Universitat Oberta de Catalunya

uoc.edu

Evaluación de la seguridad en el acceso a servicios web desde portlets integrados en gestores de contenido

Estudiante: Fernando Rubí Aguiló
Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones. Protocolos y aplicaciones de seguridad

Consultor: Amadeu Albós Raya
Profesor: Victor Garcia Font

02/06/2020

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Evaluación de la seguridad en el acceso a servicios web desde portlets integrados en gestores de contenido*
Nombre del autor:	*Fernando Rubí Aguiló*
Nombre del consultor/a:	*Amadeu Albós Raya*
Nombre del PRA:	*Victor Garcia Font*
Fecha de entrega (mm/aaaa):	06/2020
Titulación::	*Máster Universitario en Seguridad de las Tecnologías de la Información y de las Comunicaciones*
Área del Trabajo Final:	*Protocolos y aplicaciones de seguridad*
Idioma del trabajo:	*Español*
Palabras clave	*Portal Web, Portlet, Web Services*

Resumen del Trabajo:

Este trabajo se basa en el análisis de la seguridad de los *portlets* y *web services* utilizados para integrar aplicaciones en portales web.

Debido a que actualmente existe una clara tendencia a la implantación de portales web especializados para la gestión de los sistemas de información de las organizaciones, desarrollando componentes modulares, denominados *portlets* y habilitando servicios web para acceder a bases de datos, desde los mismos *portlet*s o aplicaciones para dispositivos móviles, se hace necesario establecer mecanismos para que estos desarrollos se realicen de forma segura.

Para ello, inicialmente se han desarrollado un portlet y un servicio web, y a continuación se ha hecho un análisis de los problemas que podemos encontrar en un despliegue de estos, un entorno aproximado al que podríamos encontrar en una organización, buscando detectar vulnerabilidades que afectan a algunos de los pilares básicos de la seguridad de la información, como son la autenticación, la confidencialidad o la disponibilidad.

A continuación se han analizado qué soluciones pueden mitigar las vulnerabilidades detectadas, que se han ido incorporando una a una, para poder verificar su funcionamiento separadamente.

Finalmente se ha obtenido una guía de los pasos a seguir para implementar en el desarrollo de portlet y servicios web que utilicen tecnologías como las que se han propuesto en el trabajo.

Abstract:

This work is based on the analysis of the security of portlets and web services used to integrate applications in web portals.

Because there is currently a clear trend towards the implementation of specialized web portals for managing organizations' information systems, developing modular components, called portlets, and enabling web services to access databases, from the same portals or applications for mobile devices, it is necessary to establish mechanisms for these developments to be carried out safely.

For this, initially a portlet and a web service have been developed, and then an analysis has been made of the problems that we can find in a deployment of these, an approximate environment that we could find in an organization, seeking to detect vulnerabilities that affect some of the basic pillars of information security, such as authentication, confidentiality or availability.

Below we have analyzed which solutions can mitigate the detected vulnerabilities, which have been incorporated one by one, in order to verify their operation separately.

Finally, a guide has been obtained on the steps to follow to implement portlet development and web services that use technologies such as those proposed in the work.

Índice

Índice de imágenes

1. Introducción

1.1 Contexto y justificación del Trabajo

Actualmente en el mercado es muy común encontrar empresas que utilizan gestores de contenido o *CMS* (*Content Management System*) para facilitar la publicación y gestión de contenido a los gestores de la información de las organizaciones. Dentro de estos *CMS* existen algunos más especializados, llamados portales web, que además de facilitar la gestión de contenido, también incluyen herramientas adicionales que van más allá de la gestión de contenido, como puede ser foros, gestores de documentos, noticias, etc.

Aunque los portales web incluyen una gran cantidad de herramientas, en muchas ocasiones no son las que la organización necesita. Por este motivo algunos portales han evolucionado para facilitar la inclusión de módulos adicionales, ya sean de programación propia de la organización o de terceros.

Estos módulos adicionales, denominados *portlets* suelen tener un objetivo diferente al de gestor de contenido, pueden ser una única aplicación (o ser parte de otra aplicación compuesta de diferentes *portlets*), y pueden comunicarse con otros servicios, dentro de la misma organización o con otros externos.

Damos por supuesto que las organizaciones que dan soporte a los portales web, y los servidores de aplicaciones donde se alojan, tienen sus propios mecanismos para controlar la seguridad de sus productos, publicando actualizaciones para ir solucionando las vulnerabilidades de seguridad que se van encontrando, pero ¿es posible que al integrar un *portlet* o un servicio web desarrollado por nosotros estemos creando un agujero de seguridad?

En este trabajo se intentará determinar si un desarrollo adicional a medida, para incluirse en un portal web, debería incorporar algún tipo de seguridad específica, que garantice la confidencialidad, integridad y disponibilidad, que debe proporcionar cualquier sistema de información, con algún mecanismo de seguridad proporcionada por el mismo portal web o servidor de aplicación, o simplemente, si estos ya implementan suficientes mecanismos de seguridad, y no hacer nada en este aspecto.

1.2 Objetivos del Trabajo

Cómo se indica en el titulo, el objetivo principal de este trabajo es la evaluación de la seguridad en el acceso a servicios web desde *portlet*s integrados en gestores de contenido, analizando de forma práctica que se consigue antes y después de aplicar diferentes mecanismos de seguridad en *portlet*s y servicios web.

Para alcanzar este objetivo principal es necesario determinar una serie de objetivos parciales:

- Analizar los mecanismos de seguridad que se pueden implementar en los servicios web que garanticen un mínimo de seguridad en la comunicación con las aplicaciones.
- Analizar los mecanismos de seguridad que implementan los portales web que existen en el mercado, y de qué forma se pueden integrar en los *portlets*.
- Investigar los problemas de seguridad que se plantean en el desarrollo de *portlets* y servicios web, analizando las amenazas que estos pueden generar en los entornos en los que se despliegan.
- Plantear mejoras en la seguridad que corrijan los problemas de seguridad detectados.

1.3 Enfoque y método seguido

Este trabajo está enfocado a realizar un trabajo más práctico que teórico, ya que en muchas ocasiones no hay tiempo para tener la certeza de que se está realizando una configuración segura de un sistema de información. Un entorno de pruebas que se aproxime a una situación real puede ser un punto de partida muy válido para adquirir la experiencia necesaria que requiere este tipo de responsabilidades.

Para ello se ha seguido una metodología ágil ya que esta va a permitir un desarrollo del trabajo de forma fluida. Este va a consistir en ir incorporando diferentes niveles de seguridad para ir haciendo las pruebas necesarias en cada nivel e ir sacando conclusiones que completen la memoria final.

De esta forma la estrategia va a ser la siguiente:

- Analizar que tipos de portales web existen en el mercado, seleccionando uno, para investigar los mecanismos de que dispone para garantizar la seguridad.
- Implementar un entorno de pruebas que permita aplicar alguno (o varios) de los sistemas de seguridad. Este entorno debería aproximarse a una instalación real, por lo que se deberá desarrollar un módulo y un servicio web. El objetivo de contar con un desarrollo propio es controlar el funcionamiento de estas aplicaciones para determinar en que punto se producen las vulnerabilidades, comprobando que el acceso al módulo del portal web, se realiza sólo por usuarios autorizados, y comprobando que la comunicación entre el módulo y el servicio web se realiza de forma segura.
- Analizar las vulnerabilidades detectadas, y comprobar qué mecanismo se pueden utilizar para solucionar dichas vulnerabilidades.

- Comprobar que se implementan correctamente los mecanismos analizados, validando que las vulnerabilidades se corrigen. Para ello se irán incorporando los mecanismos, a cada elemento vulnerable, y con cada incorporación se realizará una verificación de la corrección de la vulnerabilidad.
- Obtener un documento con las conclusiones del trabajo realizado, enfatizando las consideraciones mínimas que se deberían de tener en cuenta para la puesta en producción de un entorno como el planteado.

1.4 Planificación del Trabajo

Cod	Actividad	Inicio	Fin	Duración
1	Planificación			
1.1	Establecer un contexto y el problema a resolver	19/02/2020	28/02/2020	10
1.2	Definición de los objetivos	29/02/2020	01/03/2020	2
1.3	Elaborar lista de tareas	02/03/2020	03/03/2020	2
	Entrega PEC1	**03/03/2020**	**03/03/2020**	**Hito**
2	Investigación			
2.1	Estado del arte	04/03/2020	11/03/2020	8
3	Desarrollo			1
3.1	Preparación del entorno	12/03/2020	14/03/2020	3
3.2	Desarrollo de un servicio web	15/03/2020	17/03/2020	3
3.3	Desarrollo de un portlet	18/03/2020	20/03/2020	3
3.4	Análisis de vulnerabilidades	21/03/2020	28/03/2020	8
3.5	Documentación riesgos entorno	29/03/2020	30/03/2020	2
	Entrega PEC2	**31/03/2020**	**31/03/2020**	**Hito**
3.6	Análisis de mecanismos de seguridad existentes	01/04/2020	08/04/2020	8
3.7	Incorporación de un mecanismo de seguridad	09/04/2020	15/04/2020	7
3.8	Validación de las soluciones	16/04/2020	25/04/2020	10
3.9	Documentación de las soluciones	26/04/2020	28/04/2020	3
	Entrega PEC3	**28/04/2020**	**28/04/2020**	**Hito**
4	Memoria del trabajo			
4.1	Revisión de la documentación previa	29/04/2020	03/05/2020	5
4.2	Revisión general de la memoria	04/05/2020	18/05/2020	15
4.3	Elaboración de la presentación	19/05/2020	02/06/2020	15
	Entrega PEC4	**02/06/2020**	**02/06/2020**	**Hito**

1.5 Breve sumario del producto obtenido

Este trabajo está pensado para realizar un estudio práctico de los mecanismos de seguridad a nivel de aplicación que pueden aplicar en un entorno de una aplicación distribuida. Por tanto, este trabajo se podrá considerar como un producto que proporciona una guía detallada de los mecanismos existentes para asegurar tanto *portlets* como *web services* a nivel de aplicación.

1.6 Breve descripción de los otros capítulos de la memoria

El este primer capitulo se ha dado una descripción del objetivo de este proyecto, para justificar el desarrollo que se expondrá en los sucesivos capítulos.

El segundo capítulo consistirá en un estudio introductorio del estado del arte en que se encuentran los portales web, los *portlets* y los servicios web, y se procederá al estudio de los mecanismos de seguridad que ofrecen, así mismo se propondrá una serie de frameworks que podría disponer una aplicación y un servicio web real.

El tercer capítulo se propondrá un posible escenario dónde se ejecutarán las pruebas iniciales para realizar el análisis de los mecanismos de seguridad analizados en el capitulo anterior.

En el cuarto capítulo se buscará describir las vulnerabilidades que se pueden producir en el escenario planteado.

En el quinto se describirán los pasos para incorporar las medidas correctivas que se consideren más adecuadas para solucionar las vulnerabilidades detectadas, y así mismo se detallará en que momento y en que medida queda resueltas.

Finalmente, en el capitulo sexto se expondrán las conclusiones de este trabajo.

2. Estado del arte

2.1 Esquema de los elementos que componen el entorno del proyecto

Antes de entrar en detalle en el estado del arte de los elementos que forman parte del sistema de información del entorno, es interesante exponer un diagrama descriptivo que muestre la relación entre los elementos que se van a analizar.

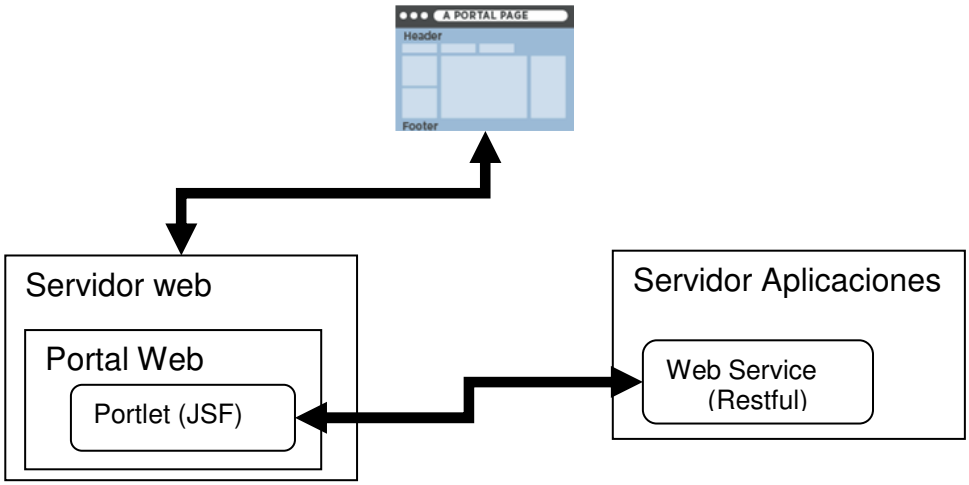


Imagen 1. Diagrama descriptivo del entorno de estudio

En el diagrama encontramos los siguientes elementos, un navegador web con el que el usuario realiza la comunicación con el portal, la máquina en la que está instalado un servidor web que contiene un portal web en la que hay desplegado un *portlet*, y la máquina en la que está instalado el servidor de aplicaciones en el que esta desplegado el *web service* y que en la que también está instalada una base de datos.

En este trabajo se estudiará que sucede entre las comunicaciones que se producen entre el usuario y el *portlet*, y entre el *portlet* y el servicio web, y como se puede controlar la autenticación y autorización a nivel de aplicación, y si finalmente es necesario asegurar la comunicación a nivel de protocolo. Pero antes de iniciar con el estudio se va a realizar una aproximación de los componentes del diagrama expuesto.

2.2 Portales web y Liferay

Como se ha comentado en la introducción los gestores de contenido o *CMS* (en inglés *Content Management System*) facilitan la gestión de contenido que las organizaciones quieren publicar. Aunque hoy en día CMS es un término muy extenso, y existen algunos más especializados que además de facilitar la gestión de contenido, incluyen herramientas adicionales que ofrecen al usuario, de forma fácil e integrada, una serie de recursos que van más allá de la gestión de contenido, como puede ser foros, gestores de documentos, noticias,

archivos digitales, etc., y por tanto se hace necesario clasificar cada uno de los diferentes tipos de CMS que existen, así podemos encontrar:

- *WCM* (Web Content Management): Este tipo se centra en la gestión de contenido web, está orientado a la creación de contenidos públicos de forma colaborativa.

- *ECM* (Enterprise Content Management): Este tipo se centra en la gestión de contenido empresarial, como puede ser software para gestionar el escaneo, almacenaje, organización y recuperación de documentos físicos.

- *DAM* (Digital Asset Management): Este tipo se centra en la gestión de activos digitales, utilizados frecuentemente por empresas de medios de comunicación para catalogar, tomar notas, almacenar, recuperar y distribuir audios, vídeos, animaciones y otros contenidos y audiovisuales digitales.

- *DXP* (Digital Experience Platform) (Este tipo de plataformas mejoran la experiencia digital de la empresa, facilitando la creación de contenido adaptable para cualquier canal digital existente a día de hoy, además de mejorar las operaciones de negocio a través de la digitalización y la integración de procesos.

Nos centraremos en este último, ya que actualmente hay una clara tendencia de portales web que están centrados en convertirse en DXP. De entre este tipo de portales web más especializados Liferay se ha posicionado como líder del mercado, si miramos como Gartner, una de las consultoras más relevantes en mundo IT, la ha posicionado como líder en su *Gartner Magic Quadrant for Digital Experience*.

Uno de los aspectos que más nos interesa en este trabajo es el de la seguridad, y Liferay implementa no sólo los mecanismos de seguridad que se supone que van intrínsicamente ligados a un portal que han de utilizar muchos usuarios, como usuarios, grupos y roles, sino que también permite integrar sistemas de seguridad externos.

De esta forma al situar un *portlet* dentro de una página privada de este portal, se está consiguiendo validar la autorización y autorización para acceder a ese *portlet*. Si a esto sumamos que Liferay permite integrar un sistema SSO (Single Sign On), o autenticación única, donde los usuarios se autentican una sola vez en un conjunto de sistemas relacionados, como puede ser un correo web, utilizando CAS (Central Authentication Service) para una autenticación federada, que utilice como base de datos de usuarios un LDAP del sistema de la organización, conseguimos solucionar una parte del problema de la autenticación para la explotación adecuada del *portlet*.

Imagen 2. Posicionamiento de Liferay según Gartner

En cuanto a vulnerabilidades, como cualquier otro portal, también está expuesta a fallos de seguridad, de hecho en la misma web de Liferay encontramos un resumen de las vulnerabilidades que se van encontrando y en qué actualización se puede encontrar la solución al problema.

2.3 Portlets

Una de las características que hacen que Liferay destaque sobre el resto es la de incorporar las especificaciones, JSR 168 y JSR 286.

Los JSR (Java Specification Request) son documentos formales que describen las especificaciones y tecnologías propuestas para añadir funcionalidades a la plataforma Java, existe un gran número de ellas, de entre estas las JSR 168 y JSR 286 definen como se han de desarrollar los componentes modulares de interfaz de usuario, denominados *portlets*. De esta forma, un *portlet* que cumpla con las especificaciones mencionadas, podrá desplegarse y ejecutarse dentro de un portal web que implemente dichas especificaciones, como liferay.

Este hecho es importante, ya que un *portlet* se ejecuta dentro de un contenedor que gestiona su ciclo de vida, realizando llamadas a los métodos equivalentes a render() y destroy() en el framework utilizado para el desarrollo, y en el origen del paradigma de los *portlets*, cada empresa realizó su propia implementación generando problemas de seguridad. El hecho de contar con una especificación fuerza a tener una estructura y un comportamiento idéntico, y sobretodo, al contar con unos parámetros de seguridad verificados por la autoridad certificadora de las especificaciones.

Por este motivo, cuando se desarrolla un *portlet* también puede ser interesante añadir un *framework* de trabajo, que permita tener el código lo más ordenado

posible y siguiendo un patrón MVC (Modelo-Vista-Controlador), y de entre todos los disponibles hay uno que resulta bastante interesante, Java Server Faces (JSF). Este framework para aplicaciones Java basadas en web simplifica el desarrollo de interfaces de usuario, ya que incorpora componentes con funcionalidades predefinidas que se pueden utilizar fácilmente. Además, también permite trabajar con librerías más especializadas, como Primefaces, que es una biblioteca de componentes para JSF que cuenta con un conjunto de componentes enriquecidos, donde destaca el uso de Ajax para la comunicación entre el navegador del cliente y el servidor, de forma que se permite trabajar como si se tratara de una aplicación de escritorio.

