone	customer
andy	x	x	x	
bob	x	x	x	
chris	x	x	x	x
dave	x		x	
ellen	x		x	x
frank	x		x	
gina			x	

Todos ellos son visibles desde la pestaña "**Asuntos**", subpestañas "**usuarios**" y "**grupos**".

En el caso de que no existan, podemos proceder a crearlos.

NOTA: la creación de los usuarios ("**Asuntos**") se realizan en el repositorio de datos que se definió en el momento de la instalación del servidor OpenAM, que en este caso será el servicio LDAP gestionado por OpenDJ, ubicado en el servidor "ldap.tfmuoc.edu".

Estos usuarios pueden ser consultados a través de OpenAM, o por mediación de cualquier software de administración LDAP, tal como la propia consola de OpenDJ, o el Directory Manager de Apache Directory.

Una vez están todos los usuarios y grupos introducidos, podremos visualizarlos a través de la opción “Asuntos” del dominio raiz:

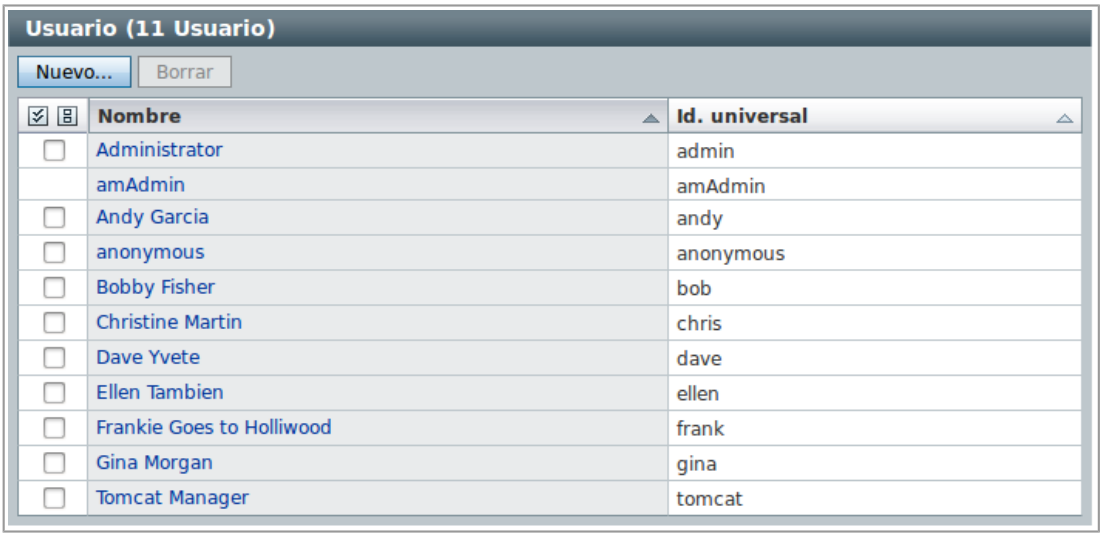

figura 10: Subjects ("Asuntos") de un Realm

4. Ahora ya solo nos queda crear la política (directiva) de seguridad, para ello seleccionaremos, dentro de la configuración del dominio raiz, la

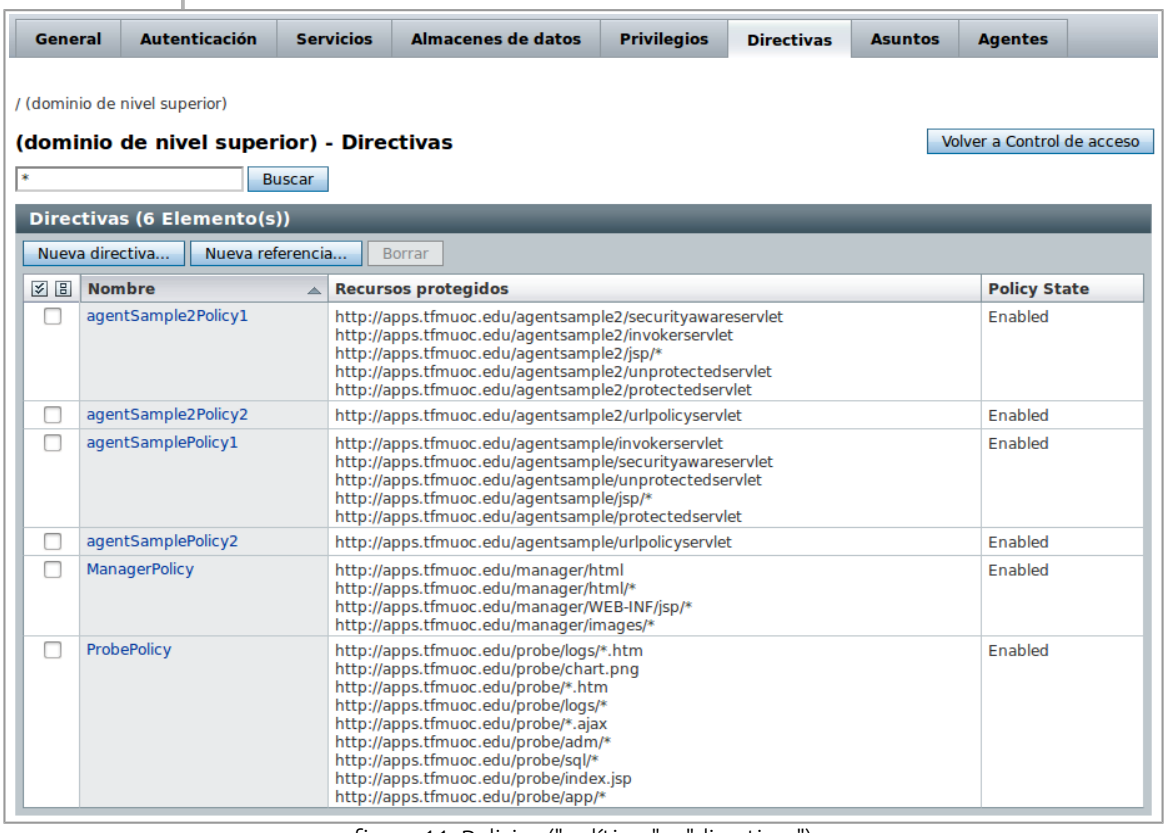

figura 11: Policies ("políticas" o "directivas")

pestaña de “Directivas”:

5. De la lista que aparece (que al inicio aparecerá vacía), se selecciona el botón "**nueva directiva**" y aparece el siguiente formulario:

figura 12: Configuración de una política (directiva)

6. Una vez informado el nombre, seleccionaremos crear una nueva regla, en la que iremos introduciendo todas las URLs de la aplicación que queramos proteger.

   Es importante disponer de información precisa de los permisos requeridos para cada usuario en cada recurso (URL) antes de proceder a

7. crear la regla de la política.

   También es altamente probable que no todas las opciones (o recursos de la aplicación) puedan ser accedidos por los mismos grupos de usuarios, por lo que se deberán crear varias directivas para la misma aplicación, asociando cada una de ellas a un grupo de usuarios/grupos distinto.

   Una directiva ya informada quedaría del siguiente modo:

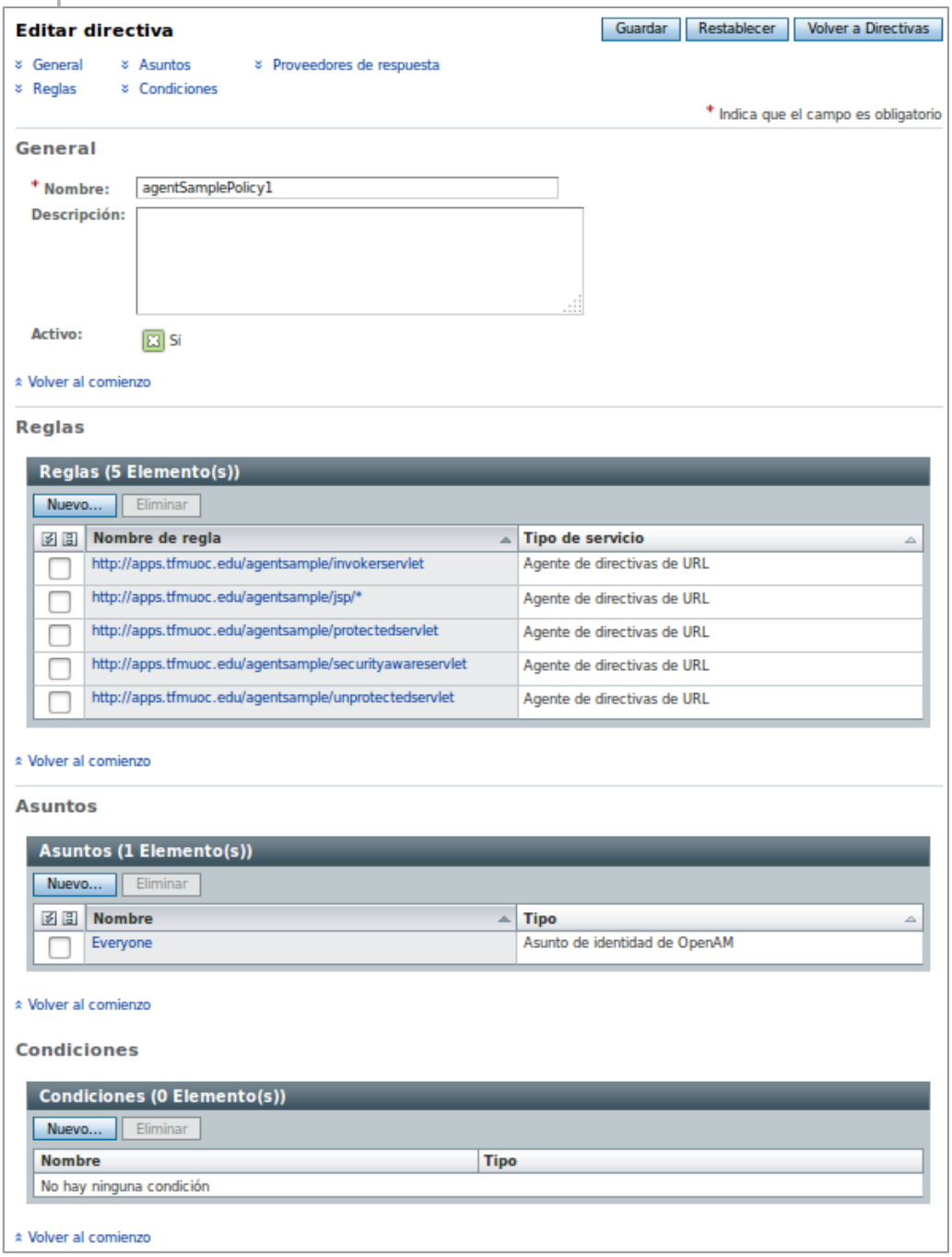

figura 13: Política ya configurada.

8. Y una regla se define de la siguiente manera:

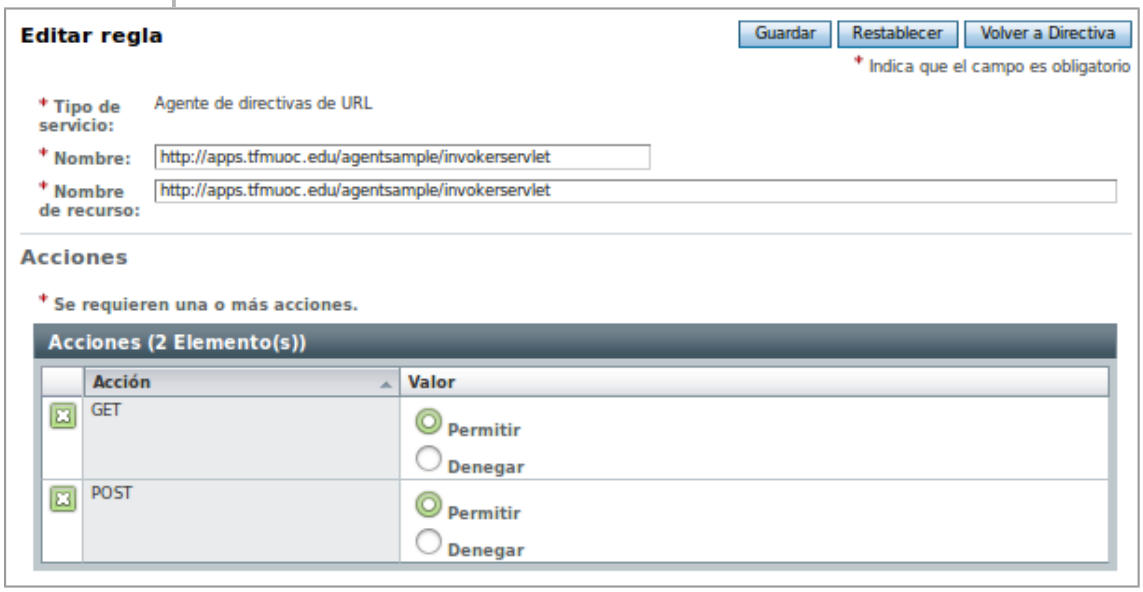

figura 14: Configuración de una regla de una política.

9. Donde definiremos el nombre del recurso protegido (yo he elegido el mismo recurso protegido por comodidad y claridad, pero puede usarse cualquier otro). Se indicará el recurso a proteger (la URL) y que acción debe realizarse al realizar un acceso HTTP GET o uno POST.

Nota: Es necesario especificar tantas reglas de acceso a los recursos protegidos como existan en la aplicación a los que pueda acceder un conjunto de usuarios específico, considerando comodines y otros elementos que permitan asociar todos ellos, ya que, de otro modo, OpenAM no permitirá el acceso a aquellos que no se han identificado dentro de la política, obteniendo efectos de ejecución en la web indeseados.
Por cada grupo de usuarios con diferentes tipos de acceso deberán definirse políticas independientes que diferencien el tipo de acceso a los recursos.

10. Una vez identificados todos recursos (URLs) a los que puede acceder cada un grupo de usuarios, procederemos a agregarlos a la política (los usuarios) pulsando el botón "Nuevo" de la tabla "Asuntos", en donde seleccionaremos los usuarios y/o grupos que tengan permisos (o no) para acceder a el grupo de reglas/recursos que hemos definido.

    Además de usuarios y/o grupos existentes en el repositorio de credenciales de OpenAM ("**Asunto de identidad de OpenAM**"), se podrán definir tipos de usuarios genéricos tales como "**Todos los usuarios autenticados**", que indicarán que sólo es necesario que el usuario disponga de una sesión válida (se haya logado) en OpenAM para poder acceder a los recursos.

11. Opcionalmente se podrán agregar condiciones de aplicación de las reglas de acceso a los recursos para los usuarios/grupos indicados tales como por ejemplo:
    1. Que la autenticación provenga de un nombre DNS o una IP concreta.
    2. Que exista un nivel de autenticación superior o inferior a un nivel de referencia dado.
    3. Alguna propiedad de sesión específica.

4. Franja horaria o de fecha, etc...

12. Es muy importante que, una vez definidas las políticas, si la aplicación dispone de partes "públicas", tales como es el caso de portales o sistemas CMS, etc... sean indicados como de acceso libre, ya que, de otro modo, OpenAM tampoco permitirá su acceso.

    Estos elementos "públicos" se deben definir como "de no aplicación" de la política, para que el agente no considere las reglas de acceso a los mismos.

    Esto se debe indicar en la configuración del agente (en nuestro caso J2EE)

figura 15: Definición de recursos "públicos" (acceso no supervisado)

Al seleccionar el Agente amAdmin y la solapa "Aplicación", veremos:

figura 16: Recursos no supervisados

13. Que nos lleva a:

figura 17: Detalle de recursos no supervisados (no aplicados)

14. En el que incluiremos como valores actuales, todas las rutas de la aplicación sobre los que las políticas creadas no deben aplicar (y el acceso debe ser público).
15. También debemos indicarle qué formulario (página) ha de presentarle al usuario cuando requiera su autenticación (pantalla de Login) y cual es la que se debe mostrar en caso de que el proceso de autenticación falle (Acceso denegado).
16. Para ello, de nuevo en la opción "**Aplicación**" de la configuración del agente J2EE, seleccionaremos "**Procesamiento de inicio de sesión**", donde introduciremos la pantalla de Login que deseemos usar (por defecto se usará la propia de OpenAM)

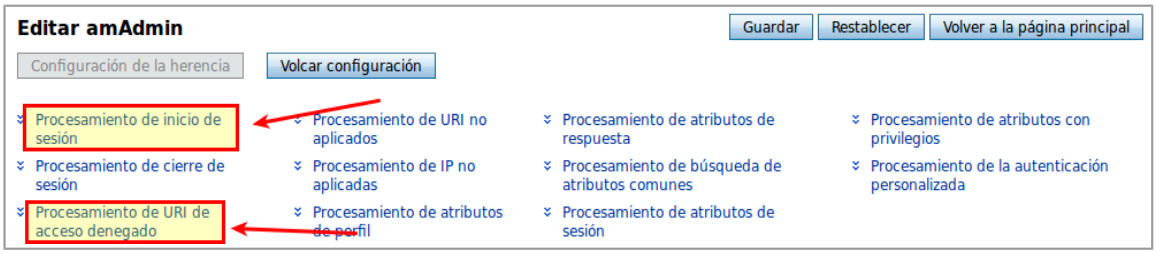

figura 18: Otros detalles de configuración

17. Y en la opción "**Procesamiento de URI de acceso denegado**", introduciremos la pantalla donde se informe de tal evento, y en nuestro caso utilizaremos una propia de la aplicación:

```
/agentsample/authentication/accessdenied.html
```

18. Y por último, es posible definir elementos que mapeen otros que pueden ser utilizados en el descriptor de despliegue de la aplicación, de forma que sea fácilmente referenciable en la misma, es decir, en este apartado se pueden relacionar nombres de "variables" que apunten a objetos "principales" que se pueden usar en el fichero WEB-APP/web.xml de las aplicaciones J2EE.

19. El objetivo de esta relación es la de ocultar información sensible de los "**Principales**" que tienen acceso a los recursos protegidos en el descriptor de la aplicación.
20. Para configurarlos, se accede en la misma opción de "**Aplicación**" del agente, y agregaremos, en la opción de "**Procesamiento de atributos con privilegios**"[13][14] a aquellos atributos privilegiados (Principales) que se deseen mapear.
21. La definición de atributos se realiza con la definición LDAP (en este caso) del grupo de usuarios con permisos de acceso.

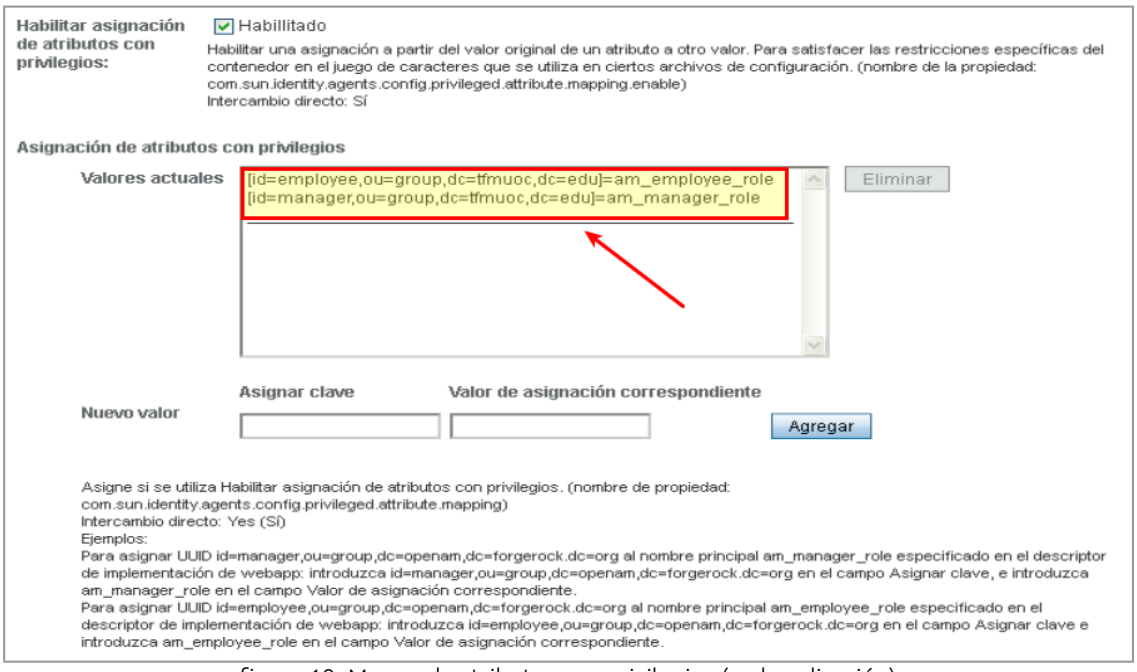

figura 19: Mapeo de atributos con privilegios (en la aplicación)

Con esto se concluye la primera integración de aplicación J2EE en el entorno de OpenAM, ahora ya sólo queda comprobar su correcto funcionamiento (ver *Anexo II – Test de funcionamiento*).

Resumen

Para proteger la aplicación "agentsample" realizaremos los siguientes pasos:

1. Desplegar la aplicación en el servidor Tomcat, quedando accesible en la URL del proxy inverso: http://apps.tfmuoc.edu/agentsample
2. Crear un perfil de agente en el Realm de nivel superior (root).
3. Instalar el agente en el servidor tomcat.
4. Verificar o crear, si procede, los grupos de usuarios y/o grupos que accederán a la aplicación.
5. Crear las directivas de las URLs sobre las que existirán accesos privilegiados, con las acciones pertinentes asociadas.
6. Configurar, dentro del perfil del agente, para la aplicación, aquellas URL que no requerirán supervisión del mismo.
7. Mapeo de Atributos privilegiados (para el descriptor J2EE de la aplicación).

13 Pg. 114 del manual de administración de OpenAM 10.1 (http://docs.forgerock.org/en/openam/10.1.0/OpenAM-10.1.0-Admin-Guide.pdf)

Configuración avanzada

En esta sección se describen algunas funcionalidades avanzadas de OpenAM, tales como el uso de certificados para autenticar a los usuarios, los niveles de autenticación o el intercambio de información entre las aplicaciones J2EE y OpenAM.

Autenticación mediante certificados

Dentro del contexto de identificación de usuarios, el uso de certificados ofrece un plus de seguridad que no ofrecen los mecanismos clásicos como el de usuario/contraseña.

Además, OpenAM permite la combinación de ambos mecanismos, lo que permite que el usuario pueda realizar su autenticación, incluso, cuando no disponga del certificado.

Será tarea de OpenAM decidir el nivel de acceso y autorización que ofrezca al usuario en función del método de autenticación que este utilice, y para ello utilizará una técnica denominada escalado de seguridad.

Proceso de Login

El proceso de autenticación[14][15] entre el usuario y OpenAM cuando se utiliza una infraestructura PKI es el siguiente:

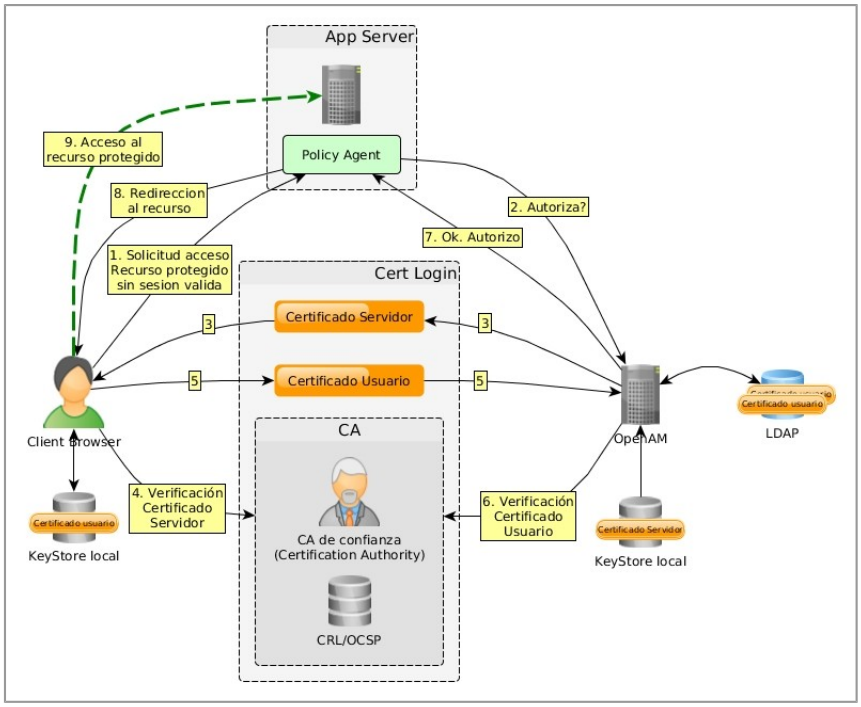

figura 20: Proceso de autenticación con certificados

Los pasos seguidos son:

1. El usuario intenta acceder a un recurso protegido y sin sesión válida.
2. El Policy Agent contacta con OpenAM y este le devuelve la URL de login, y el Policy Agent realiza un reenvío de la URL al navegador del cliente.
3. El navegador del cliente, al intentar abrir la URL de login de OpenAM (esta vez mediante protocolo HTTPS), recibe el certificado de OpenAM.

14 http://docs.oracle.com/javaee/1.4/tutorial/doc/J2EETutorial.pdf (página 1133)

4. El usuario valida con la CA de confianza (bien porque dispone del certificado en local y la cadena de certificación, o porque accede a la CA) y comprueba que el certificado del servidor es válido.
5. Si todo está OK, envía por el mismo canal al OpenAM su propio certificado.
6. OpenAM comprueba con la CA la validez del certificado del usuario, y lo identifica dentro del LDAP (localiza sus credenciales).
7. Si todo está OK y existe una política de autorización para ese recurso y usuario, envía "Allow" al Policy Agent.
8. El Policy Agent envía un reenvío HTTP al navegador del usuario hacia el recurso autorizado.
9. El usuario, finalmente, accede al recurso.

Nótese que, si el usuario ya dispone de una sesión activa y válida (véase figura 1. paso 8) en el entorno SSO, no se realiza el proceso de Logon y por tanto no se requiere el intercambio de certificados.

Observaciones

Es importante tener en cuenta que, para el proceso de intercambio de certificados entre el SSO y el usuario, será necesario el establecimiento de un canal seguro SSL/TLS, lo cual, además, requerirá que el servidor con el que se establezca el canal, esté identificado con otro certificado.

