Implementación de un sistema de Single Sign On en alta disponibilidad

Daniel Cubero
Maestría en Ciberseguridad y Privacidad
Seguridad Empresarial

Profesor Consultor: Antoni González Ciria
Profesor Responsable: Víctor García Font

01/06/2021

FICHA DEL TRABAJO FINAL

Título del trabajo:	*Implementación de un sistema de Single Sign On en alta disponibilidad*
Nombre del autor:	*Daniel Cubero Cruz*
Nombre del consultor/a:	*Antoni González Ciria*
Nombre del PRA:	*Víctor García Font*
Fecha de entrega (mm/aaaa):	06/2021
Titulación:	*Master en Ciberseguridad y Privacidad*
Área del Trabajo Final:	*Seguridad Empresarial*
Idioma del trabajo:	*Español*
Palabras clave	*Autenticación única, Control de acceso, Alta Disponibilidad*

Resumen del Trabajo (máximo 250 palabras):

En la Privacidad de la información personal, proteger la Identidad Digital, todos los sistemas necesarios para su tratamiento y el medio por el cuál estos datos llegan a transmitirse, cada vez requiere un enfoque integral, minucioso y además es un aspecto fundamental.

El Banco Central del Ecuador requiere para nuevas aplicaciones web, una tecnología de Single Sign-On, con los propósitos indicados anteriormente, infraestructura de autenticación robusta y autorización granular, correctamente estructuradas.

Este trabajo está organizado por plazos de entrega y sobre esa línea base se han realizado avances que puedan validarse, estableciendo una metodología iterativa e incremental, tomando en cuenta el análisis realizado para la implementación de cada componente.

Las actividades establecidas para la elaboración de este proyecto tienen como resultado una infraestructura de autenticación única, utilizando el servidor Central Authentication Service y un mecanismo de autorización granular con Spring Security, para acceder a las funcionalidades existentes sobre las aplicaciones web de la organización, dependiendo de los rol(es) asignado a un usuario.

Este trabajo ha tenido actividades donde prácticamente, debió hacerse uso del tiempo total de su planificación en investigación y análisis, para cumplir con los plazos de entrega. La metodología utilizada fue determinante, para la autenticación con certificados X.509 y el control de acceso granular, permitiendo alcanzar el objetivo principal y los secundarios, y; con la arquitectura establecida se logra un funcionamiento correcto en alta disponibilidad, para los componentes implementados y las aplicaciones

protegidas.

Abstract (in English, 250 words or less):

Into Privacy of personal information, protecting the Digital Identity, systems to handle it, and the medium where this data is transmitted, each time requires a comprehensive, detailed approach and is also a fundamental aspect.

The Central Bank of Ecuador requires for its new web applications, Single Sign-On technology, for purposes mentioned above, a robust authentication infrastructure and granular authorization, structured correctly.

This work is organized by deadlines and with that baseline, progress has been made, making validations, establishing an incremental and iterative methodology, taking into account analysis carried out to implement each component.

Activities established for the development of this project have resulted in a Single Sign-On infrastructure, with the CAS server and a granular authorization mechanism with Spring Security, to access to present functionalities over web applications from the organization, depending on role(s) assigned to a user.

This work has had activities where practically was obligated to use all the time to its planning on investigation and analysis, to accomplish with deadlines. The methodology utilized was determinant, to authenticate with X.509 certificates and granular access control, allowing reaching principal and secondary objectives, and; with established architecture, correct performance is achieved on high availability, to implemented components and protected applications.

Índice

Lista de figuras

Lista de Tablas

1. Introducción

En el ámbito de las aplicaciones empresariales la administración de la autenticación del usuario en diferentes servicios web de una empresa, convierte estas actividades en un desafío que se viene controlando de manera eficiente por medio de la tecnología de Single Sign On (SSO).

Por ejemplo, cuando un usuario se registra e ingresa al portal web de una empresa que provee varios servicios en internet, para realizar la personalización, cotización y compra de un auto; posteriormente puede necesitar ingresar a otro portal de la misma organización que le permita programar el mantenimiento del vehículo adquirido. Implementar las funcionalidades de SSO entre los servicios web indicados anteriormente, facilita el acceso entre los mismos al realizar la autenticación una sola vez, para que el usuario realice las operaciones que necesite con las mismas credenciales que estableció al registrarse.

Tomando en cuenta la experiencia laboral con sistemas de autenticación y control de acceso, teoría estudiada, evaluaciones y prácticas realizadas en la asignatura de Identidad Digital, esta tecnología puede complementársele con otras que permitan realizar la autorización del usuario, para acceder a diferentes funcionalidades existentes entre los portales web de la empresa, dependiendo del rol o roles adquiridos por el cliente.

La tecnología de Single Sign On en aplicaciones web empresariales es muy ventajosa por lo comentado anteriormente, aunque debe ser cuidadosamente implementada, orientándonos a la seguridad de todos los componentes involucrados, pues; como una de sus desventajas se tiene que si el sistema de autenticación única llega a ser vulnerado, pueden obtenerse las credenciales de acceso de los usuarios (entre ellos los de mayor interés), como aquellos que tengan asignados roles de alta jerarquía e ingresar libremente en todos los servicios asociados.

Además de la desventaja ya indicada, están las que se pueden producir también por las afectaciones a la disponibilidad o integridad, como ataques al sistema o fallos en los servidores, debido a que provocarían que el servicio de SSO deje de ofrecerse y no se tenga acceso a los servicios protegidos por el mismo, esto en el caso de no realizar una correcta implementación de la arquitectura que permita mantener la funcionalidad de cada componente.

1.1 Contexto y justificación del Trabajo

La realización de este trabajo está orientada a la implementación de un entorno de prueba de concepto que permita realizar una correcta presentación y la posterior aceptación con su paso a un ambiente de test, para nuevos servicios web del Banco Central del Ecuador los cuales tienen menor concurrencia de usuarios. Esta organización actualmente maneja el acceso por medio de SSO utilizando el producto comercial GetAccess de la empresa Entrust, este es un producto que provee un gran rendimiento y funcionalidades, sin embargo de alto valor económico; por esta razón han solicitado ofrecer una herramienta que

permita disminuir costos de implementación y soporte, sobre todo con sistemas que no representen una compra de licencias para su implementación con aplicaciones iniciales o adicionales, debido al coste de las mismas, por lo cual será necesario presentar una opción capaz, innovadora y de menor costo en cada rubro, manejando usuarios sobre un servicio de directorio (LDAP) como el producto que utilizan actualmente lo hace.

Este tema tiene importancia debido a que el Banco Central del Ecuador tiene confianza en las implementaciones que se ha realizado anteriormente para ellos con el sistema de SSO indicado anteriormente y busca mantener un funcionamiento similar al que ya maneja, razón por la que se nos ha solicitado realizar una presentación con una solución que maneje completamente las necesidades que se han indicado.

Debido a esto establecer una arquitectura correctamente elaborada y eficiente para su funcionalidad, será de gran ayuda en la presentación y una probable aprobación, para su ejecución en ambiente de test del cliente.

1.2 Objetivos del Trabajo

Objetivo General

Implementar un ambiente de prueba de concepto para la presentación de un sistema de autenticación única utilizando certificado digital, para la protección de servicios web del Banco Central del Ecuador.

Objetivos específicos

- Realizar la investigación bibliográfica para una adecuada implementación del producto de autenticación única y demás componentes necesarios en la arquitectura.
- Establecer una arquitectura correcta y eficiente que permita un excelente funcionamiento de todos los componentes a implementar.
- Aplicar los conocimientos obtenidos en el master de Ciberseguridad y Privacidad, para la implementación y aseguramiento de cada componente, así como de las comunicaciones entre los mismos y los usuarios que accederán a los recursos protegidos.
- Realizar pruebas de la funcionalidad del servicio de SSO y cada uno de los componentes implementados de cara al cliente final.

1.3 Enfoque y método seguido

Para conseguir los objetivos indicados en el anterior punto, esta elaboración se orientará principalmente en los plazos de las diferentes entregas de seguimiento que deben presentarse en el TFM y en esa línea base se irá realizando avances con funcionalidad que pueda validarse y se presentarán al tutor en una fecha acordada y con tiempo prudente, con el fin de que las evalúe y brinde sus observaciones, estableciendo una metodología iterativa e incremental en la implementación de los productos necesarios, con el orden que deben ser desplegados y verificados en su funcionalidad integral.

Lo anterior se logrará con la implementación de las tecnologías necesarias y la elaboración de dos aplicaciones que permitan demostrar las funcionalidades de autenticación única y autorización en la presentación de la prueba de concepto con el cliente.

1.4 Valoración de sistemas de autenticación única

Debido a que es necesario ofrecer una herramienta que permita disminuir costos de implementación y soporte. Debe presentarse una opción capaz, innovadora y en el ámbito del software libre con el fin de evitar costos por licenciamientos de cualquier tipo, por esta razón se realizará una valoración enfocada en este tipo de productos.

Se buscará una herramienta capaz de manejar diferentes mecanismos de autenticación (sobre todo con certificados x.509) e incluso la posibilidad de un doble factor, integración con el lenguaje Java, federación de identidades, integración con servicios de directorio (LDAP), compatibilidad con mecanismos de autorización y capacidad de integración con otros componentes de software libre que permitan implementar estas funcionalidades.

	OpenAM	Gluu	CAS	Keycloak	Shibboleth	LemonLDAP
Autenticación x.509	SI	SI	SI	SI	SI	SI
Factor de Autenticación Múltiple	SI	SI	SI	SI	SI	SI
Compatibilidad con Java	SI	SI	SI	SI	PARCIAL	PARCIAL
Software libre	SI	SI	SI	SI	SI	SI
Federación de identidades	SI	SI	SI	SI	SI	SI
Integración con LDAP	SI	SI	SI	SI	SI	SI
Integración con mecanismos de autorización	SI	SI	SI	SI	SI	SI
Implementación	MEDIA	MEDIA	MEDIA	MEDIA	DIFICIL	MEDIA

Tabla 1 Valoración de Sistemas de Autenticación Única / Software Libre
Fuente: sitios web de cada producto para la valoración de las funcionalidades que cada uno tiene y necesarias en este proyecto, pueden verse en la bibliografía [1].

Dentro de las valoraciones realizadas, puede verse que hay 4 soluciones idóneas para la elaboración del proyecto, sin embargo debido a experiencia laboral anterior con el servidor CAS, estudios y prácticas realizadas en la asignatura de Identidad Digital, resulta más simple para realizar una implementación organizada y por varias opciones de configuración en la autenticación con certificados x.509 que se ha encontrado en su documentación [2], este sistema sobresale con respecto a las demás alternativas.

La compatibilidad Java en CAS y con mecanismos que provean el control de acceso luce más transparente, por medio de componentes ya implementados o que se pueden agregar para diferentes aplicaciones cliente, con respecto a las demás opciones.

Tomando en cuenta la evaluación realizada, para el sistema de autenticación única, esta implementación será resuelta por medio del producto Central Authentication Service (CAS), que tiene una compatibilidad completa con aplicaciones web Java, una integración fuerte y transparente con sistemas que controlen la autorización de usuarios, para los recursos que serán protegidos.

1.5 Descripción del servicio de Central Authentication Service - CAS

CAS es un software de código libre que provee servicios de autenticación por medio de diferentes tecnologías. Este servidor está desarrollado utilizando la tecnología de servlets Java, integrado con Spring Framework para aprovechar varios de sus aspectos como Spring MVC y Spring Webflow, según indica su documentación oficial [3], todo con el fin de permitir mejoras a este producto sin mayores dificultades.

El servicio de CAS principalmente protege aplicaciones y sus datos del acceso por parte de usuarios no autenticados, por medio de la interceptación de solicitudes pidiendo recursos protegidos por CAS, estableciendo así confidencialidad para los mismos.

Los recursos protegidos son conocidos como clientes CAS, pueden ser aplicaciones que ya están habilitadas para utilizar este servicio o paquetes integrados sobre software desarrollado con diferentes lenguajes, como sucede con este proyecto, donde se han elaborado dos pequeños programas con tecnología Java.

Cada cliente se comunica con el servicio de CAS por medio de los protocolos de autenticación que son soportados [4], entre los que se tiene:

- CAS versión 1, 2 y 3
- OAuth versión 2.0
- OpenID
- OpenID Connect
- SAML versiones 1.1 y 2
- WS Federation

Estos protocolos pueden ser utilizados dependiendo de la funcionalidad entregada por el mismo y adecuada para la forma en que sea necesario autenticar a los usuarios de las aplicaciones protegidas. Para este trabajo, utilizar la implementación de CAS versión 2 ha sido suficiente, para los requerimientos de la implementación del control de acceso en alta disponibilidad, sobre las aplicaciones que se desarrollaran en JSP y la integración que brinda con Spring Security [5].

1.5.1 Protocolo de autenticación CAS

El protocolo CAS está basado en HTTP, tiene como fundamento el sistema Kerberos y se comunica a través de varias URIs que proveen las diferentes funcionalidades de sus componentes. Tomando en cuenta la documentación oficial [6] y utilizadas en la configuración de Spring Security para las aplicaciones protegidas, están las siguientes:

- /login, intercepta las solicitudes de autenticación con CAS, redirige a la URL de la página de login de CAS.
- /serviceValidate, chequea la validez de un ticket de servicio y regresa una respuesta XML indicando una autenticación exitosa, el usuario autenticado y sus atributos de ser el caso.
- /logout, controla la solicitud de logout, redirige a la URL de logout indicando el fin de la sesión en el servicio de CAS.