La facilidad de este modelo radica en que, por un lado se define la interface de usuario en un archivo .xhtml, y por otro lado, en el controlador, se definen las operaciones que se ejecutaran con los valores que le llegan del .xhtml, de forma que para las operaciones habituales de mantenimiento de tablas, se va a requerir poca programación.

Liferay conocedora del potencial que tiene esta combinación de tecnologías, facilita un conjunto de librerías especializadas, denominado *Liferay Portlet Bridge*, que dan un soporte específico a JSF (y librerías de terceros), que permite completar el ciclo de vida de las fases en que se procesa la generación del html que se envía al cliente.

2.4 Servidor web Tomcat

Ahora que ya sabemos que vamos a utilizar un portal web Liferay, también es necesario definir en que servidor web se va a desplegar, al fin y al cabo, liferay es una aplicación que se despliega en un servidor Java.

Hay que tener presente que desde la misma página de Liferay existe la posibilidad de descargar Liferay desplegado en un servidor. Este formato de empaquetamiento (*bundle*) facilita que el portal web sea puesto en funcionamiento en muy poco tiempo, un tiempo atrás facilitaba el bundle desplegado en los servidores más conocidos, ahora se proporciona únicamente un *bundle* de Tomcat, uno de los servidores más utilizados en el mercado.

El hecho de contar con un servidor veterano, Apache Software Fundation lo lanzó en 1999, puede hacer que tengamos una cierta sensación de seguridad, ya que la experiencia adquirida en esta materia le confiere un alto grado de especialización, pero también es cierto que, una vez detectada una vulnerabilidad, se pueden realizar ataques a un gran número de organizaciones que lo utilizan. Por tanto, también es necesario estar al día a las actualizaciones que se publican para aplicarla, en la medida de lo posible, cuanto antes.

De esta forma, ya que Liferay ha optimizado su portal para este servidor, se usará el bundle que se puede descargar desde su página.

2.5 Servicios web

Los servicios web también son componentes a los que se puede acceder utilizando un protocolo web estándar HTTP. Realmente son una colección de métodos (llamados *endpoints*) que se pueden invocar desde una red interna o desde Internet, y que aceptan o retornan los datos solicitados con cada interacción. Estos servicios web se basan en el intercambio de información en un formato predefinido, que puede ser XML o JSON, de forma que puedan ser utilizados con cualquier tipo de aplicación independientemente del lenguaje en que se desarrolle.

Existen dos grandes tipos, SOAP y RESTful, el primero se basa en el intercambio de mensajes XML, que requieren de un archivo de definición de su estructura (WSDL) para poder interpretarlos, y los segundos se basan en el intercambio de mensajes en formato XML o JSON, siendo este último el más utilizado y el que se utilizará en este trabajo.

El mecanismo de funcionamiento de RESTful, es realizar una petición HTTP de tipo GET, PUT, POST o DELETE, incluyendo en el body de la petición el JSON con datos si se trata de un alta o modificación, o pasando los valores por URL si se trata de una petición de datos o una eliminación. En este mecanismo es sin estado, por lo que en cada petición se envía la información necesaria para su procesado.

Para el desarrollo se ha seleccionado RESTful, desarrollado en java (aunque podría utilizarse cualquier otro lenguaje). También se utilizado además JPA (Java Persistente Api) con el objetivo de no tener que programar toda la gestión con la base de datos, ya que con simples anotaciones en las clases java se mapea un acceso a campos de las tablas de la base de datos más fácilmente.

Este tipo de servicio web también cuenta con una especificación, la JSR 311, llamada JAX-RS (*Java API for RESTful Web Services*). De la misma forma que las anteriores tecnologías para desplegar este tipo de servicios es necesario buscar un servidor que cumpla con las especificaciones descritas.

2.6 Servidor de aplicaciones Wildfly

Uno de los servidores que ha demostrado ser más robusto en el uso de la combinación de tecnología RESTful y JPA es Wildfly. Este servidor de código abierto, que cambio de nombre en 2014, y que aún se sigue llamando JBoss por la comunidad de usuarios, esta certificado para Java EE, lo que significa que cumple con las especificaciones necesarias para desplegar la tecnología propuesta en este trabajo.

2.7 Seguridad del conjunto

Ahora que ya hemos determinado cuales son los elementos que forman parte del estudio, donde se entiende que separadamente cumplen con determinadas garantías de seguridad, debemos preguntarnos que puntos quedan debilitados al utilizarlos dentro de un sistema de información desarrollado para que estos elementos trabajen conjuntamente.

Entre las dudas que se pueden plantear tenemos las siguientes:

- Un *portlet* desplegado en un portal, se puede instalar en cualquier página, pública o privada. Si por algún incidente, un *portlet* que gestiona información comprometida se incrusta en alguna página que no debería, se generaría un problema importante de seguridad. Por tanto, se hace necesario analizar qué mecanismos se pueden utilizar para que no sea posible acceder a la información del *portlet* por parte de usuarios no autorizados.

- Normalmente un *web service* desplegado en un servidor de aplicaciones está pensado para que sea consultado desde cualquier punto de la red, y por tanto, no sólo se podría utilizar desde un portlet, sino que también será posible interrogarlo desde un una aplicación móvil o incluso directamente. Por tanto, será necesario analizar que únicamente las aplicaciones autorizadas puedan realizar peticiones al web service.

- Por otra parte, también puede darse el caso de que alguien esté interesado en acceder los datos que se gestionan en el portlet analizando la información que circula por la red. Por tanto, también será necesario analizar qué pasos hay que dar para proteger el envío y recepción de la información que circula por la red.

Ahora que se ha expuesto los elementos que van a formar parte del estudio de este trabajo, vamos a centrarnos en la parte de seguridad atendiendo a las vulnerabilidades comentadas.

3. Entorno de pruebas

3.1 Servidores

Para la ejecución de las pruebas se han preparado dos máquinas virtuales (utilizando el software de virtualización Oracle VM VirtualBox). Estas dos máquinas tienen como sistema operativo Debian 10 y utilizan la versión de java OpenJDK 11 que viene instalado por defecto en el sistema.

En la máquina que se ha de utilizar para atender los servicios web, se ha instalado, por un lado, una base de datos postgres (versión 11.7), y por otro lado, el servidor Wildfly, instalados mediante el gestor de paquetes del sistema.

Wildfly también necesitará un driver de postgres y una conexión a la base de datos para que el *web service* la pueda utilizar para acceder a la base de datos.

En la máquina que se va a utilizar para el portal web, únicamente se ha instalado un *bundle* de Liferay sobre tomcat (lo que significa que el portal ya está desplegado en el servidor y se puede poner en funcionamiento de forma rápida), este se ha descargado desde la misma página de Liferay. La versión descargada es la versión 7.2.10 DXP.

Inicialmente las máquinas se habían configurado para utilizar la misma interface de red wifi del host, llamada “Adaptador Puente” en la configuración del VirtualBox, para cada una de las máquinas, pero con esta configuración no era posible capturar el tráfico con Wireshak (no se veía trafico desde el host hasta ninguna de las direcciones IP de los servidores) debido a la gestión que realiza el Wireshark. Finalmente, la configuración de la red para estos servidores ha sido la siguiente:

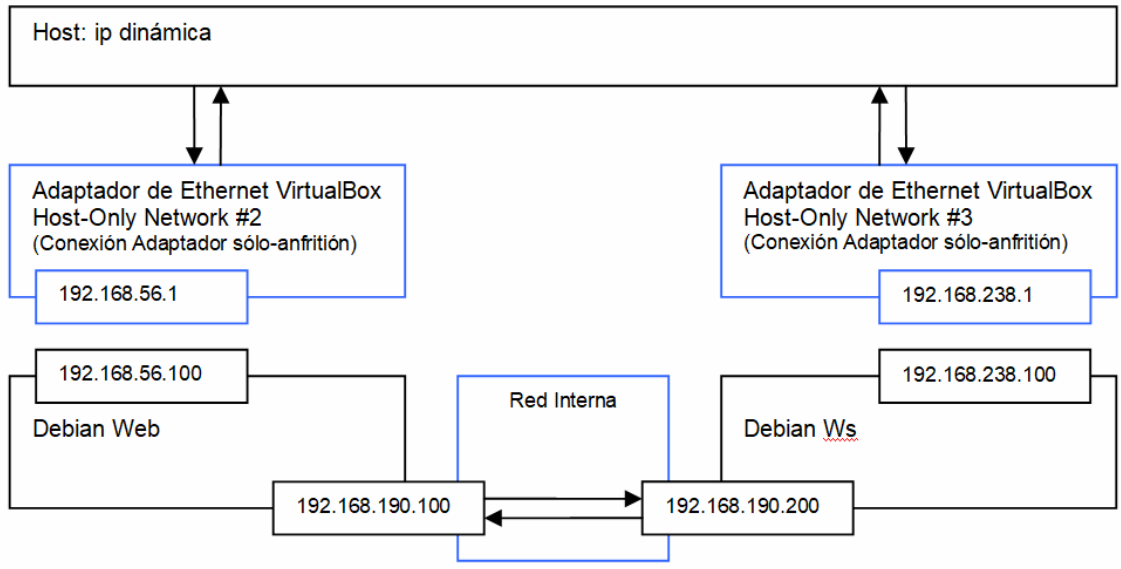


Imagen 3. Esquema de direcciones IP utilizadas en el entorno de pruebas

De esta forma desde el host puede acceder a wildfly desde el usuario. Aunque realmente el acceso a este servidor no seria necesario para el experimento, ya

que lo se que persigue es capturar la comunicación entre los dos servidores, y al situar estos en una red privada se limita el acceso al análisis de esta comunicación.

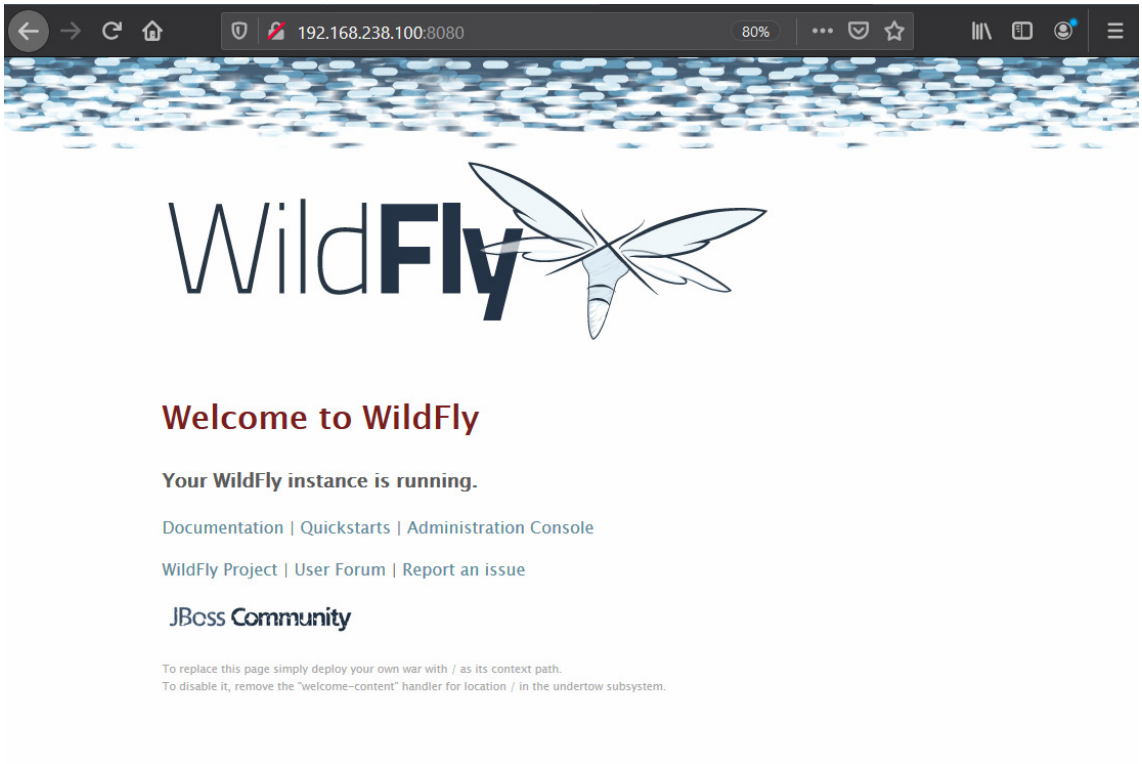


Imagen 4. Página de acceso a la consola de administración de wildfly

Por otro lado, para acceder al Liferay también se puede acceder desde el navegador del usuario:

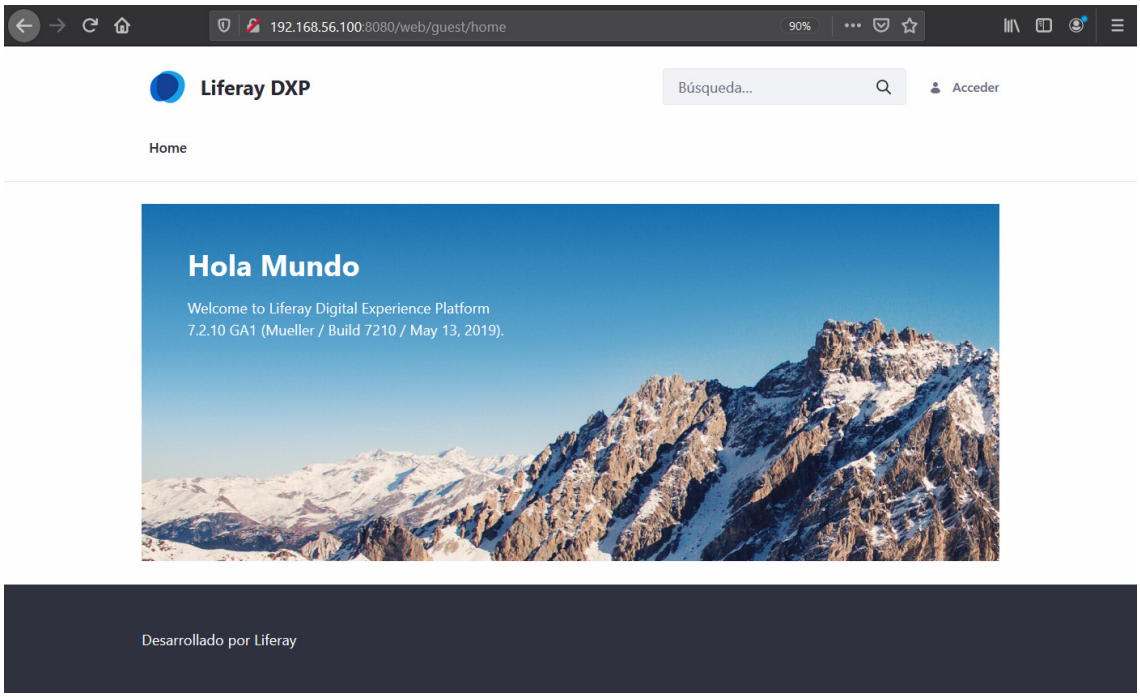


Imagen 5. Página de inicio de liferay

3.2 Aplicaciones: AplicacionWs y AplicacionPortlet

Para realizar las pruebas se han creado dos aplicaciones muy simples, que consisten, por un lado, un servicio web que contempla el mantenimiento de una única tabla de alumnos alojada en el servidor *postgres*, y por otro lado, un *portlet* que utilizando el servicio web realiza operaciones sobre esta tabla.

En el siguiente diagrama de secuencia se refleja el proceso de comunicación del entorno de pruebas:

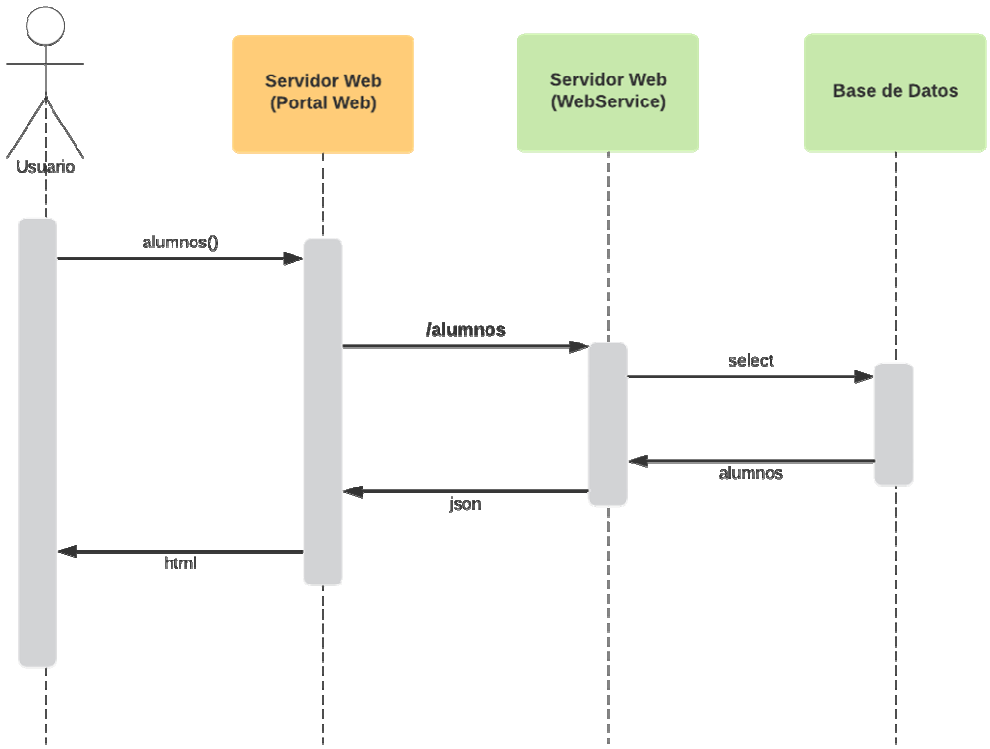


Imagen 6. Diagrama de secuencia

Un usuario que se autentica en el portal web, con su usuario y contraseña, tiene acceso a diferentes opciones, entre las que se encuentra un *portlet* que mantiene una tabla de alumnos. En este punto el usuario podrá realizar las opciones para las que este autorizado, y que serán atendidas en primer lugar por el portal web, que comunicará al *portlet* para que las procese, este las procesará solicitando o enviando peticiones a un *web service* que realizará las comunicaciones pertienentes con la base de datos.