Por otro lado, en un entorno de producción, la infraestructura x.509 la debe proporcionar una PKI (Private Key Infraestructure).

Para nuestro cometido, de la PKI, necesitaremos, al menos, los siguientes elementos:

1. Una entidad certificadora de confianza que proporcione y firme los certificados de usuario. Y que será la que nos confirme la autenticidad de los certificados de usuario.
2. Una lista de revocación (CRL), o bien, un servicio OCSP[15][16] (Online Certificate Status Protocol) con el que verificaremos el estado de revocación de los certificados (si están o no revocados).
3. Un mecanismo que permita verificar la confianza en la entidad certificadora (CA), generalmente mediante la existencia de la cadena de certificación completa.

NOTA (y lectura recomendada)

La implementación de una PKI excede los límites de este proyecto, no obstante, para esta tarea, suponiendo que implementamos una PKI con el paquete RedHat Certificate System (su equivalente para CentOS) se recomiendan la siguiente lectura:

- `https://access.redhat.com/site/documentation/en-US/Red_Hat_Certificate_System/8.1/pdf/Deploy_and_Install_Guide/Red_Hat_Certificate_System-8.1-Deploy_and_Install_Guide-en-US.pdf`

Para ilustrar este apartado, se usarán certificados autofirmados generados mediante la herramienta OpenSSL (ver anexo III. "*Creación de certificados autorfirmados*" para la identificación de sitios web mediante OpenSSL).

15 https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

La problemática del Establecimiento del canal SSL

La mayoría de los navegadores web exigen, por razones obvias de seguridad, que el intercambio de certificados se realice a través de una conexión SSL establecida (protocolo HTTPS), por tanto, es necesario que cuando este accede a la URL de la aplicación, y se requiera la autenticación de usuario, OpenAM, o el intermediario entre este y el usuario (el proxy), la establezca previamente.

Esto quiere decir que, desde el punto de vista de la autenticación, mientras el usuario navegue por la aplicación por una zona pública, no es necesario que se establezca el canal seguro y, al menos, cuando se deba realizar la identificación del mismo, se establezca dicha conexión.

Ahora bien, en una arquitectura con uno o varios intermediarios, puede existir varias combinaciones posibles para el establecimiento del canal seguro:

- Desde el usuario hacia el proxy (en la DMZ), y desde la DMZ al proxy del entorno SSO (y los servidores SSO) en claro (sin SSL).
- Desde el usuario hacia el proxy del entorno SSO directamente (en LAN), y el Proxy Inverso[16][17] de la DMZ se limita a reenviar la conexión.
- Desde el usuario hasta el proxy de la DMZ, y desde el proxy hasta el proxy SSO (en la LAN) en dos canales SSL diferenciados.
- Establecer el canal SSL desde el usuario hasta cada uno de los propios servidores SSO directamente.
- Etc.

Y dependiendo de las necesidades de seguridad de la compañía, será más adecuada una u otra configuración, lo que, puede hacer que la implementación del entorno sea una tarea realmente compleja.

La arquitectura más habitual, es la que establece la sesión SSL desde los usuarios hacia el proxy de los servidores SSO, y configurar el proxy de la DMZ para realizar el reenvío del tráfico SSL.

En este punto, hay que valorar en qué forma afecta el uso de certificados para la identificación de los servidores al uso de la facilidad "User Session Failover", ya que si se empleara un canal "extremo a extremo" con los propios servidores SSO (no el proxy), el cambio de servidor SSO en caso de fallo obligaría al restablecimiento del canal SSL debido a que la identificación (certificado) del servidor cambia.

Adicionalmente, si el proxy debe aceptar conexiones seguras (uso del canal SSL hasta el proxy), ha de poder verificar los certificados (de OpenAM) con los que se establezca y cotejarlos con la CA de confianza, certificando que los interlocutores de la conexión SSL son quienes dicen ser.

Y todo esto, además, se ha de efectuar teniendo en cuenta que debe quedar integrado en el mismo proxy junto con las zonas de las aplicaciones que no requieran conexión cifrada (zonas públicas), bajo la misma URL.

Por otro lado, cuando un usuario, o incluso personal de IT/Desarrollo, requieren el acceso a la aplicación protegida por OpenAM desde una red segura (LAN), lo razonable desde el punto de vista de la seguridad, es que empleen también conexiones SSL con el proxy SSO, o directamente con los servidores SSO.

En nuestro caso, la configuración que valoraremos es aquella en la que la conexión SSL se realiza hasta el proxy SSO, con independencia del lugar desde el que se conecten los usuarios (LAN o WAN).

16 https://blogs.adobe.com/cguerrero/2010/10/27/configuring-a-reverse-proxy-with-apache-that-handles-https-connections/

Configuración

Para poder implementar la configuración descrita, debemos disponer de, al menos, los siguientes elementos:

1. Que OpenAM haya sido instalado y configurado para usar sesiones SSL, activando la opción SSL (en el wizard de instalación inicial).
2. Certificado de servidor válido para el entorno SSO (con la extensión Server Auth), firmados por la CA de nuestra PKI.
3. Certificados de usuario que incluyan la extensión "Client Auth" y estén debidamente firmados por nuestra CA o por otra CA de confianza, que deberán ser instalados en sus respectivos navegadores (usuarios), o bien, en algún contenedor de certificados seguro (ej, tarjeta criptográfica).

   Si permitimos que el usuario proporcione certificados válidos que no hemos firmado nosotros, por ejemplo el DNIe, deberemos tener en cuenta que:

   1. Nuestro OpenAM, para poder aceptarlos, debe ser capaz de comprobar su validez con las CA que los emitieron (CNMT o la dirección general de la policía), así como poder consultar las listas de revocación correspondientes, directamente, o mediante el protocolo OCSP.
   2. En el caso de que no aceptemos autenticación mediante DNIe, entonces, necesariamente deberemos ser nosotros quienes proporcionemos al usuario su correspondiente certificado firmado.

Adicionalmente, podemos decidir que, en el caso de que seamos nosotros quienes emitamos los certificados de usuario, sean almacenados en el repositorio de credenciales de OpenAM, por ejemplo, el LDAP.

También hemos de decidir qué elemento del certificado incluido en las credenciales de usuario en el servicio LDAP (OpenDJ) será la que utilicemos para buscar (Base DN) la información de credenciales del usuario (e incluso el mismo certificado almacenado en el LDAP).

Una vez disponemos de los certificados, los pasos a realizar serán:

1. Asegurar la conexión mediante SSL/TLS desde el proxy que proporciona el acceso, a partir de la cual, el usuario podrá enviar su propio certificado.
2. Configurar OpenAM[17][18] para que el proceso de autenticación de usuario, lo realice de la siguiente manera:
   1. Si el usuario presenta un certificado válido para el entorno (no está revocado, ni caducado, ha sido emitido y firmado por nuestra CA), se le permitirá el acceso (Allow).
   2. En caso de que exista algún tipo de problema con el certificado (esté caducado, revocado o simplemente no sea de confianza), se le ofrecerá la posibilidad de pasar al segundo módulo de autenticación , que es el clásico de usuario/contraseña.

Escalado de seguridad

Es posible que, en la aplicación J2EE, existan zonas (URLs) especialmente sensibles, o que, entre aplicaciones (ej, una aplicación de RRHH amparada dentro del mismo entorno SSO) requieran medidas adicionales de seguridad,

17 http://blog.profiq.cz/2012/05/24/certification-based-authentication-with-openam-10-and-tomcat-7/

por ejemplo, que obligatoriamente el usuario deba haberse identificado mediante certificado, no siendo válida la autenticación usuario/password.

OpenAM ofrece la posibilidad de que cada módulo de autenticación sea identificado mediante un número que indica el nivel de seguridad que queremos asociar a ese módulo de autenticación.

Para configurar esta opción, deberemos acceder al Realm correspondiente (el de nivel superior en nuestro caso), y el la solapa de "**Autenticación**", en el apartado de "**Instancias de módulo**", donde aparecen los módulos de autenticación disponibles para este Realm:

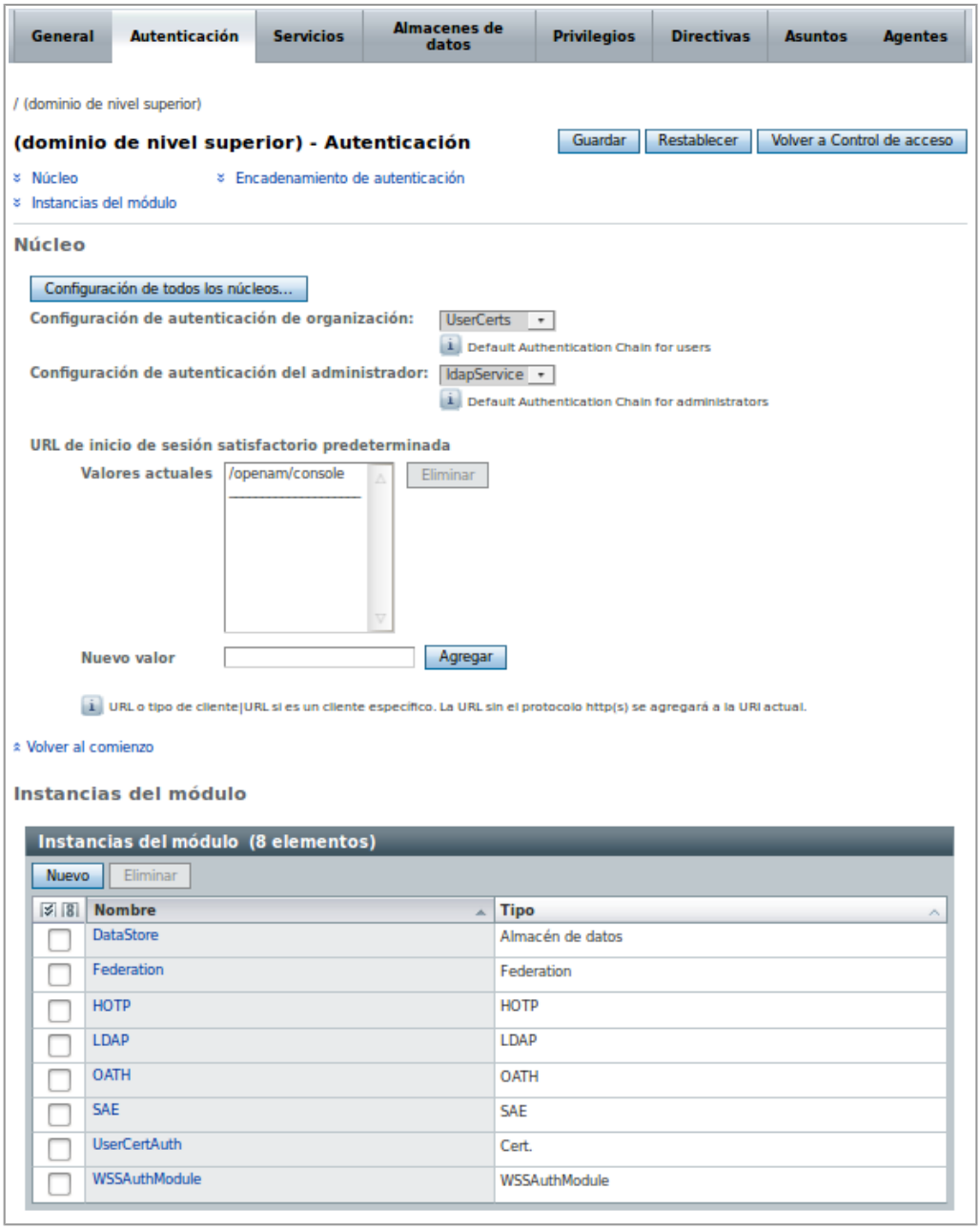

figura 21: Modulos de autenticación de un Realm. Escalado de seguridad

Donde seleccionaremos el módulo que nos interese.

Por ejemplo, para el caso del módulo "**Almacén de datos**" (*Datastore*) que es el módulo por defecto de OpenAM (el LDAP OpenDJ), únicamente permite configurar el nivel de autenticación:

figura 22: Modulo de autenticación. Nivel de autenticación.

No obstante, este parámetro se encuentra en todos los módulos de autenticación.

Funcionamiento

Una vez hemos asignado el valor del nivel de seguridad de los módulos de autenticación que emplearemos, haremos uso de ellos en la definición de las políticas de acceso a aplicaciones y/o recursos de las mismas.

Para ello, al definir una política (ver "*Creación de una política de seguridad*", pg.31), hemos de incluir la condición[18][19] de autorización de la política:

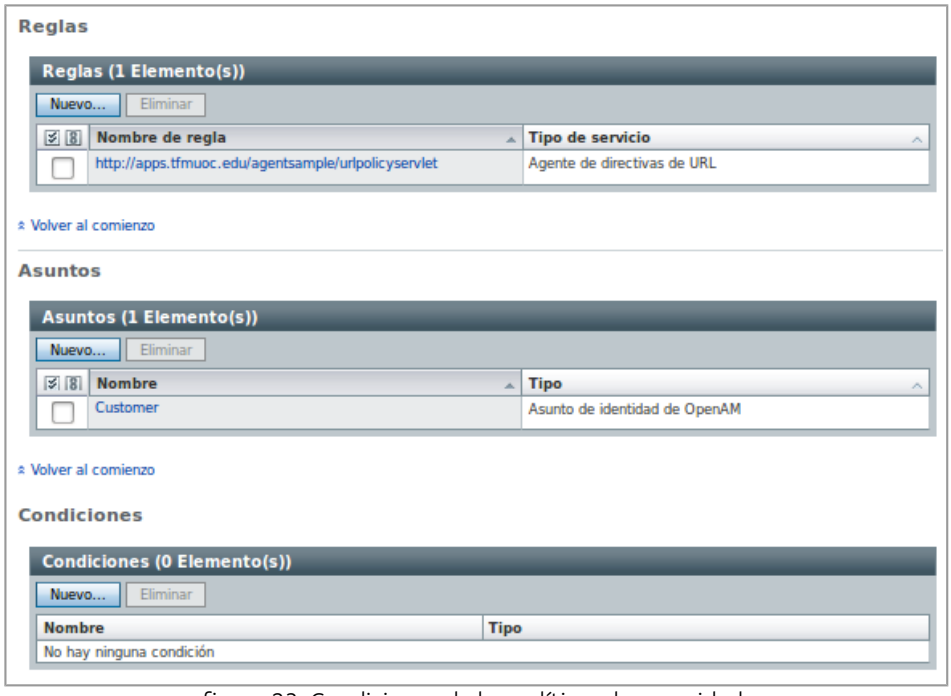

figura 23: Condiciones de las políticas de seguridad.

Estas condiciones deben cumplirse cuando los usuarios/grupos definidos en "**Asuntos**" quieren acceder al recurso definido en "**Reglas**", en caso contrario OpenAM devolverá un "Deny" al Policy Agent, que a su vez denegará el acceso al usuario (direccionando su navegador a la correspondiente página de error o denegación).

18 http://docs.oracle.com/cd/E19681-01/820-3885/gipxm/index.html

Para indicar la condición del nivel de autenticación para una política, agregaremos una condición nueva (pulsando en el botón "Nuevo") y obtendremos:

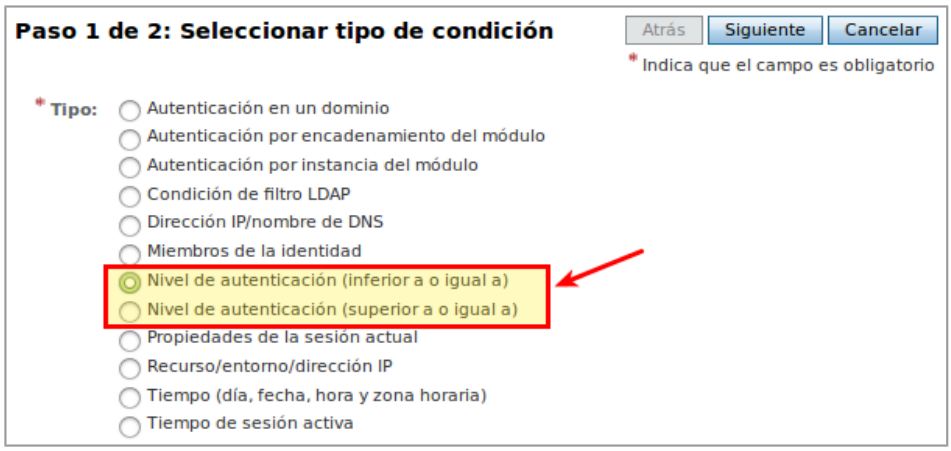

figura 26: Condiciones del nivel de autenticación

Al pulsar siguiente, nos permite terminar de configurar la condición:

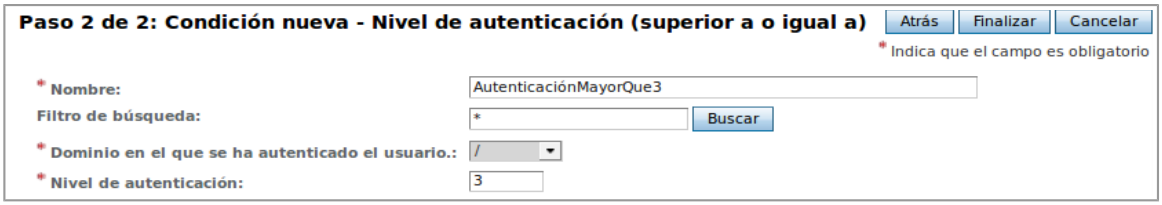

figura 24: Definición de los valores de la condición

Lo que indica que la condición se aplicará para el Realm que se especifica y sólo devolvería el valor "Allow" al Policy Agent si el módulo de autenticación utilizado por el usuario tiene definido un valor igual o superior a 3 (es imprescindible tener en cuenta que se deben haber definido los valores de autenticación de los módulos, en caso contrario, OpenAM siempre considera que el valor de un módulo es cero), y la condición definida queda:

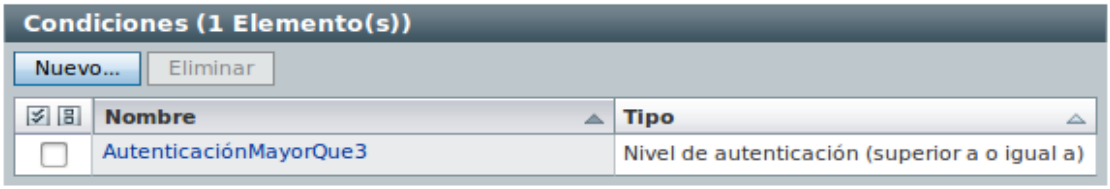

figura 25: Condición de política ya definida.

Cuando el usuario intente acceder al recurso a través de un módulo que no tiene suficiente nivel de autenticación, OpenAM rechazará su acceso, incluso aunque, como usuario/grupo disponga de autorización.

Un caso que puede ilustrar este tipo de técnicas, es el de los accesos remotos de usuarios a recursos sensibles, esto es, por ejemplo, a información protegida especialmente (ej, información médica), donde una autenticación básica con usuario/contraseña puede no resultar suficientemente segura.

Combinando condiciones

Es posible combinar varias condiciones, como por ejemplo, que el recurso sólo pueda accederse con módulos de autenticación de nivel bajo si la dirección IP del usuario pertenece a una red segura (ej, LAN).

Y se pueden definir varias políticas hacia el mismo recurso con diferentes condiciones, por ejemplo:

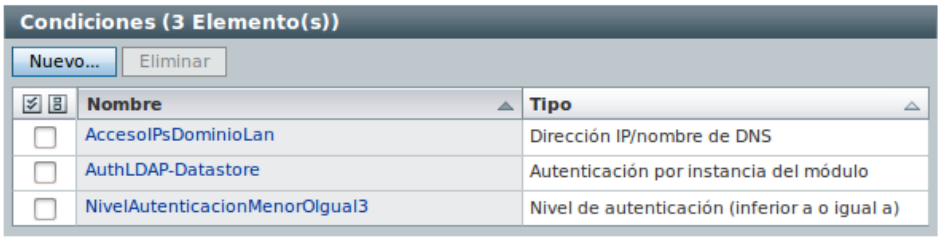

figura 27: Múltiples condiciones de una política

En este ejemplo, la política indica que cuando el usuario autorizado para acceder al recurso, acceda desde una dirección IP del rango definido en la condición, su autenticación haya sido realizada mediante el módulo DataStore (el servicio LDAP de OpenDJ que usa OpenAM) y el nivel de autenticación del modulo usado no sea superior a 3, OpenAM permitirá el acceso al recurso, y en cualquier otra condición, lo denegará.

Cuando el usuario no cumpla las condiciones, por ejemplo, del nivel de autenticación, y desee acceder al recurso, deberá caducar la sesión (cerrando el navegador) y creando las condiciones adecuadas para que el nivel de seguridad cambie adecuadamente, es decir, que debe salir del entorno SSO, y entrar, por ejemplo, con un módulo de autenticación más seguro (ej, certificados de usuario, siempre que hayamos informado dicho módulo con el nivel de seguridad adecuado).

Interacción OpenAM-Apps

Siempre que el usuario invoca las aplicaciones J2EE (Servlets) lo hace utilizando la interface **`HttpServletRequest`**, en este caso, si el usuario dispone de una sesión válida y ha sido autorizado (Allow), OpenAM deposita información de las credenciales de usuario en dicha solicitud utilizando para ello la Interface Java **`Principal`**[19][20], que se ha incluido del tro de **`HttpServletRequest`**, de forma que la aplicación podrá utilizarla, a través del método **`getUserPrincipal()`**, para realizar validaciones o cualquier otro tipo de operación.

Además, la clase HttpServletRequest[20][21] ofrece, también, otro método interesante: **`isUserInRole()`**, que retornará un Booleano indicando si el usuario pertenece, o no, a Rol determinado pasado por parámetro.

Este último detalle permite aún mayor flexibilidad en la aplicación ya que OpenAM le proporciona información de seguridad del usuario que puede ser utilizado para otros fines no contemplados en el propio OpenAM.

También es posible hacer que OpenAM ponga a disposición de la aplicación, a través del Policy Agent, más información sobre los atributos de sesión o de usuario en la cabecera HTTP, o en la cookie, tales como el nombre completo o la cuenta de correo electrónico.

Este tipo de información permite realizar customizaciones de usuario en la propia aplicación (ej, que la BD de la aplicación almacene preferencias de estilo,

19 http://docs.oracle.com/javase/1.5.0/docs/api/java/security/Principal.html
20 http://docs.oracle.com/javaee/5/api/javax/servlet/http/HttpServletRequest.html#getUserPrincipal%28%29

idioma, etc...).

Un ejemplo de código sobre cómo se obtiene información del Principal del usuario a través de la llamada HttpServletRequest:

```
import java.security.Principal;
import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServlet;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import java.io.IOException;
import javax.servlet.ServletException;

public class SecurityAwareServlet extends SampleServletBase {

    public void processRequest(HttpServletRequest request,
        HttpServletResponse response) throws ServletException, IOException
    {
        request.setAttribute("RESULT", "OK");
        request.setAttribute("DETAILS", getSecurityDetails(request));
        response.setContentType("text/html");
        RequestDispatcher dispatcher =
         request.getRequestDispatcher("/jsp/securityawareservletresult.jsp");
        dispatcher.forward(request, response);
    }

    private String getSecurityDetails(HttpServletRequest request) {
        Principal principal = request.getUserPrincipal();
        String user = (principal != null)?principal.toString():"Anonymous";
        boolean isManager = request.isUserInRole("MANAGER_ROLE");
        boolean isEmployee = request.isUserInRole("EMPLOYEE_ROLE");

        StringBuffer buff = new StringBuffer();
        buff.append("The User \"").append(user).append("\" is ");
        if (!isManager) {
            if (!isEmployee) {
                buff.append(" neither ");
            } else {
                buff.append(" not ");
            }
        }
        buff.append("a manager");
        if (isEmployee) {
            if (!isManager) {
                buff.append(", but is an employee.");
            } else {
                buff.append(" and also an employee.");
            }
        } else {
            buff.append(" nor an employee.");
        }

        return buff.toString();
    }
}
```

En las líneas marcadas en rojo se realiza la instanciación del objeto **principal** (de la clase Principal) que recoge toda la información de identidad del usuario ofrecida por OpenAM a través del Policy Agent.

En pasos posteriores se permite realizar las operaciones específicas (customización) a partir de la información del Principal del usuario que se ha obtenido.

Otras funcionalidades de OpenAM

Es obvio que el el paquete OpenAM presentas muchas más funcionalidades que las descritas en el presente documento y que son dignas de mención:

- Soporte de federación[21] de identidades, a través del uso del protocolo SAML 2.0, per permite reconocer identidades de otros dominios no gestionados por OpenAM, como por ejemplo, Google Apps.
- ForgeRock, desarrolladora de OpenAM, ofrece un completo API de desarrollo para la extensión de funcionalidades tales como, por ejemplo, el desarrollo de módulos de autenticación a medida[22] (ej. usando biometría).
- Interacción con dispositivos externos para la autenticación fuerte (OATH[23]). Este módulo permite interactuar con dispositivos de generación de tokens OATH (ej., YubiKeys o software iOS/Android) para que la autenticación de usuario se realice en dos fases, la primera y más tradicional (ej, usr/pwd) y una segunda mediante el token OATH.
- Integración con los más importantes servidores J2EE del mercado.
- Integración con entornos Cloud, aunque la versión 10.1 sólo incluía capacidad para integrarse con Amazon EC2. Sin embargo, al finalizar el presente documento ForgeRock ha publicado la versión 11 que parece extender esta capacidad a otros entornos cloud.

Y finalmente, al tratarse de un producto abierto, el potencial de crecimiento y expansión, a medio/largo plazo, no tiene límites.

Open Identity Stack de ForgeRock

Y por último, creo que es interesante hacer mención a los paquetes de la misma compañía, ForgeRock que refuerzan y extienden la funcionalidad de OpenAM y OpenDJ (tratados en el presente trabajo), tales como:

- OpenIDM. Es un producto que permite integrar en una única plataforma los procesos de gestión (altas/bajas y manteniemiento) de usuarios e identidades de los diferentes entornos de una compañía, para luego poder ser utilizados desde OpenAM.
- OpenICF. Es un framework para el desarrollo de conectores, proporcionando una capa genérica entre las aplicaciones J2EE protegidas por OpenAM y recursos foráneos.
- OpenIG. Se trata de un proxy especializado para la gestión de identidades en DMZ, permitiendo extender los entornos federados y los sistemas clásicos SSO.