Si posteriormente el acceso se realiza a otra aplicación protegida por CAS en una pestaña adicional del mismo navegador, sobre el cual ya se ha autenticado anteriormente en la primera aplicación, las URIs en funcionamiento serán:

- /login, intercepta la solicitud de login con la cookie creada en la sesión de autenticación única realizada anteriormente y la cual contiene el TGT que es una identificación cifrada con validez específica, validándolo de manera que no se requiere la pantalla de inicio de sesión.
- /serviceValidate, realizando la misma funcionalidad que en la primera autenticación se produjo.

Toda esta interacción esta presentada en un diagrama de secuencia, donde todos los actores y sus acciones están representados, puede verse en el apartado de anexos.

En la emisión y validación de tickets, establecemos el SSO cuando se genera el TGT indicando anteriormente, que es la clave de sesión única para el usuario, una vez se ha autenticado correctamente.

Entonces el servidor CAS emite un ticket de servicio ST para el acceso al recurso protegido, utilizando el TGT ubicado en la cookie el cual y lo redirige al browser el cual lo utiliza para obtener el servicio correspondiente.

El componente cliente de CAS sobre la aplicación, valida el ticket de servicio sobre el servidor CAS, este retorna la respuesta XML vista anteriormente y el servicio protegido establece una cookie de sesión opacando ticket de servicio y presentando el recurso requerido.

1.5.2 Tipos de autenticación con CAS

Existen diferentes controladores de autenticación y métodos soportados por CAS [7], como son credenciales de usuario en servicios LDAP, RDBMS, X509, entre otros.

La autenticación que se va utilizar para este proyecto inicialmente se realizará con LDAP para verificar la integración con este servicio y posteriormente se establecerá la autenticación con certificado digital estándar x.509 [8].

En el mecanismo con certificado digital se establecerá el nombre del login, por medio de la extensión de certificado SubjectAltName y su valor tipo RFC822 [9], correspondiente al email asignado en la organización, con la posibilidad de utilizarlo para mapear hacia el atributo mail y buscar al usuario dentro de la rama correspondiente, sobre el servicio de directorio.

1.5.3 Arquitectura base del servicio de CAS

Los componentes vistos anteriormente servidor y clientes CAS, protocolos y mecanismos de autenticación componen la arquitectura base se esta tecnología, como se indica en su documentación oficial [10].

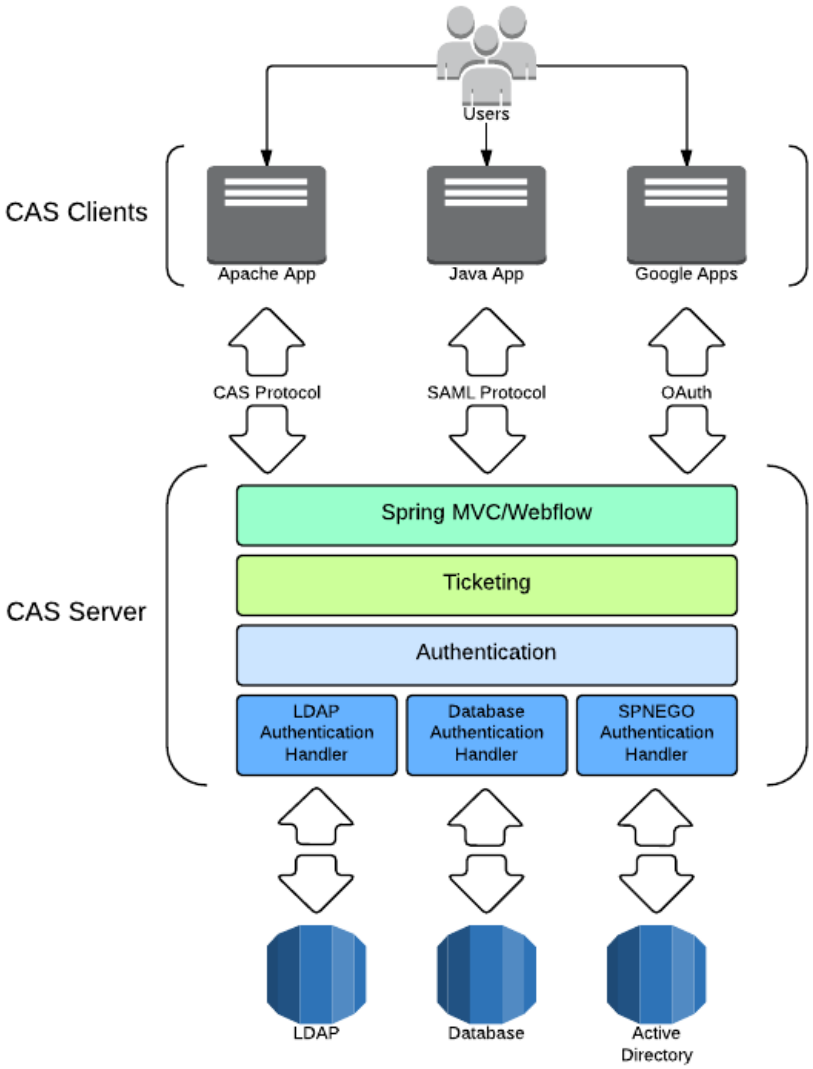


Figura 1 Arquitectura base de CAS
Fuente: sitio web de la documentación oficial para la arquitectura de CAS [10].

1.5.4. Requisitos de Hardware

La documentación oficial de Apereo CAS [11], indica que una evidencia de la comunidad sugiere un procesador de 3.00Ghz y 8GB de memoria como mínimo para un buen rendimiento. También indica que se puede afinar en más o menos recursos, según el despliegue y cantidad de solicitudes, sin dejar de tener en cuenta el tamaño en disco, sobre todo si los logs están en el mismo servidor.

Tomando en cuenta el detalle respecto al tipo de despliegue, para los recursos de las máquinas virtuales se establece capacidades mínimas para realizar una prueba de concepto, como 1GB de memoria, 1 procesador y disco de 30GB para los tres servidores que serán implementados.

1.6 Limitaciones

El proyecto tendrá dificultades principalmente en la implementación del componente de autorización, pues su configuración y desarrollo para el acceso a los recursos y usuarios con sus roles correspondientes, será la elaboración en la que más tiempo se llegue a emplear por dos situaciones a resolver:

- Obtener los roles una vez realizada la autenticación del usuario por certificado digital, puede establecer un cierto nivel de complejidad por lo que probablemente derive en una mayor cantidad de investigación y tiempo respecto a los demás componentes a implementar.
- Realizar la configuración del mecanismo de autorización para las dos aplicaciones que trabajarán con SSO y el desarrollo para permitir utilizar las funcionalidades de cada una según los roles asignados (control de acceso granular).

Con estos puntos a resolver y debido a la compatibilidad que el servidor CAS tiene con sus clientes, entre los que se encuentra integraciones con diferentes tipos de implementaciones basadas en tecnología Java, como Spring Security que por medio de librerías JAR, permite utilizar clases consumidoras de los componentes encontrados en cada una de las URLs del protocolo CAS, configurarlas para ser integradas con el servicio de autenticación y obtener atributos y roles de usuario desde el repositorio LDAP en el que serán ubicados.

Las capacidades anteriores para gestionar la autorización de usuarios, indican que centrar el análisis en Spring Security y utilizarlo para el control de acceso basado en roles, será la mejor alternativa para lograr esta funcionalidad.

Tomando en cuenta la experiencia en implementaciones de servicios de Autoridades Certificadoras, OpenLDAP, HaProxy y con versiones anteriores de CAS integrado a LDAP, para autenticación con usuario y password, estas implementaciones para autenticación con certificado x.509, pueden manejarse sin mayor inconveniente en este proyecto y permitirán designar mayor tiempo para el trabajo con los puntos a resolver.

1.7 Recursos disponibles

Para la realización del proyecto se dispone de equipos de cómputo y software, entre estos:

- 1 Laptop
  - 1TB SSD
  - Memoria 16GB
  - Procesador Intel Core i7 8th generation

- 1 Router / Firewall
- Software de virtualización VirtualBox
- Sistema operativo Debian
- OpenJDK 11
- Apache Tomcat
- OpenLDAP
- Central Authentication Service
- Lenguaje Java - JSP
- Spring Security
- HaProxy
- Servicios públicos e Internet

1.8 Planificación del Trabajo

Para la organización de los tiempos de entrega de cada actividad se establece un cuadro donde se indican los procedimientos a realizar, junto con las fechas de inicio y final para su presentación basado en las fechas establecidas para cada una de las PEC del TFM.

No.	ACTIVIDAD	FECHA DE INICIO	FECHA DE FIN	DIAS DE TRABAJO
1	**PEC 1. Plan de trabajo**	**17/02/2021**	**02/03/2021**	**14 días**
1.1	Introducción	18/02/2021	25/02/2021	8 días
1.2	Valoración económica del trabajo	26/02/2021	27/02/2021	2 días
2	**PEC 2. Entrega de seguimiento**	**03/03/2021**	**30/03/2021**	**28 días**
2.1	Arquitectura del sistema	03/03/2021	05/03/2021	3 días
2.2	Elaboración del ambiente para la prueba de concepto	06/03/2021	11/03/2021	5 días
2.3	Implementación de los nodos del servicio de OpenLDAP en modo multimaster	12/03/2021	15/03/2021	4 días
2.4	Implementación de los nodos del servicio de CAS	16/03/2021	22/03/2021	7 días
2.5	Implementación del servicio de HaProxy para OpenLDAP	23/032021	24/03/2021	2 días
2.6	Pruebas integrales del servicio de autenticación con los componentes habilitados	25/03/2021	26/03/2021	2 días
3	**PEC 3. Entrega de seguimiento**	**31/03/2021**	**27/04/2021**	**28 días**

3.1	Elaboración de aplicaciones protegidas e integraciones con CAS y Spring Security	31/03/2021	19/04/2021	20 días
3.2	Implementación del servicio de HaProxy para el servicio de CAS y las aplicaciones protegidas	20/04/2021	20/04/2021	1 día
3.3	Pruebas integrales del servicio de autenticación única y autorización con todos sus componentes habilitados	21/04/2021	23/04/2021	3 días
4	**PEC 4. Memoria final**	**28/04/2021**	**01/06/2021**	**35 días**
5	**Presentación en video**	**02/06/2021**	**08/06/2021**	**7 días**
6	**Defensa del TFM**	**14/06/2021**	**18/06/2021**	**5 días**

Tabla 2 Planificación del trabajo

1.9 Breve sumario de productos obtenidos

OpenLDAP: es un repositorio de datos que utiliza el estándar del protocolo X.500 de acceso a directorio. Su estructura es organizada como un árbol, dentro de la cual se puede establecer diferentes ramas organizativas y en ellas registros únicos en base a los atributos habilitados para ser utilizados.

Central Authentication Service (CAS): este servicio permite implementar varios mecanismos de autenticación que pueden ser utilizados para restringir el acceso a diferentes tipos de aplicaciones web. Además, puede manejar distintos factores de autenticación con el fin de asegurar más el acceso y hacerlo más confiable.

HaProxy: es un sistema de seguridad dividido en varias capas, dentro de las cuales se analiza, controla y gestiona las solicitudes realizadas a aplicaciones web. Adicionalmente, ofrece el servicio de balanceo de carga el cual puede ser implementado y monitoreado con las diferentes funcionalidades que para el mismo existen.

Spring Security: permite construir el mecanismo de autorización a diferentes aplicaciones web, por medio de roles asignados a usuarios y permitidos para los recursos protegidos, esta tecnología se puede mantener sin inconvenientes, debido a la organización y reusabilidad de estos paquetes en diferentes capas dentro de un proyecto, como en la arquitectura de software MVC.

1.10 Breve descripción de los capítulos de la memoria

a. Introducción: detalles del trabajo realizado, análisis y requerimientos del proyecto, alcance que tiene su funcionalidad, planificación y tecnología implementada para el mismo.
b. Valoración económica del trabajo: análisis del valor económico de la implementación de los componentes necesarios para el proyecto, sobre los cuales se ha establecido un margen de ganancia tomando en cuenta los requerimientos del cliente.

c. Arquitectura del sistema de SSO: distribución de los componentes implementados dentro de la infraestructura de red adecuada para este proyecto, tomando en cuenta sus necesidades, con el fin de lograr y presentar un diseño eficiente.
d. Implementación del servicio de SSO: instalación y configuración de los componentes del servicio para la autenticación única a recursos web, tomando en cuenta los requerimientos para su funcionamiento indicados por el cliente.
e. Conclusiones: ideas establecidas sobre la realización del trabajo, sus objetivos, proceso de elaboración y las soluciones que se pueden implementar posteriormente para mejorar el proyecto.
f. Glosario: términos específicos encontrados en la elaboración de los diferentes apartados, sobre la implementación de las tecnologías utilizadas en el proyecto.
g. Bibliografía: descripción de cada uno de los libros, páginas web o recursos bibliográficos consultados para la elaboración de la memoria del TFM.
h. Anexos: recursos digitales de las evidencias recogidas y utilizadas como apoyo en la implementación y documentación del proyecto.