Para el desarrollo de las dos aplicaciones se ha utilizado *Netbeans* 11.2, donde los aspectos más destacados de su desarrollo son:

- *Web service*: AplicacionWs, se genera utilizando un asistente incorporado en el Netbeans, que permite crear POJO's a partir de una tabla (o varias) de la base de datos. Netbeans también permite crear un servicio web a partir de las clases creadas anteriormente.

- *Portlet*: AplicacionPortlet, también se ha utilizado asistente incorporado en el Netbeans, que permite crear una estructura de aplicación a partir de un POJO de una clase java. Una vez generada la estructura se ha modificado para incluir el framework Primefaces y sustituido el acceso a la base de datos mediante JPA, por llamadas al *web service*, para todas las operaciones de mantenimiento habituales.

Una vez compiladas se despliegan en cada uno de los servidores:

- *Web service* desplegado en Wildfly:

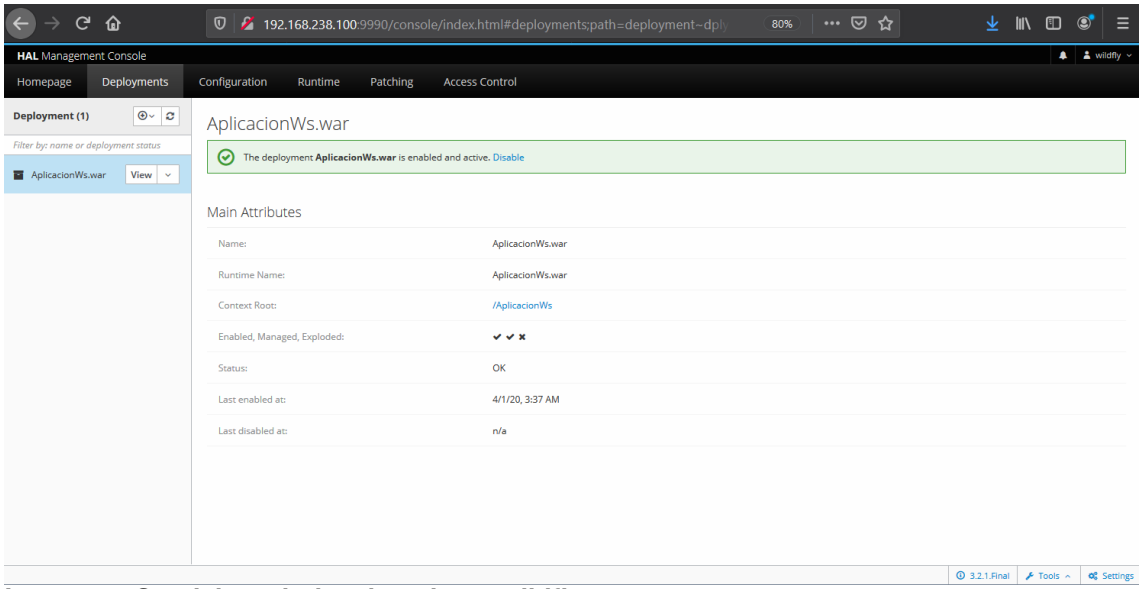


Imagen 7. Servicio web desplegado en wildfly

Para hacer una prueba de conexión se puede utilizar una petición por url:

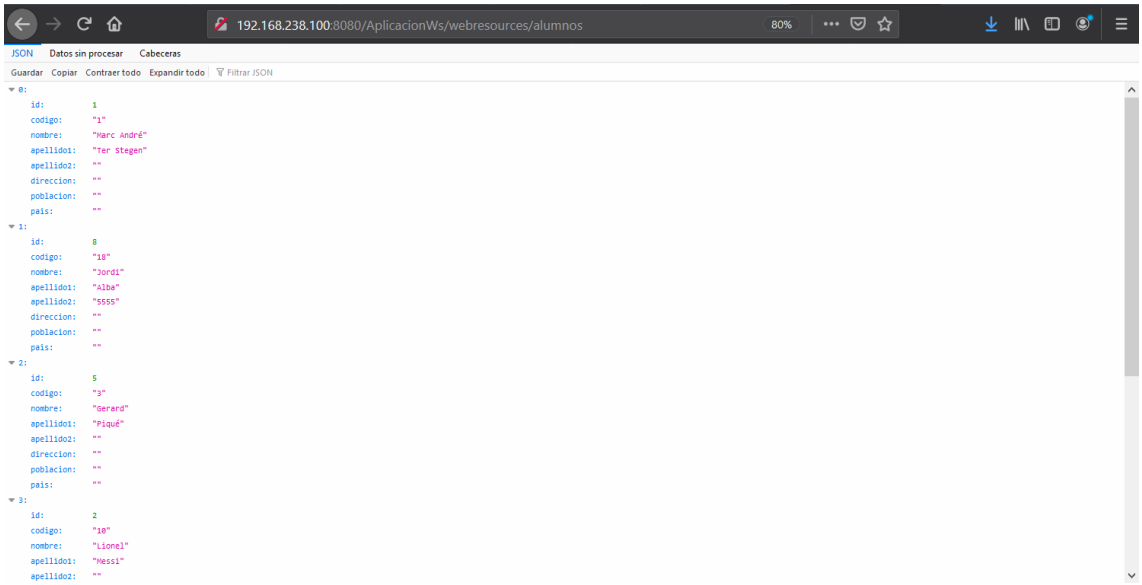


Imagen 8. Petición de datos al servicio web

- *Portlet* desplegado en Liferay:

Se ha situado el *portlet* en una página exclusivamente para trabajar con el *portlet*, para no tener interferencias de otro tipo de *portlets*.

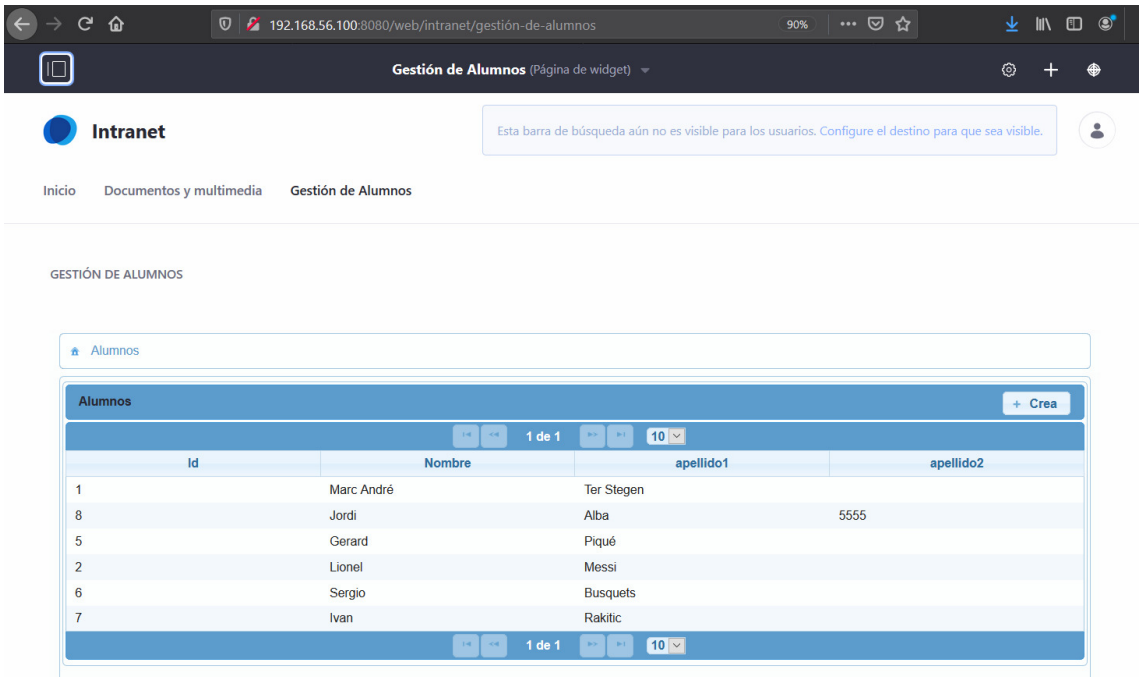


Imagen 9. Portlet funcionando en Liferay

Al seleccionar un elemento de la lista se carga el detalle de ese elemento:

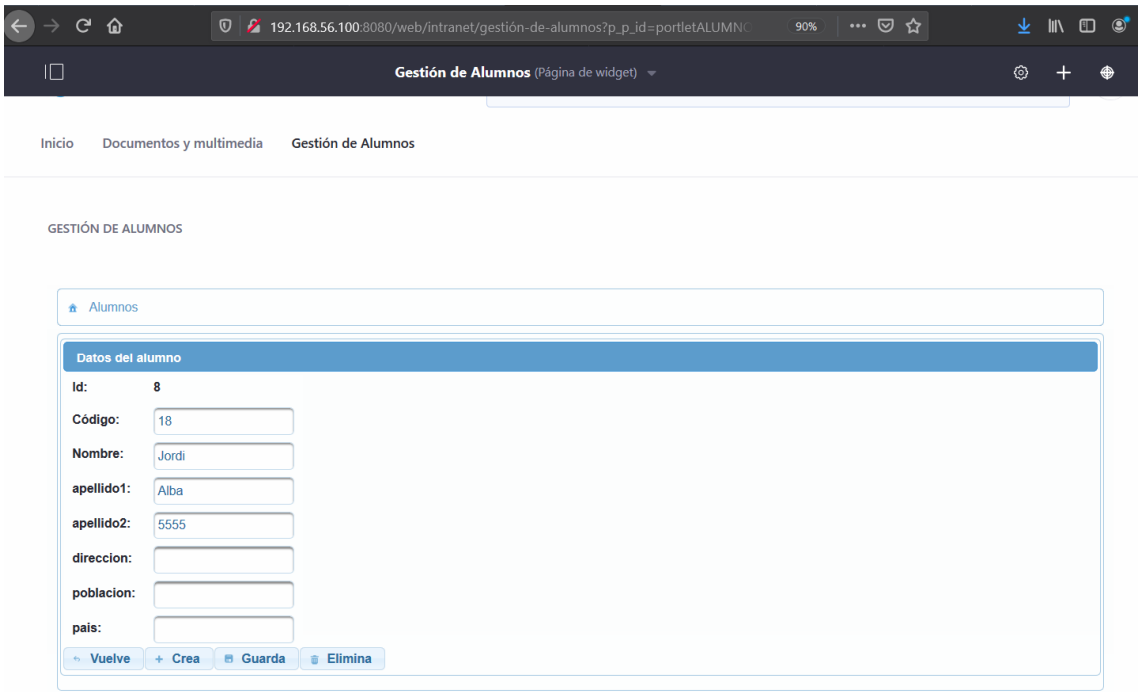


Imagen 10. Modificación de un registro

4. Análisis de datos sin protección

En este punto el objetivo es analizar el acceso a la información y como se procesan entre los diferentes los elementos del sistema de información, sin tomar ningún tipo de media. Es decir, se pretende analizar qué sucede si se utiliza una instalación básica de los componentes del sistema de información sin implementar ninguna medida de seguridad mínima.

Como parte de la preparación del entorno se han añadido alumnos a la base de datos, comprobando que tanto el *portlet* como el *web service* funcionan de forma correcta. Por tanto, para realizar las pruebas con el sistema se van a realizar diferentes interacciones con el *portlet*, solicitando un registro o todos los registros.

Las pruebas que se van a realizar serán las siguientes:

- Comprobar que el *portlet* sólo es accesible para usuarios atenticados y autorizados.

- Validar que sólo estos usuarios pueden realizar operaciones con el *portlet*.

- Comprobar que la información que circula entre los servidores se realiza de forma confidencial.

- Verificar que los servidores web sólo atienden a las partes que forman parte del sistema de información y no a otros elementos externos.

Por problemática del diseño del entorno con el software virtualización VirtualBox, no va a escuchar directamente en el tráfico generado en la red, sino que se tendrá que analizar el tráfico en cada una de las tarjetas de red virtuales.

4.1 Acceso al portlet

En esta primera prueba, el *portlet* se ha situado en una página privada a la que sólo pueden acceder usuarios autenticados, pero se podría dar el caso de que por accidente este también se despliegue en una página a la que pueden acceder otro tipo de usuarios, o incluso publica.

Pero tal como ocurre en muchos casos, este *portlet* se ha desarrollado en poco tiempo sin tener en consideración la confidencialidad de la información que esta tratando. Efectivamente puede acceder y modificar datos de alumnos, pero tal como se ha podido comprobar se puede acceder perfectamente a la información prácticamente sin restricciones.

Por tanto, se hace indispensable que se utilicen los grupos y roles facilitados por liferay para controlar la autorización al acceso a los datos.

4.2 Petición de datos al Liferay

En esta prueba vamos a analizar lo que sucede cuando, desde un navegador, se solicita el detalle de un alumno de la lista de alumnos. En el siguiente diagrama de secuencia se indica en rojo los puntos que se van analizar, el punto 1 que es la petición que llega al servidor web del usuario, y el punto 2 que es la respuesta procesada que se le envía al navegador del usuario.

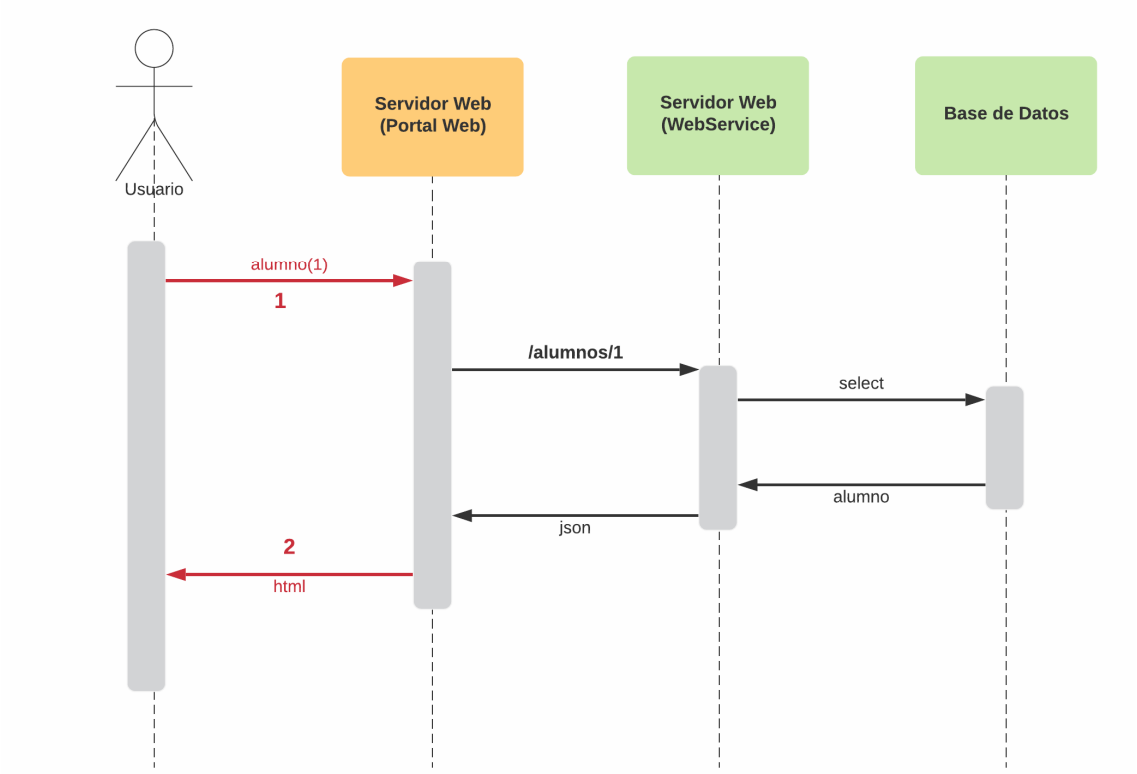


Imagen 11. Diagrama de secuencia. Petición de datos al portlet

Liferay procesa la petición, pero si capturamos la petición, que le llega al servidor web, como se muestra en la siguiente captura (punto 1 del diagrama de secuencia anterior) que se produce en la interface de red “VirtualBox Host-Only Network #2” perteneciente al servidor web donde está instalado Tomcat, y donde está desplegado el Liferay, podemos ver que la petición y sus parámetros son totalmente visibles, y por tanto, un usuario malintencionado, podría volver a repetir esa petición y obtener los datos:

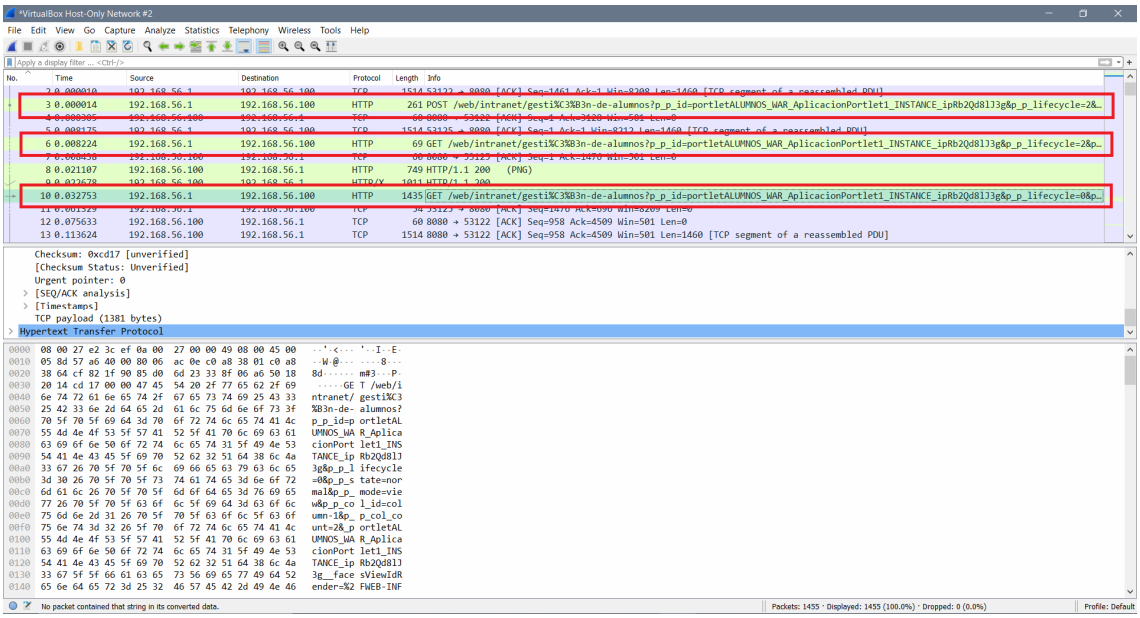


Imagen 12. Peticiones de datos a Liferay desplegado en el servidor web

A continuación se muestra la captura de la respuesta (punto 2 del diagrama de secuencia anterior), que se corresponde con el HTML que envía el *portlet* al navegador del usuario después de procesar la respuesta del *web service*, y que incluye los datos del alumno solicitado:

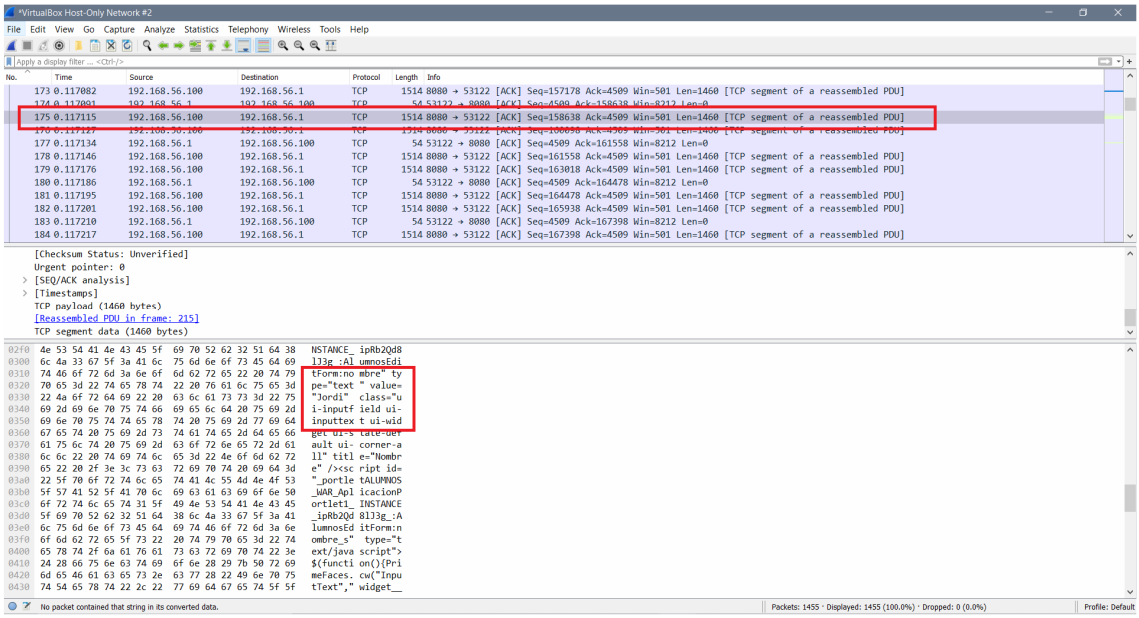


Imagen 13. Respuesta de Liferay al usuario

En esta prueba ha quedado en evidencia el problema de confidencialidad que se produce al no utilizar un protocolo de seguridad adecuado, SSL sobre HTTP, y que garantice la transmisión entre los extremos de la comunicación.

4.2 Petición de datos al web service

En esta prueba vamos a analizar lo que sucede cuando el *portlet* solicita la lista de alumnos al *web service*. Para ellos en el siguiente diagrama se ha indicado

en rojo los punto de la comunicación que se está analizando, el punto 3 la petición que le llega al *web service* del *portlet*, y el punto 4 que es el objeto json que contiene los datos de los alumnos.

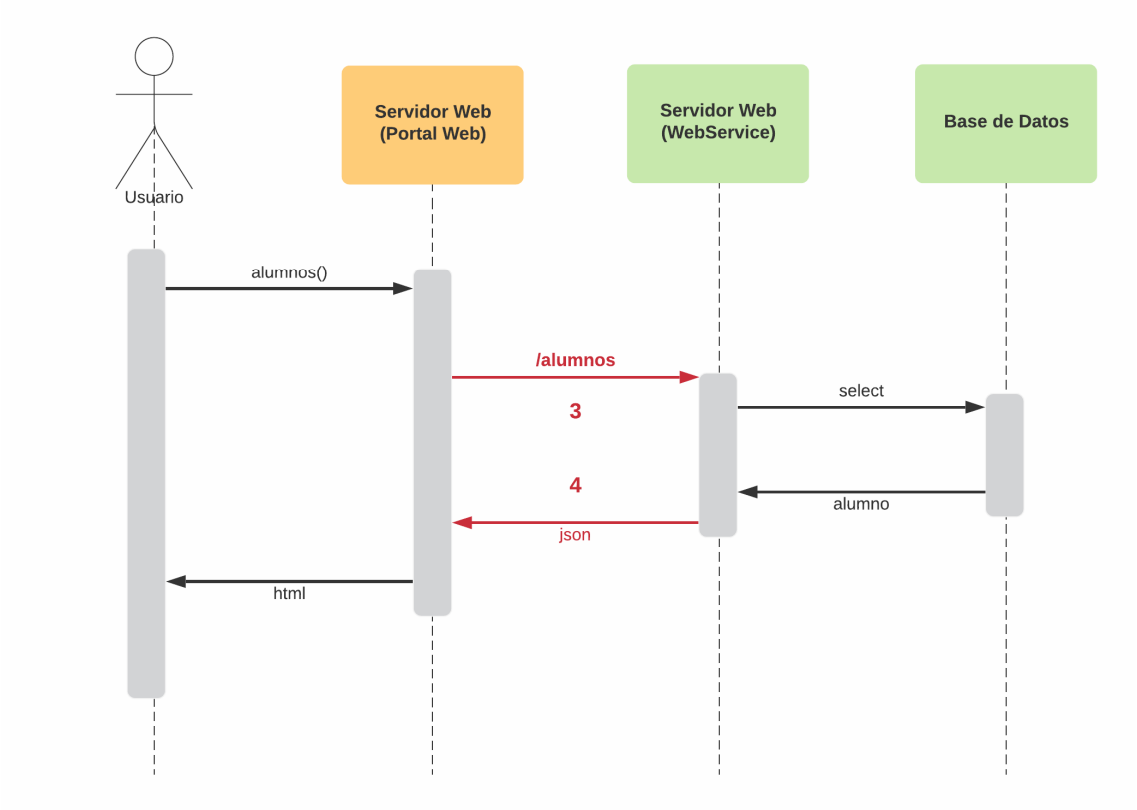


Imagen 14. Diagrama de secuencia. Petición de datos al web service

De la misma forma que en el punto anterior, a continuación se puede ver la captura que representa la petición (punto 3 del diagrama de secuencia anterior) que se produce en la interface de red “VirtualBox Host-Only Network #3” perteneciente al servidor web donde está instalado Wildfly solicitando los alumnos:

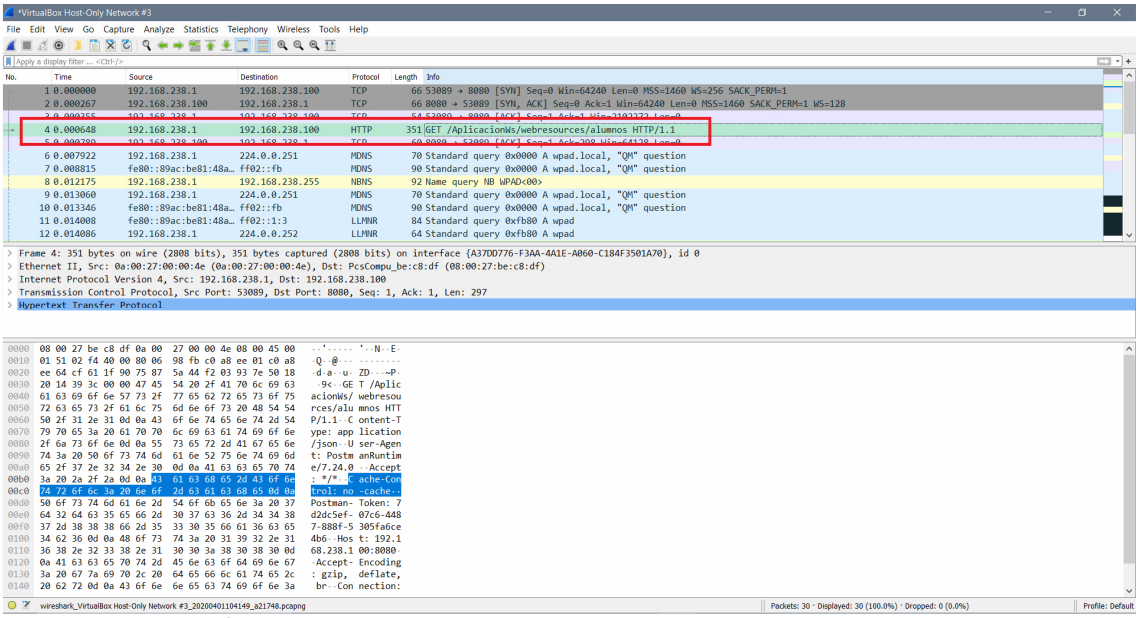


Imagen 15. Petición del portlet al servicio web

El *web service* procesa la petición (recupera los registros de la base de datos) y se devuelve un objeto json al *portlet* de Liferay:

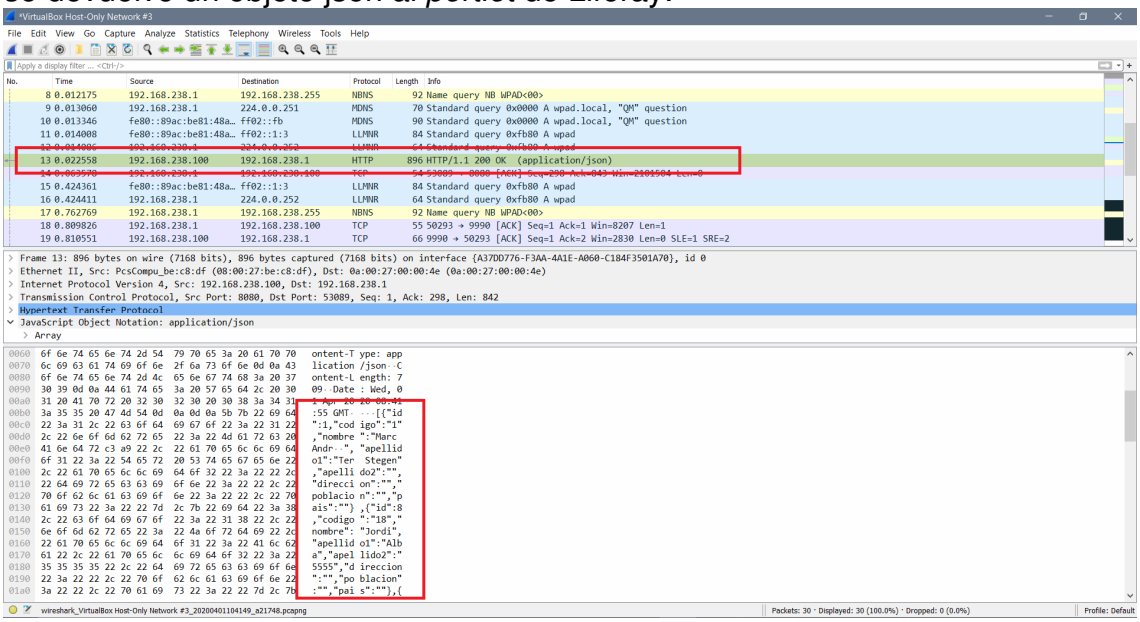


Imagen 16. Respuesta del servicio web al portlet

Como en el punto anterior, vuelve a quedar en evidencia el problema de confidencialidad en la transmisión entre los extremos de la comunicación. Pero además, un usuario que vea las peticiones que se van realizando al *web service* puede llegar a obtener una lista de todos los métodos disponibles en la API RESTful, y posteriormente lanzar los métodos a conveniencia para modificar los datos directamente sin pasar por el *portlet*.

4.3 Petición directa al servicio web

A raíz el problema detectado en el punto anterior, y tal como está montado el sistema, también es posible realizar una petición directamente al servicio web, sin pasar por el portal web. Este es uno de los grandes problemas de los *web* services, ya que en muchas ocasiones no se piensa en el grado de exposición de este tipo de API's.

Aunque con un navegador cualquiera podemos obtener datos GET de forma sencilla, para el resto de peticiones un usuario podría utilizar alguna aplicación, tipo postman, que permita incluir un json en el body de la petición y así manipular los datos del servidor a su antojo.

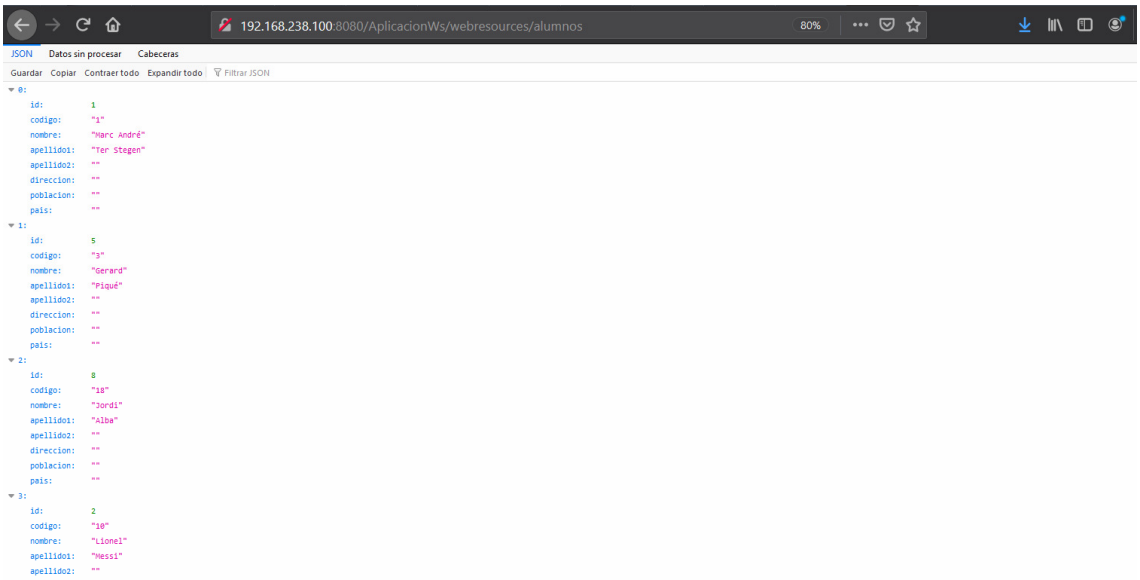


Imagen 17. Petición directa al servicio web

En la captura realizada a la interfaz de red del servidor del servicio web, se ve como la petición y respuesta se procesa de la misma forma como si se tratase del servidor web:

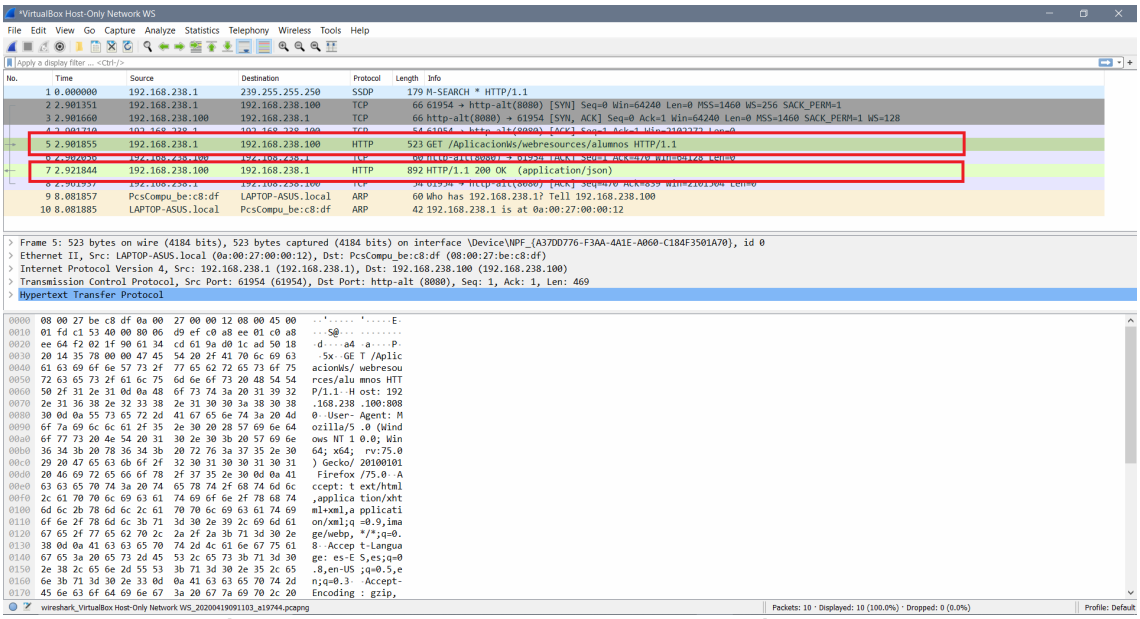


Imagen 18. Petición y respuesta del servidor web a la petición directa

Por tanto, después de ejecutar esta prueba queda en evidencia que este punto es uno de los más frágiles del sistema. No sólo se permite la ejecución directa sin autenticación, ni autorización, sino que el hecho de que un usuario externo pueda modificar la información supone un duro golpe a la integridad de la información que se está gestionando.

4.5 Conclusión de las primeras pruebas

Después de ejecutar las pruebas se puede comprobar que uno de los pilares básicos de la seguridad de la información, la confidencialidad, queda

totalmente expuesta, por un lado al situar el *portlet* en un lugar erróneo sin tener en cuenta los permisos que pueda tener la persona que está viendo los datos, y por otro lado se puede observar como la comunicación está totalmente expuesta, es decir, con un programa de escucha como Wireshak se puede obtener, sin mucha complicación, la información que está procesando Liferay, y como este, a través del *portlet*, realiza las consultas y actualizaciones a la tabla a través del servicio web.