21 http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/admin-guide/index/chap-federation.html
22 http://rabidwoodpecker.blogspot.com.es/2010/06/implementing-custom-openam.html
23 https://wikis.forgerock.org/confluence/display/openam/Configure+OpenAM+to+use+OATH+for+Strong+Authentication

Conclusiones

Frente a los mecanismos de autenticación clásicos, embutidos dentro de las propias aplicaciones Web/J2EE, OpenAM nos abre un tremendo potencial de gestión e integración de identidades en un punto único, permitiendo unificar en un sólo paquete, todas las identidades de una compañía.

Las grandes ventajas que obtenemos de esta integración son:

- Entorno Single Sign On, es decir, autenticación única de usuario para acceder a todas las aplicaciones protegidas por OpenAM, o dicho de otro modo, la gestión unificada de las sesiones de usuario dentro de todo el entorno SSO (teniendo en cuenta que se mantienen coockies por separado para la aplicación y el Single Sign On).
- Granularidad de acceso, es decir, permite definir, incluso a nivel de URL, qué acceso debe tener cada usuario/grupo
- Definir múltiples condiciones de acceso, esto es, permite establecer bajo qué condiciones de seguridad pueden ser accedidas las aplicaciones y/o recursos.
- Integración de los mecanismos de autenticación, dicho de otro modo, en una organización en la que, históricamente, existen varias maneras de autenticar usuarios (ej, Directorio Activo y LDAP), OpenAM permite utilizar repositorios de credenciales ajenos como si fueran propios, integrádolos todos en una única plataforma.
- Da capacidad de escalabilidad, permitiendo agregar nuevos módulos de autenticación, o incluso el desarrollo de módulos a medida.
- Permite desligar la aplicación J2EE del proceso de autenticación de usuarios, cediendo todo el control sobre OpenAM, pero también permite que la aplicación reciba información sobre las credenciales de usuario para uso propio.
- Aunque no ha podido ser evaluado en el presente documento, también permite la federación de identidades de diferentes entornos/dominios (ej. Google Apps), así como el uso de estándares como el protocolo SAML para el intercambio de información de seguridad con entidades que lo soporten.

En definitiva, se trata de una muy potente herramienta de seguridad y control de acceso, que permite que una compañía pueda crecer en su número de usuarios (ej, clientes) de forma ordenada en todas sus aplicaciones, así como la capacidad de integrar servicios ajenos dentro de su propio entorno.

Sin embargo, hay que considerar que la arquitectura necesaria para poder realizar el despliegue de un entorno SSO es más compleja que con los entornos J2EE tradicionales, ya que requiere, para su correcto funcionamiento, de:

- Un buen y correcto funcionamiento de un servicio de nombres (DNS). Es imprescindible que OpenAM identifique las URL/URIs de todos los recursos que debe proteger, en caso contrario, no autorizará el acceso.
- La integración de todos los servicios bajo un mismo nombre de dominio (URI), lo que implica necesariamente, el uso de balanceadores de carga/proxies inversos que permiten la integración bajo un mismo nombre de diferentes servicios J2EE (aplicaciones y SSO).
- Y en el caso de uso de una PKI para autenticación de usuarios, requiere, adicionalmente, el despliegue de un buen número de servicios que deben estar perfectamente integrados para su correcto funcionamiento.

Objetivos conseguidos

El objetivo principal de este proyecto consistía en la implantación de un entorno Single Sign On basado en OpenAM, que ha consistido en:

- Diseño de la arquitectura física necesaria (9 máquinas virtuales en su configuración final), y su implementación a través de un entorno virtualizado basado en VirtualBox.

  Incluso se han planteado, en algunos casos, opciones alternativas (ej, proxies).

- Intalación y configuración del software base (S.O., servicios anexos como DNS, Firewall, Proxies, NFS, Bds, Servidores J2EE, etc...).
- Despliegue y configuración del entorno SSO con OpenAM para la protección de 4 aplicaciones J2EE.
- Seguimiento y verificación de su funcionamiento.

Objetivos no conseguidos

La causa principal de esta desviación es el tamaño y la complejidad del proyecto, sumado a mi falta de experiencia en el despliegue de algunos de los elementos que, por otro lado, no tienen relación directa con el proyecto (pero si necesaria), por ejemplo, el despliegue de los proxies.

Estas circunstancias, ya tratadas con el director de este trabajo, hacen que el proyecto sólo haya podido completarse en la fase de despliegue del sistema Single Sign On, quedando fuera la implementación práctica de los siguientes puntos:

- Autenticación de usuarios mediante certificados, implementación de una PKI para poder ser usada desde el entorno SSO.
- Profundización (desde el punto de vista de desarrollo) del acceso de las aplicaciones J2EE a la información (Realm) de identidad de los usuarios gestionada por OpenAM (Principal).

No obstante, ambos puntos han sido tratados, al menos en su aspecto teórico, dentro del presente documento.

Posibles ampliaciones

Las propuestas de ampliación, que, obviamente, pueden incluir los objetivos no cumplidos, pueden ser:

- Autenticación de usuarios mediante certificados, a través de una PKI, por ejemplo, la implementada mediante el paquete: Fedora Certificate System (versión gnu del paquete del mismo nombre de RedHat), que se puede descargar de: http://pki.fedoraproject.org/wiki/PKI_Main_Page
- Una ampliación también muy interesante, sería explotar todos los mecanismos de federación con entornos foráneos (Google Apps, Facebook, etc...).
- Desarrollar algún tipo de aplicación J2EE orientada al uso intensivo de los servicios de OpenAM, tales como la customización de la aplicación para usuarios, etc....
- Desarrollar algún módulo de autenticación a medida utilizando el API proporcionado por ForgeRock (autores de OpenAM).
- Configuración avanzada de OpenAM combinando todo lo anterior.

Referencias bibliográficas y Citas

[1] - ForgeRock, Inc.: "**Single sign on, the basic concepts**" - http://blogs.forgerock.org/petermajor/2013/02/single-sign-on-the-basic-concepts/

[2] - Universitat Oberta de Catalunya, **Modulo 4. Single Sign-On y federación de identidades**

[3] - Wikipedia: "**Single Sign On**" - https://en.wikipedia.org/wiki/Single_sign-on

[4] - Miquel Trilla: "**Single Sign On**" - http://studies.ac.upc.edu/FIB/CASO/seminaris/1q0304/T7.ppt

[5] - Microsoft: "**Understanding Enterprise Single Sign-On**" - http://msdn.microsoft.com/en-us/library/aa745042%28v=bts.10%29.aspx

[6] - Courion Corporation: "**Enterprise Single Sign-on**" - http://www.courion.com/products/enterprise-single-sign-on.html

[7] - Dave: "**Load balancer persistence (sticky sessions)**" - http://web.archive.org/web/20090302072037/http://daveonsoftware.blogspot.com/2007/08/load-balancer-persistence-sticky.html

[8] - ForgeRock, Inc.: "**User Session Failover**" - http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/install-guide/index/chap-session-failover.html

[9] - Wikipedia: "**OpenAM**" - https://en.wikipedia.org/wiki/OpenAM

[10] - InternetNews: "**ForgeRock Extending Sun's OpenSSO Platform**" - http://www.internetnews.com/dev-news/article.php/3881681/ForgeRock+Extending+Suns+OpenSSO+Platform.htm

[11] - Oracle-Sun: "**Installing And Configuring the OpenSSO Enterprise Policy Agent on Identity Manager**" - http://docs.oracle.com/cd/E19316-01/820-4729/ggtzd/index.html

[12] - Danny Coward & Yutaka Yoshida: "**Java Servlet SpecificationVersion 2.4**" - http://download.oracle.com/otn-pub/jcp/servlet-2.4-fr-spec-oth-JSpec/servlet-2_4-fr-spec.pdf

[13] - Oracle-Sun: "**A web.xml Deployment Descriptor Elements**" - http://docs.oracle.com/cd/E14571_01/web.1111/e13712/web_xml.htm#autoId0

[14] - ForgeRock, Inc.: "**OpenAM 10.1.0 Administration Guide**" - http://docs.forgerock.org/en/openam/10.1.0/OpenAM-10.1.0-Admin-Guide.pdf

[15] - Sun Microsystems, Inc.: "**The J2EETM 1.4 Tutorial**" - http://docs.oracle.com/javaee/1.4/tutorial/doc/J2EETutorial.pdf

[16] - Wikipedia: "**Online Certificate Status Protocol**" - https://en.wikipedia.org/wiki/Online_Certificate_Status_Protocol

[17] - Carlos Guerrero: "**Configuring a Reverse Proxy with Apache that handles HTTPS connections**" - https://blogs.adobe.com/cguerrero/2010/10/27/configuring-a-reverse-proxy-with-apache-that-handles-https-connections/

[18] - N4A L: "**Certificate based authentication with OpenAM 10 and Tomcat 7**" - http://blog.profiq.cz/2012/05/24/certification-based-authentication-with-openam-10-and-tomcat-7/

[19] - Oracle-Sun: "**Sun OpenSSO Enterprise 8.0 Administration Guide - Conditions**" - http://docs.oracle.com/cd/E19681-01/820-3885/gipxm/index.html

[20] - Sun Microsystem, Inc.: "**Interface java.security.Principal**" - http://www.cis.upenn.edu/~bcpierce/courses/629/jdkdocs/api/java.security.Principal.html

[21] - Sun Microsystems, Inc: "**J2EE API. Servlets**" - http://docs.oracle.com/javaee/5/api/javax/servlet/http/HttpServletRequest.html

Documentación adicional

Para la elaboración del presente trabajo se han empleado, de forma intensiva, los siguientes documentos:

- OpenAM 10.1.0 Installation Guide
- OpenAM 10.1.0 Administration Guide
- Policy Agent 3.1.0-Xpress Installation Guide

Todos ellos descargables desde la web:

http://docs.forgerock.org/en/index.html?product=openam&version=10.1.0

Adicionalmente, para completar los manuales indicados, se ha utilizado la documentación de referencia de OpenSSO (de Oracle), usando los siguientes documentos:

- Sun OpenSSO Enterprise 8.0 Technical Overview
- Y alguna referencia puntual de los manuales:
    - Sun OpenSSO Enterprise 8.0 Installation and Configuration Guide
    - Sun OpenSSO Enterprise 8.0 Administration Guide

Todos descargables de: http://docs.oracle.com/cd/E19681-01/

ANEXOS

A continuación se han incluido dos documentos que, sin ser aspectos directamente relacionados con la configuración de OpenAM, son necesarios para poder realizar su despliegue o para comprender su funcionamiento, los cuales son:

- Despliegue de la infraestructura necesaria: donde se describe el proceso de instalación de la infraestructura, detallando aquellos aspectos que son más importantes y sin los cuales, el servicio SSO ofrecido por OpenAM no funcionaría.

  Téngase en cuenta que la configuración de la infraestructura para el desarrollo del presente trabajo ha sufrido constantes modificaciones a lo largo del desarrollo del proyecto, fruto de la inexperiencia, y ha ido adecuándose, en cada momento, a las funcionalidades de OpenAM y el entorno J2EE.

- Test de funcionamiento: donde se ilustra, desde el punto de vista del usuario, todo el proceso de intercambios de cabeceras HTTP, creación de cookies y redirecciones que aplica el Policy Agent para poder interactuar con OpenAM.

  En este caso, también ha de considerarse que el anexo dispone de dos grandes partes, la primera, correspondiente al test realizado con un sólo servidor de aplicaciones J2EE contra un servicio SSO en LAN (sin pasar por el Firewall.

  La segunda fase corresponde ya al entorno completamente desplegado e integrado en el proxy de aplicación en la DMZ.

Despliegue de la infraestructura necesaria

En este anexo se describe la infraestructura necesaria para poder desplegar un entorno SSO basado en OpenAM usando como servidor de aplicaciones J2EE el servidor Apache Tomcat.

Arquitectura

Lectura recomendada

Antes de proceder a la diseño e implementación de la arquitectura necesaria para un entorno J2EE asegurado con OpenAM, se recomienda encarecidamente la lectura de estos tres documentos:

- http://java.net/downloads/opensso/docs/architecture/auth_arch.pdf
- http://docs.huihoo.com/opensso/j2eeagent_arch.pdf
- http://docs.oracle.com/cd/E19575-01/820-3740/820-3740.pdf

En el que se describe el funcionamiento del proceso de autenticación y de cómo es utilizado por el Policy Agent (en este caso de OpenSSO, precursor de OpenAM).

A partir de esta lectura, se realiza el diseño adecuado, teniendo en cuenta las limitaciones de recursos hardware inherentes al desarrollo del presente master.

Como se ha observado, la interacción entre los distintos elementos, desde que

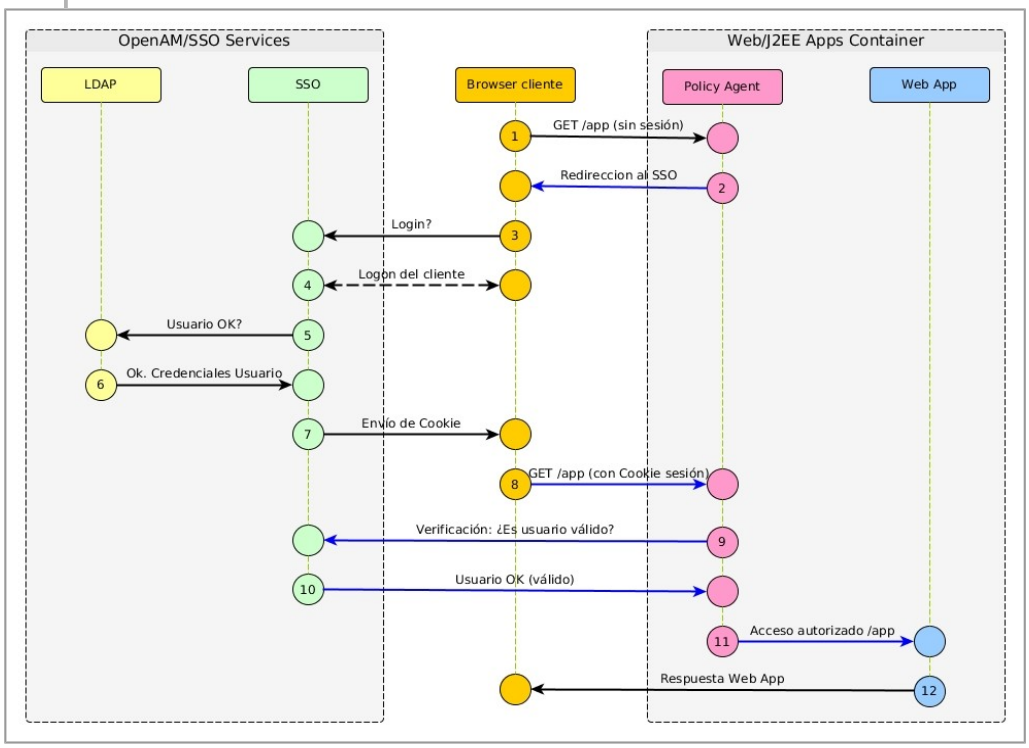

figura 28: Interacción entre el usuario y el entorno protegido por OpenAM

el usuario requiere el acceso a una URL hasta que lo obtiene o se le deniega, es la siguiente:

Lo que implica que, desde un punto de vista arquitectónico, necesitaremos lo siguiente:

- Un contenedor de aplicaciones J2EE
  - En nuestro caso, utilizaremos Apache Tomcat (v.6.0) y una aplicación J2EE de Test que se ofrece en el la propia instalación del Policy Agent de OpenAM.
- Un servicio OpenSSO
  - Que se compone de una o varias instancias de OpenAM, como servicio de autenticación/autorización, y una o varias instancias de persistencia, tanto para la configuración del propio OpenAM, como para las credenciales de los usuarios (pueden ser independientes).

    En nuestro caso, para este último elemento, usaremos OpenDJ, que es un servicio LDAP, que usaremos para ambos cometidos (persistencia de configuración y credenciales de usuarios).

Todo esto, en un entorno de producción, debe estar convenientemente protegido mediante los mecanismos

habituales de seguridad en red, es decir, que requeriremos, al menos, los siguientes elementos:

- Zonas de red diferenciadas.
  - Zona desmilitarizada: que contendrá los servicios intermediarios (proxies y DNS) que permitan el acceso externo (usuarios públicos) al servidor de aplicaciones y entorno SSO.
  - Zona segura: que contendrá el resto de elementos (Contenedor J2EE y servicio SSO).
  - Firewall: que proteja el acceso a todo lo anterior.

Por tanto, la arquitectura resultante será:

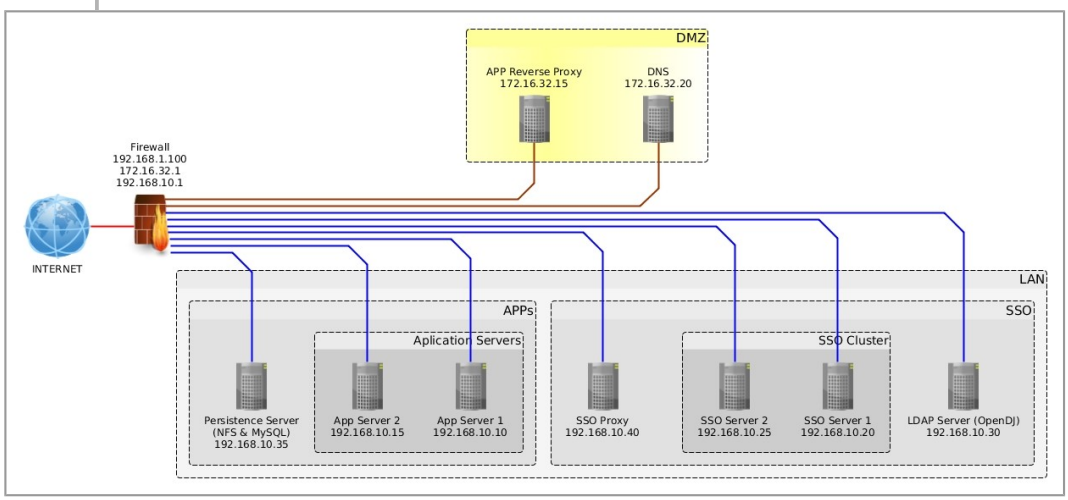

figura 29: Arquitectura física

Y si ahora damos un vistazo al modo en que interactuan los elementos del entorno, tendremos:

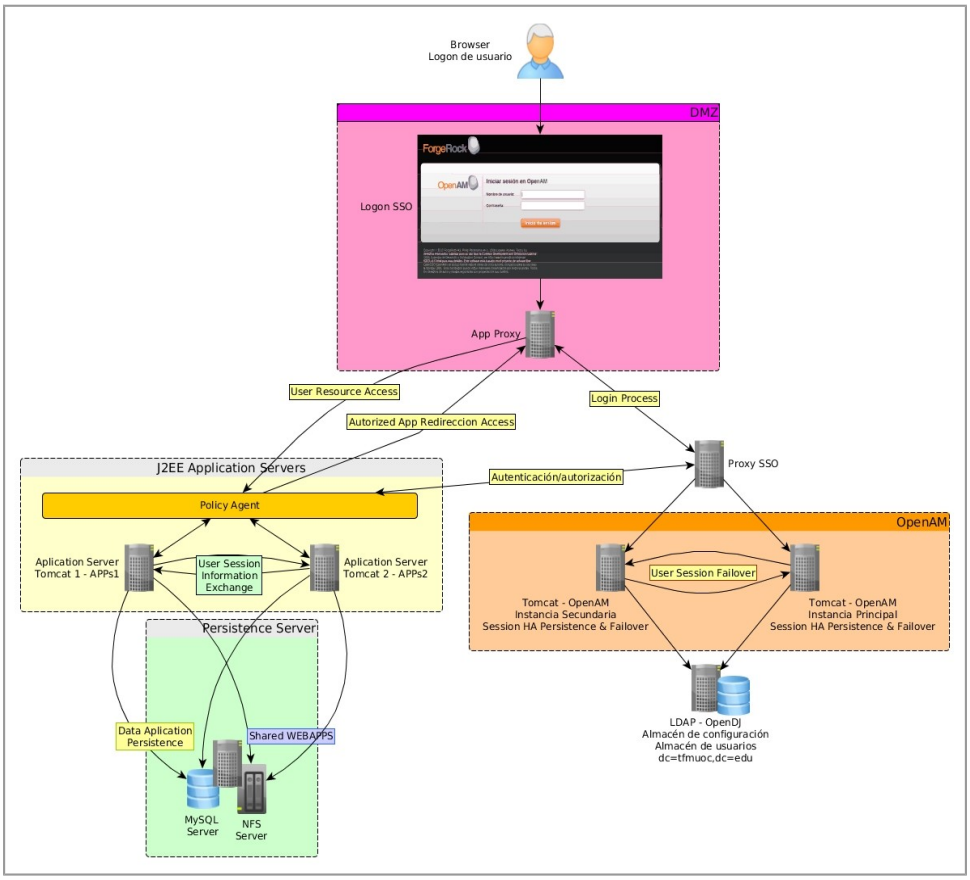

figura 30: Interacción física en el entorno SSO

A partir de este punto, en el que ya tenemos cómo debe ser nuestro entorno, procedemos a su despliegue.

El software que usaremos será:

- S.O. del servidor anfitrión (físicos): **Linux Mint x86_64 R.15**
- Servicio de virtualización: **Oracle Virtual Box 4.2**
- S.O. de servidores virtuales: **CentOS 5.9 x86_64**
- Sistema SSO: **OpenAM 10.1**
- Servidores de aplicación: **Tomcat 6.0** (aplicación Web y software de directorio y SSO).
- Servidor de directorio: **OpenDJ 2.6**
- DNS: **Bind 9.8**
- Firewall: **Iptables 1.3** (del propio kernel del Linux) y **Firewall Builder 5.1** (GUI para Iptables).
- Proxies: **Apache 2.4 y HAProxy 1.4**

Servicio de virtualización

Todas las máquinas se implementarán en el mismo entorno de virtualización, en el mismo servidor físico. Para poder independizar las redes de cada máquina virtual, se les ha agregado tantas interfaces como redes se requieren según la figura 4, y se han configurado como redes privadas (que funcionan a modo de VLAN) de forma que no es posible que las interfaces de dos redes privadas

diferentes se vean.

El direccionamiento IP utilizado es:

Maquina	Nombre DNS	if	IP	Net Mask	Gateway
Firewall	fw.tfmuoc.edu	eth0 eth1 eth2	192.168.1.100 172.16.32.1 192.168.10.1	255.255.255.0 255.255.255.0 255.255.255.0	192.168.1.20
Reverse proxy Apps	rpapps.tfmuoc.edu	eth0	172.16.32.15	255.255.255.0	172.16.32.1
DNS	dns.tfmuoc.edu	eth0	172.16.32.20	255.255.255.0	172.16.32.1
Application Server 1	apps1.tfmuoc.edu	eth0	192.168.10.10	255.255.255.0	192.168.10.1
Application Server 2	apps2.tfmuoc.edu	eth0	192.168.10.15	255.255.255.0	192.168.10.1
Persistence Server	ps.tfmuoc.edu	eth0	192.168.10.35	255.255.255.0	192.168.10.1
Reverse proxy SSO	lrpsso.tfmuoc.edu	eth0	192.168.10.40	255.255.255.0	192.168.10.1
SSO Server 1	sso1.tfmuoc.edu	eth0	192.168.10.20	255.255.255.0	192.168.10.1
SSO Server 2	sso2.tfmuoc.edu	eth0	192.168.10.25	255.255.255.0	192.168.10.1
LDAP	ldap.tfmuoc.edu	eth0	192.168.10.30	255.255.255.0	192.168.10.1

WAN: Firewall

Se trata de una máquina bastionada, con la instalación mínima del sistema operativo y que utiliza el servicio de filtrado de paquetes nativo en Linux: IPTABLES, y para facilitar su gestión, se utiliza el frontal de configuración: Firewall Builder.

DMZ: Servicio de nombres (DNS)

Como se indicó, para el servicio de nombres, utilizaremos el estándar de facto Bind (de la Internet System Consortium. http://www.isc.org/), que se denomina **named**, y es un servicio que se incluye dentro de los repositorios estándar de Linux.

El dominio sobre el que se desarrollará el laboratorio es: tfmuoc.edu

Se trata de un dominio ficticio no registrado, pero que cubrirá todas las necesidades requeridas para este mismo laboratorio. En condiciones de producción reales, será necesario que el dominio esté registrado en Internet en cualquiera de los proveedores de registro de nombres que existe (ej, por ejemplo, Register.com[24]).

Los nombres de servicio DNS asignados a cada servidor son los que se indican en la tabla de la página anterior. El direccionamiento IP para cada uno de estos servidores aparece en la tabla definida en el apartado "Configuración de red de las máquinas de la LAN/DMZ".

Adicionalmente debemos considerar que nuestro DNS ha de ser capaz de reenviar (Forward) todas las peticiones recibidas hacia otro DNS, en este caso público. Los DNS de forwarding elegidos son los correspondientes a Google (8.8.8.8 y 8.8.4.4), por tanto, la configuración queda de la siguiente manera:

```
options
{
        directory "/var/named"; // the default
        dump-file               "data/cache_dump.db";
        statistics-file         "data/named_stats.txt";
```

24 https://www.register.com/index.rcmx?ab=true

```
        memstatistics-file      "data/named_mem_stats.txt";
        forwarders { 8.8.8.8; 8.8.4.4; };
        allow-recursion { 192.168.10.0/24; 172.16.32.0/24; };
        version "Forbidden";
};
logging
{
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};
key ddns_key
{
        algorithm hmac-md5;
                                                                    secret
"souIkbHA8i5TGYmbNdcqdFhtVV3Viyn9ScvYuInBiZ4fziaJymdzl3axBzxP";
};
zone "tfmuoc.edu" in {
        file "masters/tfmuoc.edu.zone";
        type master;
};
zone "1.168.192.in-addr.arpa" in {
        file "masters/1.168.192.in-addr.arpa.zone";
        type master;
};
```

Y la zona de resolución directa de tfmuoc.edu es:

```
$ORIGIN .
$TTL 1h
tfmuoc.edu              IN              SOA             dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                        2013041904      ; serial
                                        7200            ; refresh
                                        1800            ; retry
                                        100             ; expiry
                                        100 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
$ORIGIN tfmuoc.edu.
dns             1       IN      A       192.168.1.100
rpapps          1       IN      A       192.168.1.100
```

Y las de resolución inversa son:

```
$ORIGIN .
$TTL 1h
;Definiciones de la zona inversa de tfmuoc.edu
1.168.192.in-addr.arpa      IN            SOA          dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                        2006060000      ; serial
                                        600             ; refresh
                                        120             ; retry
                                        3600            ; expiry
                                        600 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
                        IN      A       192.168.1.100
$ORIGIN 1.168.192.in-addr.arpa.
100                     IN      PTR     dns.tfmuoc.edu.
100                     IN      PTR     rpapps.tfmuoc.edu.
```

Ahora sólo queda adaptar las reglas del Firewall para que el DNS sea accesible desde el exterior y desde las máquinas del laboratorio.