2. Valoración económica del trabajo

Ya que el cliente busca disminuir costos para un sistema de SSO de sus nuevas aplicaciones desarrolladas en Java, para su cálculo se estima valores de esfuerzo, tiempo y productividad; basados en implementaciones realizadas anteriormente como GetAccess de Entrust, sin agregar el valor de licencias y con un precio mucho menor por soporte trimestral durante un año.

Costos Fijos	**Valor**
Servicios públicos e Internet	160
Depreciación equipos de computación	30.21
Total mensual	**$190.21**

Tabla 3 Estimación de Costos Fijos

En total para el proyecto se invertirá en un total de **$570.63** por costos fijos estimados, durante los tres meses de su investigación y pruebas.

Para la implementación del proyecto, soportes en cualquier momento del día y mantenimientos trimestrales, una vez aceptada la presentación del mismo, necesita un persona para realizará estas actividades durante un año, con un sueldo mensual promedio de **$1900**.

Entonces tenemos: **1 persona * 15 meses * $1900**. Esto establece un costo estimado de **$28500**.

Con el cálculo anterior y sumándole los costos fijos, tenemos: **$28500 + $570.63 = $29,070.63 como costo total estimado del proyecto**.
Tomando en cuenta los valores calculados junto con los conocidos de la implementación ya realizada de la herramienta comercial Getaccess, pero en este caso para proyectos con propósitos de fase inicial, se acuerda un valor con el cliente entre 40% y 50% menor a la del ejecutado con el producto de Entrust.

Con lo anterior en mente se establece un valor total del proyecto de $60,000. Así pues, calculando el retorno de la inversión se tiene:

ROI = ($60,000 - $29,070.63) / 29,070.63
ROI = 1.064 veces el valor de la inversión inicial.

Con el resultado obtenido en el ROI, se puede indicar que es un proyecto que tiene viabilidad económica.

3. Arquitectura del sistema de SSO

Debido a que este trabajo está orientado a ser una prueba de concepto con la cual se pretende la aceptación para su implementación en un ambiente de test, sobre la infraestructura del Banco Central del Ecuador.

Se diseña una arquitectura teniendo en cuenta dos firewalls con el fin de establecer una red perimetral para brindar mayor seguridad en el caso de recibir ataques e intrusiones [12], para lograr una arquitectura mayormente protegida y compatible con la que está implementada en el ambiente de test del cliente.

Sobre la red perimetral se implementa el servicio de HaProxy como balanceador, este es una simulación de los balanceadores appliance y proxies que son utilizados en el entorno del cliente, para el acceso a los servicios web protegidos y los de autenticación única.

Las comunicaciones hacia el exterior, tras el primer firewall lógicamente pueden tener menos restricciones [13], sin embargo está pensado originalmente para establecer la protección del servicio de balanceo en la red perimetral, como el único punto de acceso hacia los servicios correspondientes en la red interna, por medio de las reglas de filtrado para la apertura de los puertos necesarios para acceder a las aplicaciones web que solicitan los usuarios y el servicio de autenticación con certificado X.509.

Tras el segundo firewall donde se encuentra la red interna, están todos los servicios implementados para realizar la autenticación única con certificado digital, como los nodos para el servicio central de autenticación y los del repositorio de usuarios LDAP, estos últimos accedidos a través de un balanceador ubicado en la misma red.

Adicionalmente, sobre la red interna se encuentran dos servidores de aplicaciones con despliegues en par, para la ejecución de los recursos web protegidos.

El tráfico entre el servicio de HaProxy en la DMZ, el servidor CAS y las aplicaciones protegidas está restringido únicamente para acceder a cada uno de sus servicios, por medio de reglas de filtrado que permiten el acceso a los puertos necesarios, con el fin de limitar ataques a la red interna [14].

La seguridad de las comunicaciones se realiza implementando el cifrado de las mismas, hacia los servicios HaProxy de la DMZ, el servidor CAS y los servicios protegidos se implementa el protocolo HTTPS.

Además, se establece el cifrado de la comunicación entre el servicio de HaProxy de la red interna, los servidores OpenLDAP y estos últimos con el servidor CAS.

A continuación, el diagrama de la arquitectura establecida para la implementación de los componentes del sistema de autenticación única y las aplicaciones protegidas.

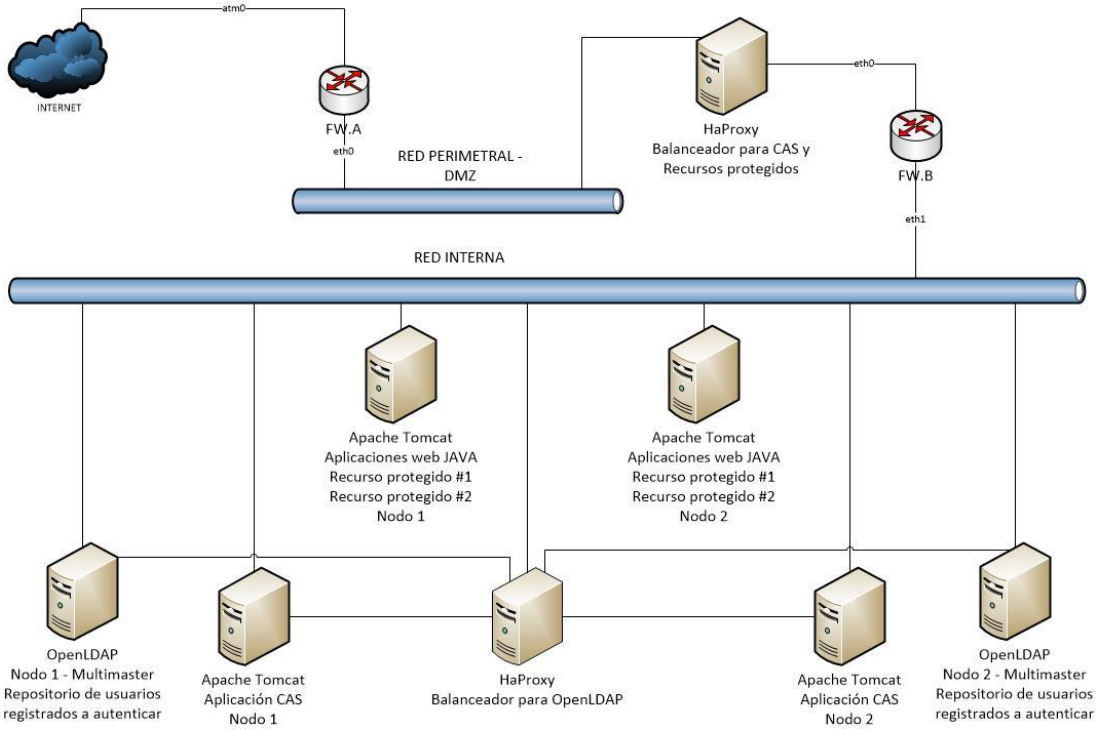


Figura 2 Arquitectura del sistema de SSO

4. Implementación del servicio de SSO

En los apartados siguientes se detallarán las actividades y tareas realizadas, para la implementación del ambiente para la prueba de concepto, servicio de autenticación única por medio de certificado digital, control de acceso hasta un nivel granular, repositorio LDAP de usuarios y roles, aplicaciones para prueba de protección y servicio de alta disponibilidad para CAS y los recursos protegidos.

4.1 Elaboración del ambiente para la prueba de concepto

En este apartado se encuentra las tareas realizadas para implementar el hardware virtual, sistemas base y herramientas necesarias para la prueba de concepto.

4.1.1 Instalación y configuración de sistemas operativos en máquinas virtuales

Se ha realizado la implementación de tres máquinas virtuales con Debian versión 10, utilizando Virtualbox, para los nodos OpenLDAP, CAS, Aplicaciones Protegidas y HaProxy. La distribución de los componentes en cada máquina virtual, será de la siguiente manera:

- Máquina virtual nodo 1 para OpenLDAP, servidor CAS, Apache Tomcat para aplicaciones protegidas y HaProxy (balanceador de los servicios de OpenLDAP).
- Máquina virtual nodo 2 para OpenLDAP, servidor CAS, Apache Tomcat para aplicaciones protegidas.
- Máquina virtual para HaProxy (balanceador en la red perimetral, para el servicio de CAS y aplicaciones protegidas)

Figura 3 Instalación de Debian

4.1.2 Instalación y configuración de los servidores de aplicación Apache Tomcat para las aplicaciones protegidas

Antes de la implementación del servidor Apache Tomcat en los nodos de la red interna, para las aplicaciones protegidas, será necesario instalar la versión de JDK compatible, para este caso OpenJDK 11.

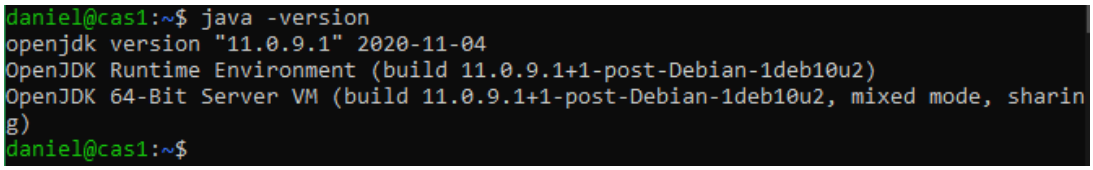


Figura 4 Versión de OpenJDK instalada

El servidor de aplicaciones utilizado es Apache Tomcat versión 8, descargado y desempaquetado para utilizarlo en el path /opt/tomcat. Se crea un usuario específico llamado tomcat, para ejecutar este servicio.

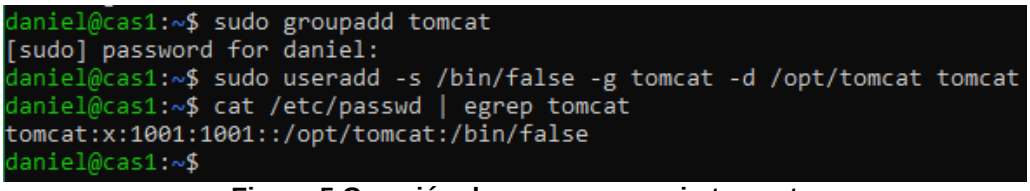


Figura 5 Creación de grupo y usuario tomcat

Para ejecutar el servicio se establece permisos de propiedad para usuario y grupo tomcat, sobre el path de la instalación de Apache Tomcat y a todas las carpetas dentro de ese directorio.

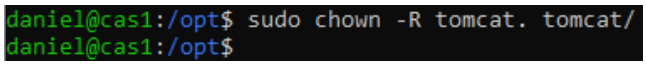


Figura 6 Permiso de propiedad al usuario del servidor de aplicaciones

Adicional se realiza la creación del archivo tomcat.service del servicio systemd en el directorio /etc/systemd/system.

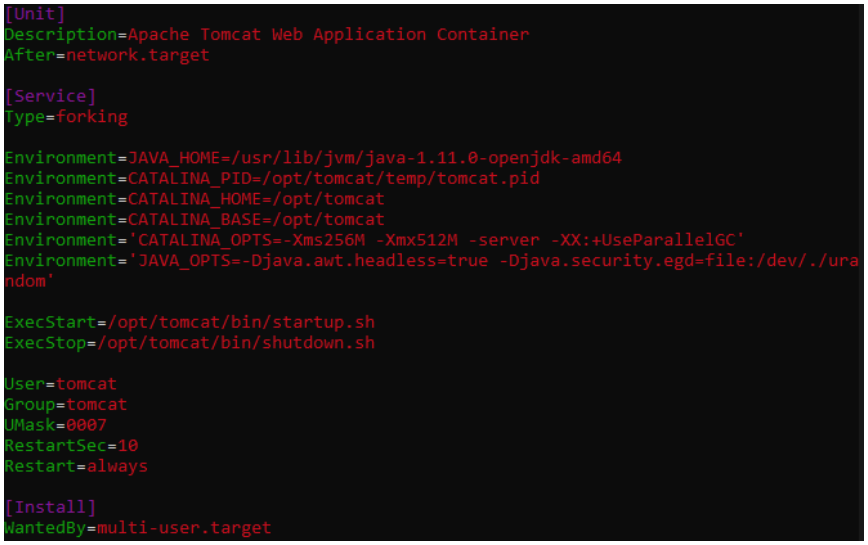


Figura 7 Configuración systemd para el servicio Apache Tomcat

La configuración de autenticación y autorización se ha habilitado en ambos nodos del servidor de aplicaciones.

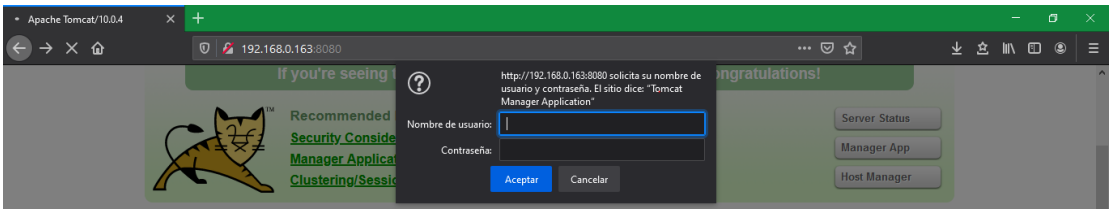


Figura 8 Autenticación en Apache Tomcat sobre el nodo1

4.1.3 Hardening de los sistemas operativos instalados

Para este apartado es necesario tener en cuenta las posibles amenazas que se pueden implantar, con el fin de evitarlas, por esta razón se han realizado varias tareas fundamentales [15], para asegurar los sistemas base en los que se implementará cada componente para el servicio de autenticación única en alta disponibilidad.