Además de que el hecho de que sea posible acceder a los *endpoints* del *web service* directamente sin pasar por el portal web, o que se pueda simular una petición al *portlet*, hace que la integridad, otro de los pilares de la seguridad de la información quede totalmente en entredicho. Y como consecuencia, al poder manipular la información se podría alterar de tal forma que afecte también a la disponibilidad.

De la misma forma, tal como está montado el sistema no hay ningún mecanismo que pueda garantizar la trazabilidad y no repudio de las operaciones que se llevan a cabo.

5. Medidas correctivas

Una de las medidas más lógicas para proteger los extremos de la conexión, es utilizar el protocolo TLS sobre HTTP, de esta forma se garantizar la confidencialidad, ya que solamente podrá acceder a la información las personas que participan en la comunicación al estar cifrados los mensajes, la integridad, ya que se garantiza que la información no se modifica durante la transmisión, y la autenticación al intercambiar sus certificados los participantes en la comunicación.

Pero no se puede delegar toda la seguridad de un sistema de información en un único mecanismo que pueda quedar comprometido en algún momento determinado, hay que buscar mecanismos que a nivel de aplicación permitan garantizar un mínimo de características de seguridad. Y además hay que realizarlo en los dos elementos principales del sistema planteado, en el *portlet* y en el *web service*.

5.1 Aplicando seguridad al portlet

Al integrar el *portlet* dentro de un portal web como Liferay, ya tenemos medio camino hecho, sólo falta terminarlo.

Liferay proporciona un estructura de seguridad basada en usuarios, grupos y roles, donde habitualmente, los usuarios son añadidos a grupos de usuarios, y a los grupos de usuarios de les van añadiendo roles que permiten acceder a determinadas paginas privadas.

Por tanto, sólo hay que implementar los métodos necesarios para recuperar la información del usuario y grupo al que pertenece para permitir o no el acceso.

En el siguiente diagrama de secuencia vemos en que punto se integraría el mecanismo. Si un usuario accede a una página en la que se ha situado el *portlet*, el *portlet* accede al context de liferay para obtener el usuario y grupos a los que pertenece. A continuación si el grupo es el grupo autorizado para acceder a los datos, GRUPO_INTRANET, podrá utilizar el portlet, y si no pertenece se mostrará un mensaje indicando que no tiene permiso.

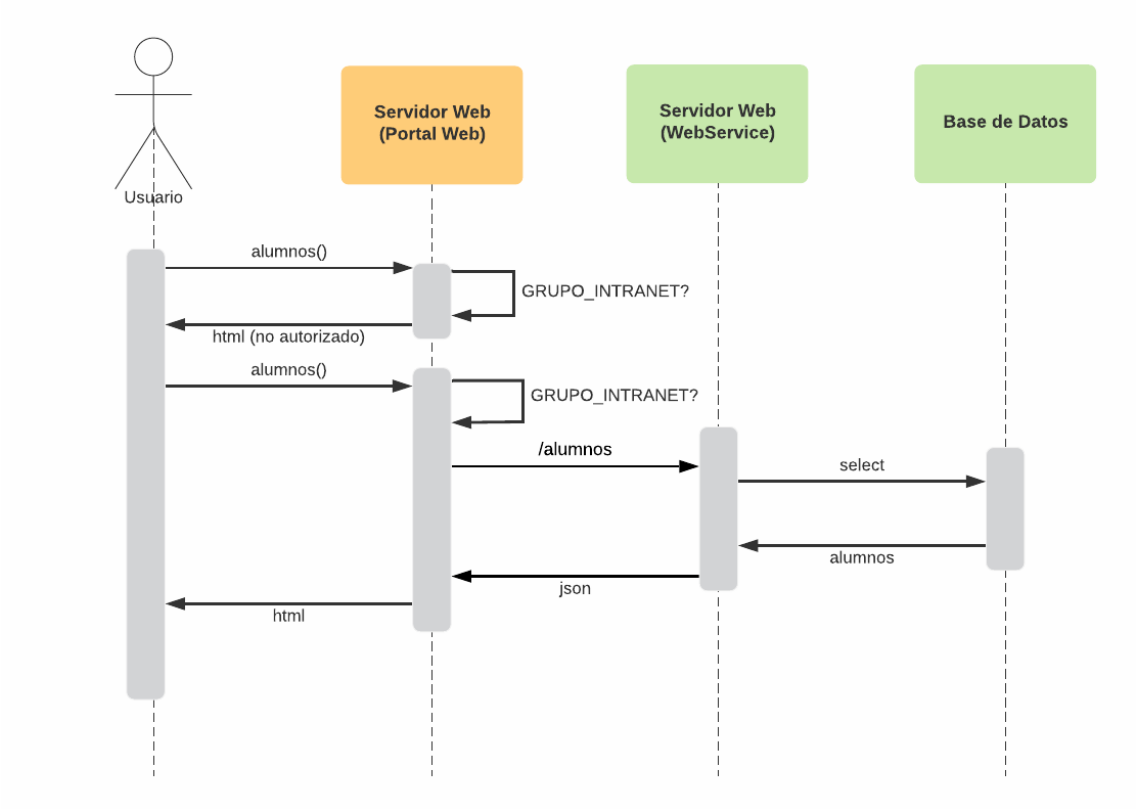


Imagen 19. Diagrama de secuencia. Mecanismo de autorización en el portlet

A continuación se muestra la información que salía antes y después de implementar la medida para un usuario sin autenticar:

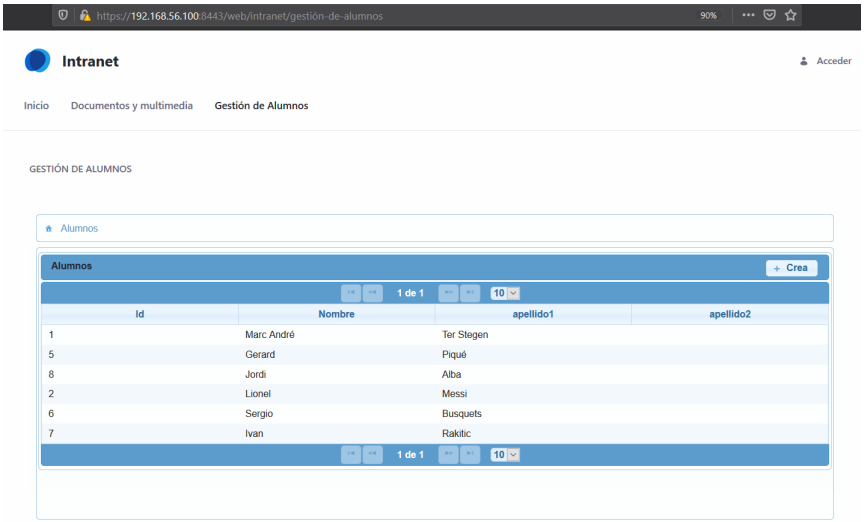


Imagen 20. Acceso antes de la implementación del mecanismo

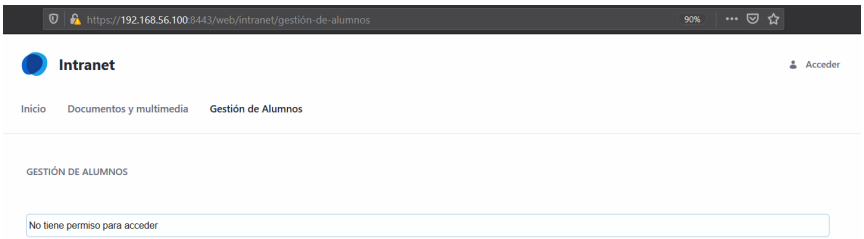


Imagen 21. Acceso después de la implementación del mecanismo

5.2 Securización de los servicios web RESTful

El servicio web es uno de los elementos más comprometidos en el esquema propuesto, por tanto a continuación se describirán los mecanismos que se pueden utilizar para mejorar su seguridad.

Dentro de los servicios web REST existe una gran variedad de métodos de autenticación y de formas de garantizar la confidencialidad de la información. Centrándonos en la autenticación, existen tres métodos principales:

- Básica
- Basada en cookies
- Basada en tokens
- OAuth 2.0

Hay que tener en cuenta que estamos tratando con servicios REST, lo que significa que no se mantiene el estado de la conexión en el servidor, y por tanto, en cada petición es necesario comunicar las credenciales para verificar la autorización. Así cada uno de estos métodos tiene sus pros y contras:

1. Autenticación básica: Utiliza un usuario y contraseña para la autenticación que se comunica dentro del encabezado HTTP denominado (HTTP *Authorization*). El problema de este tipo de autorización es que aunque el usuario y la contraseña estén codificadas, cualquier transmisión de datos se puede interceptar mediante una ataque *Man-In-The-Middle*, y por tanto, se hace obligatorio utilizar una conexión TLS/HTTPS.

2. Autenticación basada en *cookies*: Ha sido muy utilizado durante mucho tiempo, pero en este tipo de comunicaciones se guarda el estado (stateful) de la comunicación, ya que, al identificarse, el servidor registra los datos y genera un identificador de sesión que se almacena en la cookie del cliente, y que se utilizará mientras dure la comunicación.

3. Autenticación basada en *tokens*: En este tipo de autenticación sin estado (stateless), también se utiliza un usuario y contraseña para realizar la autenticación, pero sólo en la primera petición, ya que el servidor generará un token, basado en las credenciales, que se puede guardar en una base de datos y que retornará al cliente para que se lo envíe codificado en cada petición nueva que haga. Normalmente, estos tokens están codificados con la fecha y hora, para que en caso de un ataque *Man-In-The-Middle*, no se puedan utilizar pasado un tiempo establecido. Así mismo, el token se puede configurar con una caducidad definida, de forma que pasado un tiempo los clientes se tengan que volver a identificar de nuevo.

4. Autorización abierta OAuth 2.0: Es un método que permite utilizar proveedores de servicios externos que, mediante una API, facilitan la autenticación de los usuarios ya que el usuario utiliza las credenciales de esos proveedores de servicios para autenticarse al servicio REST. El proveedor de servicios generará un *token* que se utilizará en toda la comunicación.

5.2.1 Autorización basada en tokens: JWT

Para la securización del servicio web de este trabajo se ha seleccionado la autorización basada en *tokens*, ya que actualmente es la que está siendo la más utilizada. Al utilizar JSONs para la comunicación entre las partes, se va comprobar que beneficios se consiguen al utilizar el concepto de JSON Web Tokens (JWT).

Para entender un poco más cómo funciona el sistema JWT, hay entender los pasos que hay que realizar para habilitar la comunicación, estos son:

1. El usuario se autentica enviando un usuario y contraseña.
2. El servicio web valida el usuario y genera un token con una caducidad determinada.
3. Se envía el token al usuario para que lo guarde.
4. El usuario a partir de este momento enviará el token, en el header de la petición, para todas las peticiones que realice.
5. Para cada petición el servicio web, validará el token
6. Si el token es válido se procesará la petición, en caso contrario se volverá a requerir la autenticación del punto 1.

En el siguiente diagrama muestra la secuencia completa del proceso JWT incorporado al modelo de entorno que estamos construyendo:

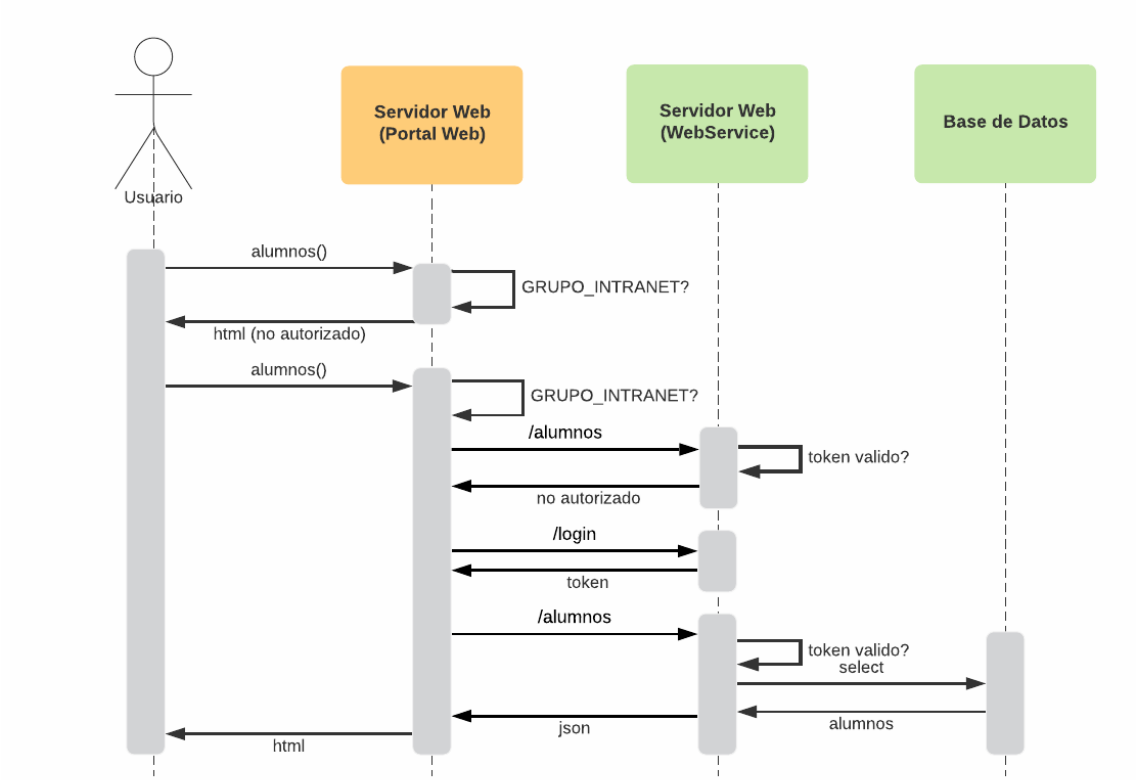


Imagen 22. Diagrama de secuencia. Mecanismo de autorización en el servlet

El JWT es una cadena que está formada por tres partes denominadas el ***Header*****,** el ***Payload*** y la ***Signature*****,** separadas por un punto y codificadas en base64, un ejemplo seria el siguiente:

eyJhbGciOiJIUzUxMiJ9.eyJzdWIiOiJwb3J0YWwiLCJpc3MiOiJodHRwOi8vMTkyLjE2OC4yMzguMTAwOjgwODAvQXBsaWNhY2lvbldzL3dlYnJlc291cmNlcy91c2Vycy9sb2dpbiIsImlhdCI6MTU4ODM0MjM4MCwiZXhwIjoxNTg4MzQ1OTgwfQ.AkMcpyoQV8-C8FUQOHAQxLGvwTBTNZNW0z9_mygjbCqQnOw13nouhRAsVemtyxP_D58WYt7ZhufGmuaAcKrYDg

El *Header* del *token* está formada por dos partes: el tipo y la codificación utilizada.

El *Payload* está compuesto por los llamados JWT *Claims* que definen los atributos que definen el *token*. Estos son, *sub* que dentifica el sujeto del *token* (como puede ser el usuario), *iat*: que identifica la fecha de creación del *token*, y *exp*: identifica a la fecha de expiración del token.

La *Signature*, está formada por los anteriores componentes, *Header* y *Payload*, codificados en Base64 y encriptada con una clave secreta, y que se puede utilizar para comprobar que todo está bien.

Por tanto, para que este mecanismo funcione será necesario implementar una parte en el servicio web y otra en el *portlet*. En el servicio web, es necesario crear un filtro que se incorporará a todos los *endpoints* que se quieran proteger, y un *endpoint* exclusivo para realizar la autenticación del usuario. En el *portlet*, se ha de modificar el flujo de las peticiones para incluir el *token* en todas las peticiones que se realicen, y para controlar que cuando se requiera el *token,* o este caduque, se realice un proceso de autenticación de forma que el usuario no perciba que se está realizando.

El punto más delicado de este sistema es la comunicación de las credenciales al servicio web para obtener el *token*. Para simplificar el ejemplo se ha utilizado un usuario específico, portal, que será el único que se puede conectar al servidor web, en condiciones normales, lo lógico es implementar un sistema de autenticación recuperando datos del LDAP de la organización.

5.2.2 Validando el mecanismo JWT

Una vez implementado el mecanismo JWT en el servicio web y en el *portlet* se ha comprobado su funcionamiento de la siguiente forma:

Petición no autorizada: Se realiza una petición al servicio web, sin pasar por el portal, utilizando la aplicación Postman que permite realizar consultas a endpoints de forma sencilla.

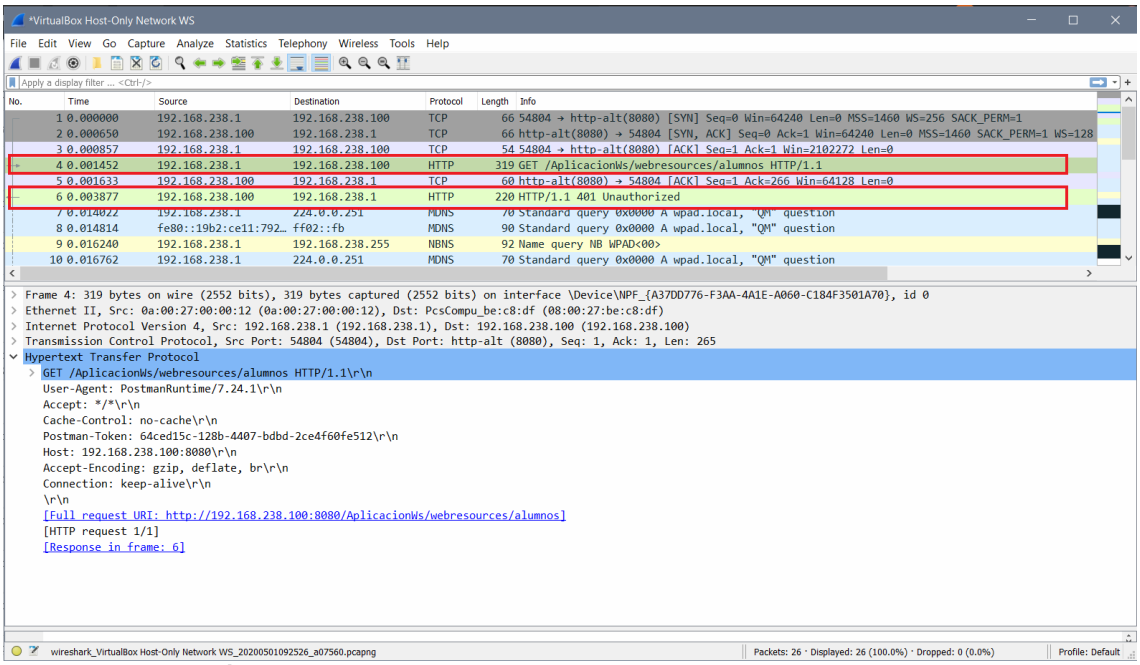


Imagen 23. Petición y respuesta al endpoint sin token

En esta captura al interface de red de la máquina donde está instalado el servicio web, se ve como llega la petición, y como esta es rechazada como no autorizada (401).