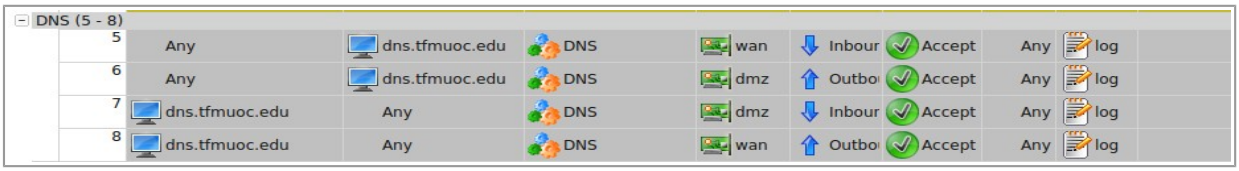

figura 31: Reglas de acceso al servicio DNS

La resolución de nombres de la DMZ y la LAN debería realizarse con un DNS desplegado en la LAN, pero no ha sido implementado en este proyecto por falta de recursos hardware. La solución pasaría, en este caso, con informar convenientemente todos los ficheros **/etc/hosts** de cada servidor indicando todas las Ips necesarias para el acceso a todos los servicios del entorno.

De todos modos, y en aras de la comodidad, durante el proceso de despliegue y test, el servicio DNS se ha configurado con todas las zonas (LAN, DMZ y WAN) de forma que, desde mi portátil, pudiera acceder a todos los recursos, manteniendo un único punto de definición de nombres.

La configuración y las zonas dns utilizadas durante el despliegue son:

```
options {
        directory "/var/named"; // the default
        dump-file               "data/cache_dump.db";
        statistics-file         "data/named_stats.txt";
        memstatistics-file      "data/named_mem_stats.txt";
        forwarders { 8.8.8.8; 8.8.4.4; };
        allow-recursion { 192.168.10.0/24; 172.16.32.0/24; };
        version "Forbidden";
};
logging {
        channel default_debug {
                file "data/named.run";
                severity dynamic;
        };
};
key ddns_key {
        algorithm hmac-md5;
secret
"souIkbHA8i5TGYmbNdcqdFhtVV3Viyn9ScvYuInBiZ4fziaJymdzl3axBzxP";
};
zone "tfmuoc.edu" in {
        file "masters/tfmuoc.edu.zone";
        type master;
};
zone "10.168.192.in-addr.arpa" in {
        file "masters/10.168.192.in-addr.arpa.zone";
        type master;
};
zone "1.168.192.in-addr.arpa" in {
        file "masters/1.168.192.in-addr.arpa.zone";
        type master;
};
zone "32.16.172.in-addr.arpa" in {
        file "masters/32.16.172.in-addr.arpa.zone";
        type master;
};
```

Y la zona de resolución directa de tfmuoc.edu es:

```
$ORIGIN .
$TTL 1h
tfmuoc.edu              IN              SOA           dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                        2013041904      ; serial
                                        7200            ; refresh
                                        1800            ; retry
                                        100             ; expiry
                                        100 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
$ORIGIN tfmuoc.edu.
dns             1       IN      A       192.168.1.100
rpapps          1       IN      A       192.168.1.100
dns-dmz         1       IN      A       172.16.32.20
rpapps          1       IN      A       172.16.32.15
apps1           1       IN      A       192.168.10.10
apps2           1       IN      A       192.168.10.15
ps              1       IN      A       192.168.10.35
lrpsso          1       IN      A       192.168.10.40
ssos1           1       IN      A       192.168.10.20
ssos2           1       IN      A       192.168.10.25
ldap            1       IN      A       192.168.10.30
```

Y las de resolución inversa son:

```
$ORIGIN .
$TTL 1h
;Definiciones de la zona inversa de tfmuoc.edu
10.168.192.in-addr.arpa      IN           SOA         dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                        2006060000      ; serial
                                        600             ; refresh
                                        120             ; retry
                                        3600            ; expiry
                                        600 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
                        IN      A       172.16.32.20

$ORIGIN 10.168.192.in-addr.arpa.
10                      IN      PTR     apps1.tfmuoc.edu.
15                      IN      PTR     apps2.tfmuoc.edu.
20                      IN      PTR     ssos1.tfmuoc.edu.
25                      IN      PTR     ssos2.tfmuoc.edu.
30                      IN      PTR     ldap.tfmuoc.edu.
35                      IN      PTR     ps.tfmuoc.edu.
40                      IN      PTR     lrpsso.tfmuoc.edu.
```

```
$ORIGIN .
$TTL 1h
;Definiciones de la zona inversa de tfmuoc.edu
1.168.192.in-addr.arpa      IN           SOA         dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                        2006060000      ; serial
                                        600             ; refresh
                                        120             ; retry
                                        3600            ; expiry
                                        600 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
                        IN      A       172.16.32.20
```

```
$ORIGIN 1.168.192.in-addr.arpa.
100                     IN      PTR     dns.tfmuoc.edu.
100                     IN      PTR     rpapps.tfmuoc.edu.
```

```
$ORIGIN .
$TTL 1h
;Definiciones de la zona inversa de tfmuoc.edu
32.16.172.in-addr.arpa      IN            SOA          dns.tfmuoc.edu.
dnsadmin.tfmuoc.edu. (
                                          2006060000      ; serial
                                          600             ; refresh
                                          120             ; retry
                                          3600            ; expiry
                                          600 )           ; minimum
                        IN      NS      dns.tfmuoc.edu.
                        IN      A       172.16.32.20

$ORIGIN 32.16.172.in-addr.arpa.
15                      IN      PTR     rpapps.tfmuoc.edu.
20                      IN      PTR     dns.tfmuoc.edu.
```

DMZ: Servicio de Proxy

A lo largo del proyecto, y debido a la falta de experiencia, se han ido probando diferentes combinaciones de proxies, teniendo en cuenta todas las circunstancias de URLs de acceso, políticas e integración con OpenAM, especialmente en el apartado de redirección del Policy Agent cuando este debe presentar la página de Login/Logout/error, etc... enviada desde OpenAM.

Se ha podido observar la importancia de una correcta resolución de nombres de dominio, así como la integración de todos los recursos J2EE, es decir, las aplicaciones y el entorno SSO, para el correcto funcionamiento de todo el entorno.

Las combinaciones de proxy probadas son las siguientes:

- Apache con el módulo mod_jk explicada en el apartado anterior que, si bien funciona, no es especialmente estable ya que a veces no es capaz de recuperar los procesos de Worker del balanceador de carga cuando uno de los servicios balanceados falla.
- Y la tercera combinación se realiza, de nuevo, con Apache, pero con el módulo mod_proxy_ajp. Esta opción que también usa el protocolo AJP (de forma similar mod_jk), y permite publicar exclusivamente los recursos que deben ser visibles desde Internet, mostrando un error de inexistencia cuando se intenta acceder a recursos no publicados.

  Esta tercera opción es la que, finalmente, he utilizado para dar acceso al servidor de aplicaciones J2EE, y de paso, para integrar el acceso al servicio SSO.

  Es obvio que esta solución permite que, bajo el mismo nombre DNS sean accedidos, tanto las aplicaciones del entorno J2EE, como el entorno SSO, que puede publicar el nombre del proxy de la LAN o directamente los dos servidores SSO (opción elegida).
- HAProxy tanto en el entorno SSO como en el entorno APPs, esta solución, explicada en el apartado “Balanceador de carga (HAProxy) descrito al final del presente anexo, ha demostrado ser muy estable, aunque genera mucho tráfico en la modalidad Round Robin, ya que el proceso está contínuamente interrogando a los servidores sobre los que balancea, incluso aunque no tengan carga.

Además presenta otro inconveniente con respecto al enmascaramiento de los recursos J2EE, permitiendo el acceso a todos los recursos ofrecidos por los servidores, ya que sólo se limita a realizar un balanceo hacia las IPs/nombres, trasladando a los servidores cualquier petición HTTP sobre cualquier recurso hecho sobre el nombre del balanceador.

Desde el punto de vista de la seguridad, los recursos que no interesa ofrecer a redes públicas deberían quedar ocultos como inexistentes, cosa que este servicio no hace, incluso aunque OpenAM no permita su acceso.

DMZ: Proxy de aplicación (primera aproximación)

El software empleado es Apache, en su versión 2.2.3 rev 83.

Para poder gestionar las sesiones y el reparto de carga entre las instancias de las aplicaciones de tomcat, el proyecto apache dispone de conectores para diferente uso, y en este caso en concreto proporciona uno específico denominado mod_jk, que se encarga de realizar la distribución de sesiones entre diferentes "workers" o elementos de carga que son los que finalmente se conectan a los diferentes servidores Tomcat.

> **Nota**: en la versión CentOS 5.9 usada para este proyecto, la versión de apache es la indicada más arriba, y en paquete compilado (binario) del módulo mod_jk descargable del repositorio PBONE (no estándar) es la versión 1.2.30-2.7, que presenta un bug al iniciarse en el contexto de Apache:
>
> *`/etc/httpd/modules/mod_jk.so: undefined symbol: ap_get_server_description`*
>
> Este bug está reconocido por RedHat (CentOS) y recomienda el cambio a una versión superior o bien de Apache o del módulo, y como, en el momento de escribir estas líneas, no existen versiones binarias más nuevas en esta distribución de Linux, he optado por descargar los fuentes del módulo y compilarlo con el entorno de desarrollo de Apache dentro de una máquina virtual montada ex profeso para esta finalidad.

La configuración del módulo se debe realizar dentro del fichero de configuración del servidor Web Apache, sito en /etc/httpd/conf/httpd.conf, en este fichero se indica que quedan incluidos todos los ficheros *.conf que sean colocados en el directorio /etc/httpd/conf.d/, por tanto situaremos allá todos nuestros ficheros de configuración del módulo.

Nuestro primer fichero, que denominaremos loadbalancer_jk.conf contendrá lo siguiente:

```
LoadModule jk_module modules/mod_jk.so

# Fichero de configuracion del modulo
JkWorkersFile /etc/httpd/conf.d/workers.properties

# Ficheros de LOG del modulo
JkShmFile /var/log/lb_shm/loadbalancer_jk.shm
JkLogFile /var/log/loadbalancer.log

# Nivel de detalle en Log (de mas a menos): debug, info, warn, error,
trace
JkLogLevel info
JkLogStampFormat "[%a %b %d %H:%M:%S %Y]"
JkOptions +ForwardKeySize +ForwardURICompat -ForwardDirectories

# --- Monitorizacion de aplicaciones (workers)
JkMount /probe agentsample
```

```
JkMount /probe/* agentsample

JkMount /manager manager
JkMount /manager/* manager

# --- Monitor del balanceador de carga
<Location /jkmanager>
       JkMount lb_status
       Order deny,allow
       Deny from all
       Allow from localhost
</Location>
```

La primera línea del fichero (en negrita) indica a Apache cual es el módulo que ha de cargar, en este caso será el mod_jk que es el módulo que se encargará de gestionar el balanceo de carga entre las aplicaciones de tomcat.

Del resto de elementos del fichero, podemos destacar los elementos más importantes[25]:

- JkWorkersFile: indica el nombre del fichero (incluido su path) donde se almacenará la configuración de los "workers" que desplegará el módulo, y que serán los encargados de gestionar el balanceo de carga y el estado entre los nodos que gestionan las aplicaciones de tomcat.
- JkMount: se trata de un identificador virtual de punto de montaje, que se usa también para identificar los "workers" que formarán el sistema de balanceo de carga, por claridad, es recomendable vincular el punto de montaje con la aplicación que balancean (especificado en el primer parámetro del JkMount), ya que, la URL resultante de acceso a la aplicación balanceada será:

  http://balanceador/worker

  Siendo "worker" el identificador del JkMount (último parámtro).
- JkShmFile: este fichero contiene información de configuración y runtime de los "workers" del balanceador de carga y todos sus miembros, y contiene información compartida entre ellos tal como información de estatus, carga de trabajo individual de cada "worker" e información adicional compartida entre todos ellos.

Como se ha visto, la directiva JkWorkersFile (en azul) indica el fichero en el que se almacenará la configuración específica del módulo, cuyo fichero contiene lo siguiente:

```
# Dos "workers", el que controla el estado y el lb_app1 propiamente
dicho.
worker.list=lb_status,agentsample,manager

# El worker "lb_status" es del tipo status (monitor)
worker.lb_status.type=status

# Defino la variable "lan_net" con el valor que quiero
lan_net=192.168.10

# defino los "sub" workers "apps1" y "apps2" que para los servidores
# de aplicacion e indico el protocolo "ajp13" la IP del Host y el
# puerto de acceso (tomcat), tambien se podria dar todo en el "host"
# con el valor de la IP:PORT
```

25 https://tomcat.apache.org/connectors-doc/reference/apache.html

```
worker.apps1.type=ajp13
worker.apps1.host=$(lan_net).10
worker.apps1.port=8080

worker.apps2.type=ajp13
worker.apps2.host=$(lan_net).15
worker.apps2.port=8080

# Y asocio al punto de montaje "agentsample" del tipo "lb" los dos
workers
worker.agentsample.type=lb
worker.agentsample.balance_workers=apps1,apps2
worker.agentsample.sticky_session=1

# Y asocio al punto de montaje "manager" del tipo "lb" los dos
workers
worker.manager.type=lb
worker.manager.balance_workers=apps1,apps2
worker.manager.sticky_session=1
```

Dentro del fichero de configuración del módulo, podemos destacar las siguientes directivas:

- worker.list: contiene la lista, separada con comas, de los "workers" del balanceador de carga, es imprescindible que se usen los nombres utilizados en todas las directivas JkMount del fichero previo (en nuestro caso *loadbalancer_jk.conf*).

  Salvo que el balanceo se realice para todas las aplicaciones (especificado en el parámtro JkMount), se recomienda definir un "worker" por aplicación que se desee balancear.

- worker.[nombre_nodo].{type|host|port}: define la IP, puerto y tipo de protocolo a usar por esa "instancia/servidor" tomcat con el que un worker se deberá conectar para gestionar el balanceo de carga.

  En definitiva, define nodos AJP de tomcat sobre los que realizar el balanceo de carga.

  - type: indica el protocolo de intercambio de información entre los diferentes workers, por defecto será el protocolo AJP en su release 1.3
  - host: indica la dirección IP del servidor Tomcat con el que repartirá carga (con el resto de servidores), y al que se le asignará un Worker del Proxy.
  - port: puerto donde escuchará el conector de protocolo AJP del servidor Tomcat sobre el que se realiza el balanceo (el de la dirección IP del parámetro anterior).

- worker.[nombre_worker].{type|balance_workers|sticky_session}: indica, para el worker indicado en el nombre, qué tipo de worker es, el número de elementos con los que se ha de conectar para gestionar su balanceo de carga, y el modo en que gestionará las sesiones de usuariosentre ellos.
  - type: indica de qué tipo será el "worker", tipo LB (Load Balancer) o Status.
  - balance_workers: dispondrá de una lista, separada de comas, de los workers encargados de repartir y asignar las sesiones a los servidores de aplicación (Tomcat). En nuestro caso habrá uno por nodo servidor.

- **sticky_session**: le indica el modo en que debe mantener las sesiones de usuario y reencaminarlas (rerouting) en caso de fallo del servidor de aplicación.

Una vez configurado el entorno, el acceso a las aplicaciones Tomcat a través del proxy inverso debe ser autorizado y filtrado a través del firewall de la siguiente manera:

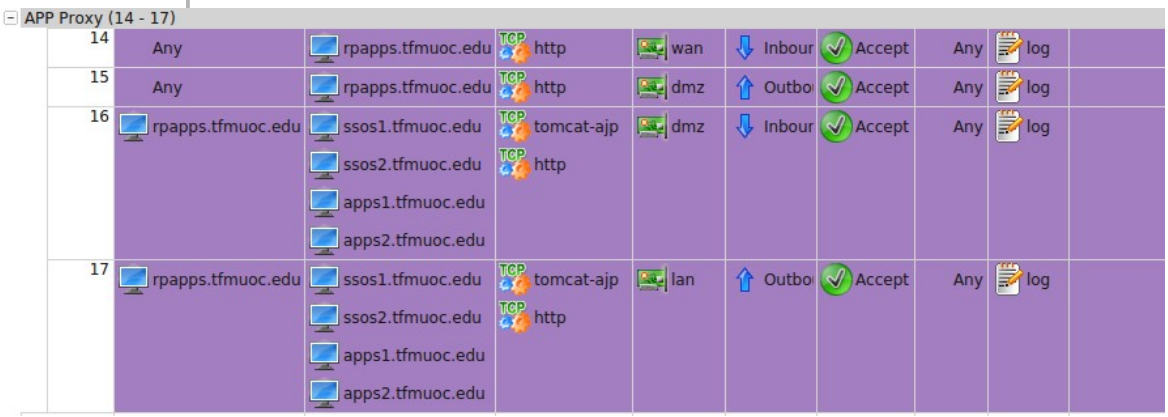

figura 32: Reglas de acceso al proxy inverso de aplicaciones

Y con una regla de NAT:

figura 33: Nat de acceso al proxy de aplicaciones

DMZ: Proxy de aplicación (Segunda aproximación)

La segunda configuración de proxy también utiliza el protocolo AJP para el balanceo entre los dos nodos de aplicación, pero usa el módulo mod_proxy_ajp, en vez del mod_jk, y resulta más eficiente y fácil de configurar.

En este caso, la configuración se encuentra almacenada en el fichero **/etc/httpd/conf.d/proxy_ajp.conf**, cuyo contenido es:

```
LoadModule proxy_ajp_module modules/mod_proxy_ajp.so
<VirtualHost *:80>
        ProxyRequests Off
        ProxyPreserveHost On
        ProxyVia On
        CustomLog /var/log/httpd/apps.tfmuoc.edu.log combined
        LogLevel debug
        RewriteEngine On
        RewriteCond %{REQUEST_URI} ^/(portal)$
        RewriteRule ^(.*)$ http://%{HTTP_HOST}$1/ [R=301,L]
        <Proxy *>
                Order deny,allow
                Allow from all
        </Proxy>
        <Proxy balancer://aplicaciones>
                BalancerMember ajp://apps1.tfmuoc.edu:8009 route=apps1
                BalancerMember ajp://apps2.tfmuoc.edu:8009 route=apps2
        </Proxy>
        <Location /agentsample>
                ProxyPass       balancer://aplicaciones/agentsample
        </Location>
```

```
        <Location /agentsample2>
                ProxyPass        balancer://aplicaciones/agentsample2
        </Location>
        <Location /manager>
                ProxyPass        balancer://aplicaciones/manager
        </Location>
        <Location /probe>
                ProxyPass        balancer://aplicaciones/probe
        </Location>
        <Proxy balancer://singlesignon>
                BalancerMember ajp://ssos1.tfmuoc.edu:8009 route=ssos1
                BalancerMember ajp://ssos2.tfmuoc.edu:8009 route=ssos2
        </Proxy>
        <Location /openam>
                ProxyPass balancer://singlesignon/openam
        </Location>
         <Location /balancer-manager>
                  SetHandler balancer-manager
                  Order Deny,Allow
                  Deny from all
                  Allow from 192.168.1.31
         </Location>
</VirtualHost>
```

Y las partes más importantes de esta configuración son:

- La directiva "balancer" define un balanceador de carga (a priori genérico).
- "BalancerMember" define qué servidores recibirán las peticiones de balanceo (ajp:) indica que el protocolo usado para acceder a las instancias (java) de los servidores Tomcat es AJP, identificando estas instancias con el parámetro "route", que deberán definirse en la configuración de los servidores J2EE correspondientes.
- En el ejemplo, "singlesignon" es un nombre arbitrario que identifica al balanceador, al que, desde la clausula "Location" se asocia con el recurso publicado, en este ejemplo "openam".

La "Location" "balancer-manager" es un recurso virtual ofrecido por el proxy para monitorizar el balanceo de carga.

En definitiva, con esta configuración conseguiremos que, al nombre del proxy (en su acceso público), que será http://apps.tfmuoc.edu/, se le pueda asociar como propio los recursos /openam, /agentsample, etc... Es decir:

- La URL http://apps.tfmuoc.edu/openam será balanceada a alguno de los servidores http://ssos1.tfmuoc.edu/openam o http://ssos2.tfmuoc.edu/openam dentro de la LAN.
- La URL http://apps.tfmuoc.edu/agentsample (o cualquiera de las demás), será balanceada a alguno de los servidores de aplicación http://apps1.tfmuoc.edu/agentsample o el servidor http://apps2.tfmuoc.edu/agentsample.

De ese modo quedan integrados todos los recursos bajo el nombre DNS apps.tfmuoc.edu.

La configuración del Firewall no varía, y por tanto, las reglas indicadas en la primera aproximación son válidas para esta.

DMZ: Balanceador de carga (tercera aproximación)

Y por último, hemos probado otro balanceador de carga denominado HAProxy, el cual es mencionado en la propia documentación de OpenAM[26], y que representa una configuración bastante simple e intuitiva, y que, además, también permite el uso de las "Sticky Sessions"[27], o las llamadas sesiones persistentes.

En este caso, en nuestro servidor CentOS de la DMZ que hará el trabajo del balaneo de carga (rpssos.tfmuoc.edu) bastará con instalarle el paquete correspondiente de HAProxy.

Desgraciadamente este paquete aún no pertenece a los repositorios estándar, y no es posible localizarlo mediante YUM, así que para simplificar el proceso de instalación y evitar descargar los fuentes y todo el entorno de compilación, he procedido a buscarlo en la librería de paquetes rpm.pbone.net[28].

Una vez descargado se instala con el comando:

```
yum localinstall -nogpgcheck haproxy-1.3.26-1.el5.x86_64.rpm
```

Esto revisará las dependencias del paquete y si queda alguna pendiente de resolver, descargará todos los paquetes necesarios para solucionarla.

Una vez instalado, crea el fichero de configuración: /etc/haproxy/haproxy.cfg
Cuyo contenido dejaremos del siguiente modo:

```
global
    log         127.0.0.1 local2
    chroot      /var/lib/haproxy
    pidfile     /var/run/haproxy.pid
    maxconn     4000
    user        haproxy
    group       haproxy
    daemon

defaults
    mode        http
    log         global
    option      dontlognull
    option      httpclose
    option      httplog
    option      forwardfor
    option      redispatch
    timeout connect 10000
         # default 10 second time out if a backend is not found
    timeout client 300000
    timeout server 300000
    maxconn     60000
    retries     3

frontend http-in
    bind *:80
    default_backend apps

backend apps
    cookie SERVERID insert nocache
    balance roundrobin
    server  apps1 192.168.10.10:80 cookie 01 id 1001 check inter 2000
rise 2 fall 5
    server  apps2 192.168.10.15:80 cookie 02 id 1002 check inter 2000
rise 2 fall 5
```

26 https://wikis.forgerock.org/confluence/display/openam/5+Extending+to+a+Dual+Instance+Deployment
27 http://www.networkinghowtos.com/howto/sticky-session-load-balancing-with-haproxy/
28 ftp://ftp.pbone.net/mirror/download.fedora.redhat.com/pub/fedora/epel/5/x86_64/haproxy-1.3.26-1.el5.x86_64.rpm

```
    errorfile 400 /usr/share/haproxy/400.http
    errorfile 403 /usr/share/haproxy/403.http
    errorfile 408 /usr/share/haproxy/408.http
    errorfile 500 /usr/share/haproxy/500.http
    errorfile 502 /usr/share/haproxy/502.http
    errorfile 503 /usr/share/haproxy/503.http
    errorfile 504 /usr/share/haproxy/504.http
```

En este fichero podemos observar las siguientes secciones (es posible una configuración más fina a través del manual[29] del producto):

- Global: donde se indican todas las opciones del demonio haproxy, tal como el usuario que lo ejecutará, el fichero de pid, dónde almacenará el log (syslog), etc..
- Defaults: en esta sección se configuran los parámetros propios del mecanismo de balanceo de carga y que serán válidos para todos los "balanceadores", tales como los timeouts, el modo de trabajo por defecto, etc...
- Frontend: que será el punto de entrada de las conexiones de usuario, esto es, cómo entrarán los usuarios (puerto de enlace, que en nuestro caso es el 8080) y cómo reenviará estas solicitudes hacia el "backend" que es donde realmente se produce el balanceo.
- Backend: donde se indican todos los servidores/puerto sobre los que se balancearán las sesiones de usuario, y en los que se indica el modo de balanceo (round-robin) y de qué manera tratará las sesiones (cookies).
    - cookie: parámetro que activa la persistencia de las sesiones basadas en cookies.
        - insert: indica que la cookie persistente se inserta en las respuestas del servidor.
        - nocache: evita que la cookie sea cacheada.
    - balance roundrobin, esto es balanceo equilibrado de una conexión a cada servidor alternativamente.
    - server: describe el servicio sobre el que se balancea la conexión, habrá tantos como servidores, en nuestro caso Tomcats, dispongamos.
        - [*Identificador*] (apps1 y apps2) es un identificador interno de servicio balanceado, útil para hacer un seguimiento en los logs.
        - IP:PUERTO: donde se enviarán las solicitudes HTTP balanceadas.
        - cookie: indica todos los parámetros insertados por HAProxy que tendrá las cookies que serán enviadas a cada uno de los servidores.
    - errorfile: son las páginas HTML de error que HAProxy mostrará cuando ocurra algún problema.

Una vez configurado, procederemos a iniciar el servicio HAProxy y a verificar su funcionamiento a través de la URL: http://apps.tfmuoc.edu/agentsample, y veremos que nos aparece la pantalla de la aplicación (siempre que no necesitemos iniciar sesión en OpenAM, en caso contrario, mostrará el Logon de OpenAM)

Por último, sólo nos queda configurar el Firewall de forma similar a como se indicó en las anteriores aproximaciones.