Las primeras acciones se realizaron en la instalación del sistema operativo, únicamente con el sistema base y sin otro servicio adicional. Sobre los nodos en la red interna, únicamente se accede realizando un NAT al servicio de SSH y este acceso será deshabilitado una vez finalizadas las actividades y activado cuando sea necesario.

Se ha eliminado la respuesta a ping por medio de /proc de manera persistente, sobre cada uno de los servidores del proyecto.

```
root@cas1:~# echo "net/ipv4/icmp_echo_ignore_all = 1" >> /etc/sysctl.conf
root@cas1:~# sysctl -p
net.ipv4.icmp_echo_ignore_all = 1
```

Figura 9 Des habilitación de respuesta a ping en servidores

Protección del cargador de arranque GRUB de los sistemas operativos instalados, utilizando PBKDF2 con el comando sudo grub-mkpasswd-pbkdf2 para cifrar la clave. Una vez realizada, verificamos que se ha actualizado la configuración de GRUB, con el súper usuario y password que se ha establecido en el archivo 40_custom en el path /etc/grub.d, con el comando sudo vim /boot/grub/grub.cfg.

```
### BEGIN /etc/grub.d/40_custom ###
# This file provides an easy way to add custom menu entries.  Simply type the
# menu entries you want to add after this comment.  Be careful not to change
# the 'exec tail' line above.
set superusers="root"
password_pbkdf2 root grub.pbkdf2.sha512.10000.BF455CC0749F9651AD78A7E46ABF559238B519E
55C6DA461EC69AEE0FA0C2B23F04903C32A07A7F42FFB80754CA0E021CF0B81A7FA0247DF8CC7886BE2C7
5961.26ACA079F3BC94469175BF000C0AFAAE1BBC564EB0C10C076DFA5A0E31E1D0B76AE26D2AD87B6CCB
23285EB5D7A3A3F0BFDD97DEBA9F8012C42CFA49C3B3F905
### END /etc/grub.d/40_custom ###
```

Figura 10 Configuración de Grub con usuario y password

Se realiza la instalación de la aplicación lynis, para un procedimiento de auditoría de seguridad sobre los sistemas operativos instalados [16]. En los resultados, existe 47 sugerencias muchas de ellas sobre gestión de password para los usuarios del sistema y que pueden tomarse en cuenta y manejarse sin inconveniente en ambientes de producción.

También puede verse un mensaje de warning sobre el firewall iptables del sistema el cual no ha sido habilitado y configurado, pues esta funcionalidad se utilizará con el firewall UFW.

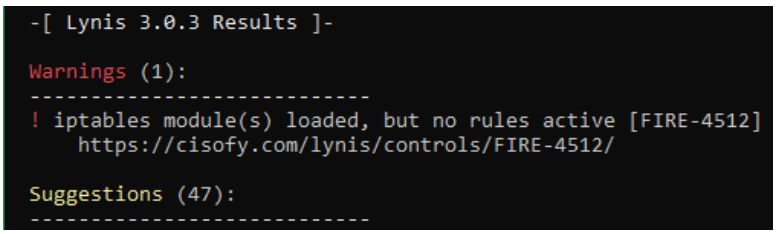


Figura 11 Mensaje de la aplicación lynis

4.1.4 Implementación de firewall UFW en los sistemas operativos

Instalación del firewall UFW en todos los servidores para la implementación del servicio de SSO, se presenta el procedimiento en el servidor de HaProxy 1 en la DMZ.

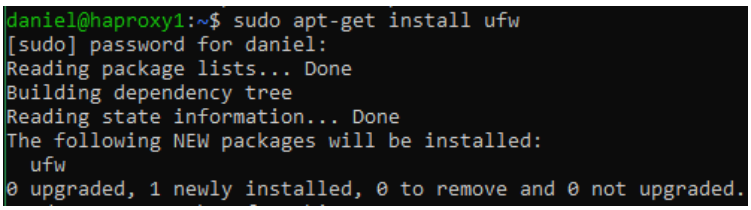


Figura 12 Instalación de firewall UFW

Activación del firewall UFW, sobre el cual se irá abriendo los puertos necesarios para cada servicio.

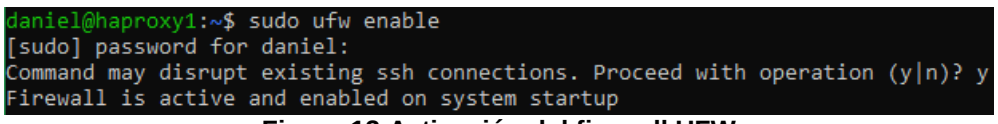


Figura 13 Activación del firewall UFW

4.1.5 Generación de llaves y certificados digitales para asegurar las comunicaciones de OpenLDAP, CAS y aplicaciones protegidas

Ejecución del comando openssl que permite crear las claves criptográficas RSA de tamaño 2048 bits y certificado x509 auto firmado con validez de 1 año, para el servicio de OpenLDAP.

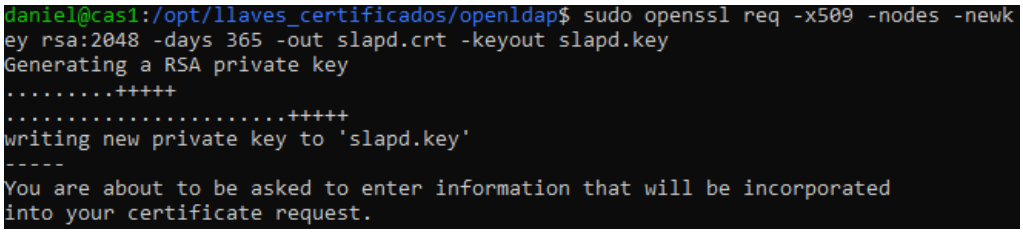


Figura 14 Comando openssl para generar llaves criptográficas de OpenLDAP

Ejecución del comando keytool que permite crear las llaves criptográficas RSA de tamaño 2048 bits y certificado x509 auto firmado con validez de 1 año, para los servicios CAS y Apache Tomcat de las aplicaciones protegidas.

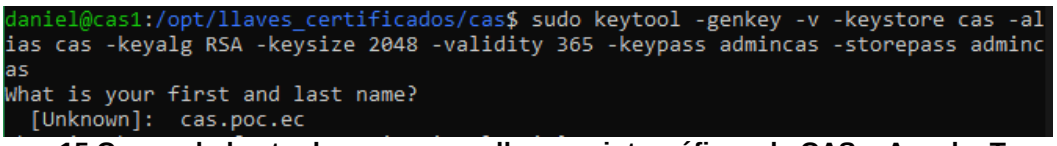


Figura 15 Comando keytool para generar llaves criptográficas de CAS y Apache Tomcat

4.2 Implementación de los nodos del servicio de OpenLDAP en modo multimaster

En este apartado se encuentra las tareas realizadas para implementar el servicio de OpenLDAP y replicación en modo multimaster.

4.2.1 Instalación y configuración inicial de los servicios de OpenLDAP

Instalación y configuración inicial de OpenLDAP y herramientas utilitarias de este servicio.

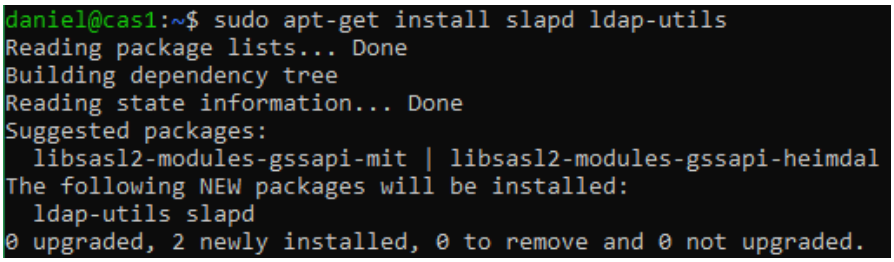


Figura 16 Instalación y configuración de OpenLDAP

Definición del password del usuario administrador del servicio de openldap.

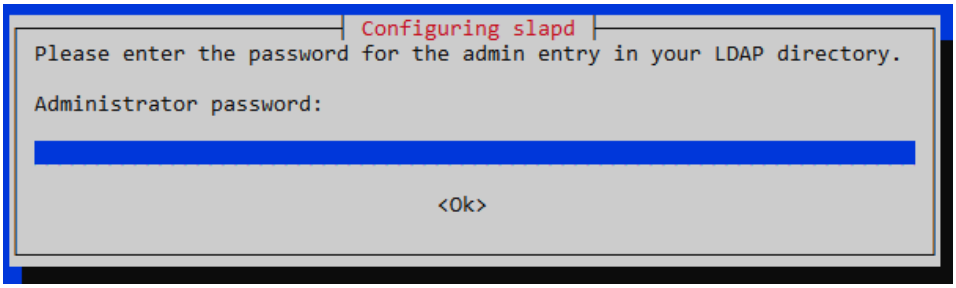


Figura 17 Configuración del password para administrador de OpenLDAP

Se permite el acceso sobre el firewall al servicio OpenLDAP, inicialmente en su puerto por defecto.

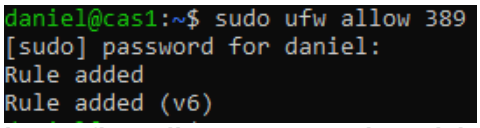


Figura 18 Permiso en firewall para acceso al servicio de OpenLDAP

4.2.2 Creación del árbol de usuarios y roles para autenticación con CAS

Definición de los usuarios que se registrarán y utilizarán para la autenticación con el servicio de CAS.

```
daniel@cas1:/opt/archivos_openldap$ cat usuarios_cas-apps.ldif
### usuarios
dn: uid=danielc,ou=usuarios, o=cas-apps, dc=poc,dc=ec
userPassword: {SSHA}5MxbX1nQ6Myx7pKxgAuqcGjyyAZHjhSq
uid: danielc
ou: usuarios
objectClass: person
objectClass: inetOrgPerson
cn: Daniel Cubero
sn: Cubero
title: Gerente
mail: danielc@poc.ec
```

Figura 19 Definición de usuarios para CAS

Definición de roles para usuarios y aplicaciones protegidas.

```
###roles
dn: cn=OPERADORES,ou=roles,o=cas-apps,dc=poc,dc=ec
objectclass: groupofnames
cn: OPERADORERS
ou: operador
member: uid=danielc, ou=usuarios, o=cas-apps,dc=poc,dc=ec
```

Figura 20 Definición de roles para CAS

Definición y configuración del módulo memberof, para habilitar la pertenencia de usuarios a grupo o roles.

```
daniel@cas1:/opt/archivos_openldap$ cat memberof.ldif
dn: cn=module,cn=config
cn: module
objectClass: olcModuleList
olcModuleLoad: memberof
olcModulePath: /usr/lib/ldap

dn: olcOverlay={0}memberof,olcDatabase={1}mdb,cn=config
objectClass: olcConfig
objectClass: olcMemberOf
objectClass: olcOverlayConfig
objectClass: top
olcOverlay: memberof
olcMemberOfDangling: ignore
olcMemberOfRefInt: TRUE
olcMemberOfGroupOC: groupOfNames
olcMemberOfMemberAD: member
olcMemberOfMemberOfAD: memberOf
```

Figura 21 Configuración de módulo memberof

Definición y configuración del módulo para integridad referencial de los atributos utilizados con el módulo memberof.

```
daniel@cas1:/opt/archivos_openldap$ cat refint_mod.ldif
dn: cn=module,cn=config
cn: module
objectClass: olcModuleList
olcModulePath: /usr/lib/ldap
olcModuleLoad: refint.la

dn: olcOverlay={1}refint,olcDatabase={1}mdb,cn=config
objectClass: olcConfig
objectClass: olcOverlayConfig
objectClass: olcRefintConfig
objectClass: top
olcOverlay: {1}refint
olcRefintAttribute: memberof member manager owner
```

Figura 22 Configuración de módulo de integridad referencial para atributos memberof

4.2.3 Configuración de replicación y modo multimaster sobre los servidores OpenLDAP

Configuración del módulo proveedor de réplica, con las clases de objeto, overlay del servicio y registro de funcionamiento.

```
daniel@cas1:/opt$ cat syncprov.ldif
dn: olcOverlay=syncprov,olcDatabase={1}mdb,cn=config
objectClass: olcOverlayConfig
objectClass: olcSyncProvConfig
olcOverlay: syncprov
olcSpSessionLog: 100
```

Figura 23 Configuración de módulo proveedor de réplica

Configuración de réplica multimaster, donde se establece la URL del proveedor de réplica, credenciales y base de búsqueda desde donde se traerán los datos, alcance de la búsqueda y tipo de conexión. Para establecer el modo multimaster se agrega la configuración de MirrorMode en true.

```
dn: olcDatabase={1}mdb,cn=config
changetype: modify
add: olcSyncRepl
olcSyncRepl: rid=001
  provider=ldap://cas2.poc.ec:389/
  bindmethod=simple
  binddn="cn=admin,dc=poc,dc=ec"
  credentials=Password.123
  searchbase="dc=poc,dc=ec"
  scope=sub
  schemachecking=on
  type=refreshAndPersist
  retry="30 5 300 3"
  interval=00:00:05:00
-
add: olcMirrorMode
olcMirrorMode: TRUE
```

Figura 24 Configuración de réplica y modo multimaster

4.2.4 Verificación de la réplica y sincronización de servidores OpenLDAP

Para realizar esta validación se ha agregado el atributo member y su valor, donde se ubica los DNs de los usuarios que tienen asignados el rol correspondiente y se ha constatado el cambio realizado sobre ambos nodos.