Petición autorizada: Antes de realizar una petición de lógica de negocio, se realizará una petición de autorización.

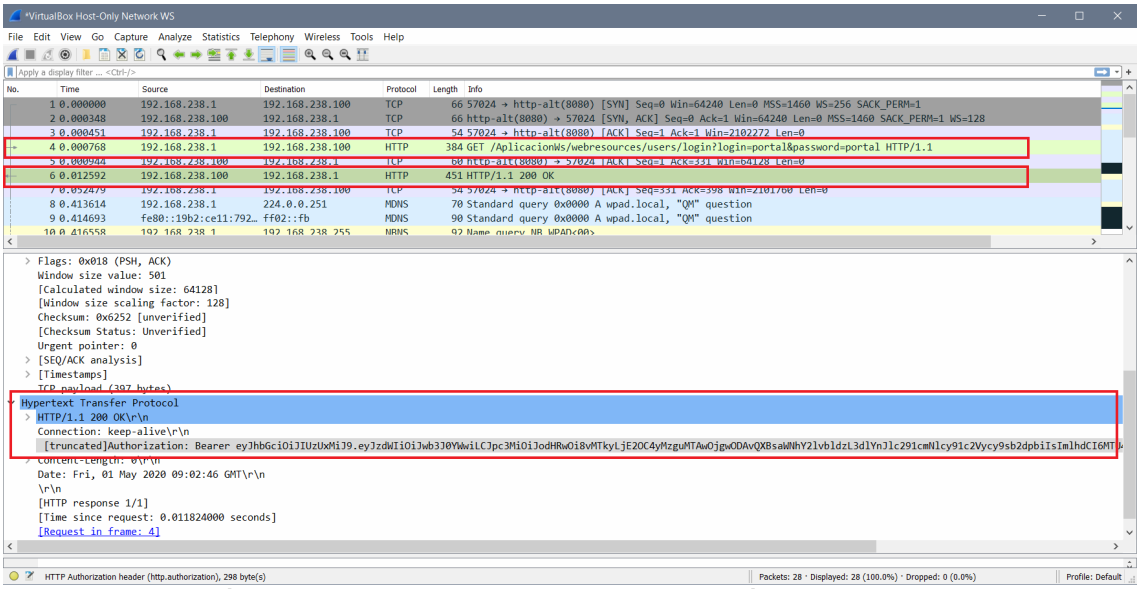


Imagen 24. Petición y respuesta al endpoint de autenticación

En esta captura al *endpoint* de autenticación se ve el usuario y su contraseña se pasa como parámetro en claro, dejando en evidencia uno de los puntos débiles de este mecanismo. La respuesta del servidor incluye el *token* en la respuesta, el *token*, que empieza por *Bearer* (portador), es una cadena codificada con información del usuario y con validez temporal.

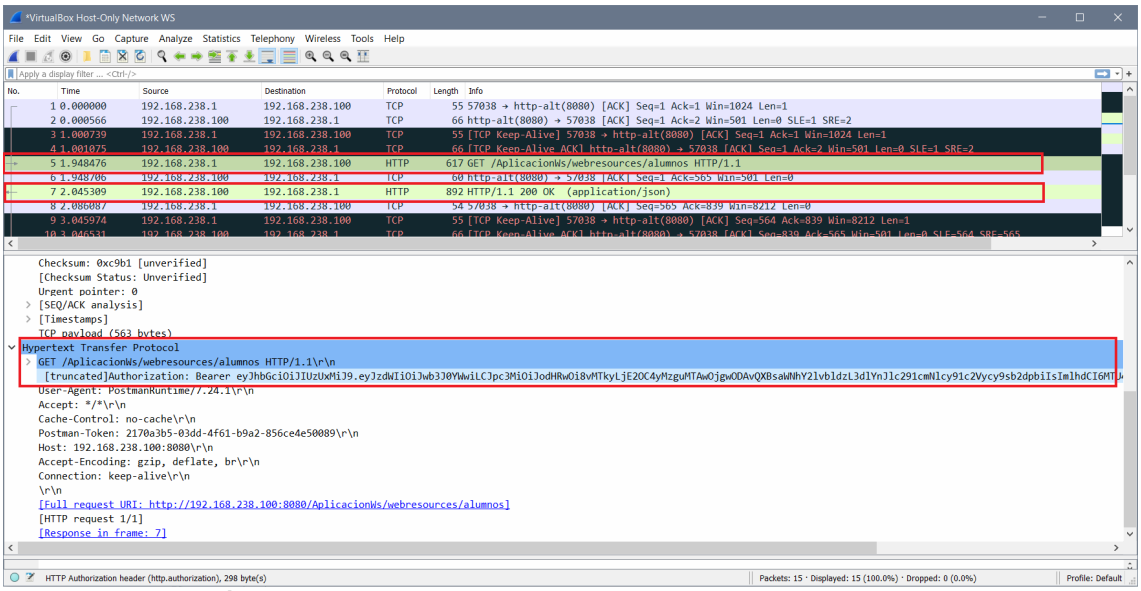


Imagen 25. Petición y respuesta al endpoint con token

En esta captura se puede comprobar como la misma petición, que no se autorizo previamente, es autorizada al incluir el *token* en el *heather* de la petición.

5.3 Implementación del protocolo SSL/TTL en el servicio Web

En este punto lo más lógico es habilitar la conexión SSL/TTL para terminar de asegurar la confidencialidad de la autenticación. Donde una habilitada en el servidor se procede a realizar la misma petición se solicitud del *token* para poder realizar las peticiones de negocio.

En la siguiente captura se ve como se realiza la petición de autenticación, pasando el usuario y contraseña (si las que se ven en claro)

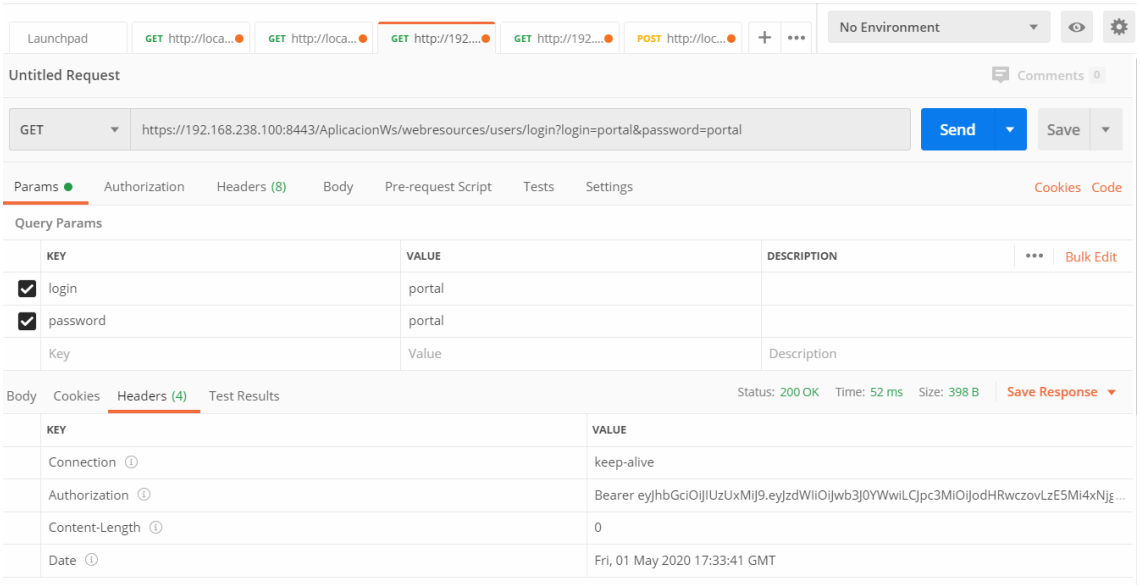


Imagen 26. Petición al endpoint de autenticación a través de https

Y en Wireshark ya no se puede capturar el tráfico http, únicamente se puede ver como se establece la conexión segura, se envía la petición cifrada y se retorna cifrada igualmente.

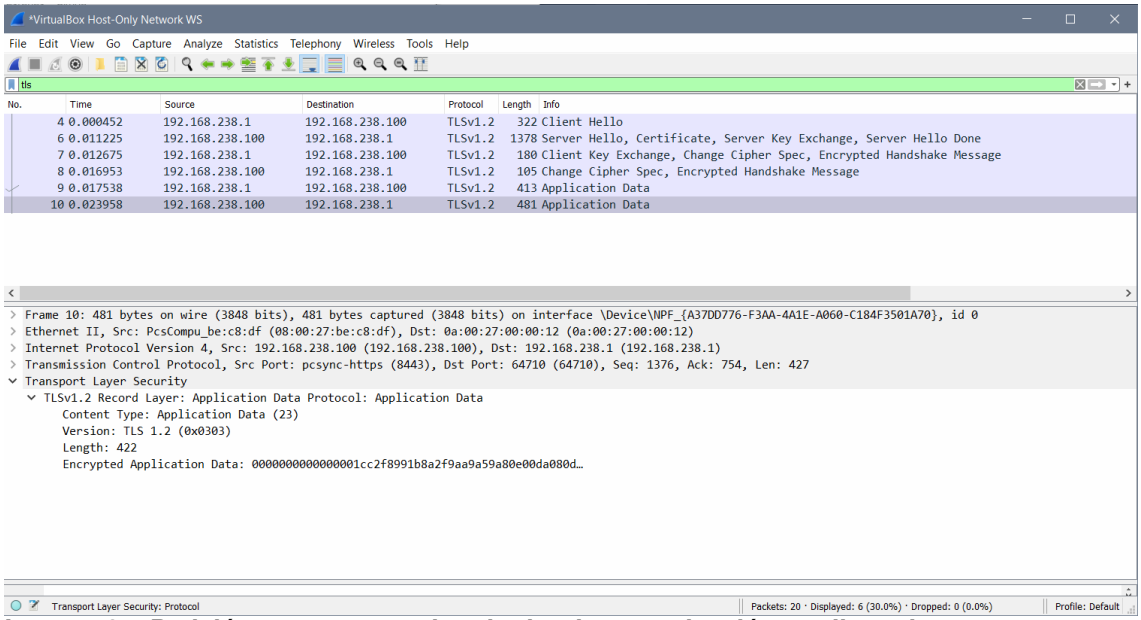


Imagen 27. Petición y respuesta al endpoint de autenticación mediante https

De la misma forma si se hace una petición a un *endpoint* de lógica de negocio con el token retornado en la operación anterior.

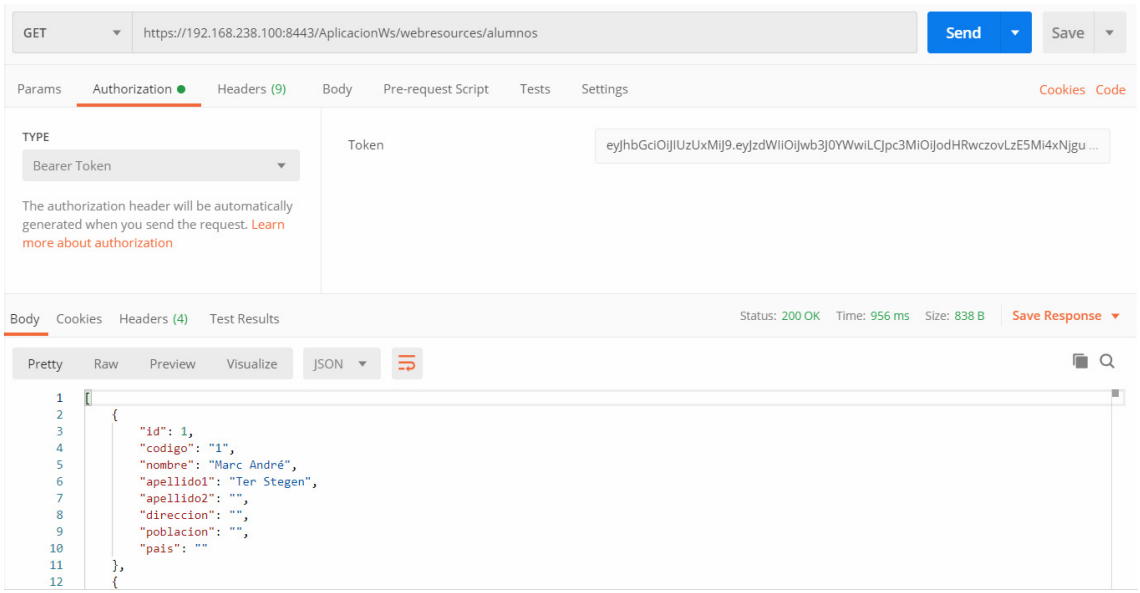


Imagen 28. Petición al endpoint con token y mediante https

El resultado es el mismo la comunicación está cifrada y la confidencialidad asegurada, como se puede ver en la siguiente captura.

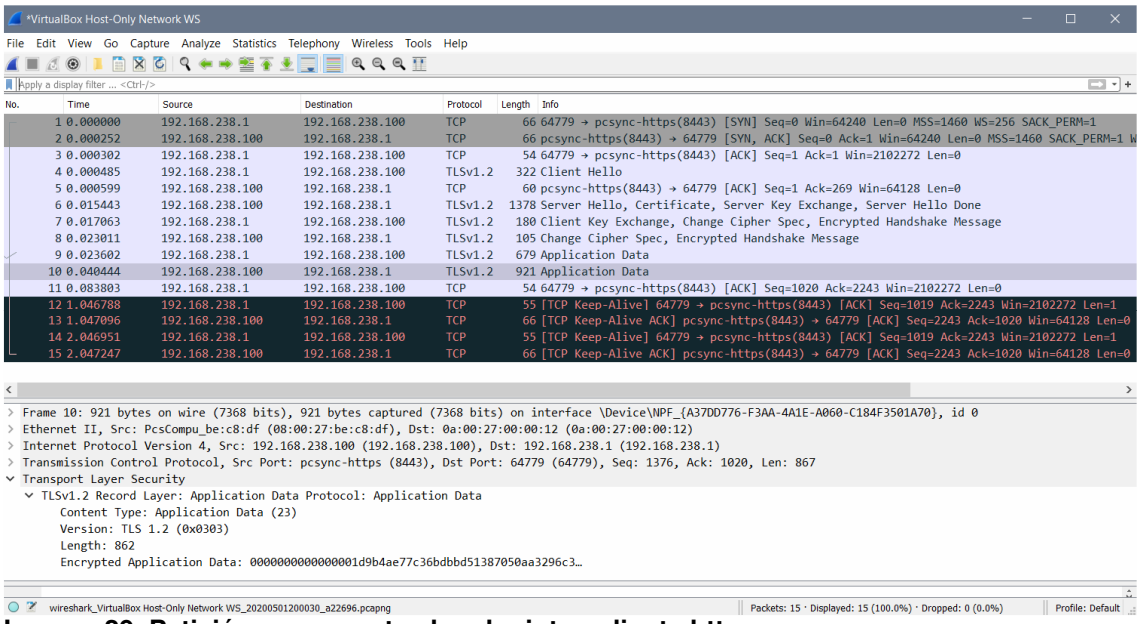


Imagen 29. Petición y respuesta al endpoint mediante https.

5.4 Implementación del protocolo SSL/TTL en el portal Web

Para solucionar, otro de los problemas de seguridad, detectado en el punto 4.1, donde se podían capturar en claro los datos que eran enviados y recibidos al navegador web del usuario, es necesario con la implementación del protocolo SST/TTL en el portal web.

Para ello se ha procedido a generar un certificado autofirmado para el servidor tomcat donde está instalado Liferay. Así mismo ha sido necesario importar el certificado autofirmado utilizado para cifrar el servidor de aplicaciones wildfly, creado en el punto anterior, dentro del almacén de certificados de la JVM del servidor donde está instalado tomcat, para que el *portlet* funcionase correctamente con el mecanismo JWT. Este problema se producía al no poder validarse el certificado digital del servidor Wildfly (de la misma forma que ocurre cuando nos conectamos a una página segura pero que tiene un certificado no válido y el navegador muestra una advertencia que es necesario aceptar para poder continuar).

Por tanto, con el sistema ya operativo se ha procedido a realizar la misa prueba realizada en el punto 4.1.