29 http://cbonte.github.io/haproxy-dconv/configuration-1.5.html

LAN: Servicio de aplicaciones J2EE

El servicio de aplicaciones está basado en el servidor J2EE Apache Tomcat, en su versión 6.0. Su instalación consiste en descomprimir el paquete descargado de la web del fabricante y definir las variables de entorno adecuadas (CATALINA_HOME, etc...) apuntando al directorio en el que se ha desempaquetado.

Una vez en funcionamiento, la interacción entre el usuario (desde el navegador) y el servidor J2EE debería quedar de la siguiente manera:

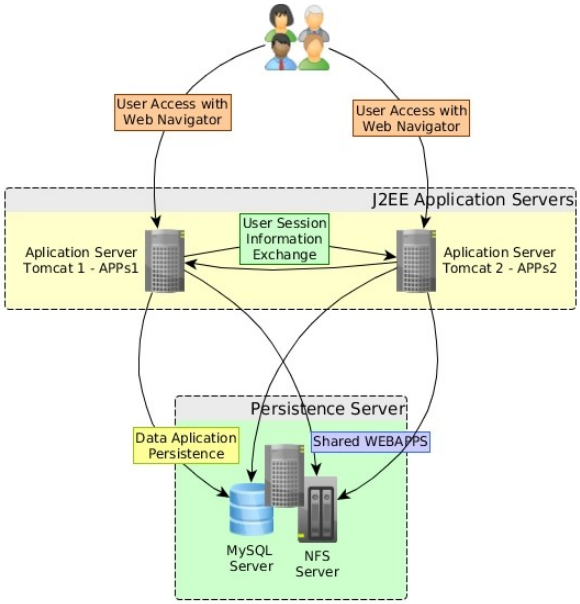

figura 34: Interaccción entre el usuario y el servidor J2EE

La redundancia, pensada para entornos de producción, se realiza por motivos de alta disponibilidad, que procura la existencia del servicio con tolerancia a fallos en uno de los dos servidores.

Por tanto, en este caso, al tratarse de un entorno con más de un servidor de aplicaciones, y puesto que, como se ha mencionado con anterioridad, es necesario que el entorno se encuentre clusterizado, los accesos de los usuarios a las aplicaciones se puede realizar con independencia del servidor que atienda el acceso (que siempre será único).

La clusterización implica que todos los servidores que forman el cluster han de ser capaces de ofrecer el mismo servicio de aplicación, y para esto, todos ellos han de ejecutar la misma aplicación (o todas), que tomarán de una misma ubicación de red compartida (SAN, NAS, etc...).

Adicionalmente, la persistencia de estas aplicaciones debe poder ser accedida con independencia del servidor que esté ofreciendo el servicio, por lo que el acceso al motor de base de datos donde la aplicación dispone de sus datos debe ser posible para todos los nodos que forman el cluster.

Persistencia de la sesión de usuario

Cuando un usuario accede por primera vez a una aplicación, completando el proceso de autenticación correctamente, le es asignado un identificador de sesión (que se almacenará en la cookie) y que durará lo que esté programado en la aplicación para las sesiones de usuario.

Si, por ejemplo, debido a un fallo en uno de los servidores de aplicación, el usuario es obligado a conectarse al otro (en la misma aplicación), la sesión no será identificada como válida debido a que no ha sido creada y registrada por la aplicación a la que se ha migrado el usuario.

Por tanto, para solucionar este problema, es necesario que ambas aplicaciones (la misma en los dos servidores) permitan el intercambio de la información de las sesiones de usuario, de tal manera que cuando este cambie de servidor, pueda seguir operando con la aplicación ya que será válida en ambos servidores.

Para conseguir eso, los servidores Tomcat se configuran para que intercambien información de las sesiones de usuario que tienen activas, de tal manera que, cuando un servidor deja de responder, el otro es capaz de hacerse cargo de las sesiones de usuario, pues siguen siendo válidas en el que aún responde.

Para gestionar el intercambio de información de sesiones entre los servidores Tomcat, Apache desarrolló un protocolo, denominado AJP[30] (**A**pache **J**serv **P**rotocol, que actualmente se encuentra en su release 1.3).

Además, este protocolo se diseñó para que un web server (Apache) pudiera trabajar en modo proxy hacia aplicaciones que se ejecutaban en uno o varios servidores de aplicación (Tomcat), permitiendo balanceo de carga.

La activación del protocolo implica que se deban configurar los conectores (de protocolo) dentro del servidor de aplicaciones, a través de los cuales puedan intercambiar información entre los nodos del cluster, y con un proxy inverso que balancee las conexiones de usuarios.

Clusterización

Llegado este punto, para configurar un cluster de servidores de aplicaciones, necesitaremos, en primer lugar, un repositorio de ficheros común, donde almacenaremos las aplicaciones, es decir, que ambos servidores (o los que formen el cluster) puedan ver el mismo Path donde se almacenen las aplicaciones que deben desplegarse.

Esto se resuelve mediante el uso de un servicio de almacenamiento en red (NAS, SAN o un servicio de ficheros de red), que en nuestro caso será un servidor NFS al que conectaremos ambos nodos (en `$CATALINA_HOME/webapps`).

Adicionalmente, si las aplicaciones necesitan persistencia de datos (normalmente mediante una base de datos), será necesario implementar un servicio de estas características. En este caso las aplicaciones desplegadas usan la base de datos relacional MySQL, que será desplegada en el mismo servidor de almacenamiento de ficheros (NFS) por motivos exclusivamente de economía de recursos, aunque en un entorno productivo, estos servicios deberían ser independientes y, probablemente, también clusterizados.

Queda fuera del alcance del presente proyecto la descripción del proceso de instalación y configuración de los servicios de persistencia NFS y MySQL.

Una vez dispongamos de los dos servidores de aplicaciones con almacenamiento y base de datos compartida, procederemos a configurarlos, para ello tendremos en el fichero (`$CATALINA/conf/server.xml`) la siguiente cláusula de configuración[31]:

```
<Connector port="8009" protocol="AJP/1.3" redirectPort="8443" />
```

30 http://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html
31 En el anexo II se incluye el fichero server.xml completo.

Esta directiva indica qué puerto TCP será utilizado para emplear el protocolo AJP, necesario para la comunicación entre los nodos del cluster (para el intercambio de información de las sesiones de usuarios, entre otras cosas).

También necesitamos identificar la o las máquinas virtuales de Java del nodo para que otros elementos (otros nodos o un balanceador de carga) pueda contactar con este e interrogarle sobre el estado o transferirle información de sesiones, etc..., para ello usaremos la siguiente directiva:

```
<Engine name="Catalina" defaultHost="localhost" jvmRoute="apps1">
```

De este modo, dependiendo de dónde hayamos definido el "Engine" (Host, Service, etc...) indicaremos un nombre para que el resto de elementos de red puedan identificarlo. Véase que este nombre es el usado en la definición de "Workers" en el balanceador de carga de Apache (apartado "*Configuración proxy RP a las APPs Tomcat*" del presente documento).

Además, tendremos:

```
<Cluster className="org.apache.catalina.ha.tcp.SimpleTcpCluster"
channelSendOptions="8">
     <Manager className="org.apache.catalina.ha.session.DeltaManager"
         expireSessionsOnShutdown="false"
         notifyListenerOnReplication="true"/>
     <Channel className="org.apache.catalina.tribes.group.GroupChannel">
     <Membership className="org.apache.catalina.tribes.membership.McastService"
         address="228.0.0.4"
         port="45564"
         frequency="500"
         dropTime="300"/>
     <Sender className="org.apache.catalina.tribes.transport.ReplicationTransmitter">
         <Transport
className="org.apache.catalina.tribes.transport.nio.PooledParallelSender"/>
     </Sender>
     <Receiver className="org.apache.catalina.tribes.transport.nio.NioReceiver"
         address="192.168.10.10"
         port="4000"
         autoBind="100"
         selectorTimeout="5000"
         maxThreads="6"/>
    <Interceptor
className="org.apache.catalina.tribes.group.interceptors.TcpFailureDetector"/>
    <Interceptor
className="org.apache.catalina.tribes.group.interceptors.MessageDispatch15Interceptor"/
>
    </Channel>
    <Valve className="org.apache.catalina.ha.tcp.ReplicationValve" filter=""/>
    <Valve className="org.apache.catalina.ha.session.JvmRouteBinderValve"/>
    <ClusterListener
className="org.apache.catalina.ha.session.JvmRouteSessionIDBinderListener"/>
    <ClusterListener
className="org.apache.catalina.ha.session.ClusterSessionListener"/>
</Cluster>
```

En esta configuración podemos destacar los siguientes elementos:

- El tipo de Cluster: definido como "**SimpleTCPCluster**"[32] (azul claro), que es el tipo de cluster más sencillo y utiliza un multicast simple para el intercambio de información entre nodos.
- El gestor de sesiones "Manager"[33], que indica el modo en que deben gestionarse las sesiones:
  - Cuando sel servidor se para: indica que la sesión no debe ser caducada en el resto de nodos del cluster (false), facilitando así la persistencia de la sesión activa. Si este parámetro es true, indica que al cerrar el servidor, el "Caller" notifica al resto de nodos que sus sesiones se deben caducar para todos los nodos.

32 https://tomcat.apache.org/tomcat-5.5-doc/catalina/docs/api/org/apache/catalina/cluster/tcp/SimpleTcpCluster.html
33 https://tomcat.apache.org/tomcat-7.0-doc/config/cluster-manager.html

    - Y también se indica (**notifyListenerOnReplication**), que los listener deben replicar y notificar cambios en las sesiones de usuario.
- Definirá los parámetros de red de Multicast (en rojo) para la comunicación del nodo presente al resto de nodos.
- Y definirá los parámetros de red del receptor de las comunicaciones multicast del resto de nodos hacia el presente (en azul).
- El resto de parámetros terminan de configurar el Binder de jvmRouter para identificar los nodos/aplicaciones, y otros parámetros de red necesarios.

NOTA IMPORTANTE

Al final del proyecto, una vez ha sido desplegado todo el entorno de aplicación junto con el entorno SSO, se ha comprobado que no se ha de configurar los dos servidores Tomcat de aplicación en modo Cluster, ya que esta configuración buscaba el intercambio de sesiones entre los dos nodos, función que es ofrecida por OpenAM en la persistencia de sesión SSO.

Es decir, la configuración básica (sin Single Sign On), requiere que los servidores Tomcat estén clusterizados para poder compartir la información de las sesiones de usuario (y poder transferirse el control de las sesiones entre si).

Al introducir un SSO, que se encarga de gestionar las sesiones de usuarios dentro de todo el entorno, hace innecesario el uso de la configuración clusterizada de los servidores de aplicación.

Por otro lado, para aquellas aplicaciones que utilicen un sistema de autenticación de usuarios no integrado en la misma, podrán hacer uso del Realm[34], que se definirá de la siguiente manera:

```
<Realm className="org.apache.catalina.realm.LockOutRealm">
   <Realm  className="org.apache.catalina.realm.JDBCRealm"
       driverName="com.mysql.jdbc.Driver"
       connectionURL="jdbc:mysql://192.168.10.35:3306/eberom"
       connectionName="root" connectionPassword="root"
       userTable="user" userNameCol="user_name"
       userCredCol="password" userRoleTable="user_role"
       roleNameCol="role_name" />
</Realm>
```

En esta configuración, el Realm (por defecto, Tomcat usa otro Realm ("org.apache.catalina.realm.UserDatabaseRealm", que lee del recurso UserDatabase, que a su vez lee del fichero tomcat-user.xml)), será el JDBCRealm, y encontrará la información de las credenciales de usuarios almacenada nuestra base de datos MySQL en las tablas UserTable y UserRoleTable, en las columnas indicadas.

Esta configuración debe existir en ambos nodos Tomcat, que, al compartir también, la carpeta de despliegue de aplicaciones, ya estaremos en condiciones de arrancar el cluster Tomcat.

Llegado este punto, cuando un usuario se conecte a una aplicación, deberá especificar a qué nodo del cluster debe conectarse, pero si este nodo falla, perderá la conexión (pero no la sesión) que podrá retomar si se conecta al otro nodo.

34 No existe una traducción literal, dentro del ámbito IT, de la palabra Realm, pero en términos históricos, hacía referencia al dominio que se podía ejercer sobre algo (ej, sobre un señorío feudal). En términos IT vendría a referirse a una entidad externa que autentica a un usuario en el acceso a la aplicación que lo solicita.

Para evitar esto, desde la DMZ, se implementa el proxy que, por un lado, permite migrar las conexiones entre ambos nodos en función de la carga y disponibilidad, y por otro, hace transparente la conexión a los usuarios, ofreciendo una única URL de conexión con independencia del nodo al que se conecte.

Persistencia de las aplicaciones J2EE contenidas

Al tratarse de un cluster de servidores de aplicación, a priori han de ejecutar las mismas instancias de aplicaciones J2EE, para facilitar esta tarea y evitar inconsistencias, lo apropiado es que el directorio de despliegue de aplicaciones se encuentre compartido entre todas las instancias de Tomcat.

Cualquier servicio de para compartir ficheros (Samba, NFS, etc...) puede servir. En nuestro caso, utilizaremos el servicio por defecto para Linux, NFS.

Bastará con exportar, mediante el fichero /etc/exports, un directorio que servirá para contener el directorio de despliegue, y en cada servidor de aplicación, habrá que montar (mount -t nfs …) el directorio exportado.

LAN: Servicio SSO

Una vez hemos montado y desplegado el entorno J2EE para nuestras aplicaciones, que inicialmente tendrán el sistema de autenticación propio de las mismas, y configurado el entorno de alta disponibilidad y balanceo de carga, tendremos un entorno clásico de aplicaciones, pero cada una con un mecanismo de autenticación propio.

Es el objetivo de este proyecto unificar todos los mecanismos de autenticación en un único entorno SSO.

En nuestro caso el producto seleccionado es OpenAM[35], este producto es un gestor de acceso unificado, en el que se proporcionan todas las herramientas necesarias para:

1. Poder sustituir los mecanismos nativos de autenticación de las aplicaciones por uno centralizado.
2. Disponer de un repositorio de credenciales de usuarios único (usualmente LDAP).
3. Proporcionar los mecanismos de persistencia y gestión de sesión de usuario que permitan la superveniencia y validez de las sesiones a lo largo de todo el entorno, y permita, además, cierto grado de tolerancia a fallos (clusterización y balanceo de carga).
4. También aporta toda la infraestructura necesaria para poder federar el entorno con otros sistemas de autenticación unificados (SSO) y poder establecer las relaciones de confianza necesarias para poder realizar validaciones cruzadas.

Dentro de la misma Web del producto, también se proporcionan herramientas complementarias tales como: servicio de directorio LDAP (OpenDJ) o un Identity Manager (OpenIDM) que complementan el servicio de SSO proporcionado por OpenAM.

35 https://en.wikipedia.org/wiki/OpenAM

Arquitectura a implementar

Como se mencionó en apartados anteriores, el servicio de autenticación de usuarios, que es quien, en definitiva, permite el acceso a las aplicaciones, es un servicio crítico, lo que implica, necesariamente, que debe estar configurado de forma que permita la caída de uno de los servicios (OpenAM) sin que ello impida que el servicio general siga funcionando.

Para ello OpenAM implementa lo que se denomina "Session Failover"[36], que consiste en disponer de, al menos, dos servicios OpenAM que se intercambian información de las sesiones en tiempo real y que, cuando uno de ellos deja de responder, el otro es capaz de seguir haciéndolo en idénticas condiciones.

Por otro lado, un elemento crucial para que OpenAM pueda funcionar en alta disponibilidad, es el repositorio donde se almacenarán las credenciales de los usuarios.

En el caso de OpenAM, el producto incorpora embebido un motor LDAP basado en su producto OpenDJ, que hace que OpenAM pueda funcionar de forma autónoma. El problema del uso del repositorio embebido es que cuando uno de los dos servicios OpenAM falle, el otro deberá poder acceder al repositorio de credenciales, bien porque estas estén replicadas, bien porque el servicio de directorio se almacena en un recurso remoto y compartido con las medidas de clusterización y réplica adecuadas.

En este caso, lo apropiado es que no se utilice el almacenamiento local a OpenAM y se emplee un repositorio remoto compartido.

Adicionalmente, y en aras de poder soportar altas cargas de autenticación y gestión de sesión de usuarios, se recomienda el despliegue de un servicio de balanceo de carga, que será implementado por un proxy inverso.

En nuestro caso podríamos haber seguido utilizando el servicio de proxy de Apache del mismo modo que lo hicimos para la aplicaciones tomcat, máxime teniendo en cuenta que el propio OpenAM es una aplicación J2EE que puede ser ejecutada en el contexto de Tomcat.

Sin embargo he decidido usar otro mecanismo de Proxy HTTP para poder ofrecer una alternativa comparable al Proxy inverso de Apache, el software usado en este caso será HAProxy[37].

En definitiva, la arquitectura a implementar será:

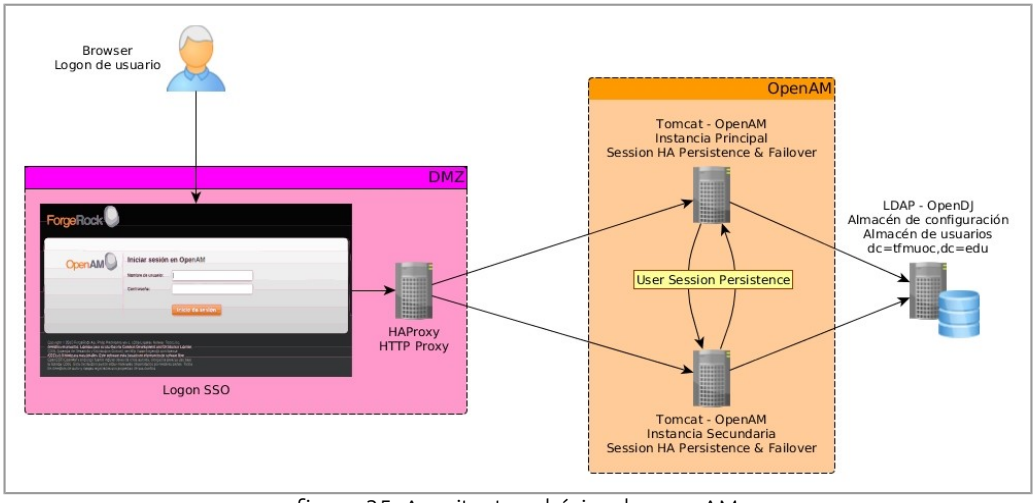

figura 35: Arquitectura básica de openAM

36 http://openam.forgerock.org/openam-documentation/openam-doc-source/doc/install-guide/index/chap-session-failover.html
37 http://haproxy.1wt.eu/

En nuestro entorno, el repositorio de configuración y usuarios quedará almacenado en un único servicio LDAP, sin embargo, en un entorno productivo y más seguro debería estar almacenado en un sistema replicado de, al menos, dos servicios de directorio con las medidas de protección adecuadas (clusterizados, etc...).

Servicio de directorio LDAP (OpenDJ)

Lo primero que vamos a implementar es el servicio de persistencia para OpenAM, es decir, el almacén de credenciales de usuarios y de configuración de nuestro servicio OpenAM, para ello utilizaremos el producto OpenDJ que pertenece al mismo fabricante.

Se trata de un software íntegramente desarrollado en Java que es autónomo, esto es, no requiere de servidor de aplicaciones.

Procederemos a descargar el paquete adecuado para el sistema operativo usado (CentOS 5.9) de la web del fabricante[38], y ejecutaremos el comando:

```
yum localinstall –nogpgcheck opendj-2.6.0-1.noarch.rpm
```

Esto hará que el comando Yum compruebe dependencias y si falta alguna, descargue los paquetes que le falten para su correcto funcionamiento.

Su instalación se realiza por defecto en /opt/opendj, pero el servicio no está configurado, para ello podemos elegir entre la opción de usar un entorno gráfico (si hemos instalado las Xwindows) o mediante línea de comando mediante el comando

```
/opt/opendj/setup --cli
```

Esto arranca un wizard de configuración que permite completar la puesta en marcha del servicio.

De los parámetros requeridos en el proceso usaremos los siguientes valores:

1. El usuario root del directorio (en notación DN): cn=DIrectory Manager
2. Contraseña del usario root.
3. Nombre del servidor (FQN): ldap.tfmuoc.edu
4. Puerto LDAP para la conexión de clientes: 389
5. Puerto administrativo: 4444
6. Desea la creación de la base DN del servidor, diremos que si, e introduciremos: dc=tfmuoc,dc=edu
7. ¿Queremos cargar usuarios de ejemplo? En nuestro caso, al tratarse de un laboratorio, seleccionamos la carga automática de unos cuantos usuarios (10) de ejemplo (opción 4).
8. No usaremos SSL ni TLS (su uso implica, además, la creación de un certificado de servidor, que puede ser creado “al vuelo” en el mismo wizard).
9. Y deseamos que el servicio quede arrancado al finalizar la configuración.

Ahora se muestra un resúmen, que aceptaremos y con ello concluirá la configuración de nuestro servicio LDAP.

38 https://download.forgerock.com/downloads/enterprise/opendj/2.6.0/opendj-2.6.0-1.noarch.rpm

Ahora configuraremos un script de arranque automático, que podemos generar nosotros mismos o hacerlo automáticamente con el script: **`/opt/opendj/bin/create-rc-script`**

En cualquier caso el inicio y parada del servicio de directorio se realiza mediante los scripts **`/opt/opendj/bin/start-ds`** y **`/opt/opendj/bin/stop-ds`** respectivamente.

Servicio SSO (OpenAM)

OpenAM es una aplicación tipo J2EE y, en consecuencia, puede ejecutarse en cualquier entorno compatible. En nuestro laboratorio hemos elegido utilizar Tomcat versión 6, que es la que el propio fabricante de OpenAM recomienda para la versión 10.1, que será la que usaremos.

Por tanto, procederemos a la instalación del servidor de aplicaciones:

1. Descargaremos la aplicación[39].
2. Y la descomprimiremos en `/usr/share/apache-tomcat-6.0.37`
3. Por si no lo hemos hecho, también es necesario descargarse la versión JDK 6[40], acorde con esta versión de tomcat y recomendada por OpenAM.

   > *Nota curiosa: obsérvese que en la web de descarga de la JDK, que pertenece al dominio de Oracle, se está usando el producto OpenSSO para autenticar la sesión de descarga, esto es, cuando se inicia la descarga, si no se está autenticado, el sistema automáticamente lanzará la solicitud de Logon a través de OpenSSO, sin embargo, si previamente si se ha realizado el Logon, no lo solicitará al realizar la descarga.*

4. También procederemos a instalar el paquete de la JDK descargada, y en el fichero `/etc/profile`, definiremos las variables: `$JAVA_HOME` y agregaremos la `$JAVA_HOME/bin` a la variable de sistema `PATH`.
5. Asimismo, también crearemos el script de arranque y parada automático del servicio de Tomcat que agregaremos al sistema (en `/etc/init.d`) y daremos de alta como servicio correspondiente.

   A este fichero hemos de incluirle el PATH de donde haya quedado instalado el Java a través de la variable `JAVA_HOME`.
6. Este proceso lo realizaremos en los dos servidores destinados a ejecutar las instancias de OpenAM, cuyos nompres serán
   - **`ssos1.tfmuoc.edu (192.168.10.20)`**
   - **`ssos2.tfmuoc.edu (192.168.10.25)`**

instancia principal

Una vez dispongamos los dos servicios de Tomcat (se puede verificar su funcionamiento en las URLs de los servidores en el puerto por defecto para tomcat: 8080), podemos proceder a instalar OpenAM en ambos, para ello descargaremos el producto[41] de la web del fabricante y lo descomprimiremos en local.

De el contenido del paquete descargado podemos observar varios elementos, que corresponden a los diferentes elementos del servicio que pueden ser

39 http://ftp.cixug.es/apache/tomcat/tomcat-6/v6.0.37/bin/apache-tomcat-6.0.37.tar.gz

40 Aviso: se requiere un usuario válido en el sitio de Oracle, no obstante, la URL sería: http://download.oracle.com/otn/java/jdk/6u45-b06/jdk-6u45-linux-x64-rpm.bin

41 https://download.forgerock.com/downloads/enterprise/openam/openam10/10.1.0/openam_10.1.0.zip

configurados, así como diferentes configuraciones del propio servidor OpenAM.

A nosotros el paquete que nos interesa es el siguiente:

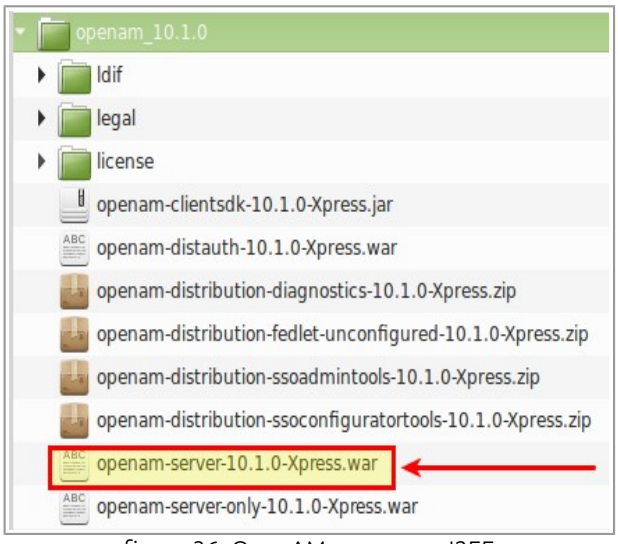

figura 36: OpenAM como app J2EE

Y de este directorio, copiaremos el fichero "`openam-server-10.1.0-Xpress.war`" en el directorio `/usr/share/apache-tomcat-6.0.37/webapps`, y lo renombraremos simplemente a "openam.war".

Para la correcta ejecución del servicio de OpenAM, la configuración de memoria de la máquina virtual de Tomcat no es suficiente, por tanto, además, es necesario que en el script de arranque del servicio Tomcat (`/etc/init.d/tomcat`) se incluyan los siguientes parámetros (justo después de donde se indique la variable `JAVA_HOME`.

```
JAVA_OPTS="-Xmx1024m -XX:MaxPermSize=256m"
export JAVA_OPTS
```

Llegado a este punto, procederemos a arrancar el servicio TOMCAT y a través del fichero de log `/usr/share/apache-tomcat-6.0.37/logs/catalina.out`, podremos ver la evolución del despliegue de la aplicación OpenAM.