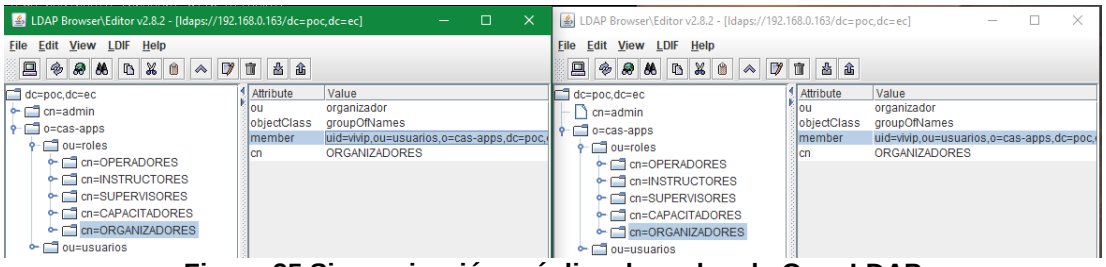


Figura 25 Sincronización - réplica de nodos de OpenLDAP

4.2.5 Implementación de protocolo seguro para OpenLDAP

Configuración de las directivas TLS sobre el archivo tls_openldap.ldif en el path /opt/llaves_certificados/openldap, para utilizar la clave privada y el certificado auto firmado.

```
dn: cn=config
changetype: modify
add: olcTLSCACertificateFile
olcTLSCACertificateFile: /opt/llaves_certificados/openldap/slapd.crt
-
replace: olcTLSCertificateFile
olcTLSCertificateFile: /opt/llaves_certificados/openldap/slapd.crt
-
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /opt/llaves_certificados/openldap/slapd.key
```

Figura 26 Configuración de llaves criptográficas y certificado para LDAP seguro

Configuración del archivo slapd en el path /etc/default, para establecer el protocolo ldap seguro (ldaps).

```
# slapd normally serves ldap only on all TCP-ports 389. slapd can also
# service requests on TCP-port 636 (ldaps) and requests via unix
# sockets.
# Example usage:
# SLAPD_SERVICES="ldap://127.0.0.1:389/ ldaps:/// ldapi:///"
SLAPD_SERVICES="ldap:/// ldapi:/// ldaps:///"
```

Figura 27 Configuración para ejecución de protocolo LDAP seguro

Permisos en firewall UFW, para acceso al puerto 636 del servicio OpenLDAP.

```
daniel@cas1:/opt/llaves_certificados/openldap$ sudo ufw allow ldaps
Rule added
Rule added (v6)
```

Figura 28 Permiso en firewall para acceso al servicio de OpenLDAP seguro

4.3 Implementación de los nodos del servicio de CAS

En este apartado se encuentra las tareas realizadas para implementar el servidor CAS.

4.3.1 Instalación y configuración inicial para los servicios de CAS

Se muestra la instalación en el primer nodo, utilizando el comando git, para la clonación del proyecto CAS Overlay Template versión 6.3.x.

```
daniel@cas1:~/proyecto-cas-overlay$ sudo git clone https://github.com/apereo/cas-ov
erlay-template.git
Cloning into 'cas-overlay-template'...
remote: Enumerating objects: 30, done.
```

Figura 29 Clonación del template CAS Overlay

Explotación del archivo WAR construido previamente para obtener el archivo de configuración application.properties y la carpeta services, desde el path build/cas-resources, dentro del proyecto CAS clonado con el fin de ubicarlo en el path src/main/resources/, para realizar configuraciones y despliegues.

```
daniel@cas1:/opt/proyecto-cas-overlay/cas-overlay-template$ sudo ./gradlew explodeW
ar

> Task :explodeWarOnly
Exploded WAR into /opt/proyecto-cas-overlay/cas-overlay-template/build/cas

> Task :explodeWar
Exploded WAR resources into /opt/proyecto-cas-overlay/cas-overlay-template/build/ca
s-resources

Deprecated Gradle features were used in this build, making it incompatible with Gra
dle 7.0.
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/6.8.1/userguide/command_line_interface.html#sec:command
_line_warnings

BUILD SUCCESSFUL in 7s
7 actionable tasks: 4 executed, 3 up-to-date
```

Figura 30 Explotación del fichero war de CAS

Configuración del archivo graddle.properties, para generación del artefacto WAR de CAS para despliegue en modo standalone.

```
# Versions
cas.version=6.3.2
springBootVersion=2.3.7.RELEASE

# Use -jetty, -undertow to other containers
# Or blank if you want to deploy to an external container
appServer=-tomcat
executable=false

tomcatVersion=9.0.43

group=org.apereo.cas
sourceCompatibility=11
targetCompatibility=11
```

Figura 31 Configuración del archivo gradle.properties

Configuración del keystore y puerto 9443, sobre el archivo application.properties, ubicado en src/main/resources/, dentro del proyecto CAS clonado.

```
##
# CAS Web Application Embedded Server SSL Configuration
#
server.ssl.key-store=file:/etc/cas/cas.jks
server.ssl.key-store-password=admincas
server.ssl.key-password=admincas
server.ssl.enabled=true
##
# CAS Web Application Embedded Server Configuration
#
server.port=9443
server.servlet.context-path=/cas
```

Figura 32 Configuración de keystore y puerto seguro de CAS

Configuración de la URL del servidor CAS, sobre el archivo cas.properties, ubicado en el path etc/cas/config/, dentro del proyecto CAS clonado.

```
cas.server.name=https://cas.poc.ec:9443
cas.server.prefix=${cas.server.name}/cas

logging.config=file:/etc/cas/config/log4j2.xml
```

Figura 33 Configuración de URL de acceso al servicio de CAS

Configuración del módulo para autenticación LDAP, sobre el archivo build.gradle.

```
    // CAS dependencies/modules may be listed here statically...
    implementation "org.apereo.cas:cas-server-webapp-init:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-ldap:${casServerVersion}"
}
```

Figura 34 Definición de módulo para autenticación con LDAP sobre CAS

Configuración para la autenticación con LDAP, sobre el archivo cas.properties, ubicado en el path etc/cas/config/, dentro del proyecto CAS, utilizando el FQDN cas-apps.poc.ec establecido en la creación del certificado digital, para asegurar el protocolo de comunicación de OpenLDAP.

```
#LDAP AUTHENTICATION
cas.authn.ldap[0].type=AUTHENTICATED
cas.authn.ldap[0].ldap-url=ldaps://cas-apps.poc.ec:636
cas.authn.ldap[0].base-dn=ou=usuarios, o=cas-apps, dc=poc,dc=ec
cas.authn.ldap[0].search-filter=uid={user}
cas.authn.ldap[0].bind-dn=cn=admin, dc=poc,dc=ec
cas.authn.ldap[0].bind-credential=Password.123
cas.authn.ldap[0].principal-attribute-list=memberOf:Roles,cn:Nombre,uid:usuario,mai
l
cas.authn.ldap[0].trust-certificates=file:/etc/cas/slapd.cer
```

Figura 35 Configuración de autenticación LDAP de CAS

Permisos en firewall UFW, para acceso al puerto 9443 del servicio CAS

```
daniel@cas1:/opt/proyecto-cas-overlay/cas-overlay-template$ sudo ufw allow 9443
Rule added
Rule added (v6)
```

Figura 36 Permiso en firewall para acceso a autenticación LDAP

4.3.2 Verificación inicial de la funcionalidad de los servicios de CAS (autenticación con usuario y password, sobre OpenLDAP)

Inicio de sesión a la aplicación CAS desde navegador en la URL https://cas.poc.ec:9443, autenticando con credenciales de usuario en LDAP.

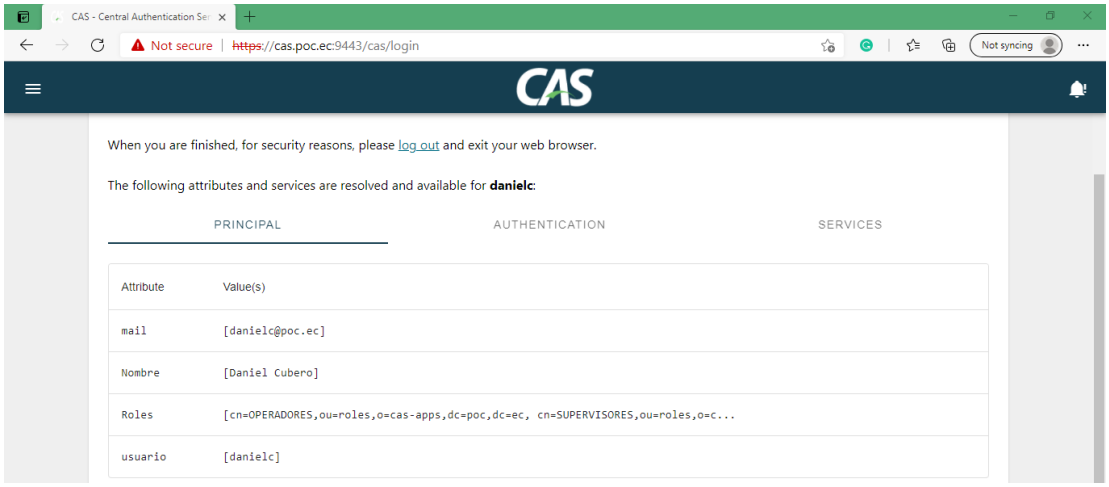


Figura 37 Autenticación LDAP sobre CAS

4.3.3 Configuración de autenticación con certificado digital para sobre CAS

Configuración del módulo para autenticación X.509, sobre el archivo build.gradle.

```
    // CAS dependencies/modules may be listed here statically...
    implementation "org.apereo.cas:cas-server-webapp-init:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-ldap:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-x509-webflow:${casServerVersion}"
}
```

Figura 38 Definición de módulo para autenticación con certificados X.509 sobre CAS

Configuración para la autenticación con X.509, sobre el archivo application.properties, ubicado en el path src/main/resources/, dentro del proyecto CAS.

```
##
#X.509 authentication
server.ssl.trust-store=file:/etc/cas/truststore.jks
server.ssl.trust-store-password=trustx.509
server.ssl.client-auth=WANT
```

Figura 39 Configuración de autenticación X.509 de servidor de aplicaciones CAS

Creación del almacén de certificados truststore.jks, para importar los certificados de Autoridades Certificadoras emisoras de credenciales de los usuarios finales.

```
daniel@cas1:/etc/cas$ sudo keytool -keystore truststore.jks -alias cas-apps -import
 -file /opt/llaves_certificados/CARaiz/caraiz.cer
[sudo] password for daniel:
Enter keystore password:
Re-enter new password:
Owner: O=cas-apps, DC=poc, DC=ec
Issuer: O=cas-apps, DC=poc, DC=ec
Serial number: 3d7f0a9d204148f8306f0ec8177363f52380c690
```

Figura 40 Creación del almacén para certificados de AC

Configuración para la autenticación con X.509, sobre el archivo cas.properties, ubicado en el path etc/cas/config/, dentro del proyecto CAS. Se habilita un

puerto extra para la autenticación separada con certificado digital X.509, utilizando el FQDN cas.poc.ec establecido en la creación del certificado digital, para asegurar la comunicación de este servicio.

```
#X.509 AUTHENTICATION
cas.authn.x509.webflow.port=9446
cas.authn.x509.webflow.client-auth=want
cas.authn.x509.principal-type=RFC822_EMAIL
cas.authn.x509.reg-ex-trusted-issuer-dn-pattern=O=cas-apps.+
cas.authn.x509.reg-ex-subject-dn-pattern=.+
cas.authn.x509.ldap.ldap-url=ldaps://cas-apps.poc.ec:636
cas.authn.x509.ldap.base-dn=ou=usuarios, o=cas-apps, dc=poc,dc=ec
cas.authn.x509.ldap.search-filter=uid={user}
cas.authn.x509.ldap.bind-dn=cn=admin, dc=poc,dc=ec
cas.authn.x509.ldap.bind-credential=Password.123
cas.authn.x509.ldap.trust-certificates=file:/etc/cas/slapd.cer
```

Figura 41 Configuración de autenticación X.509 de CAS

Permisos en firewall UFW, para acceso al puerto 9446 del servicio CAS

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ sudo ufw allow 9446
Rule added
Rule added (v6)
daniel@cas1:/opt/llaves_certificados/CARaiz$
```

Figura 42 Permiso en firewall para acceso a autenticación X.509

4.3.4 Verificación de la funcionalidad de los servicios de CAS

Generación de la clave criptográfica privada, para la firma digital del certificado de CA Raíz emisora de los certificados de usuario final.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ sudo openssl genrsa -aes256 -out carai
z.key 4096
Generating RSA private key, 4096 bit long modulus (2 primes)
.....................................................................................
.....................................+++++
.....................................................+++++
e is 65537 (0x010001)
Enter pass phrase for caraiz.key:
Verifying - Enter pass phrase for caraiz.key:
daniel@cas1:/opt/llaves_certificados/CARaiz$ ls -lth
total 4.0K
-rw------- 1 root root 3.3K Mar 24 20:04 caraiz.key
```