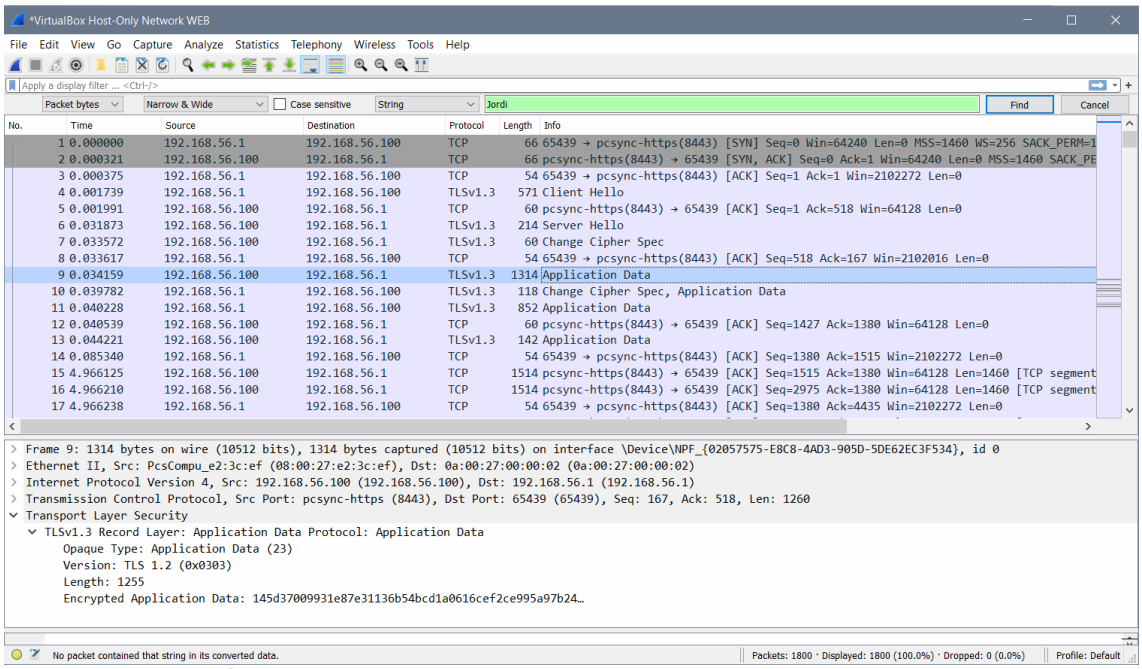


Imagen 30. Petición y respuesta al portlet mediante https

Y como se puede comprobar, ni los valores incluidos en la petición del usuario, ni los valores devueltos al usuario por el servidor, son visibles.

6. Conclusiones

Mi objetivo principal a la hora de iniciar este trabajo era la de complementar el desarrollo de *portlets* y *web services*, que actualmente estamos desarrollando en la empresa donde trabajo, con los principios y conceptos de seguridad adquiridos en la realización del Master, para conseguir una guía de los pasos a seguir en desarrollos futuros.

Es por este motivo que este trabajo es muy práctico, intentado ir directamente a solucionar los problemas más evidentes, que intentando buscar problemas que quizá quedan fuera del alcance de este proyecto, como puedan ser vulnerabilidades específicas como la inyección de código sql o el cross site-scripting.

Con la realización de las pruebas ha quedado en evidencia que a la hora de desarrollar una aplicación es necesario no sólo tener en cuenta la implementación de los requisitos funcionales detectados, sino que también se debe ir más allá, con la incorporación de mecanismos de seguridad que garanticen una solución integral y no generen otro tipo de problemas, como es el de la seguridad.

La parte más costosa para la realización de las pruebas ha sido la configuración del entorno, ya que no es mi trabajo habitual, y lo que a priori debería haber sido lo más sencillo se convertido en un problema. Ya que al utilizar un portátil con máquinas virtualizadas, el programa de captura de paquetes Wireshak no escuchaba el tráfico generado internamente por las máquinas virtuales, sino que sólo a la interfaz wifi. Finalmente, creando interfaces de red virtuales y redes internas, se ha podido alcanzar un objetivo doble, hacer que el experimento funcione y quitar el polvo mis conocimientos de redes.

Por otra parte, la implementación del los mecanismos no es todo lo funcional que pretendía, ya que faltaría integrar el mecanismo JWT con un sistema LDAP de la organización para la consulta de los usuarios que pueden tener acceso al *web service*, y no introducir un usuario y contraseña fijo como se ha hecho para realizar las pruebas.

Tengo muy claro que todo el sistema es mejorable. Pero aún así, creo que lo analizado, desarrollado y validado, es un buen comienzo para completar un sistema más a seguro de lo que tenemos actualmente en nuestra empresa.

Referencias

[1] Liferay, ¿Qué es un CMS? [En línea]. [Consultado: marzo 2020] <https://www.liferay.com/es/resources/l/content-management-system>

[2] Linuxize, How to Install WildFly (JBoss) on Debian 9 [En línea]. [Consultado: marzo 2020] <https://linuxize.com/post/how-to-install-wildfly-on-debian-9/>

[3] INCIBE, Gestión de sesiones web: ataques y medidas de seguridad [En línea] [Consultado: abril 2020] <https://www.incibe.es/extfrontinteco/img/File/intecocert/Formacion/EstudiosInformes/gestion_sesiones_web_seguridad.pdf>

[4] NTUN, Specification-based security analysis of REST APIs [En línea]. [Consultado: abril 2020] <https://ntnuopen.ntnu.no/ntnu-xmlui/bitstream/handle/11250/2560780/18062_FULLTEXT.pdf?sequence=1&isAllowed=y>

[5] Configuración de un certificado local SSL con WildFly 10 [En línea]. [Consultado: abril 2020] <https://videlcloud.wordpress.com/2017/06/26/configuracion-de-un-certificado-local-ssl-con-wildfly-10/>

[6] Oreilly, How a RESTful API server reacts to requests [En línea]. [Consultado: abril 2020] <https://www.oreilly.com/content/how-a-restful-api-server-reacts-to-requests/>

[7] ITDO, ¿Cuál es el mejor método de autentificación en un API REST? [En línea]. [Consultado: abril 2020] <https://www.itdo.com/blog/cual-es-el-mejor-metodo-de-autentificacion-en-un-api-rest/>

[8] Securizar un API REST utilizando JSON Web Tokens [En línea]. [Consultado: abril 2020] <https://www.adictosaltrabajo.com/2017/09/25/securizar-un-api-rest-utilizando-json-web-tokens/>

[9] Autenticación con JSON Web Tokens [En línea]. [Consultado: abril 2020] <https://www.oscarblancarteblog.com/2017/06/08/autenticacion-con-json-web-tokens/>

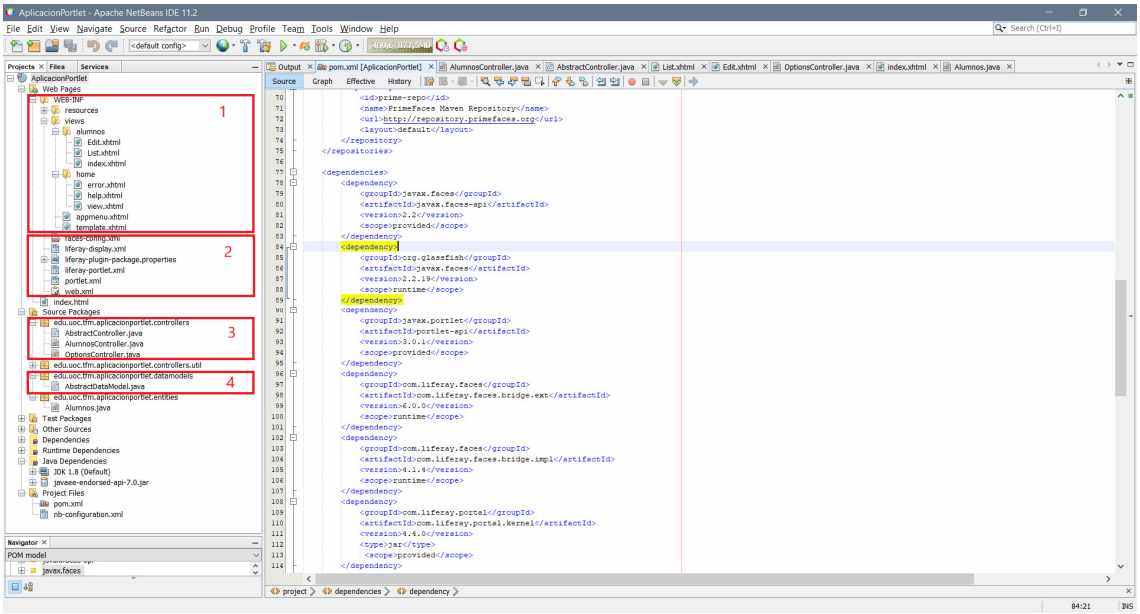
[10] Breaking Token JWT or JWT Exponed [En línea]. [Consultado: abril 2020] https://www.hackplayers.com/2017/07/breaking-token-jwt-o-jwt-exposed.html>

[11] Securing JAX-RS Endpoints with JWT [En línea]. [Consultado: abril 2020] <https://antoniogoncalves.org/2016/10/03/securing-jax-rs-endpoints-with-jwt/>

[12] Role Based Authentication on Restful endpoints in java [En línea]. [Consultado: abril 2020] <https://stackoverflow.com/questions/55559309/role-based-authentication-on-restful-endpoints-in-java>

Anexo 1. Aspectos destacados del portlet

El *portlet* se ha desarrollado con el IDE Netbeans 11.2 como un proyecto maven:



En esta captura se puede apreciar la estructura del proyecto Aplicación que se corresponde con el *portlet* utilizado para realizar las pruebas. En esta estructura se puede apreciar los siguientes puntos:

1. Contiene los archivos .xhtml, que representan la vista del paradigma MVC, que se procesarán para generar el código html que se enviará al navegador.

En este caso podemos apreciar el código perteneciente a List.xhtml, dónde se pueden ver las llamadas a los métodos de un controlador determinado, en este caso alumnosController:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<ui:composition xmlns="http://www.w3.org/1999/xhtml"
                xmlns:ui="http://java.sun.com/jsf/facelets"
                xmlns:h="http://java.sun.com/jsf/html"
                xmlns:f="http://java.sun.com/jsf/core"
                xmlns:p="http://primefaces.org/ui">

    <h:form id="ListAlumnosForm">

        <p:panel header="#{i18n['ListAlumnosTitle']}" >

            <f:facet name="header">
                <h:outputLabel value="#{i18n['ListAlumnosTitle']}"/>
                <p:outputPanel style="float:right;">
                    <p:commandButton id="createbtn" icon="ui-icon-plus"
                                     value="#{i18n['Create']}"
                                     action="#{alumnosController.prepareListCreate}" ajax="false"/>
                </p:outputPanel>
            </f:facet>

            <p:dataTable id="datalist" widgetVar="wvdatalist" var="item"
                value="#{alumnosController.itemsDataModel}"
                selection="#{alumnosController.selected}" selectionMode="single"
                rowKey="#{item.id}" paginator="true" rows="10" rowsPerPageTemplate="10,20,30"
```

```
                currentPageReportTemplate="{currentPage} de {totalPages}"
                paginatorTemplate="{FirstPageLink} {PreviousPageLink} {CurrentPageReport}
{NextPageLink} {LastPageLink} {RowsPerPageDropdown}"
                emptyMessage="#{i18n['NoRows']}">

                <p:ajax event="rowSelect" listener="#{alumnosController.onRowSelectClassic}" />

                <p:column>
                    <f:facet name="header">
                        <h:outputText value="#{i18n['ListAlumnosTitle_id']}"/>
                    </f:facet>
                    <h:outputText value="#{item.id}"/>
                </p:column>

                <p:column>
                    <f:facet name="header">
                        <h:outputText value="#{i18n['ListAlumnosTitle_nombre']}"/>
                    </f:facet>
                    <h:outputText value="#{item.nombre}"/>
                </p:column>

                <p:column>
                    <f:facet name="header">
                        <h:outputText value="#{i18n['ListAlumnosTitle_apellido1']}"/>
                    </f:facet>
                    <h:outputText value="#{item.apellido1}"/>
                </p:column>

                <p:column>
                    <f:facet name="header">
                        <h:outputText value="#{i18n['ListAlumnosTitle_apellido2']}"/>
                    </f:facet>
                    <h:outputText value="#{item.apellido2}"/>
                </p:column>

            </p:dataTable>

        </p:panel>

    </h:form>

</ui:composition>
```

2. Son los archivos de configuración del *portlet* que especifican como se debe desplegar (nombre, página principal, etc).

Entre estos podemos destacar el web.xml, especialmente dónde se define el servlet principal de la aplicación, integrado en las clases javax de JSF:

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns="http://java.sun.com/xml/ns/javaee" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
version="3.0">

    <filter>
        <filter-name>PrimeFaces FileUpload Filter</filter-name>
        <filter-class>org.primefaces.webapp.filter.FileUploadFilter</filter-class>
    </filter>
    <filter-mapping>
        <filter-name>PrimeFaces FileUpload Filter</filter-name>
        <servlet-name>Faces Servlet</servlet-name>
    </filter-mapping>

    <context-param>
        <param-name>javax.faces.PROJECT_STAGE</param-name>
        <param-value>Development</param-value>
    </context-param>

    <context-param>
        <param-name>javax.faces.WEBAPP_RESOURCES_DIRECTORY</param-name>
        <param-value>/WEB-INF/resources</param-value>
    </context-param>

    <servlet>
        <servlet-name>Faces Servlet</servlet-name>
        <servlet-class>javax.faces.webapp.FacesServlet</servlet-class>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <security-constraint>
        <display-name>Prevent direct access to Facelet XHTML</display-name>
        <web-resource-collection>
            <web-resource-name>Facelet XHTML</web-resource-name>
            <url-pattern>*.xhtml</url-pattern>
        </web-resource-collection>
        <auth-constraint/>
    </security-constraint>
    <context-param>
        <param-name>primefaces.THEME</param-name>
```

```
        <param-value>redmond</param-value>
    </context-param>
</web-app>
```

Y el portlet.xml que define el tipo de *portlet*, su nombre o los puntos de dentrada en la aplicación:

```
<?xml version="1.0" encoding="UTF-8"?>

<portlet-app xmlns="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd
http://java.sun.com/xml/ns/portlet/portlet-app_2_0.xsd"
    version="2.0">
    <portlet>
        <portlet-name>portletALUMNOS</portlet-name>
        <display-name>portletALUMNOS</display-name>
        <portlet-class>javax.portlet.faces.GenericFacesPortlet</portlet-class>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.view</name>
            <value>/WEB-INF/views/alumnos/index.xhtml</value>
        </init-param>
        <init-param>
            <name>javax.portlet.faces.defaultViewId.help</name>
            <value>/WEB-INF/views/home/help.xhtml</value>
        </init-param>
        <expiration-cache>0</expiration-cache>
        <supports>
            <mime-type>text/html</mime-type>
            <portlet-mode>view</portlet-mode>
            <portlet-mode>help</portlet-mode>
        </supports>
        <portlet-info>
            <title>Gestión de Alumnos</title>
            <short-title>ALUMNOS</short-title>
            <keywords>Gestión de Alumnos</keywords>
        </portlet-info>
        <security-role-ref>
            <role-name>administrator</role-name>
        </security-role-ref>
        <security-role-ref>
            <role-name>guest</role-name>
        </security-role-ref>
        <security-role-ref>
            <role-name>power-user</role-name>
        </security-role-ref>
        <security-role-ref>
            <role-name>user</role-name>
        </security-role-ref>
    </portlet>
</portlet-app>
```

3. Son las clases java que contienen la lógica de aplicación, es decir los métodos que son utilizados para atender peticiones o realizar procesos. Representa a la parte de controlador del paradigma MVC. Entre las clases podemos destacas las siguientes:

AbstractController.java, esta clase abstracta contiene los métodos definidos básicos que utilizaría cualquier entidad para su mantenimiento, aunque en este trabajo se utilice sólo una entidad (alumnos). Entre las funciones más destacadas encontramos, las funcione getToken() que se encarga de solicitar un token válido al servicioweb para establecer la conexión, o las peticiones de datos al webservice, como findAll() para solicitar todos los registros o restCreate() para crear nuevos registros:

```
    public String getToken() {
        try {
        WebTarget resource = client.target( this.webserviceurl ).path("/users/login")
                .queryParam("login", "portal")
                .queryParam("password", "portal");
        Response response =  resource.request().get();
        String authorizationHeader = response.getHeaderString( "Authorization");
        return authorizationHeader.substring(7);
        } catch (JSONException e) {
            System.out.println("** AbstractController :: getToken() :: error :: " + e );
            return null;
        }
    }

    public void setToken(String token) {
        this.token = token;
    }
```

```
    public List<T> findAll() {
```

```
        try {
            WebTarget resource = this.webTarget;
            String jsonListString = resource
                    .request(MediaType.APPLICATION_JSON)
                    .header( "Authorization", "Bearer " + getToken() )    // Añadido para enviar el token recuperado
                    .get(String.class);
            return convertToList(jsonListString);
        } catch (JSONException e) {
            System.out.println("** AbstractController :: findAll() :: error :: " + e );
            JsfUtil.addErrorMessage(e, "Excepció llançada quan s'intentava obtenir una llista de registres");
            return null;
        }
    }
```

```
  private boolean restCreate(T entity) throws Exception {
        boolean result = true;
        try {

            prepareToSend(entity);  // Función sobreescrita de la classe abstracta para validar los campos

             // Envio de los datos al webservice
            Response response = webTarget
                    .request( javax.ws.rs.core.MediaType.APPLICATION_JSON )
                    .header( "Authorization", "Bearer " + getToken() )    // Añadido para enviar el token recuperado
                    .post( javax.ws.rs.client.Entity.entity( entity , javax.ws.rs.core.MediaType.APPLICATION_JSON),
                            Response.class );

            // Comprobación de la respuesta del webservice
            if (response.getStatus() != Response.Status.OK.getStatusCode() ) {  // NO_CONTENT=204   OK=200
                result = false;
                throw new Exception( "No se ha podido crear el registro. Código: " + response.getStatus() );
            }

            // Si llega hasta aquí, se habrá procesado correctamente y será necesario actualizar los datos
            // de la pantalla.
            String jsonManufactureEntity = response.readEntity(String.class);
            Constructor<T> constructor = itemClass.getConstructor(new Class[]{String.class});
            this.setSelected( (T) constructor.newInstance( jsonManufactureEntity) );

        } catch (Exception ex) {
            System.out.println(" AbstractController :: restCreate() : error : " + ex );
            Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
        }
        return result;
    }
```

Por otro lado, en OptionsController.java, se han añadido métodos para controlar si un usuario determinado puede acceder o no al *portlet*, en función de si está asignado o no a un grupo de usuarios de liferay. Para ello:

Primero se recupera el usuario conectado, del contexto de liferay:

```
    private User user;
    public User getUser() {
        try {
            FacesContext facesContext = FacesContext.getCurrentInstance();
            ExternalContext externalContext = facesContext.getExternalContext();
            ThemeDisplay themeDisplay = (ThemeDisplay) externalContext.getRequestMap().get(WebKeys.THEME_DISPLAY);
            user = themeDisplay.getRealUser();
        } catch (Exception e) {
            System.out.println("optionsController : error : getUser : " + e);
        }
        return user;
    }

    public void setUser(User user) {
        this.user = user;
    }
```