Al finalizar el arranque, podremos conectar a la pantalla inicial de configuración de OpenAM a través de la URL: http://ssos1.tfmuoc.edu:8080/openam obteniendo lo siguiente:

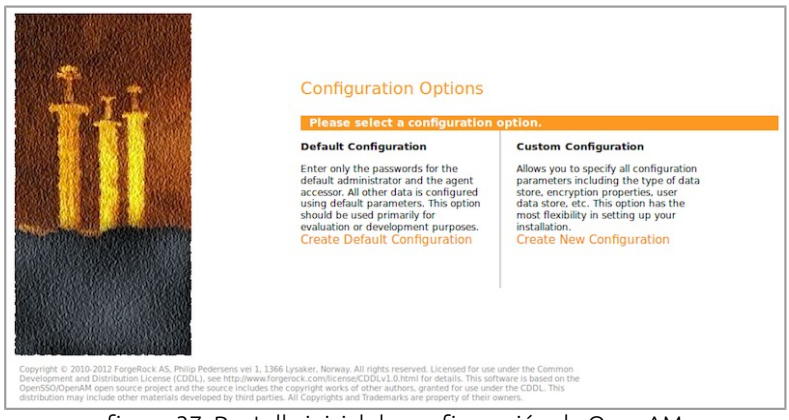

figura 37: Pantalla inicial de configuración de OpenAM

Donde se nos ofrece crear una configuración por defecto, que hará uso del repositorio de configuración y usuarios en el OpenDJ embebido, o crear una nueva configuración, que será la que nosotros seleccionaremos.

Lo primero se nos solicitará la contraseña

figura 38: Contraseña del administrador de OpenAM

Y a continuación los datos del servidor en curso, cuyos datos dejaremos por defecto.

Es importante destacar que el dominio de la Cookie debe ser el mismo que hay registrado en la base DN del servicio LDAP sobre el que crearemos toda la estructura de credenciales de usuarios, por tanto, es necesario que el servidor Tomcat sobre el que estamos desplegando OpenAM tenga el nombre DNS (FQN) bien definido, o bien a través de un DNS, como es nuestro caso, o bien mediante el fichero /etc/hosts (sólo para casos de entornos de desarrollo y test).

En la siguiente pantalla definiremos dónde debe almacenarse la configuración (Configuration Store), y la completaremos de la siguiente manera:

figura 39: Almacén de datos de OpenAM (configuración)

Como podemos ver, hemos de indicarle los parámetros con los que configuramos el servicio de directorio del apartado anterior:

- Tipo de almacén de configuración, que para nostros será: OpenDJ
- No usaremos SSL.
- El puerto de acceso al LDAP: 389
- La clave de cifrado de la conexión al servicio LDAP

- La raiz DN del servicio de directorio que ya hemos configurado: dc=tfmuoc,dc=edu
- El usuario administrador de la raiz del servicio de directorio: cn=DIrectory Manager
- Y su contraseña.

Si existe algún tipo de incongruencia con el servicio de directorio (no hay conectividad, la contraseña o el usuario administrador no es válido, etc...) nos será advertido sobre la marcha.

Y aceptamos, pasando a la siguiente pantalla:

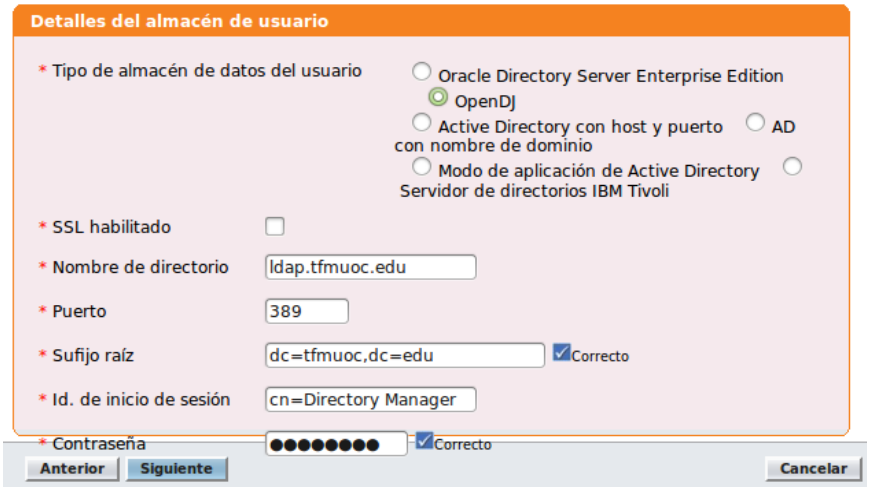

figura 40: Almacén de credenciales de usuario

En esta pantalla especificaremos dónde se almacenarán las credenciales de usuario, que puede ser en el mismo directorio LDAP (nuestro caso) o en otro distinto, por ejemplo, si estuviéramos implementando el servicio en un entorno Windows con un directorio activo completamente implementado.

De nuevo, el tipo de almacén seguirá siendo OpenDJ, instalado en el servidor de directorio, con el mismo puerto, base DN y usuario administrador.

En la siguiente parte se pregunta si OpenAM va a ser montado en un entorno con balanceo de carga y con persistencia de sesión (Session Failover), y tal y como se describía al inicio del presente capítulo, se realizará con esta opción, por tanto, los datos se informarán de la siguiente manera:

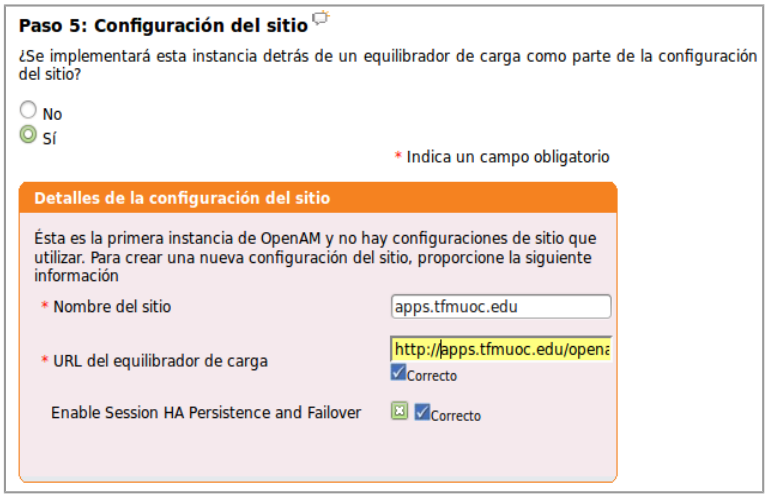

figura 41: Nombre de la URL de acceso al entorno OpenAM

Se solicita la contraseña para el agente de directivas:

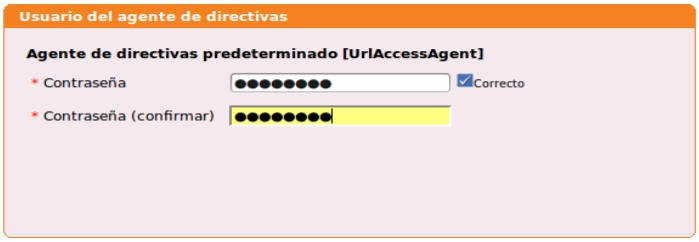

figura 43: Contraseña por defecto del Policy Agent

Y finalmente, el wizard nos mostrará un resúmen de la configuración que vamos a aplicar al nuestra primera instancia de OpenAM:

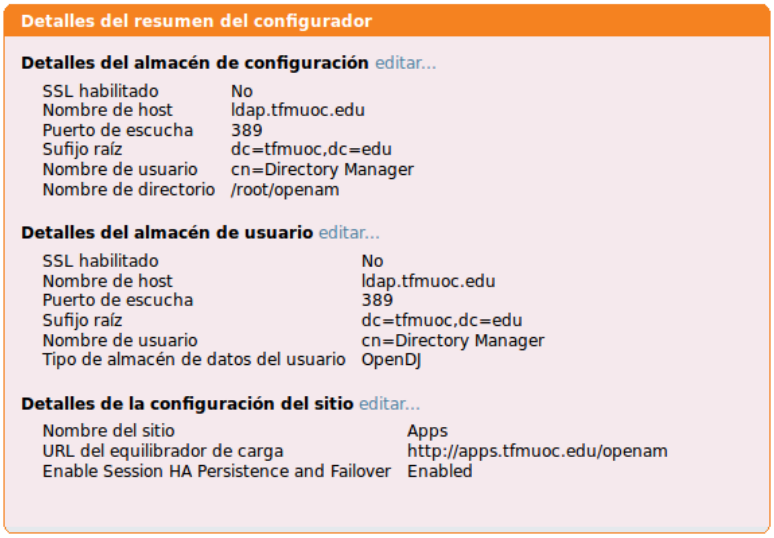

figura 44: Resúmen de la instalación (previo)

Si aceptamos, el sistema procede a configurar toda la instancia, mostrando un progreso del proceso de configuración. Al finalizar (con éxito) nos mandará a la pantalla de Logon de OpenAM:

figura 42: Login del OpenAM

Instalación de la instancia secundaria

Esta fase es más sencilla puesto que aprovecha la configuración ya existente para la instancia primaria de OpenAM.

En el segundo servidor de Tomcat (`ssos2.tfmuoc.edu`) en el que ya hemos configurado el servidor de aplicación de forma idéntica al primero, incluyendo la variable de parámetros de la máquina virtual (`JAVA_OPTS`), copiaremos el fichero openam.war que ya copiamos en el primer servidor, en la misma ubicación, pero en el segundo servidor, y procederemos a iniciar el servicio Tomcat.

En la URL (http://ssos2.tfmuoc.edu:8080/openam) veremos de nuevo la página de bienvenida (ver página 27), donde seleccionaremos la opción "Crear nueva configuración".

Introduciremos, del mismo modo, la contraseña del usuario administrador, y en la siguiente pantalla obtendremos lo siguiente:

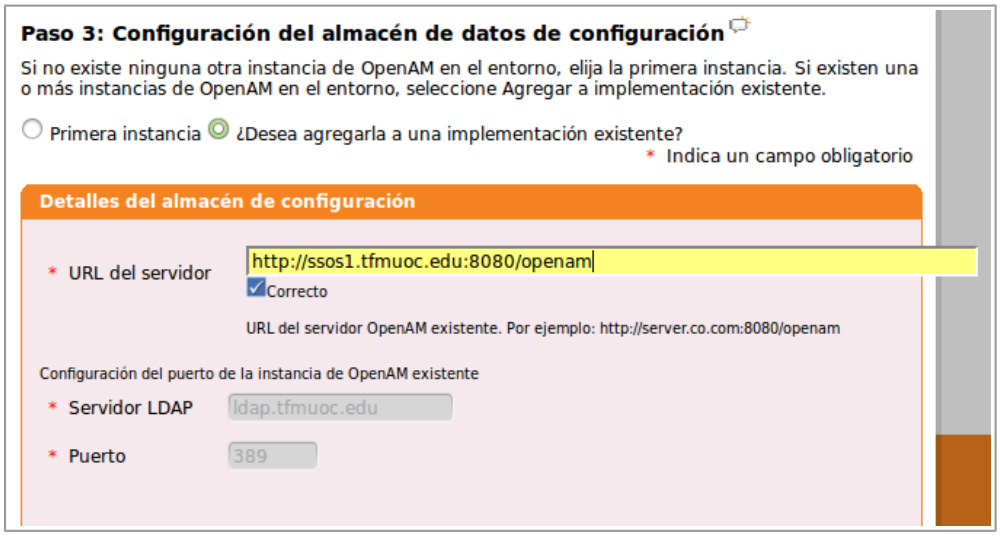

figura 45: Almacén de datos de la segunda instancia de OpenAM

En este caso, al tratarse de una segunda instancia, seleccionaremos (en la parte superior del recuadro) la opción "Agregarla a una implementación existente"... e introduciremos la URL de la aplicación OpenAM del primer nodo (http://ssos1.tfmuoc.edu:8080/openam).

El wizard verifica que se trata de una instancia OpenAM correcta y procede a cargar los datos del servicio LDAP ya configurado (que muestra en la misma pantalla).

Al avanzar a la siguiente pantalla, no será necesario indicarle que el entorno estará bajo un sistema de balanceo de carga puesto que ya lo indicamos en el primer servidor.

Nota sobre el uso de balanceadores

El uso de balanceadores de carga, en esta configuración, permite emplear correctamente la facilidad "Session Failover" ya que, de otro modo, aunque los dos servidores dispongan de la información de sesión (porque disponen de la facilidad activada, por si mismos, no informan al usuario que debe cambiar de URL (al otro servidor SSO).
Esta tarea compete al balanceador, que es quien intermamente redirige al usuario, a través de la misma URL, a uno u otro servidor en función de la carga o la respuesta (nula si ha caído), además de permitir integrar en un único punto de acceso (URL, la aplicación y el entorno SSO).

Sincronización de tiempos (Servicio NTP)

Es crucial, en un entorno clusterizado, que todos los relojes de todos los servidores que intervienen en este entorno estén sincronizados, ya que, en caso contrario, podrían producirse efectos inconsistentes a la hora de gestionar, por ejemplo, las coockies de usuario cuando son transferidas de uno a otro servidor (no sincronizado).

Para resolver este problema instalaremos en todos los servidores el servicio NTP, que puede ser obtenido de los repositorios estándar de CenOS, y que instalaremos mediante el comando YUM.

La configuración por defecto de este servicio, que configuraremos para que arranque de forma automática con los servidores, es válida y puede mantenerse por defecto.

Ahora únicamente queda autorizar en el Firewall el acceso a Internet para poder sincronizarse, para ello insertaremos las siguientes reglas:

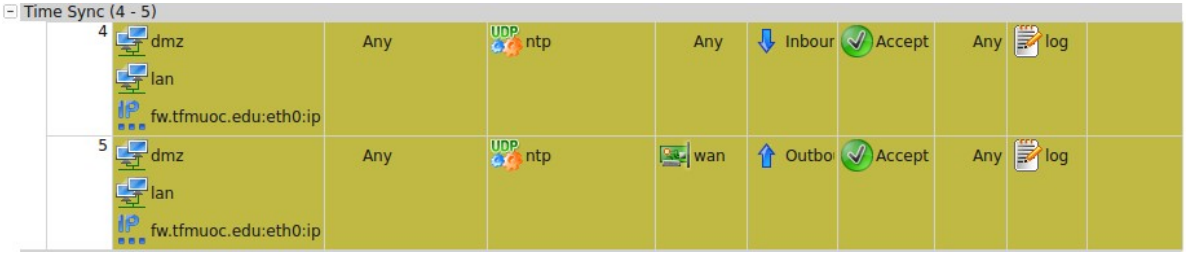

figura 46: Reglas de acceso al servicio NTP

En este punto ya dispondremos de toda la infraestructura desplegada, pero aún no configurada.

Test de funcionamiento

NOTA IMPORTANTE

El desarrollo inicial de este anexo corresponde a las primeras fases del proyecto y toma los datos del test de funcionamiento a través de un único servidor de aplicaciones, sin utilizar el proxy de acceso, y utilizando de forma diferenciada un proxy de LAN para el acceso al servicio SSO.

Ambos nombres DNS para estos servicios son:

- apps1.tfmuoc.edu
- lrpsso.tfmuoc.edu

Se ha agregado, al final del presente anexo, una segunda recogida de información sobre el funcionamiento del entorno SSO con todo el entorno ya integrado (aplicaciones J2EE y OpenAM) bajo el proxy de aplicación sito en la DMZ y que responde al nombre DNS:

- apps.tfmuoc.edu

Punto de partida

Tal y como ya hemos podido ver en el apartado "Caso práctico" (pg. 27 del presente documento), partiremos de la base de que ya disponemos Tomcat funcionando, y OpenAM correctamente instalado con la opción Session Failover funcionando y un balanceador de carga (proxy) que nos da acceso al mismo, tenemos que, para proteger la aplicación "agentsample" hemos realizado los siguientes pasos:

1. Desplegamos la aplicación en el servidor Tomcat, quedando accesible en la URL http://apps1.tfmuoc.edu:8080/agentsample
2. Creado un perfil de agente en el Realm de nivel superior (root).
3. Instalado el agente en el servidor tomcat.
4. Verificado o creados, si procede, los grupos de usuarios y/o grupos que accederán a la aplicación.
5. Creadas las directivas de las URLs sobre las que existirán accesos privilegiados, con las acciones pertinentes asociadas.
6. Configurados, dentro del perfil del agente, aquellas URL que no requerirán supervisión del mismo.
7. Mapeo de Atributos privilegiados (para el descriptor J2EE de la aplicación).

Ahora procedemos a verificar que realmente la aplicación funciona como esperamos.

Lo primero de todo es realizar el despliegue de la aplicación dentro del servidor Tomcat, para ello usaremos el fichero de la propia aplicación: agentsample.war, ubicado en: **$DIRECTORIO_INSTALACION_AGENTE/sampleapp/dist**, y lo copiaremos en el directorio de aplicaciones del servidor Tomcat **$CATALINA_HOME/webapps**, al hacerlo, el propio TOMCAT procederá a su despliegue.

Una vez finalizado el despliegue, podremos acceder a la URL de la página principal de la aplicación.

Esta página está configurada como URI sobre la que no aplica las políticas:

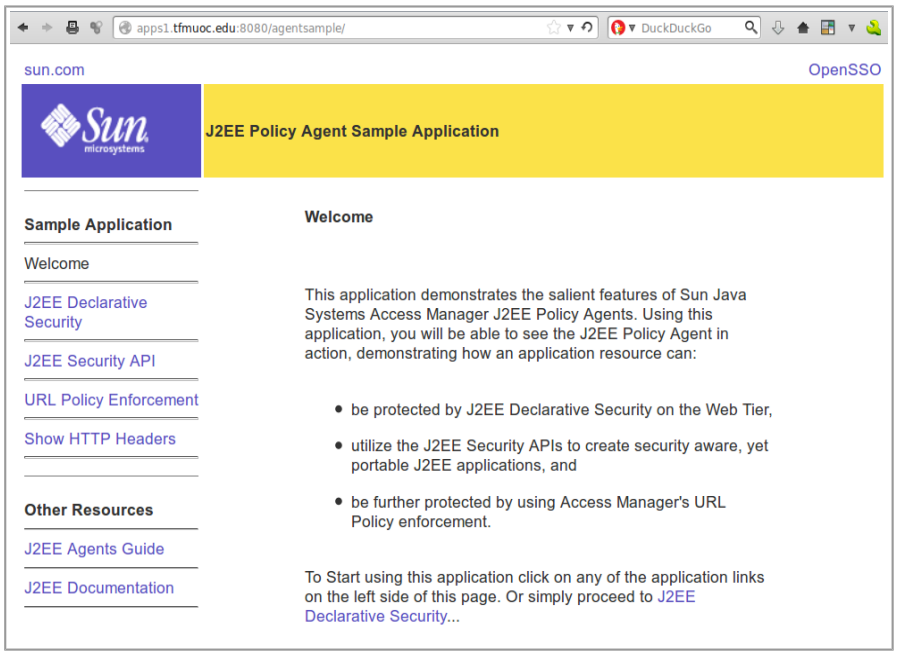

figura 47: Página principal aplicación de test "AgentSample"

Y efectivamente, OpenAM nos autoriza la entrada sin ningún tipo de autenticación, y mientras nos movamos por la aplicación invocando URIs que constan como no aplicables en la política de seguridad, la navegación no requerirá autenticación.

Recordemos que las URIs definidas como públicas son:

- `/agentsample/public/*`
- `/agentsample/images/*`
- `/agentsample/styles/*`
- `/agentsample/index.html`
- `/agentsample`

Pero si, por ejemplo, intentamos acceder a una URL que, en OpenAM, ha sido definida como protegida en la política de seguridad, quedando la configuración así:

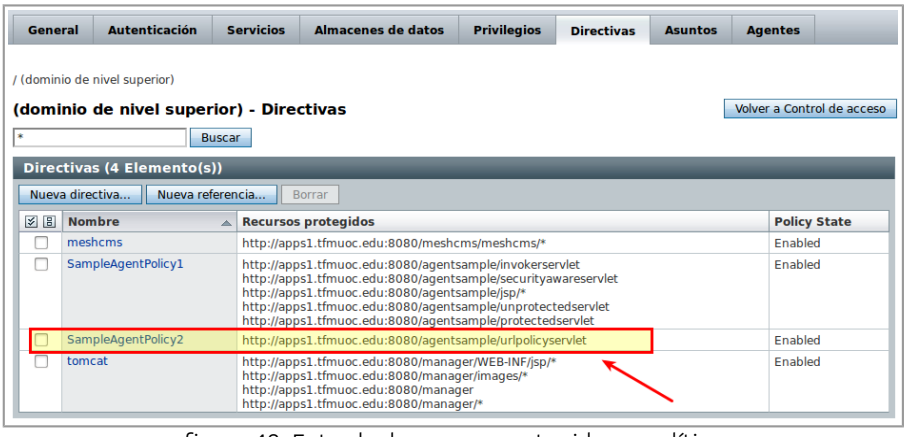

figura 48: Entrada de recurso protegido en política

Si intentamos acceder a dicha URL, veremos lo siguiente:

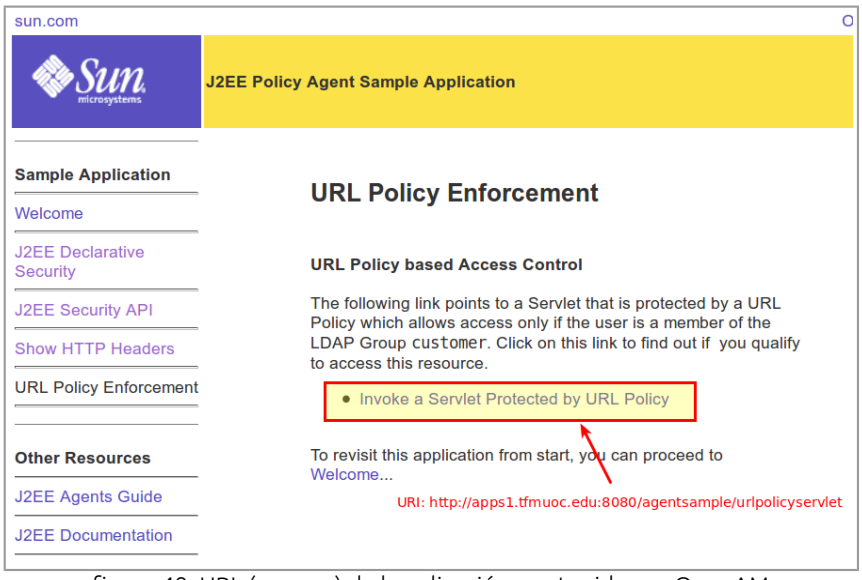

figura 49: URL (recurso) de la aplicación, protegida por OpenAM

Al acceder, el Policy Agent lo intercepta, nos direcciona a OpenAM (ver la URL) a la pantalla de login:

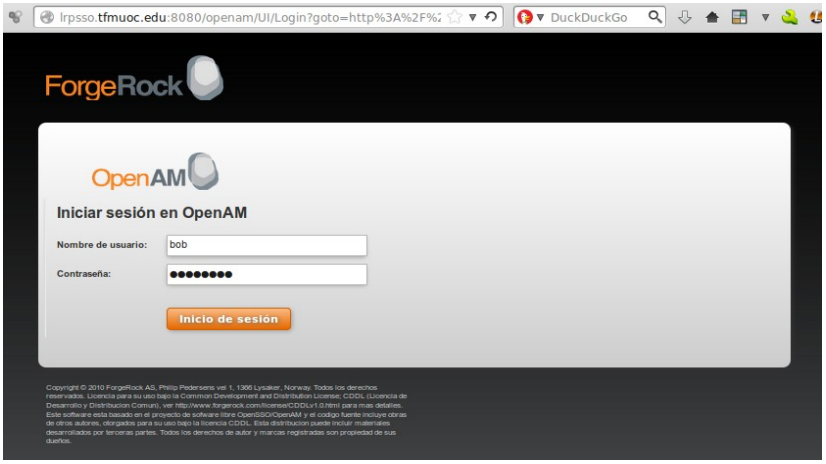

figura 50: Login de OpenAM redireccionado desde el recurso protegido

Funcionamiento: cabeceras HTTP y cookies

Ahora vamos a ver cómo se produce el intercambio de cookies de sesión entre el servidor de aplicación (a través del Policy Agent) y el servidor de SSO (OpenAM).

Para ello limpiamos en el navegador todo rastro de navegación previo, incluidas todas las cookies y procedemos a acceder a la URL principal de la aplicación, que está supervisada por el Policy Agent.

Veamos, URL: http://apps1.tfmuoc.edu:8080/agentsample

Las cabeceras HTTP[42] intercambiadas son:

```
http://apps1.tfmuoc.edu:8080/agentsample

GET /agentsample HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 302 Movido temporalmente
Server: Apache-Coyote/1.1
Location: http://apps1.tfmuoc.edu:8080/agentsample/
Transfer-Encoding: chunked
Date: Mon, 25 Nov 2013 00:32:19 GMT
-------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/

GET /agentsample/ HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Connection: keep-alive

HTTP/1.1 200 OK
[...][43]
-------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/images/sun_logo.gif

GET /agentsample/images/sun_logo.gif HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept: image/png,image/*;q=0.8,*/*;q=0.5
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://apps1.tfmuoc.edu:8080/agentsample/
Connection: keep-alive

HTTP/1.1 200 OK
[...]
-------------------------------------------------------
```

Esto indica que en la carga de la página principal de la aplicación, el Policy Agent ha interceptado el acceso a la URL, ha consultado a OpenAM con respecto a la política de seguridad correspondiente y, fruto de la autorización de OpenAM, ya que la página accedida no queda bajo la supervisión del agente, ha autorizado su acceso sin necesidad de registrar ninguna sesión, ha redirigido ("**HTTP/1.1 302 Movido temporalmente**" como respuesta dada desde el servidor de aplicación) el navegador a la respuesta proporcionada por el servidor OpenAM.

Además, como no se ha producido inicio de sesión, el navegador tampoco ha

42 Captura realizada con el plugin de Firefox "Live HTTP headers 0.17", que se puede descargar de la URL: http://livehttpheaders.mozdev.org/

43 Se ha eliminado parte de la información de respuesta del servidor para facilitar su lectura.

generado ninguna cookie.