Figura 43 Generación de llave criptográfica privada de CA

Generación y firma del certificado digital de CA Raíz.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ sudo openssl req -x509 -new -nodes -ke
y caraiz.key -sha256 -days 365 -out caraiz.cer
Enter pass phrase for caraiz.key:
You are about to be asked to enter information that will be incorporated
into your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [EC]:ec
dominio (3 letras) [uio]:poc
nombre de la organizacion []:cas-apps
Email Address []:
```

Figura 44 Generación de certificado auto firmado para CA

Generación de la clave criptográfica privada, para la firma digital de la solicitud de certificado de usuario final.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ sudo openssl genrsa -aes256 -out danie
lc.key 2048
Generating RSA private key, 2048 bit long modulus (2 primes)
.....................................................................................
.....................+++++
.........................+++++
e is 65537 (0x010001)
Enter pass phrase for danielc.key:
Verifying - Enter pass phrase for danielc.key:
```

Figura 45 Generación de llave criptográfica privada de usuario

Generación y firma de la solicitud de certificado de usuario final.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ openssl req -new
-sha256 -key danielc.key -subj "/DC=ec/DC=poc/O=cas-apps/OU=us
uarios/UID=danielc" -out danielc.csr -config config_ext_cert -
extensions v3_req
Enter pass phrase for danielc.key:
daniel@cas1:/opt/llaves_certificados/CARaiz$ ls -lth
total 36K
-rw-r--r-- 1 daniel daniel 1.1K Mar 25 15:57 danielc.csr
```

Figura 46 Generación de solicitud de certificado para usuario

Generación y firma del certificado digital de usuario final.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ openssl x509 -req
 -in danielc.csr -CA caraiz.cer -CAkey caraiz.key -CAcreateser
ial -out danielc.cer -days 350 -sha256 -extfile config_ext_cer
t -extensions v3_req
Signature ok
subject=DC = ec, DC = poc, O = cas-apps, OU = usuarios, UID =
danielc
Getting CA Private Key
Enter pass phrase for caraiz.key:
daniel@cas1:/opt/llaves_certificados/CARaiz$ ls -lth
total 40K
-rw-r--r-- 1 daniel daniel 1.6K Mar 25 16:04 danielc.cer
```

Figura 47 Generación de certificado para usuario firmado por CA

Exportación de certificado y llave privada criptográfica de usuario final a formato PKCS#12.

```
daniel@cas1:/opt/llaves_certificados/CARaiz$ sudo openssl pkcs
12 -export -clcerts -in danielc.cer -inkey danielc.key -out da
nielc.p12
[sudo] password for daniel:
Enter pass phrase for danielc.key:
Enter Export Password:
Verifying - Enter Export Password:
daniel@cas1:/opt/llaves_certificados/CARaiz$ ls -lth
total 44K
-rw------- 1 root   root   2.7K Mar 25 16:08 danielc.p12
```

Figura 48 Exportación de llave privada y certificado de usuario en formato PKCS#12

Inicio de sesión a la aplicación CAS desde navegador en la URL https://cas.poc.ec:9446, con autenticación de certificado X.509 del usuario.

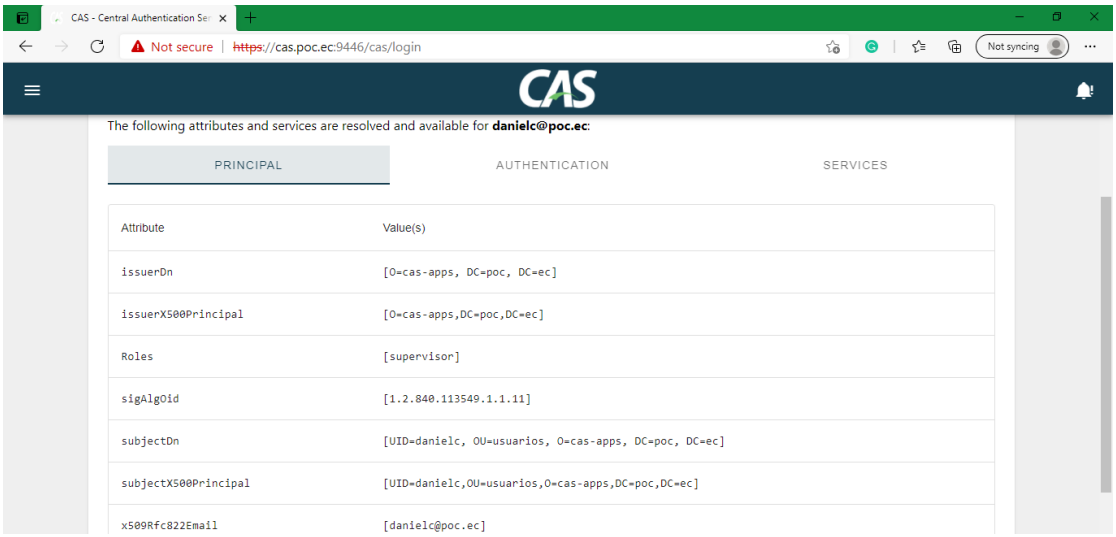


Figura 49 Autenticación X.509 sobre CAS

4.4 Implementación del servicio de HaProxy para OpenLDAP

En este apartado se encuentra las tareas realizadas para implementar el servicio de HaProxy para OpenLDAP. Para establecer esta funcionalidad es necesario crear scripts que permitan verificar el funcionamiento del servicio de LDAP y en base al estado del mismo definir una respuesta en formato HTML, para que sea enviada y entendida por el balanceador.

El procedimiento de instalación y configuración de los scritps necesarios para el monitoreo del servicio de OpenLDAP, está en el apartado de anexos.

4.4.1 Instalación y configuración del servicio de HaProxy para balanceo de OpenLDAP.

Instalación de HaProxy para OpenLDAP sobre el nodo1 interno.

```
daniel@cas1:/opt/proyecto-cas-overlay/cas-overlay-template$ sudo apt-get install ha
proxy
[sudo] password for daniel:
Reading package lists... Done
Building dependency tree
Reading state information... Done
Suggested packages:
  vim-haproxy haproxy-doc
The following NEW packages will be installed:
  haproxy
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
```

Figura 50 Instalación de HaProxy para el servicio de OpenLDAP

Configuración de HaProxy para balanceo del servicio de OpenLDAP, utilizando el puerto 636 abierto anteriormente en UFW y redirigido al puerto 1636 que utiliza el servicio LDAP.

```
listen ldap_cas
  bind *:636
  mode tcp
  timeout connect 60s
  timeout client 240s
  timeout server  240s
  balance roundrobin
  option tcpka
  option httpchk
  stick-table type ip size 1m expire 3h
#  stick on src
  server ldap1_cas 10.1.2.10:1636 check port 49202 inter 12000 fall 5
  server ldap2_cas 10.1.2.11:1636 check port 49202 inter 12000 fall 5
```

Figura 51 Configuración de HaProxy para el servicio de OpenLDAP

4.4.2 Verificación de la funcionalidad del servicio de HaProxy para OpenLDAP

Luego de bajar el segundo nodo se valida la conexión con el servicio de LDAP.

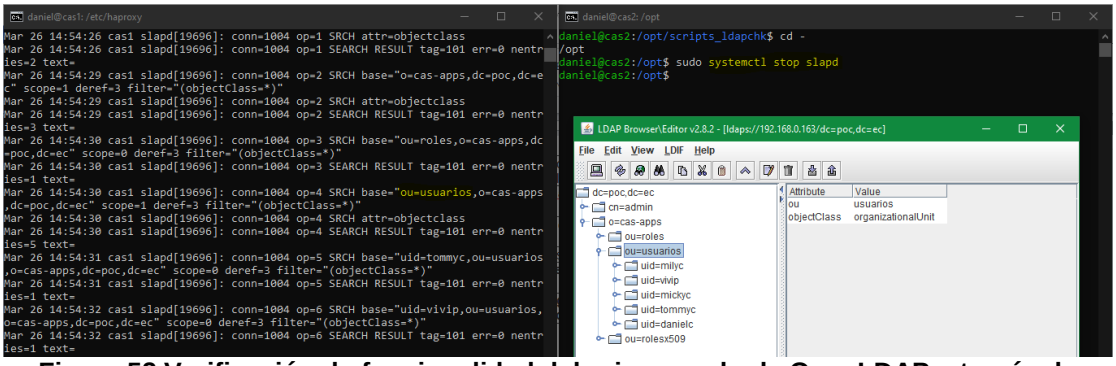


Figura 52 Verificación de funcionalidad del primer nodo de OpenLDAP a través de HaProxy

Posteriormente se finaliza el servicio en el primer nodo y se valida el servicio de LDAP.

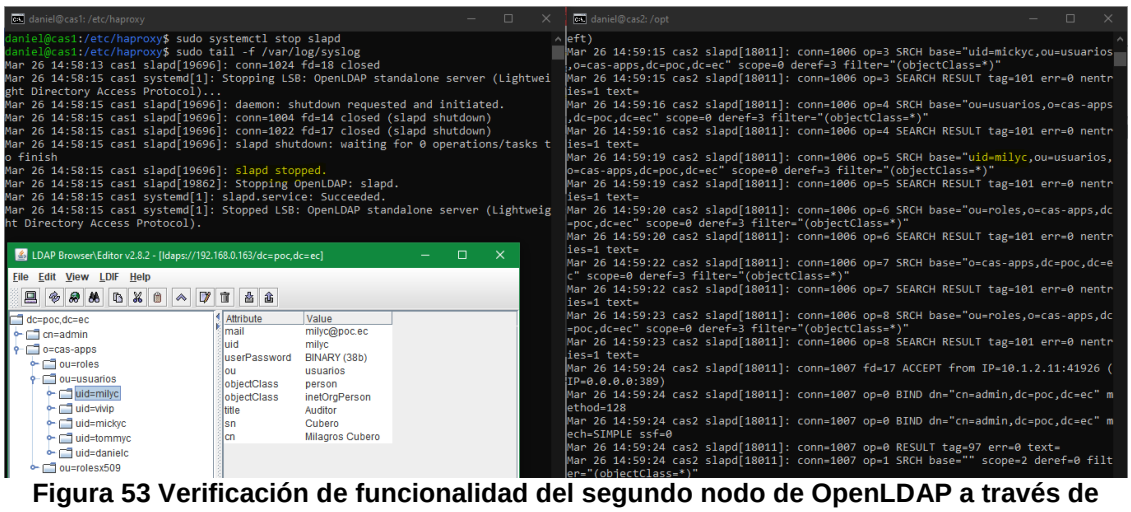


Figura 53 Verificación de funcionalidad del segundo nodo de OpenLDAP a través de HaProxy

4.5 Pruebas integrales del servicio de autenticación con los componentes habilitados

Verificación de la funcionalidad de autenticación exitosa a través del balanceo en HaProxy, habilitado para el servicio de OpenLDAP.

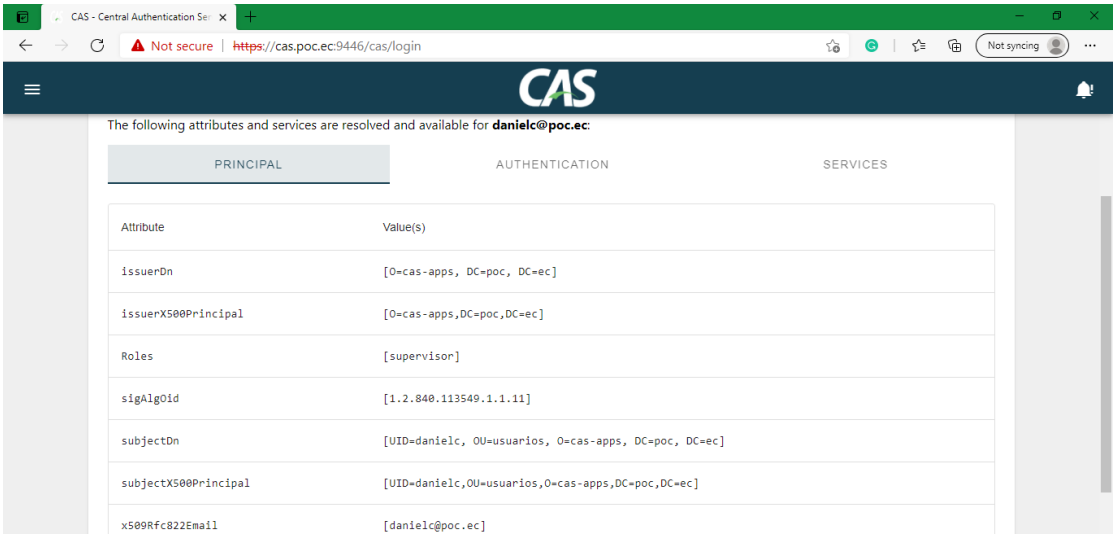


Figura 54 Validación de la funcionalidad de autenticación X.509 a través de HaProxy

Luego de bajar el segundo nodo de LDAP se valida la conexión con el servicio de autenticación, sin inconvenientes.