A continuación se comprueba si el usuario pertenece al grupo autorizado:

```
    private String accessgroup = "GRUPO_INTRANET";
    private String groupuser;
    public String getGroupuser() {
        try {
            long groupid=0;
            this.groupuser = "";
            List<UserGroup> userGroupList = UserGroupLocalServiceUtil.getUserGroups(0,
                                            UserGroupLocalServiceUtil.getUserGroupsCount());
            for (UserGroup userGroup : userGroupList) {   // Tots els grups del liferay
                if (userGroup.getName().equalsIgnoreCase( this.accessgroup ) ) { // Si és el mateix grup
                    groupid =  userGroup.getUserGroupId();
                }
            }
            List<User> users = UserLocalServiceUtil.getUserGroupUsers(groupid); // Usuaris del grup autoritzat
            for (User user : users) {
                if ( user.getUserId() == this.getUser().getUserId() ) {       // Si l'usuari es troba dins la llista
                        this.groupuser = "GRUPO_INTRANET";                    // del grupo podrà accedir
                }
            }
        } catch (Exception e) {
            System.out.println("optionsController : error : getRol : " + e);
        }
        return this.groupuser;
    }
```

```
public void setGroupuser(String groupuser) {
    this.groupuser = groupuser;
}
```

Finalmente, un atributo determinará si el usuario puede ver o no la aplicación:

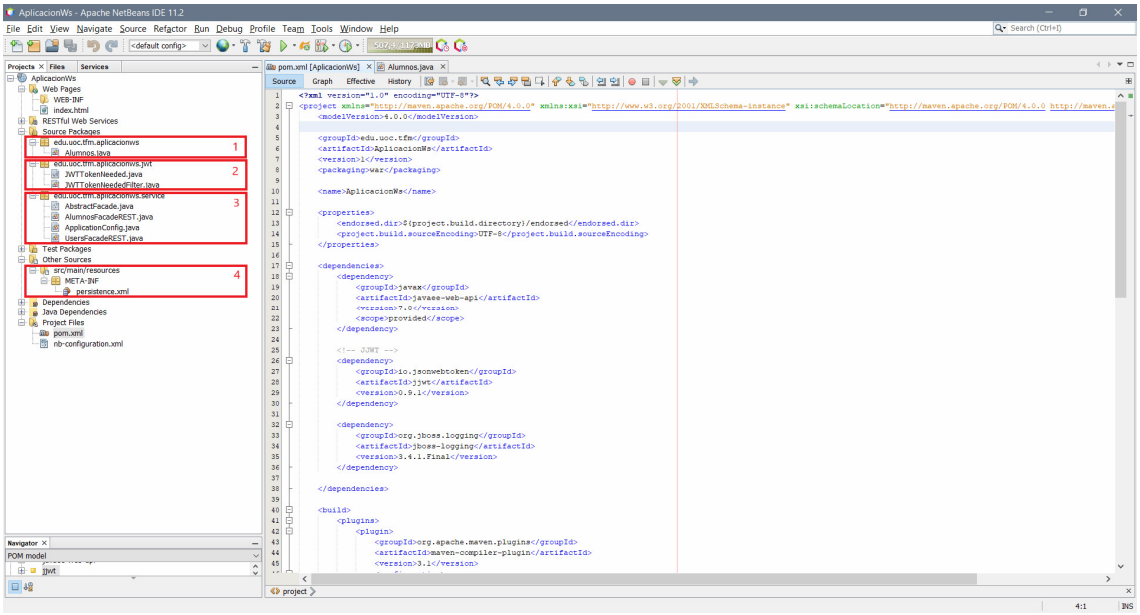
```
private boolean permis;
public boolean getPermis() {
    if (this.accessgroup.compareTo( this.getGroupuser() )==0 ) {
        return true;
    }
    return false;
}

public void setPermis(boolean permis) {
    this.permis = permis;
}
```

4. Es la clase que representa el modelo del paradigma MVC. Contiene métodos para establecer o recuperar atributos de una entidad, y así mismo como un constructor que se encarga de leer el json recibido para procesarlo con los métodos adecuados:

```
public Alumnos(String jsonentity) throws JSONException, ParseException{
    try {
        JSONObject obj = new JSONObject(jsonentity);
        if (obj.get("id")!=JSONObject.NULL) {           this.id = obj.getInt("id");                         }
        if (obj.get("codigo")!=JSONObject.NULL) {       this.codigo = obj.getString("codigo");              }
        if (obj.get("nombre")!=JSONObject.NULL) {       this.nombre = obj.getString("nombre");              }
        if (obj.get("apellido1")!=JSONObject.NULL) {    this.apellido1 = obj.getString("apellido1");        }
        if (obj.get("apellido2")!=JSONObject.NULL) {    this.apellido2 = obj.getString("apellido2");        }
        if (obj.get("direccion")!=JSONObject.NULL) {    this.direccion = obj.getString("direccion");        }
        if (obj.get("poblacion")!=JSONObject.NULL) {    this.poblacion = obj.getString("poblacion");        }
        if (obj.get("pais")!=JSONObject.NULL) {         this.pais = obj.getString("pais");                  }
    } catch (JSONException e) {
        System.out.println("Alumnes :: constructor() :: error :: " + e);
    }
}
```

Anexo 2. Aspectos destacados del webservice

El webservice se ha desarrollado con el IDE Netbeans 11.2 como un proyecto maven:



1. Alumnos.java, representa el mapeo entre campos de la tabla alumnos y los atributos de la clase. Donde hay declaradas una serie de consultas que se utilizaran posteriormente en las funciones que atenderán las peticiones, y también la utilización de una secuencia de postres para la generación de los identificadores únicos de los registros.

```
@Entity
@Table(name = "alumnos")
@XmlRootElement
@NamedQueries({
    @NamedQuery(name = "Alumnos.findAll", query = "SELECT a FROM Alumnos a"),
    @NamedQuery(name = "Alumnos.findById", query = "SELECT a FROM Alumnos a WHERE a.id = :id"),
    @NamedQuery(name = "Alumnos.findByCodigo", query = "SELECT a FROM Alumnos a WHERE a.codigo = :codigo"),
    @NamedQuery(name = "Alumnos.findByNombre", query = "SELECT a FROM Alumnos a WHERE a.nombre = :nombre"),
    @NamedQuery(name = "Alumnos.findByApellido1", query = "SELECT a FROM Alumnos a WHERE a.apellido1 = :apellido1"),
    @NamedQuery(name = "Alumnos.findByApellido2", query = "SELECT a FROM Alumnos a WHERE a.apellido2 = :apellido2"),
    @NamedQuery(name = "Alumnos.findByDireccion", query = "SELECT a FROM Alumnos a WHERE a.direccion = :direccion"),
    @NamedQuery(name = "Alumnos.findByPoblacion", query = "SELECT a FROM Alumnos a WHERE a.poblacion = :poblacion"),
    @NamedQuery(name = "Alumnos.findByPais", query = "SELECT a FROM Alumnos a WHERE a.pais = :pais")})
public class Alumnos implements Serializable {

    private static final long serialVersionUID = 1L;

    @Id
    @SequenceGenerator(name="alumnos_id_seq", sequenceName="alumnos_id_seq",  allocationSize=1)
    @GeneratedValue(strategy = GenerationType.SEQUENCE,  generator="alumnos_id_seq")
    @Column(name = "id" , updatable=false)
    private Integer id;

    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 10)
    @Column(name = "codigo")
    private String codigo;

    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 100)
    @Column(name = "nombre")
    private String nombre;

    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 100)
    @Column(name = "apellido1")
    private String apellido1;

    @Size(max = 100)
    @Column(name = "apellido2")
```

```
    private String apellido2;
```

2. Las dos clases JWTTokenNeeded y JWTTokenNeededFilter facilitan la declaración de un filtro que se ha de utilizar en todos los métodos de la API RESTful que permiten comprobar si la petición contiene un token válido.

```
@Provider
@JWTTokenNeeded
@Priority(Priorities.AUTHENTICATION)
public class JWTTokenNeededFilter implements ContainerRequestFilter {

    private static Logger logger = Logger.getLogger(JWTTokenNeededFilter.class);

    @Override
    public void filter(ContainerRequestContext requestContext) throws IOException {

        // Get the HTTP Authorization header from the request
        String authorizationHeader = requestContext.getHeaderString(HttpHeaders.AUTHORIZATION);
        logger.info("#### authorizationHeader : " + authorizationHeader);

        // Check if the HTTP Authorization header is present and formatted correctly
        if (authorizationHeader == null || !authorizationHeader.startsWith("Bearer ")) {
            logger.info("#### invalid authorizationHeader : " + authorizationHeader);
            throw new NotAuthorizedException("Authorization header must be provided");
        }

        // Extract the token from the HTTP Authorization header
        String token = authorizationHeader.substring("Bearer".length()).trim();

        try {

            // Validate the token
            String keyString = "simplekey";
            Key key = new SecretKeySpec(keyString.getBytes(), 0, keyString.getBytes().length, "DES");

            Jwts.parser().setSigningKey(key).parseClaimsJws(token);
            logger.info("#### valid token : " + token);

        } catch (Exception e) {
            logger.info("#### invalid token : " + token);
            requestContext.abortWith(Response.status(Response.Status.UNAUTHORIZED).build());
        }
    }
}
```

3. Las clases contenidas en este paquete son las que corresponden a la lógica de funcionamiento del webservice. Entre las que destaca:

AbstractFacade.java que contiene las funciones que van a interactuar con la base de datos, es decir, consultas, altas, bajas y modificaciones.

```
  public T edit(T entity) throws PersistenceException, Exception {
        try {
            getEntityManager().merge(entity);
            getEntityManager().flush();
        } catch (PersistenceException e) {
            System.out.println("AbstractFacade :: edit() :: PersistenceException :: " + e);
            System.out.println("AbstractFacade :: edit() :: PersistenceException :: cause ::  " +
                                                    e.getCause().getCause().getMessage());
            throw new Exception(e.getCause().getCause().getMessage());
        } catch (Exception ex) {
            System.out.println("AbstractFacade :: edit() :: Exception :: " + ex);
            Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
            return null;
        }
        return entity;
    }

    public void remove(T entity) throws PersistenceException, Exception {
        try {
            getEntityManager().remove(getEntityManager().merge(entity));
            getEntityManager().flush();
        } catch (PersistenceException e) {
            System.out.println("AbstractFacade :: remove() :: PersistenceException :: " + e);
            System.out.println("AbstractFacade :: remove() :: PersistenceException :: cause ::  " +
                                                    e.getCause().getCause().getMessage());
            throw new Exception(e.getCause().getCause().getMessage());
        } catch (Exception ex) {
            System.out.println("AbstractFacade :: remove() :: Exception :: " + ex);
            Logger.getLogger(this.getClass().getName()).log(Level.SEVERE, null, ex);
        }
    }

    public T find(Object id) {
        return getEntityManager().find(entityClass, id);
    }

    public List<T> findAll() {
        javax.persistence.criteria.CriteriaQuery cq = getEntityManager().getCriteriaBuilder().createQuery();
        cq.select(cq.from(entityClass));
        return getEntityManager().createQuery(cq).getResultList();
    }
```

AlumnosFacadeREST, que implementa los métodos que se pueden utilizar para interrogar a la API:

```
@Stateless
@Path("alumnos")
public class AlumnosFacadeREST extends AbstractFacade<Alumnos> {

    private static Logger logger = Logger.getLogger(AlumnosFacadeREST.class);

    @PersistenceContext(unitName = "AplicacionWsPU")
    private EntityManager em;

    public AlumnosFacadeREST() {
        super(Alumnos.class);
    }

    @POST
    @Override
    @Consumes({ MediaType.APPLICATION_JSON})
    @JWTTokenNeeded
    public Alumnos create(Alumnos entity) {
        try {
            super.create(entity);
        } catch (Exception ex) {
            System.out.println("AlumnosFacadeREST :: create() :: error :: " + ex);
            logger.info("#### AlumnosFacadeREST:: create() :: error :: " + ex);
        }
        return entity;
    }

    @PUT
    @Path("{id}")
    @Consumes({MediaType.APPLICATION_JSON})
    @JWTTokenNeeded
    public Alumnos edit(@PathParam("id") Integer id, Alumnos entity) {
        try {
            super.edit(entity);
        } catch (Exception ex) {
            System.out.println("AlumnosFacadeREST :: edit() :: error :: " + ex);
            logger.info("#### AlumnosFacadeREST :: edit() :: error ::  " + ex);
        }
         return entity;
    }
```

UsersFacadeREST, que implementa los métodos de login para validar el usuario, y si el método authenticate no se conecta a ninguna base de datos para verificar el usuario y el usuario y contraseña es la que se puede apreciar (por falta de tiempo no se ha implementado algo más relevante)

```
@Path("users")
@Produces(APPLICATION_JSON)
@Consumes(APPLICATION_JSON)
@Transactional
public class UsersFacadeREST {

    @Context
    private UriInfo uriInfo;

    @GET
    @Path("/login")
    public Response authenticateUser(@QueryParam("login") String login,
            @QueryParam("password") String password) {
        try {
            // Aquí iría el código de validación del usuario y contraseñas proporcionados,
            // por ejemplo validándolo contra una base de datos...
            authenticate(login, password);
            // Si todo es correcto, generamos el token
            String token = issueToken(login);
            // Devolvemos el token en la cabecera "Authorization".
            // Se podría devolver también en la respuesta directamente.
            return Response.ok().header(HttpHeaders.AUTHORIZATION, "Bearer " + token).build();

        } catch (Exception e) {
            return Response.status(Response.Status.UNAUTHORIZED).build();
        }
    }
    private String issueToken(String login) {
        //Calculamos la fecha de expiración del token
        Date issueDate = new Date();
        Calendar calendar = Calendar.getInstance();
        calendar.setTime(issueDate);
        calendar.add(Calendar.MINUTE, 60);
        Date expireDate = calendar.getTime();

        String keyString = "simplekey";
        Key key = new SecretKeySpec(keyString.getBytes(), 0, keyString.getBytes().length, "DES");

        //Creamos el token
        String jwtToken = Jwts.builder()
                .setSubject(login)
                .setIssuer(uriInfo.getAbsolutePath().toString())
                .setIssuedAt(issueDate)
                .setExpiration(expireDate)
                .signWith(SignatureAlgorithm.HS512, key)
```

```
                .compact();
        return jwtToken;
    }

    private void authenticate(String login, String password) throws Exception {
        if (login.compareTo("portal")!=0 || password.compareTo("portal")!=0 ) {
            throw new SecurityException("Invalid user/password");
        }
    }

}
```

4. Para finalizar, persistente.xml define el proveedor de persistencia que se utilizará, eclipselink, y la cadena de conexión que va a utilizar, y que se encuentra definida en la configuración del servidor.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.1" xmlns="http://xmlns.jcp.org/xml/ns/persistence"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/persistence
http://xmlns.jcp.org/xml/ns/persistence/persistence_2_1.xsd">
    <persistence-unit name="AplicacionWsPU" transaction-type="JTA">
        <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
        <jta-data-source>java:/uocdb</jta-data-source>
        <exclude-unlisted-classes>false</exclude-unlisted-classes>
        <shared-cache-mode>NONE</shared-cache-mode>
        <validation-mode>NONE</validation-mode>
    </persistence-unit>
</persistence>
```

Anexo 3. Certificados digitales

Unos de los puntos más evidentes para asegurar el sistema era la de cifrar las comunicaciones entre los elementos que forman parte del sistema de información, utilizando el cifrado de HTTP con SSL.

Para ello se han tenido que generar certificados autofirmados, tanto para tomcat como para wildfly. Pero además el certificado autofimado de wildfly se ha tenido que incorporar en el almacén de certificados de la JVM del servidor donde esta instalado tomcat, ya que sino se producía un error en relación con el certificado (ya que no era considerado de confianza al ser autofirmado, de la misma forma que ocurre cuando accedemos a un sitio web con un certificado no emitido por una autoridad de confianza y aparece un mensaje de error en el navegador) que hacia que el mecanismo de inicio de JWT no se pudiese completar.

Generación del certificado autofirmads para el servidor web Wildfly:

```
sudo keytool -genkey -alias admin -keyalg RSA -keystore admin.keystore
-validity 10950

sudo keytool -genkey -alias wildfly -keyalg RSA -keystore
wildfly.keystore -ext san=ip:192.168.190.200

Introduzca la contraseña del almacén de claves:
Volver a escribir la contraseña nueva:
¿Cuáles son su nombre y su apellido?
  [Unknown]:  192.198.190.200
¿Cuál es el nombre de su unidad de organización?
  [Unknown]:  tfm
¿Cuál es el nombre de su organización?
  [Unknown]:  uoc
¿Cuál es el nombre de su ciudad o localidad?
  [Unknown]:  palma
¿Cuál es el nombre de su estado o provincia?
  [Unknown]:  illes balears
¿Cuál es el código de país de dos letras de la unidad?
  [Unknown]:  es
¿Es correcto CN=192.198.190.200, OU=tfm, O=uoc, L=palma, ST=illes
balears, C=es?
  [no]:  si
```

Generación del certificado autofirmads para el servidor web Tomcat:

```
keytool -genkeypair -alias servercert -keyalg RSA -dname "CN=Web
Server,OU=Unit,O=Organization,L=City,S=State,C=ES" -keypass password -
keystore server.jks -storepass password

keytool -genkeypair -alias liferay -keystore liferay.p12 -storetype
pkcs12 -keyalg RSA -dname
"CN=liferay,OU=Unit,O=Organization,L=City,S=State,C=ES" -keypass
password -storepass password

keytool -exportcert -alias liferay -file liferay.cer -keystore
liferay.p12 -storetype pkcs12 -storepass password

keytool -importcert -keystore server.jks -alias liferay -file
```

```
liferay.cer -v -trustcacerts -noprompt -storepass password
```

Importación del certificado de Wildfly en el almacén de claves de la JVM de

```
sudo keytool -importcert -file 192_168_190_200.crt -alias wildflyip -
keystore   /usr/lib/jvm/java-11-openjdk-amd64/lib/security/cacerts   -
storepass changeit
```

Nótese que para realizar correctamente la importación se ha tenido que cambiar el nombre de wildfly.keystore a 192_168_190_200.crt para que el certificado coincida con el dominio al cual se accede.