Si ahora accedemos a un recurso de la aplicación que ha sido protegido mediante políticas de seguridad, por ejemplo: http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet, pasará lo siguiente:

```
http://apps1.tfmuoc.edu:8080/agentsample/public/urlpolicy.html

GET /agentsample/public/urlpolicy.html HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://apps1.tfmuoc.edu:8080/agentsample/
Connection: keep-alive

HTTP/1.1 200 OK
[...]
-------------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet

GET /agentsample/urlpolicyservlet HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer:
http://apps1.tfmuoc.edu:8080/agentsample/public/urlpolicy.html
Connection: keep-alive

HTTP/1.1 302 Movido temporalmente
Server: Apache-Coyote/1.1
Location: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample%2Furlpolicyservlet
Content-Length: 0
Date: Mon, 25 Nov 2013 00:36:52 GMT
-------------------------------------------------------------
http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A%2F
%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample%2Furlpolicyservlet

GET /openam/UI/Login?goto=http%3A%2F%2Fapps1.tfmuoc.edu
%3A8080%2Fagentsample%2Furlpolicyservlet HTTP/1.1
Host: lrpsso.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer:
http://apps1.tfmuoc.edu:8080/agentsample/public/urlpolicy.html
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Cache-Control: private
Pragma: no-cache
```

```
Expires: 0
X-DSAMEVersion: OpenAM 10.1.0-Xpress (2013-February-07 15:45)
AM_CLIENT_TYPE: genericHTML
Set-Cookie:
AMAuthCookie=AQIC5wM2LY4Sfcw41MgI8_pcrWhihKiBPYyAhCtMTN5pN0s.*AAJTSQA
CMDIAAlNLABQtMzAzMzcxMTIyNjc5Mjc3ODA1MAACUzEAAjAz*;
Domain=.tfmuoc.edu; Path=/
Set-Cookie: amlbcookie=03; Domain=.tfmuoc.edu; Path=/
Set-Cookie: JSESSIONID=FB81C0A0F73F2D6A4933EB2387EDCE34.ssos2;
Path=/openam
Set-Cookie: SERVERID=02; path=/
Content-Type: text/html;charset=UTF-8
Content-Length: 7720
[...]
```

Como podemos observar, la llamada a la URL protegida ha provocado que el Policy Agent responda al navegador, de nuevo, con un error de redirección "**HTTP/1.1 302 Movido temporalmente**", pero esta vez si que hace que el navegador acceda a la URL de login en OpenAM, que es lo que presenta al usuario.

Y las cookies que aparecen en el navegador son las siguientes:

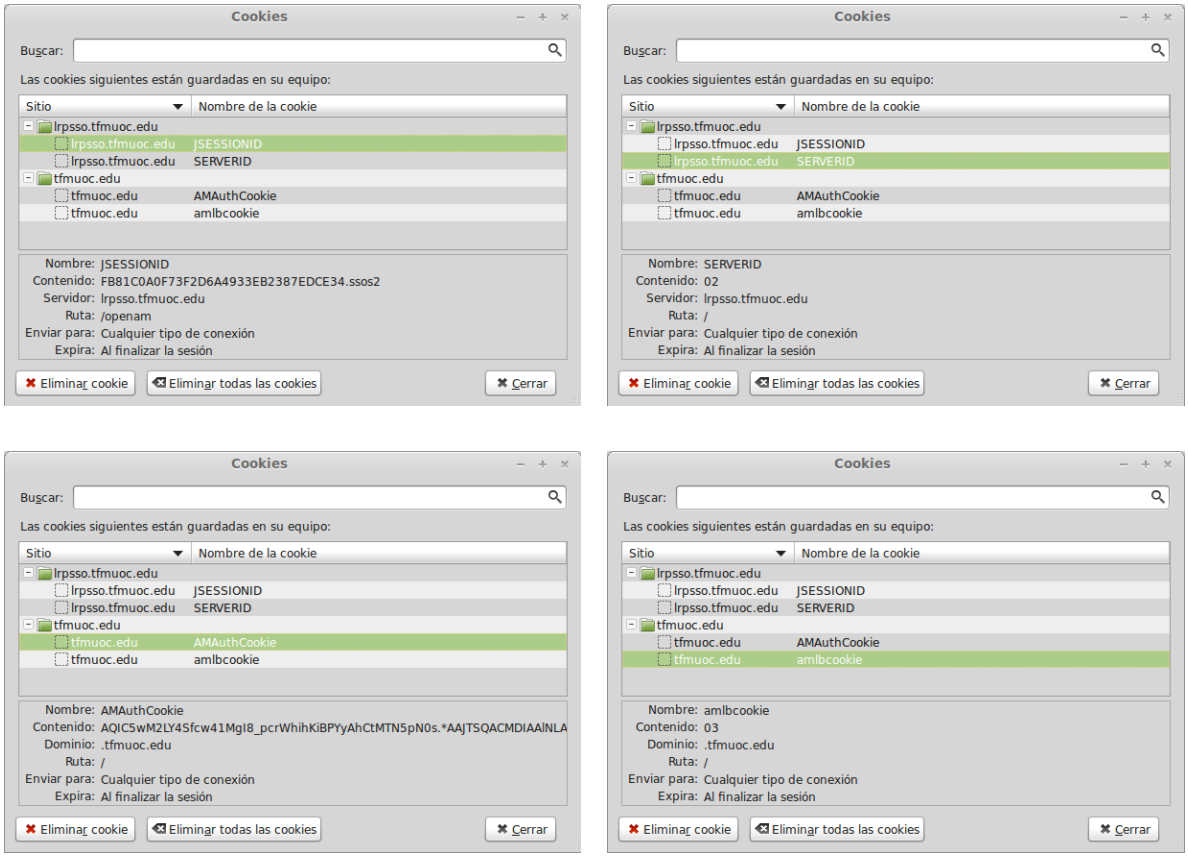

figura 51: Cookies. Redirección al SSO para Logon

Esto quiere decir que el mero acceso a la pantalla de Login genera cuatro Cookies: la cookie **SERVERID** indica qué servidor SSO nos ha atendido, la **amlbcookie** la genera el balanceador de carga y las otras dos son de sesión, sin embargo no ha habido autenticación (todavía) de usuario, en este punto

podemos observar que, si el usuario, en un tiempo prudencial, no ha concluido el proceso de login, la sesión de login. Estas cookies pertenecen al servidor OpenAM, no al servidor de aplicación y están vinculadas al proceso de autenticación de usuario.

Ahora iniciamos la sesión con un usuario válido, pero que no tiene permisos para acceder al recurso protegido seleccionado: BOB

En ese caso, el intercambio de cabeceras HTTP que se produce es el siguiente:

```
http://lrpsso.tfmuoc.edu:8080/openam/UI/Login

POST /openam/UI/Login HTTP/1.1
Host: lrpsso.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample
%2Furlpolicyservlet&gx_charset=UTF-8
Cookie: JSESSIONID=FB81C0A0F73F2D6A4933EB2387EDCE34.ssos2;
AMAuthCookie=AQIC5wM2LY4SfcwQWZzvUX343GJn8SgYVs3nfe4FYdM-CdQ.*AAJTSQA
CMDIAAlNLABQtNDAwNjM4MzY0MjY0OTM0MDMxMQACUzEAAjAz*; SERVERID=02;
amlbcookie=03
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 233
IDToken1=bob&IDToken2=bob12345&IDButton=Inicio+de+sesi
%C3%B3n&goto=aHR0cDovL2FwcHMxLnRmbXVvYy5lZHU6ODA4MC9hZ2VudHNhbXBsZS91
cmxwb2xpY3lzZXJ2bGV0&gotoOnFail=&SunQueryParamsString=Z3hfY2hhcnNldD1
VVEYtOCY%3D&encoded=true&gx_charset=UTF-8

HTTP/1.1 302 Found
Server: Apache-Coyote/1.1
Cache-Control: private
Pragma: no-cache
Expires: 0
X-DSAMEVersion: OpenAM 10.1.0-Xpress (2013-February-07 15:45)
AM_CLIENT_TYPE: genericHTML
X-AuthErrorCode: 0
Set-Cookie:
iPlanetDirectoryPro=AQIC5wM2LY4SfcwQWZzvUX343GJn8SgYVs3nfe4FYdM-CdQ.*
AAJTSQACMDIAAlNLABQtNDAwNjM4MzY0MjY0OTM0MDMxMQACUzEAAjAz*;
Domain=.tfmuoc.edu; Path=/
Set-Cookie: AMAuthCookie=LOGOUT; Domain=.tfmuoc.edu; Expires=Thu,
01-Jan-1970 00:00:10 GMT; Path=/
Location: http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet
Content-Length: 0
Date: Thu, 25 Nov 2013 00:44:43 GMT
-------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet

GET /agentsample/urlpolicyservlet HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
```

```
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample
%2Furlpolicyservlet&gx_charset=UTF-8
Cookie: amlbcookie=03;
iPlanetDirectoryPro=AQIC5wM2LY4SfcwQWZzvUX343GJn8SgYVs3nfe4FYdM-CdQ.*
AAJTSQACMDIAAlNLABQtNDAwNjM4MzY0MjY0OTM0MDMxMQACUzEAAjAz*
Connection: keep-alive

HTTP/1.1 302 Movido temporalmente
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=232E772463A84790D68DB28A9F15B31D.apps1;
Path=/agentsample
Location:
http://apps1.tfmuoc.edu:8080/agentsample/authentication/accessdenied.
html?goto=http%3A%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample
%2Furlpolicyservlet
Content-Length: 0
Date: Mon, 25 Nov 2013 00:44:50 GMT
------------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/authentication/accessdenied.
html?goto=http%3A%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample
%2Furlpolicyservlet

GET /agentsample/authentication/accessdenied.html?goto=http%3A%2F
%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample%2Furlpolicyservlet HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample
%2Furlpolicyservlet&gx_charset=UTF-8
Cookie: JSESSIONID=232E772463A84790D68DB28A9F15B31D.apps1;
amlbcookie=03;
iPlanetDirectoryPro=AQIC5wM2LY4SfcwQWZzvUX343GJn8SgYVs3nfe4FYdM-CdQ.*
AAJTSQACMDIAAlNLABQtNDAwNjM4MzY0MjY0OTM0MDMxMQACUzEAAjAz*
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Accept-Ranges: bytes
Etag: W/"6765-1227639322000"
Last-Modified: Tue, 25 Nov 2008 18:55:22 GMT
Content-Type: text/html
Content-Length: 6765
Date: Mon, 25 Nov 2013 00:44:50 GMT
------------------------------------------------------------
```

En la primera cabecera HTTP, el cliente (en azul) envía las cookies recibidas al Policy Agent, que es quien contacta con OpenAM que, como hemos visto en el apso anterior, produce el reenvío a la pantalla de Login. Cuando el usuario se autentica, OpenAM le envía al navegador del usuario la cookie iPlanetDirectoryPro (que se puede configurar en la consola de administración de OpenAM en el perfil del agente), y que contiene el Token de sesión SSO.

Esta cookie es la que mantendrá la sesión del usuario hasta que caduque o cierre el navegador.

El usuario vuelve a reenviar la cookie de sesión (iPlanetDirectoryPro) al Policy Agent, que reenvía al OpenAM, y que, ahora si, responde con un Deny en el acceso al recurso solicitado por el usuario a través del reenvío del navegador a la página "**/agentsample/authentication/accessdenied.html**".

Mostrando la página:

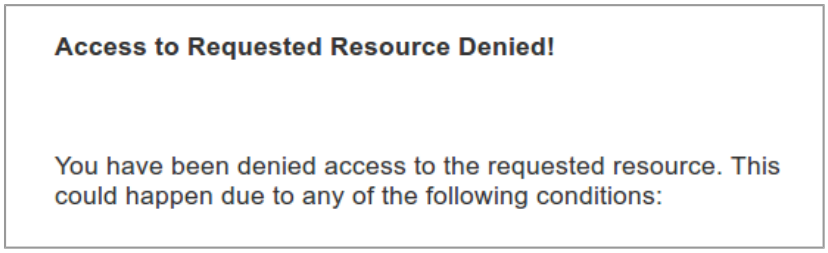

figura 53: Acceso denegado por OpenAM

Además, aunque se trate de un usuario válido en OpenAM que no ha tenido acceso al recurso, si tendrá una sesión válida dentro del entorno SSO a través de la cookie **iPlanetDirectoryPro**.

Con esta cookie podrá acceder a aquellos recursos a los que tenga permiso sin necesidad de realizar el proceso de autenticación.

Esta cookie tiene el tiempo de caducidad definido en OpenAM, aunque puede ser cancelada desde la consola.

figura 52: Cookies. Login aceptado

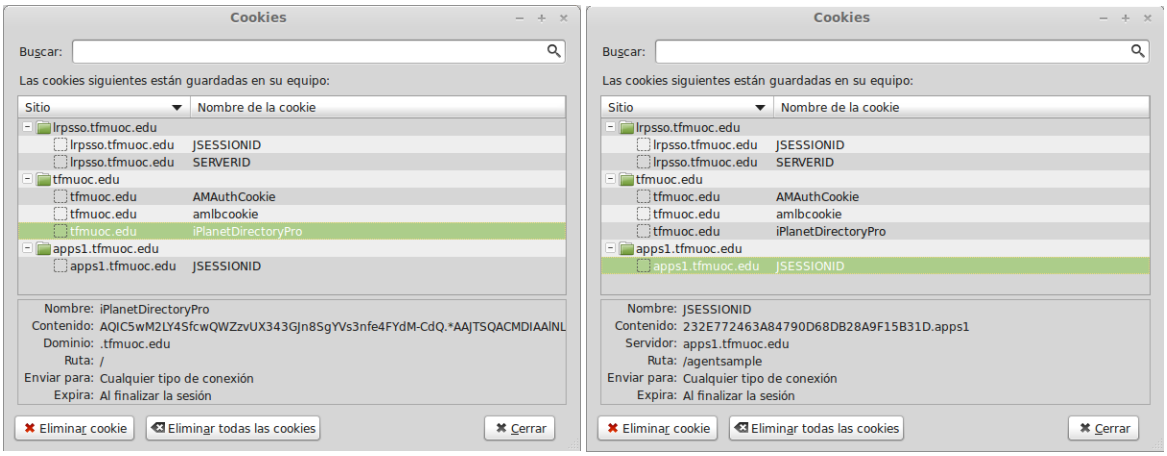

figura 54: Cookies. Login Aceptado (II)

Podemos ver la sesión del usuario en la consola del OpenAM:

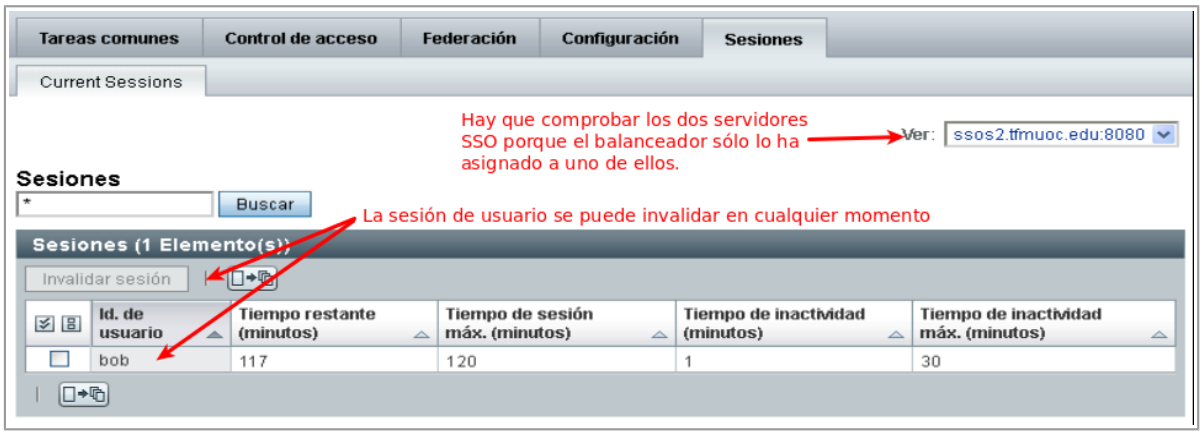

figura 55: Sesión válida de usuario en OpenAM (actualmente conectado)

Si ahora probamos con un usuario autorizado que, en el caso de la URL elegida: http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet, son los que pertenecen al grupo “admin”, es decir:

figura 56: Grupo "admin"

Nota práctica del laboratorio

En aras de la claridad, antes de proceder a probar de nuevo el acceso, esta vez con un usuario autorizado, limpiaré todo el historial de navegación y todas las cookies, y también invalidaré todas las sesiones de usuarios en OpenAM (salvo amAdmin).

De ese modo no habrá rastro de otras sesiones y la captura de cabeceras HTTP y cookies para esta prueba no quedará desvirtuada.

Así que accederemos de nuevo a la aplicación, y en concreto a la URL protegida, lo que nos llevará de nuevo a la pantalla de Login, donde procedemos a autenticarnos con un usuario autorizado, como por ejemplo, "chris", y veremos que ahora si, OpenAM nos autoriza el acceso y veremos la página siguiente:

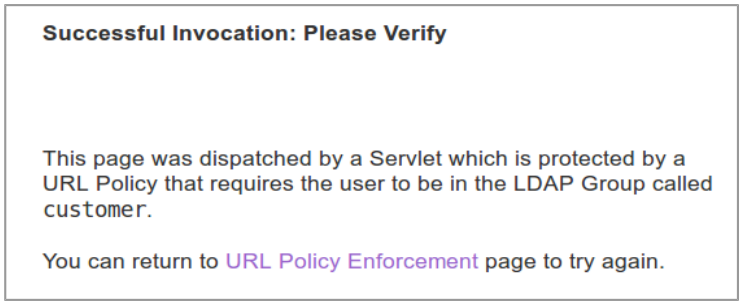

figura 57: Acceso al recurso autorizado por OpenAM

El intercambio de cabeceras HTTP y cookies antes de producirse el proceso de login es similar en cualquier tipo de usuario ya que, para OpenAM, aún no se ha producido la autenticación, por tanto el escenario sería el mismo que el descrito en el Login anterior (pg. 85 a 86 del presente anexo).

Y después del proceso de Logon (página a la que hemos llegado redireccionados por el Policy Agent), el intercambio de cabeceras HTTP queda del siguiente modo:

```
http://lrpsso.tfmuoc.edu:8080/openam/UI/Login

POST /openam/UI/Login HTTP/1.1
Host: lrpsso.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample%2Furlpolicyservlet
Cookie: JSESSIONID=784739652AF24AB88D56E18DDB0113AB.ssos1;
AMAuthCookie=AQIC5wM2LY4SfczuX7IyGMrJMcMkyizEC5Bh2vcc1Nl0oeo.*AAJTSQA
CMDIAAlNLABQtNzc0ODIyOTg4MzA5ODY2MDkwNAACUzEAAjAx*; amlbcookie=01;
SERVERID=01
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 209
IDToken1=chris&IDToken2=chris123&IDButton=Inicio+de+sesi
%C3%B3n&goto=aHR0cDovL2FwcHMxLnRmbXVvYy5lZHU6ODA4MC9hZ2VudHNhbXBsZS91
cmxwb2xpY3lzZXJ2bGV0&gotoOnFail=&SunQueryParamsString=&encoded=true&g
x_charset=UTF-8

HTTP/1.1 302 Found
Server: Apache-Coyote/1.1
```

```
Cache-Control: private
Pragma: no-cache
Expires: 0
X-DSAMEVersion: OpenAM 10.1.0-Xpress (2013-February-07 15:45)
AM_CLIENT_TYPE: genericHTML
X-AuthErrorCode: 0
Set-Cookie:
iPlanetDirectoryPro=AQIC5wM2LY4SfczuX7IyGMrJMcMkyizEC5Bh2vcc1Nl0oeo.*
AAJTSQACMDIAAlNLABQtNzc0ODIyOTg4MzA5ODY2MDkwNAACUzEAAjAx*;
Domain=.tfmuoc.edu; Path=/
Set-Cookie: AMAuthCookie=LOGOUT; Domain=.tfmuoc.edu; Expires=Thu,
01-Jan-1970 00:00:10 GMT; Path=/
Location: http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet
Content-Length: 0
Date: Sun, 01 Dec 2013 20:08:53 GMT
-------------------------------------------------------------
http://apps1.tfmuoc.edu:8080/agentsample/urlpolicyservlet

GET /agentsample/urlpolicyservlet HTTP/1.1
Host: apps1.tfmuoc.edu:8080
User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86_64; rv:25.0)
Gecko/20100101 Firefox/25.0
Accept:
text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: es-ES,es;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Referer: http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A
%2F%2Fapps1.tfmuoc.edu%3A8080%2Fagentsample%2Furlpolicyservlet
Cookie: amlbcookie=01;
iPlanetDirectoryPro=AQIC5wM2LY4SfczuX7IyGMrJMcMkyizEC5Bh2vcc1Nl0oeo.*
AAJTSQACMDIAAlNLABQtNzc0ODIyOTg4MzA5ODY2MDkwNAACUzEAAjAx*
Connection: keep-alive

HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Set-Cookie: JSESSIONID=0E7CA552B13A4872E1A8A685FA6984F4.apps1;
Path=/agentsample
Content-Type: text/html
Content-Length: 6421
Date: Mon, 25 Nov 2013 09:53:15 GMT
-------------------------------------------------------------
```

En este caso, veremos que la respuesta del servidor OpenAM es la entrega la cookie de sesión (**iPlanetDirectoryPro**) y la de de redirigir ("**HTTP/1.1 302 Found** ") al navegador de usuario al recurso que ha solicitado, ya que tiene autorización de acceso.

Las cookies que, en este caso, se producen, son las siguientes:

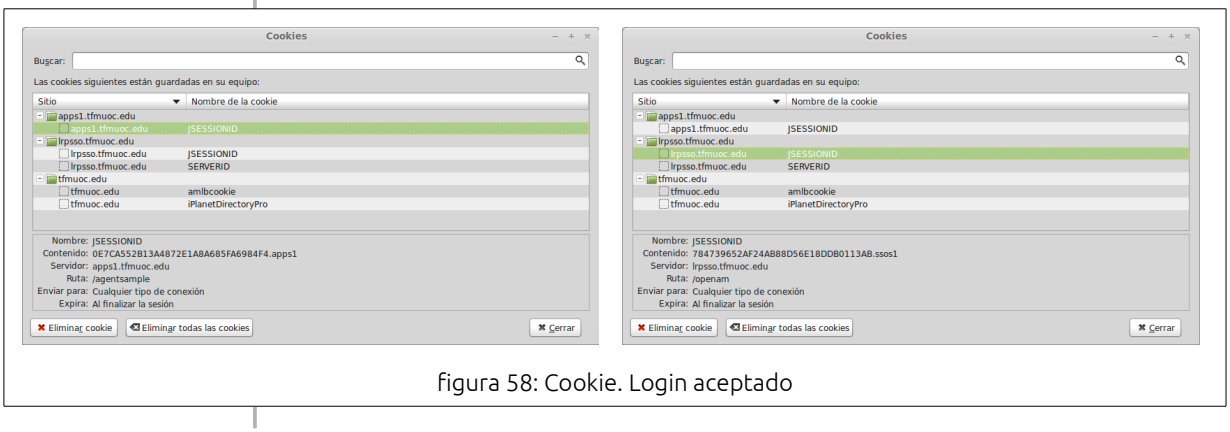

figura 58: Cookie. Login aceptado

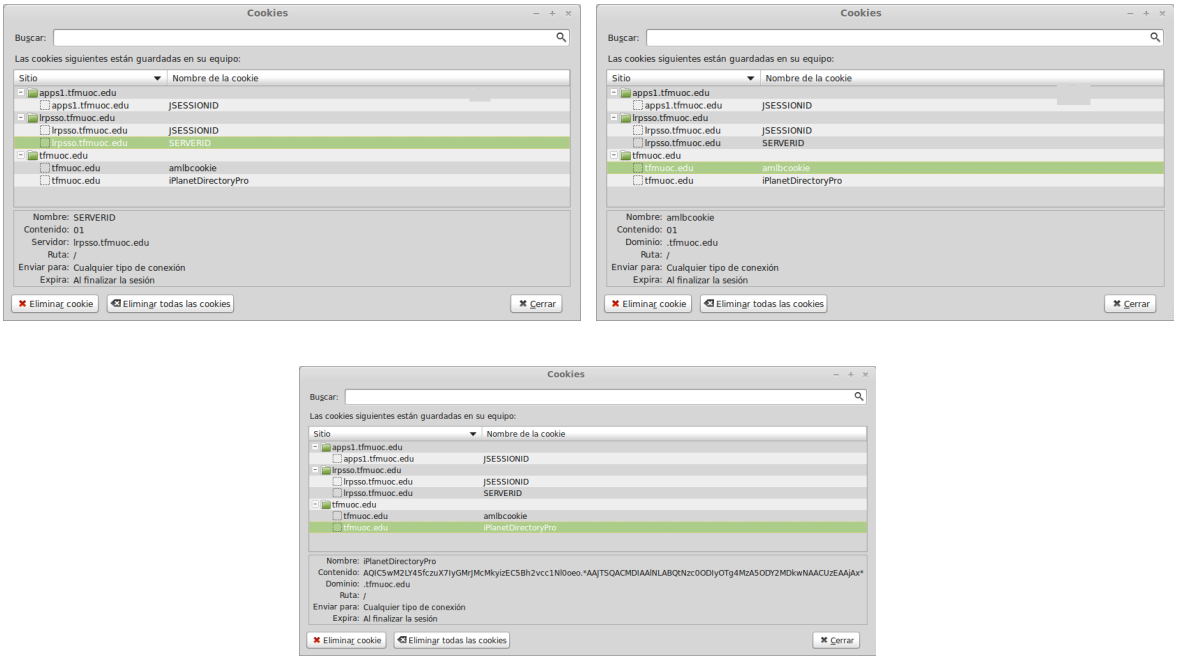

figura 59: Cookies. Login aceptado (II).

Conclusión

Como hemos podido ver, el intercambio de mensajes entre el navegador del usuario, el Policy Agent y el servidor SSO sigue el esquema mostrado en la figura 2 del presente documento, obteniendo como resultado la creación de 2 tipos de cookies:

- **Cookies de servidor**, referentes al servidor de aplicación y al servidor SSO asignado por el balanceador.
- **Cookies de dominio**, que serán dos, la que contiene el token de sesión SSO (iPlanetDirectoryPro) y la que nos indica qué servidor SSO nos ha atendido la cookie (amlbcookie). El token de sesión es el que no variará mientras el usuario esté conectado (y no le caduque la sesión) y será la utilizada para autenticar al usuario en todos los recursos gestionados por el entorno SSO (OpenAM).

Integrando la aplicación Manager de Tomcat en OpenAM

Ahora procederemos a la integración de la aplicación “Manager” que viene de serie con la instalación de Tomcat, y que hace uso del Realm básico incrustado en la propia configuración del servidor de aplicaciones.