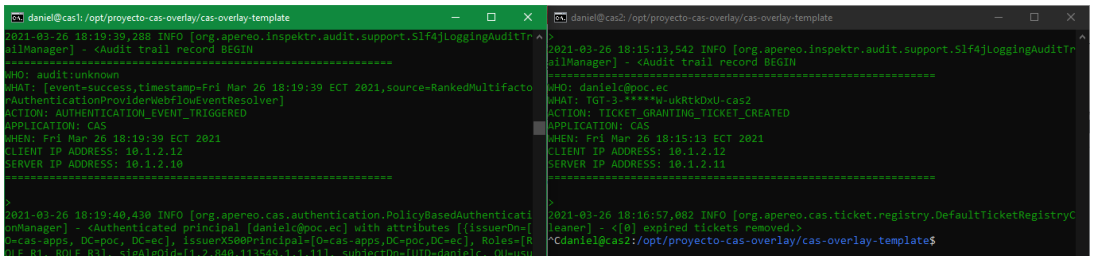


Figura 55 Validación de la autenticación X.509 y HaProxy, solo con el servicio del primer nodo de OpenLDAP

Posteriormente se finaliza el servicio de LDAP en el primer nodo y se valida el servicio de autenticación, también sin inconvenientes.

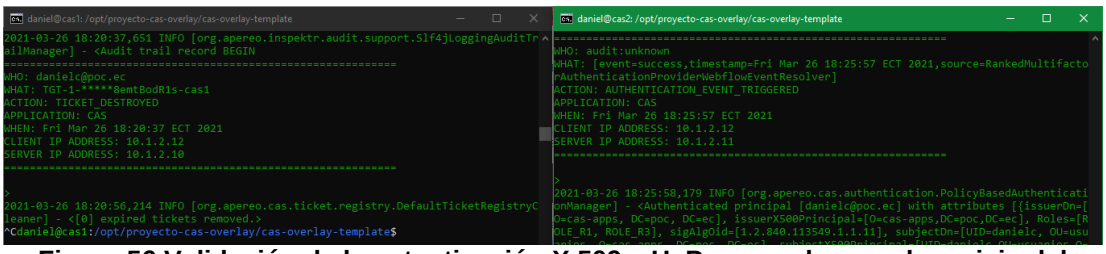


Figura 56 Validación de la autenticación X.509 y HaProxy, solo con el servicio del segundo nodo de OpenLDAP

4.6 Elaboración de aplicaciones protegidas e integraciones con CAS y Spring Security

En este apartado se encuentra las tareas realizadas para elaborar las aplicaciones protegidas e integrarlas con CAS y Spring Security.

4.6.1 Detalle de las aplicaciones desarrolladas antes de ser protegidas

Se desarrollaron dos aplicaciones simples utilizando JSP, para integrarlas al servicio de CAS, autenticar a las mismas utilizando la autenticación con certificado digital y establecer la funcionalidad de Single Sign On.

La primera aplicación permite el acceso a una lista que corresponde a funcionalidades de administración y operación, establecidas sobre menús destinados para capacitaciones, talleres y eventos. Dentro de los cuales cada usuario tendrá el acceso debido, dependiendo del rol que tenga asignado y en el caso de intentar ingresar a un recurso que no le sea permitido, encontrará una interfaz indicando la denegación correspondiente.

```
<h3>Men&uacute; Talleres</h3></td>

    <p><a href="talleres/gestionar_operadores.jsp">Gestionar operadores</a>
    <p><a href="talleres/gestionar_talleres.jsp">Gestionar talleres</a>
    <p><a href="acceso_denegado.jsp">Gestionar operaciones</a> </p>
```

Figura 57 Aplicación 1 para prueba de concepto

La segunda aplicación tiene una funcionalidad más simple, donde el usuario al ingresar tiene dos sitios a los que puede acceder, cuando su rol es de Supervisor, podrá ingresar a un sitio correspondiente al mismo. Los usuarios con otros roles podrán seguir a una interfaz también protegida y en donde se puede establecer operaciones que tengan destinadas.

```
<p>
Usuario: <%= request.getUserPrincipal() %>
</p>

<p><a href="protegido/index.jsp">Sitio protegido</a></p>
<p><a href="protegido/superior/index.jsp">Sitio para rol Supervisor</a></p>
```

Figura 58 Aplicación 2 para prueba de concepto

4.6.2 Configuración de registro y distribución de tickets a ser utilizado con alta disponibilidad en los servicios de CAS

Sobre la documentación de Apereo CAS se recomienda implementaciones basadas en cache para el registro de tickets, para despliegues en alta disponibilidad, mientras que el registro por default (en memoria) se puede establecer en entornos pequeños [17].

Existen varias implementaciones para registros de tickets en memoria con alto rendimiento, entre los que se indica en la web de Apereo están:

- Default: es el servicio de registro de tickets predeterminado en la autenticación con CAS. En un entorno de cluster activo/activo, esta implementación no es recomendada, pues no realiza una distribución de los tickets registrados entre los nodos [18].
- Ehcache: utiliza la versión 3 Ehcache, para el almacenamiento de tickets, junto con un módulo llamado Terracota en el que se apoya para realizar la distribución del cache entre los nodos de un cluster CAS. Debido a que Ehcache depende completamente de Terracota, sería una opción con poca afinidad, para la implementación en ambientes de alta disponibilidad, pues establece otro componente (en cluster) a ser administrado [19].
- Ignite: es un componente con tolerancia a fallos sobre el módulo de almacenamiento de tickets, soporta distribución de cache replicados sobre TLS en varios nodos, esta es una implementación recomendada para ambientes de alta disponibilidad [20].
- Memcached: almacena chache de tickets entre los nodos del cluster, sobre uno de ellos almacena una llave de almacenamiento de este sistema, que pasa por una función resumen, para identificar el mismo. Este componente establece el nodo en el que va a ubicar la clave, por medio de una función matemática que lo determina.

  Por esta razón en entornos de alta disponibilidad se debe monitorear su funcionamiento, para el caso en que un servidor caiga, pueda recalcular en cuál de ellos poner ahora la llave. En caso de un fallo y se permite la re autenticar el usuario, CAS no se vería afectado en su funcionamiento [21].
- Infinispan: es un sistema que también funciona con cache distribuida, puede ser utilizada como una librería java o como un módulo independiente que puede accederse remotamente. Aunque puede ser utilizado en entornos de alta disponibilidad, necesita varias implementaciones extra para su completo funcionamiento [22].
- Hazelcast: mantiene un backup del conjunto de registros de tickets de cada nodo entre los servidores CAS, para el manejo fuerte de la consistencia de datos, por lo que es fuertemente recomendado en implementaciones de CAS con alta disponibilidad [23].

Debido a lo anterior y la recomendación encontrada en la documentación oficial de CAS, puede implementarse Hazelcast con mucha seguridad, para la gestión del registro de tickets de autenticación de usuarios en alta disponibilidad.

El servicio de registro de tickets debe establecerse en el archivo build.gradle, con la configuración debajo, para habilitar la implementación del componente Hazelcast.

```
    // CAS dependencies/modules may be listed here statically...
    implementation "org.apereo.cas:cas-server-webapp-init:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-ldap:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-x509-webflow:${casServerVersion}"
    implementation "org.apereo.cas:cas-server-support-hazelcast-ticket-registry:${c
asServerVersion}"
}
```

Figura 59 Definición de módulo para registro y distribución de tickets Hazelcast sobre CAS

Configuración del componente Hazelcast en el archivo cas.properties, sobre el path /opt/proyecto-cas-overlay/cas-overlay-template/etc/cas/config, tomando en cuenta los dos nodos habilitados para el servicio de CAS.

```
#gestion de tickets con hazelcast
cas.ticket.registry.hazelcast.cluster.members=10.1.2.10,10.1.2.11
cas.ticket.registry.hazelcast.cluster.port=5703
cas.ticket.registry.hazelcast.cluster.backup-count=2
cas.ticket.registry.hazelcast.cluster.instance-name=cas1
cas.ticket.registry.hazelcast.cluster.tcpip-enabled=true
```

Figura 60 Configuración de Hazelcast sobre CAS

Se habilita el puerto 5703 en el firewall de los nodos del servicio CAS, para permitir la comunicación de componentes Hazelcast.

```
daniel@cas1:/opt/proyecto-cas-overlay/cas-overlay-template/etc/cas/config$ sudo ufw
 allow 5703
[sudo] password for daniel:
Rule added
Rule added (v6)
```

Figura 61 Permiso en firewall para acceso a puerto de cluster Hazelcast

4.6.3 Integración de aplicaciones protegidas con el servicio de CAS

Cada aplicación que se busca proteger debe ser registrada sobre el servicio de CAS, para que tenga la autorización del servicio de autenticación configurado y luego de esto será configurada con Spring Security para manejar la integración con CAS y también proveer la autorización a los módulos de cada sistema.

4.6.3.1 Registro de las aplicaciones autorizadas para proteger con CAS

Las aplicaciones que serán protegidas, deben registrarse sobre los nodos del servicio de CAS, para establecer la protección de las mismas. Existen diferentes tipos de registros para los servicios a proteger, este proyecto utilizará el registro por medio de JSON, debido a que es un tipo de registro que se puede utilizar en alta disponibilidad, donde se debe llevar a cabo un detalle de las definiciones de las aplicaciones agregadas y el cual debe ser replicado en cada uno de los nodos CAS disponibles en el cluster [24].

La configuración para poder utilizar el registro de servicios con JSON se debe declarar en el archivo build.gradle, como se define a continuación.

```
    // CAS dependencies/modules may be listed here statically...
    implementation "org.apereo.cas:cas-server-webapp-init:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-ldap:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-x509-webflow:${casServerVersion}"
    implementation "org.apereo.cas:cas-server-support-hazelcast-ticket-registry:${casServerVersion}"
    implementation "org.apereo.cas:cas-server-support-json-service-registry:${casServerVersion}"
}
```

Figura 62 Definición de módulo para registro de servicios con JSON sobre CAS

Posteriormente es necesario configurar el acceso al path donde se encuentran los ficheros JSON, que contienen la definición del registro de cada aplicación, sobre el directorio etc/cas/config de la instalación de CAS, editar el archivo cas.properties de la siguiente manera.

```
#registro de servicios
cas.service-registry.json.location=file:/etc/cas/services
```

Figura 63 Configuración de registro de servicio con JSON sobre CAS

En el path /etc/cas se debe crear la carpeta services, como se ha establecido en la configuración anterior.

```
daniel@cas1:/etc/cas$ mkdir services
daniel@cas1:/etc/cas$ chown -R daniel. services/
```

Figura 64 Creación de la carpeta que contiene los ficheros JSON de los servicios

Ahora es posible crear los archivos JSON de registro del servicio en la carpeta creada, con el fin de que la aplicación correspondiente sea protegida por CAS.

```
daniel@cas1:/etc/cas/services$ ls
poc1app-1.json  poc2app-1.json
```

Figura 65 Archivos JSON de servicios de las aplicaciones protegidas

Las definiciones en los archivos de registro de las aplicaciones autorizadas a utilizar el servicio de CAS, puede realizarse de la siguiente manera.

```
{
        "@class" : "org.apereo.cas.services.RegexRegisteredService",
        "serviceId" : "https://cas.poc.ec/poc2app.*",
        "name" : "poc2app",
        "id" : 2,
        "evaluationOrder" : 1
}
```

Figura 66 JSON servicio aplicación poc2app

4.6.3.2 Configuración de Spring Security para las aplicaciones protegidas

Las configuraciones iniciales para lograr lo indicado anteriormente, deben realizarse en el archivo descriptor de la aplicación (web.xml), sobre el cual se establece la referencia al archivo de configuración principal de Spring Security y los componentes iniciales que se deben cargar para su funcionamiento [25].

El archivo web.xml completo de una de las aplicaciones protegidas se puede ver en los anexos.

```
<context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>
        /WEB-INF/applicationContext-security.xml
    </param-value>
</context-param>

<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

Figura 67 Configuración web de la aplicación poc1app para utilizar Spring Security

El archivo de configuración para Spring Security contiene las referencias a los componentes necesarios y sus configuraciones para manejar la autenticación utilizando CAS, referenciando a las URLs correspondientes, con el fin de realizar las operaciones de login, validación del ticket de servicio y el logout del sistema [26]. En este caso el archivo nombrado applicationContext-security.xml completo de una de las aplicaciones protegidas se puede ver en los anexos.