Además, podemos ver que en el documento de despliegue de la aplicación, utiliza un sistema propio de Login (en el fichero WEB-INF/web.xml de la aplicación):

```
<!-- Define the Login Configuration for this Application -->
<login-config>
  <auth-method>BASIC</auth-method>
```

```
  <realm-name>Tomcat Manager Application</realm-name>
</login-config>
```

El repositorio de credenciales de usuario lo obtiene del fichero $CATALINA_HOME/conf/tomcat-users.xml, que podría contener lo siguiente:

```
<tomcat-users>
  <role rolename="manager-gui"/>
  <user username="tomcat" password="s3cr3t" roles="manager-gui"/>
</tomcat-users>
```

El cual es utilizado a través de la configuración de Realm que se almacena en el fichero $CATALINA_HOME/conf/server.xml (se muestra sólo un extracto):

```
<Resource name="UserDatabase" auth="Container"
        type="org.apache.catalina.UserDatabase"
        description="User database that can be updated and saved"
        factory="org.apache.catalina.users.MemoryUserDatabaseFactory"
        pathname="conf/tomcat-users.xml" />
[…]
<Realm className="org.apache.catalina.realm.UserDatabaseRealm"
              resourceName="UserDatabase"/>
```

Ahora, según la documentación de la aplicación, para acceder a la misma, se requiere que el usuario disponga de un rol de "manager-gui", y para aprovechar los usuarios que se crearon para la aplicación anterior, vamos a asignar este rol a, por ejemplo, la usuaria "chris".

Como ya disponemos del Policy Agent instalado, no es necesario proceder, y aprovecharemos la configuración del mismo para agregar todas las reglas necesarias para el acceso a esta aplicación.

Configuración de política(s)

Accederemos, en la carpeta de "Control de Acceso" al dominio de nivel superior:

figura 61: Realm. Dominio de seguridad.

Ahora definiremos una "Directiva" que llamaremos "tomcat":

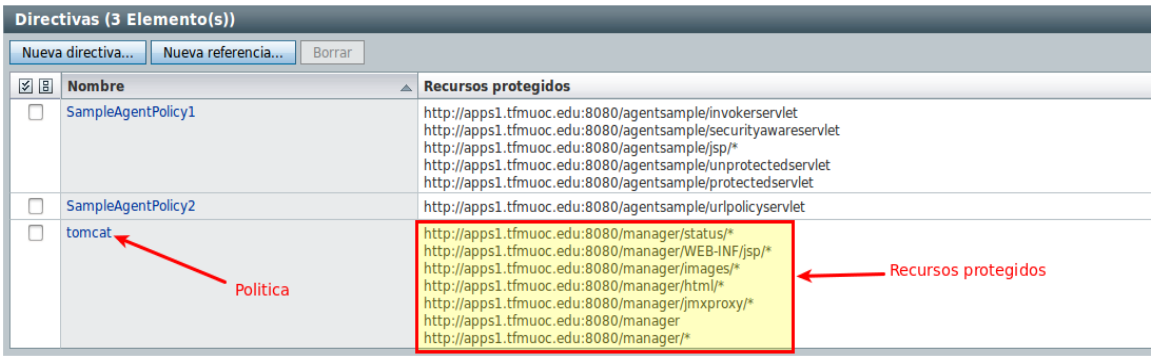

figura 60: Política para la aplicación "Manager" de Tomcat.

Y para cada recurso compartido (que inicialmente no habrá ninguno), crearemos una nueva regla de autorización de acceso GET/POST. por ejemplo:

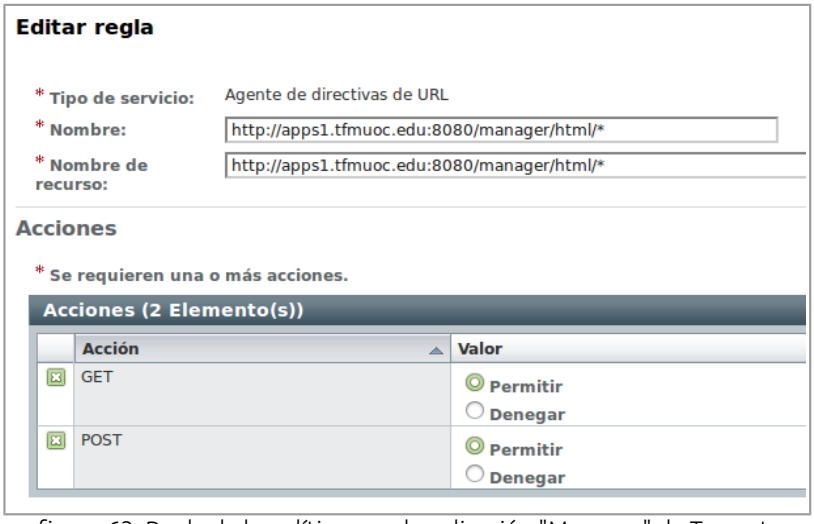

figura 63: Regla de la política para la aplicación "Manager" de Tomcat.

Introduciendo una regla por cada recurso a proteger, los cuales son:

```
http://apps1.tfmuoc.edu:8080/manager/status/*
http://apps1.tfmuoc.edu:8080/manager/WEB-INF/jsp/*
http://apps1.tfmuoc.edu:8080/manager/images/*
http://apps1.tfmuoc.edu:8080/manager/html/*
http://apps1.tfmuoc.edu:8080/manager/jmxproxy/*
http://apps1.tfmuoc.edu:8080/manager
http://apps1.tfmuoc.edu:8080/manager/*
```

Adicionalmente agregaremos los asuntos que tienen permisos de acceso, que como hemos dicho, son aquellos que tienen el rol “manager-gui”:

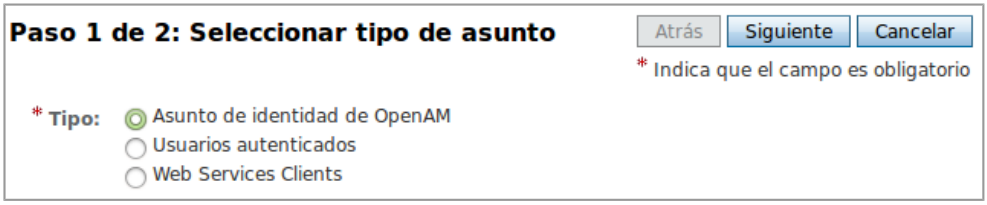

figura 64: Selección del tipo de usuarios con acceso al recurso

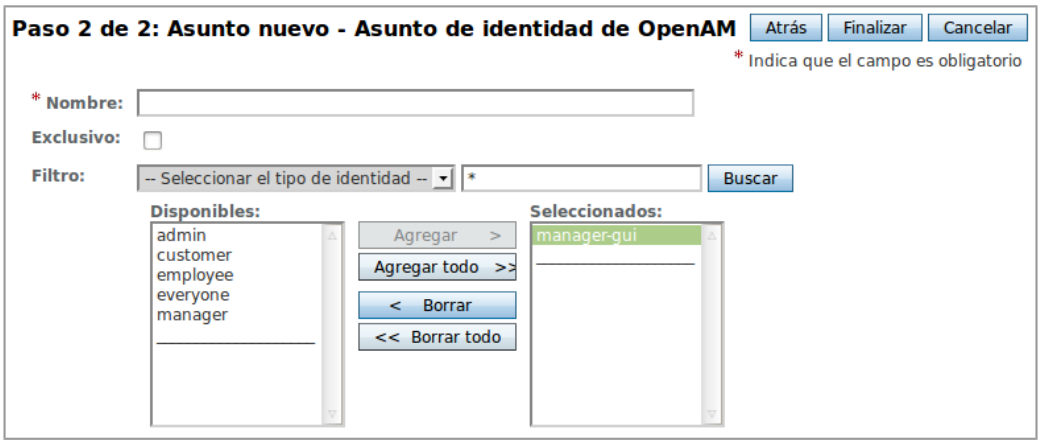

figura 62: Selección de los usuarios/grupos de acceso al recurso

Quedando

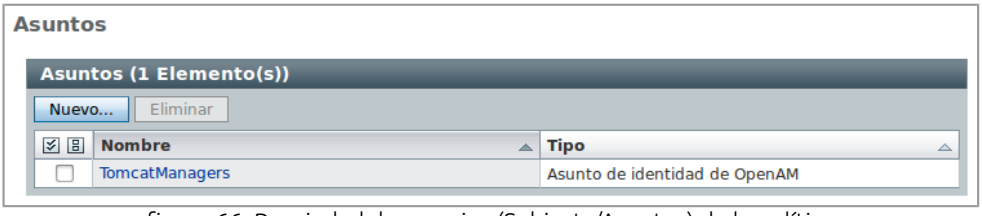

figura 66: Propiedad de usuarios (Subjects/Asuntos) de la política

El problema de la integración

Y ahora podemos proceder a probar accediendo a la URL de la aplicación.

http://apps1.tfmuoc.edu:8080/manager

El primer efecto es que, a diferencia de la aplicación anterior (agentsample), no tiene elementos declarados como públicos (ver pg. 29), el mero acceso a la aplicación requerirá autenticación de usuario y el Policy Agent interceptará el acceso y nos redireccionará a la página de Login de OpenAM.

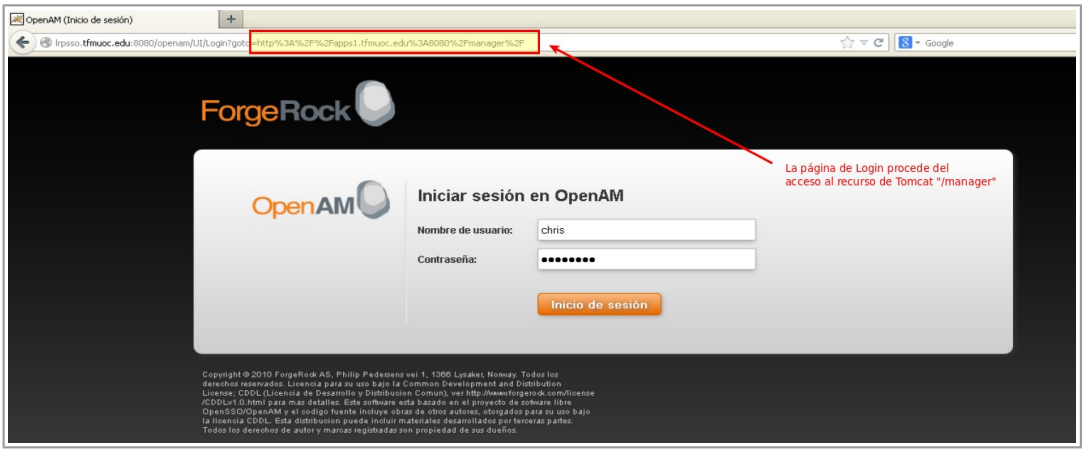

figura 65: Login (OpenAM) de la aplicación "Manager" de Tomcat

Se puede observar el salto a la página de Login a partir de la URL:

```
http://lrpsso.tfmuoc.edu:8080/openam/UI/Login?goto=http%3A%2F
%2Fapps1.tfmuoc.edu%3A8080%2Fmanager%2F
```

Lo que indica que la aplicación está protegida por OpenAM. Sin embargo, al acceder con el usuario autorizado, la aplicación, que dispone de un sistema propio de identificación de usuarios, muestra de nuevo un diálogo de Login, que no pertenece a OpenAM, si no a la propia aplicación.

Mostrando el diálogo:

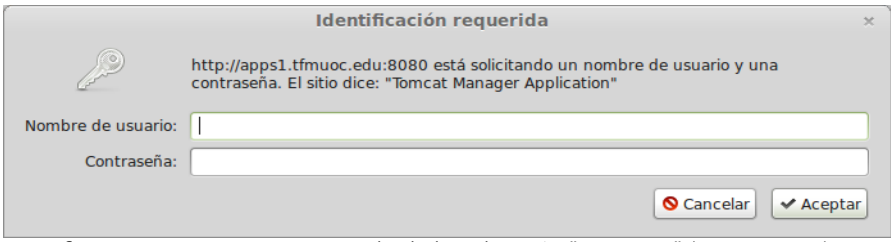

figura 68: Login no supervisado de la aplicación "Manager" (no OpenAM)

Y para OpenAM, el diálogo de Login que presenta la aplicación será un componente más, y de aquí no pasará, ya que en la instalación del agente, en la configuración de Tomcat, ha sido sustituido el Realm por defecto del servidor de aplicación por la clase que gestiona el Realm de OpenAM.

Además, dentro de OpenAM, el usuario empleado para acceder a la aplicación (chris) está logado correctamente:

En definitiva, lo que quiere decir que, para una correcta integración de las aplicaciones con un entorno SSO deben prescindir de los sistemas de autenticación de usuarios propios y ceder todo el control al sistema SSO, ya que, de lo contrario, ocurrirán cosas como lo que pasa con la aplicación Manager de Tomcat.

Por tanto, la integración de aplicaciones en un entorno SSO debe utilizar

figura 67: Sesiones de usuarios activas en OpenAM

todos los mecanismos de autenticación proporcionados por el mismo, lo cual, además, permite que los usuarios puedan acceder, de forma unificada, a todas las aplicaciones (ya que la gestión de usuarios pertenece a OpenAM y no a ellas mismas) en el nivel de autorización que se haya definido en cada política.

Test completo con Proxy de aplicación en la DMZ

Ahora que ya hemos podido ver el funcionamiento del entorno SSO, y de cómo el usuario se autentica ante OpenAM, y que este le envía el token de sesión SSO a través de la cookie iPlanetDirectoryPro, ahora procederemos a ver todo funcionando con la integración completa del proxy AJP de sesión funcionando e integrado con OpenAM.

En este caso realizaremos la prueba con la aplicación “agentsample2”, que se ha obtenido del paquete de instalación del PolicyAgent de la versión 3.3.

La página inicicial (sobre la que hemos definido que no aplican las políticas) es:

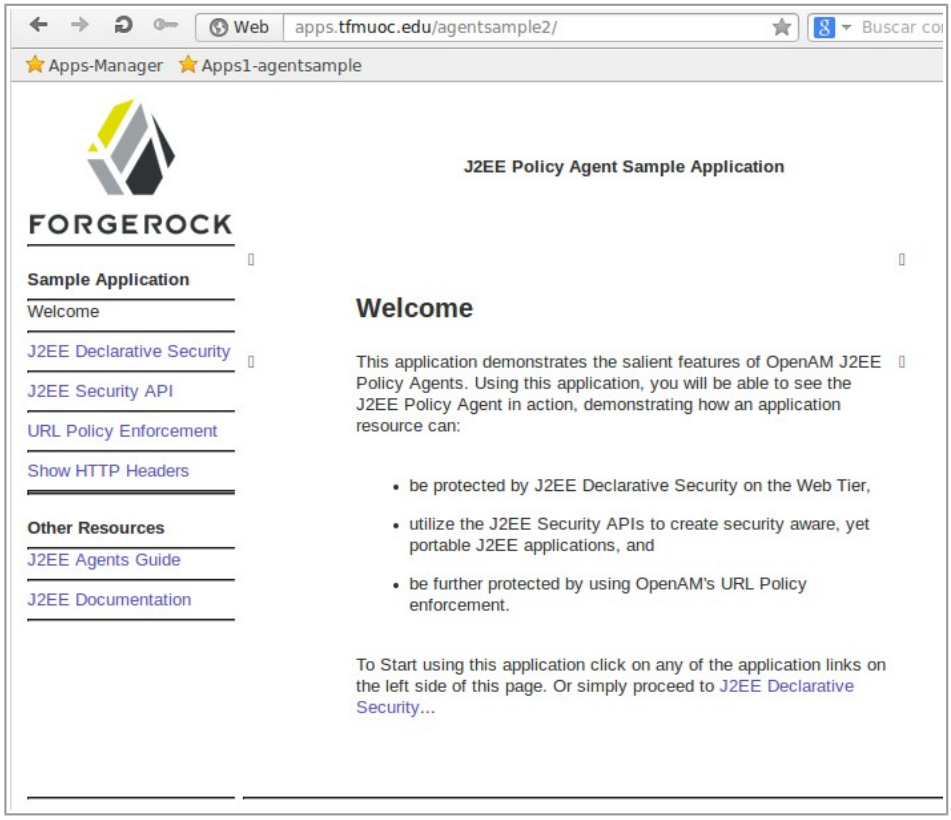

figura 70: Página inicial AgentSample2

Ahora intentamos acceder a un recurso protegido:

http://apps.tfmuoc.edu/agentsample2/urlpolicyservlet

Y la secuencia de navegación obtenida desde Chrome (F12):

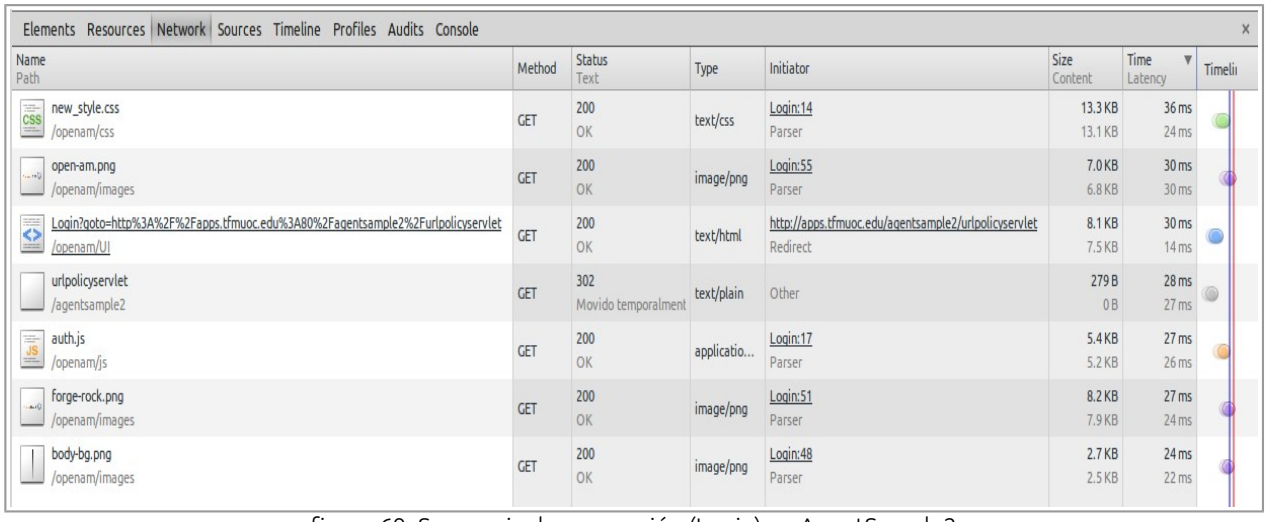

figura 69: Secuencia de navegación (Login) en AgentSample2

Concretamente, podemos ver que la página iniciadora de la redirección es: http://apps.tfmuoc.edu/agentsample2/public/urlpolicy.html

Y la página a la que se redirecciona es:

http://apps.tfmuoc.edu/openam/UI/Login?goto=http%3A%2F%2Fapps.tfmuoc.edu%3A80%2Fagentsample2%2Furlpolicyservlet

Y nos aparece la página de Login de OpenAM para que el usuario se autentique:

figura 72: Login SSO

Nos autenticamos con un usuario autorizado (chris) y la secuencia de navegación obtenida es:

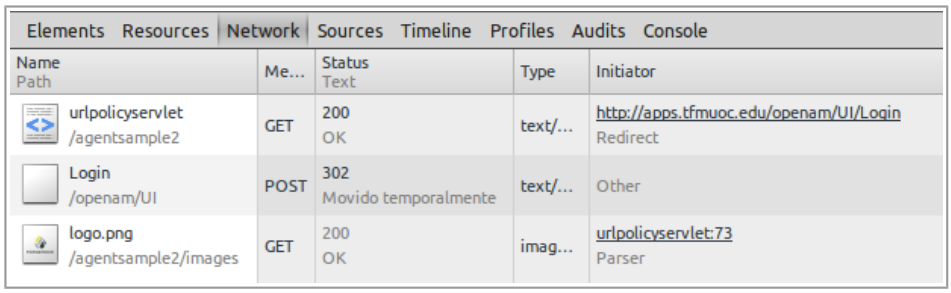

figura 73: Secuencia con sesión de usuario autorizada (Allow).

Y si vemos cómo quedan las cookies:

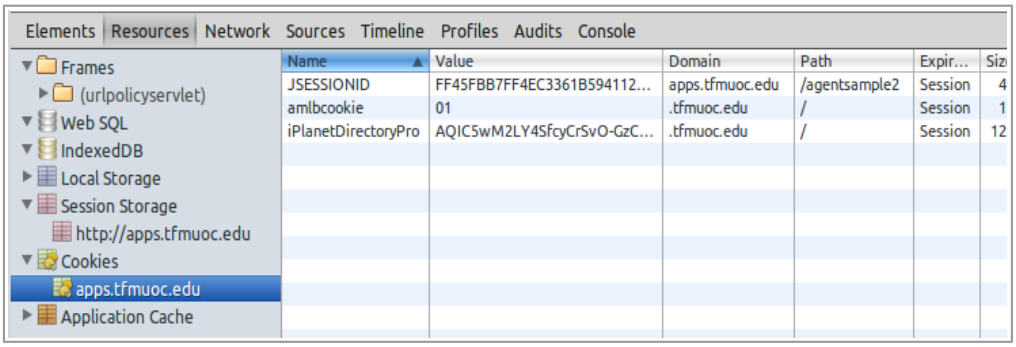

figura 71: Cookies de sesión (Allowed). AgentSample2

Donde, como podemos ver, tenemos la cookie de la aplicación (**JSESSIONID**) y la que contiene el token SSO (**iPlanetDirectoryPro**).

Ahora, para comprobar que, efectivamente, la sesión SSO está establecida y el token es válido en otra aplicación, procederemos a acceder, con la misma sesión (de la usuaria Chris) a la aplicación /Agentsample:

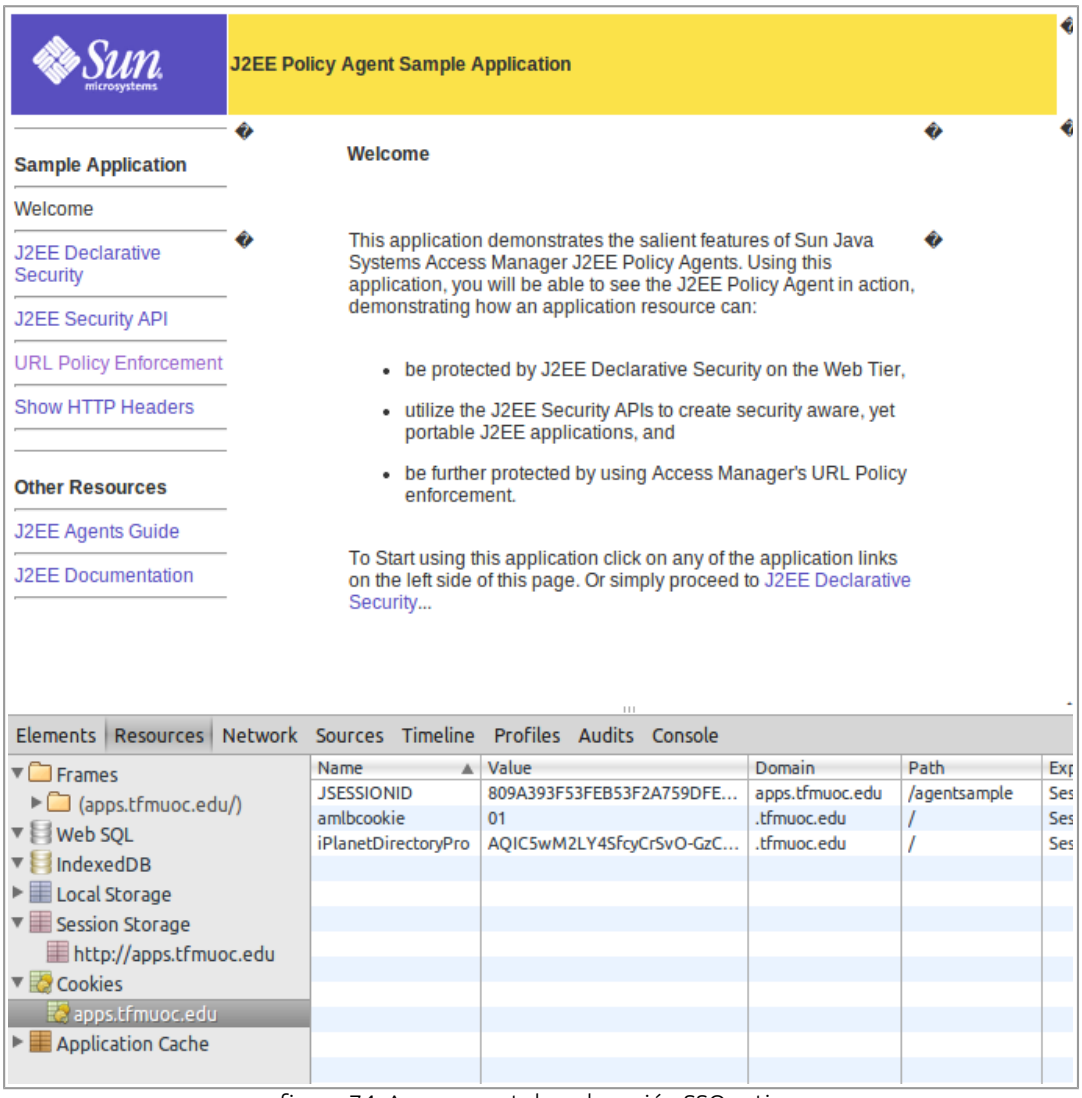

figura 74: Acceso con token de sesión SSO activo.

Y entramos en la URL (en la que Chris si tiene acceso):

http://apps.tfmuoc.edu/agentsample/urlpolicyservlet

Y obtenemos la página de acceso autorizado:

figura 75: Acceso autorizado

Y vemos que OpenAM no ha redirigido al navegador a una página de Login debido a que ya tenía un token SSO de sesión válido, y que, además, tenía autorización para acceder al recurso.

Por último, hemos podido observar, también, que el nombre de dominio proporcionado por el proxy de aplicación: apps.tfmuoc.edu, no ha cambiado en ningún momento, ni siquiera al realizarse la autenticación de usuario, quedando todo perfectamente integrado en un único nombre DNS.