```
<bean id="casAuthenticationEntryPoint" class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">
    <property name="loginUrl" value="https://cas.poc.ec:446/cas/login"></property>
    <property name="serviceProperties" ref="serviceProperties"></property>
</bean>

<!--
    Handles the CAS ticket processing.
 -->
<bean id="casAuthenticationProvider" class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
    <property name="userDetailsService" ref="userService"></property>
    <property name="serviceProperties" ref="serviceProperties"></property>
    <property name="ticketValidator">
        <bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">
            <constructor-arg index="0" value="https://cas.poc.ec:446/cas"></constructor-arg>
        </bean>
    </property>
    <property name="key" value="cas"></property>
</bean>

<!-- This filter handles a Single Logout Request from the CAS Server -->
  <bean id="singleLogoutFilter" class="org.jasig.cas.client.session.SingleSignOutFilter"/>
<!-- This filter redirects to the CAS Server to signal Single Logout should be performed -->
  <bean id="requestSingleLogoutFilter" class="org.springframework.security.web.authentication.logout.LogoutFilter">
          <constructor-arg value="https://cas.poc.ec:446/cas/logout"/>
          <constructor-arg>
            <bean class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler"/>
          </constructor-arg>
          <property name="filterProcessesUrl" value="/j_spring_cas_security_logout"/>
  </bean>
```

Figura 68 Referencia a las URLs de los componentes de CAS para configuración de Spring Security

Sobre el archivo applicationContext-security.xml se establece además las configuraciones necesarias para realizar proceso de autorización a los módulos del sistema, con los roles que tienen acceso a los mismos, definiendo así su nivel de autorización.

```
<security:http entry-point-ref="casAuthenticationEntryPoint" auto-config="true">

    <security:intercept-url pattern="/protegido/eventos/**" access="ROLE_ORGANIZADOR, ROLE_SUPERVISOR"></security:intercept-url>
    <security:intercept-url pattern="/protegido/capacitaciones/**" access="ROLE_CAPACITADOR, ROLE_SUPERVISOR"></security:intercept-url>
    <security:intercept-url pattern="/protegido/talleres/**" access="ROLE_OPERADOR, ROLE_INSTRUCTOR, ROLE_SUPERVISOR"></security:intercept-url>
    <security:intercept-url pattern="/protegido/**" access="ROLE_CAPACITADOR, ROLE_SUPERVISOR, ROLE_OPERADOR, ROLE_INSTRUCTOR, ROLE_ORGANIZADOR"></security:intercept-url>

    <security:custom-filter position="CAS_FILTER" ref="casAuthenticationFilter"></security:custom-filter>
    <security:logout logout-success-url="https://cas.poc.ec:9446/cas/logout?service=https://cas.poc.ec/poclapp"/>
    <security:custom-filter ref="requestSingleLogoutFilter" before="LOGOUT_FILTER"/>

</security:http>
```

Figura 69 Configuración de autorización a los recursos protegidos dentro de las aplicaciones

También se realiza la configuración de conexión al servicio LDAP, para buscar y obtener el usuario autenticado, por medio del nombre de usuario (atributo mail) sobre su entrada en la rama correspondiente del árbol de directorio, así como los roles que tiene asignados, con el fin de realizar las autorizaciones apropiadas.

```
<security:ldap-server id="ldapServerContext" manager-dn="cn=admin,dc=poc,dc=ec" manager-password="Password.123" url="ldap://cas-apps.poc.ec:1389" root="dc=poc,dc=ec" port="1389"
 />

<security:ldap-user-service id="userService"
    server-ref="ldapServerContext"
    user-search-base="ou=usuarios, o=cas-apps, dc=poc,dc=ec"
    user-search-filter="(mail={0})"
    group-search-base="ou=roles, o=cas-apps, dc=poc,dc=ec"
    group-role-attribute="ou"
    group-search-filter="(member={0})"
    role-prefix="ROLE_" />
```

Figura 70 Configuración de recuperación de usuario y roles asignado para control de acceso a los recursos protegidos

4.6.3.3 Desarrollo de la autorización a las funcionalidades de los recursos protegidos

Una vez que el usuario accedió a los recursos, debe restringírsele también el acceso a sus funcionalidades, tomando en cuenta el nivel de autorización.

Para ambas aplicaciones se utiliza el objeto request para obtener el rol o roles que tiene asignado el usuario y los cuales le autorizan a acceder a las funcionalidades dentro de cada módulo, sobre las que no lo tengan se indicará la denegación correspondiente, estableciendo así un control de acceso granular.

```
<h3>Men&uacute; Talleres</h3></td>

<%if (request.isUserInRole("ROLE_OPERADOR") && request.isUserInRole("ROLE_INSTRUCTOR")){%>
    <p><a href="talleres/gestionar_operadores.jsp">Gestionar operadores</a> *</p>
    <p><a href="talleres/gestionar_talleres.jsp">Gestionar talleres</a> *</p>
    <p><a href="talleres/gestionar_operaciones.jsp">Gestionar operaciones</a> *</p>

<%}  else if (request.isUserInRole("ROLE_INSTRUCTOR")){%>
    <p><a href="talleres/gestionar_operadores.jsp">Gestionar operadores</a> *</p>
    <p><a href="talleres/gestionar_talleres.jsp">Gestionar talleres</a> *</p>
    <p><a href="acceso_denegado.jsp">Gestionar operaciones</a> </p>

<% } else if(request.isUserInRole("ROLE_OPERADOR")){%>
    <p><a href="acceso_denegado.jsp">Gestionar operadores</a> </p>
    <p><a href="acceso_denegado.jsp">Gestionar talleres</a> </p>
    <p><a href="talleres/gestionar_operaciones.jsp">Gestionar operaciones</a> *</p>

<% } else if (request.isUserInRole("ROLE_SUPERVISOR")){%>
    <p><a href="acceso_denegado.jsp">Gestionar operadores</a> </p>
    <p><a href="talleres/gestionar_talleres.jsp">Gestionar talleres</a> *</p>
    <p><a href="acceso_denegado.jsp">Gestionar operaciones</a> </p>
<% } else {%>
    <p><a href="acceso_denegado.jsp">Gestionar operadores</a> </p>
    <p><a href="acceso_denegado.jsp">Gestionar talleres</a> </p>
    <p><a href="acceso_denegado.jsp">Gestionar operaciones</a> </p>
<% } %>
```

Figura 71 Control de acceso granular sobre la aplicación poc1app

4.6.4 Despliegue de las aplicaciones protegidas integradas con CAS y Spring Security

Antes de la realizar el despliegue de las aplicaciones desarrolladas y configuradas, debe realizarse la configuración del servidor de aplicaciones para que sea utilizado a través de protocolo seguro.

4.6.4.1 Configuración de Tomcat para el despliegue de los recursos a ser protegidos

Configuración de protocolo seguro, con el keystore creado anteriormente para CAS y Apache Tomcat, para para acceso a las aplicaciones protegidas.

```
<Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
                   maxThreads="150" SSLEnabled="true">
            <SSLHostConfig>
                    <Certificate certificateKeystoreFile="/etc/cas/cas.jks"
                                 certificateKeystorePassword="admincas" type="RSA" />
            </SSLHostConfig>
</Connector>
```

Figura 72 Configuración de protocolo seguro para Apache Tomcat

Verificación del puerto 8443 sobre el que Apache Tomcat, recibirá las solicitudes y el cual será el único para este servicio, habilitado sobre el firewall.

```
daniel@cas1:/opt$ sudo ufw allow 8443
Rule added
Rule added (v6)
```

Figura 73 Permiso en firewall para acceso a puerto seguro de Apache Tomcat

4.7 Implementación del servicio de HaProxy para el servicio de CAS y las aplicaciones protegidas

La instalación general del servicio de HaProxy que se encuentra en la DMZ, para el balanceo de los servicios de CAS y las aplicaciones protegidas una vez desplegadas, está realizada de manera similar a la hecha en la red interna para el balanceo del servidor de OpenLDAP y que se puede ver en el apartado 4.4.1.

4.7.1 Configuración del servicio de HaProxy

La configuración de este servicio de HaProxy utiliza el puerto 443, que es redirigido al puerto 8443 que utiliza el servicio de Apache Tomcat en la red interna, donde están desplegadas.

```
frontend lbl_cas_apps
    bind *:443
    mode tcp
    log             global
    use_backend cas_apps

backend cas_apps
    mode tcp
    log             global
    balance roundrobin
    option tcp-check
    stick-table type ip size 1m expire 2h
#    stick on src
    server cas1 10.1.2.10:8443 weight 1 maxconn 512 check
    server cas2 10.1.2.11:8443 weight 1 maxconn 512 check

frontend lbl_cas_auth
    bind *:446
    mode tcp
    log             global
    use_backend cas_auth

backend cas_auth
    mode tcp
    log             global
    balance roundrobin
    option tcp-check
    stick-table type ip size 1m expire 2h
#    stick on src
    server cas1 10.1.2.10:9446 weight 1 maxconn 512 check
    server cas2 10.1.2.11:9446 weight 1 maxconn 512 check
```

Figura 74 Configuración de HaProxy para el servicio de CAS y Apache Tomcat

Permisos en firewall UFW, para acceso al puerto 443 para el servicio de Apache Tomcat, también se da acceso de la misma forma al puerto 446 para CAS, para el balanceo desde el servicio de HaProxy.

```
daniel@haproxy1:/etc/haproxy$ sudo ufw allow 443
Rule added
Rule added (v6)
```

Figura 75 Permiso en firewall para acceso a puerto de HaProxy

4.7.2 Verificación de la funcionalidad del servicio de HaProxy

Para realizar una validación básica de la funcionalidad del servicio de HaProxy, desde el navegador se envía una consulta al path raíz del servicio de Apache Tomcat con el puerto 443, establecido para el balanceo, debido a que la aplicación de administración está restringida para el acceso desde la IP del balanceador, presenta una página indicando que el recurso no ha sido encontrado, sin embargo con esto se verifica su funcionamiento.

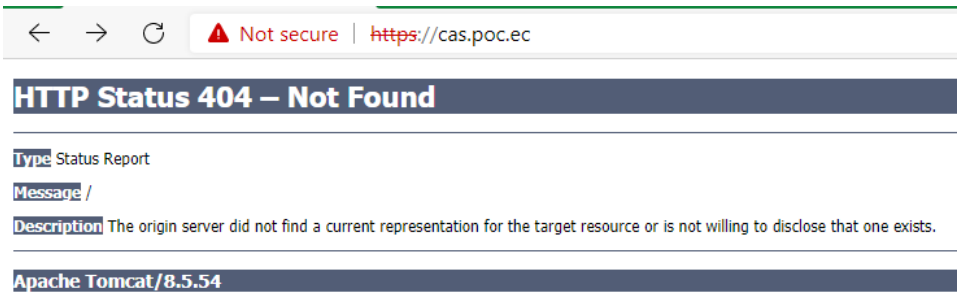


Figura 76 Validación de la funcionalidad del servicio de HaProxy

4.8. Pruebas integrales del servicio de autenticación única y autorización con todos sus componentes habilitados

Una vez implementada las funcionalidades de autenticación, autorización sobre las aplicaciones ahora protegidas y luego desplegadas en el servidor de aplicaciones, cada una puede y debe probarse, enviando la solicitud hacia HaProxy de la DMZ en el puerto 443, con el contexto correspondiente a cada aplicación, primero con poc1app.

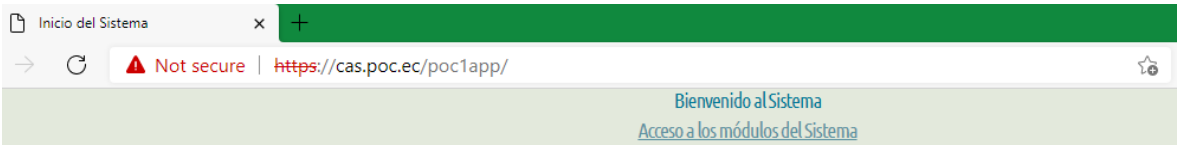


Figura 77 Acceso a la aplicación poc1app

Una vez seleccionado el certificado correspondiente, emitido por la autoridad certificadora para la cual se ha permitido el acceso de sus usuarios finales, tendremos la presentación de la aplicación protegida.

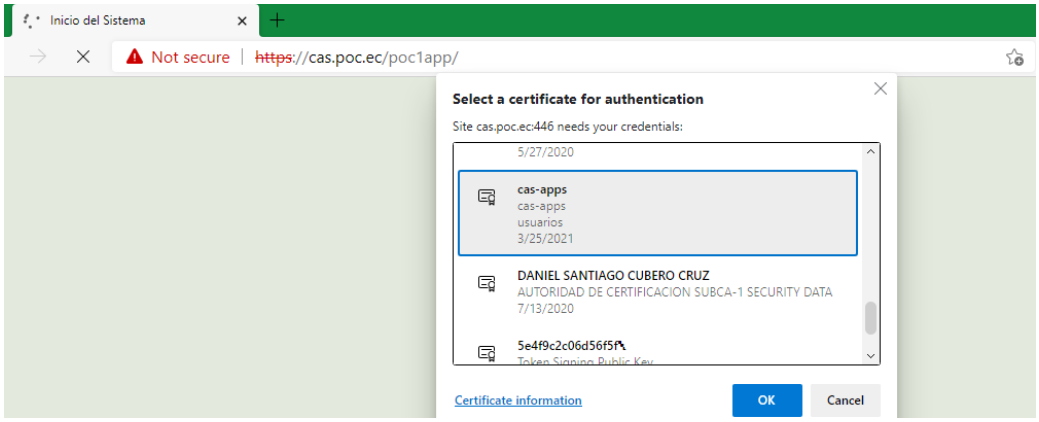


Figura 78 Autenticación X.509 para la aplicación poc1app

Sobre el mismo navegador si realizamos la solicitud de acceso a la aplicación poc2app, con la URL correspondiente, vemos que el proceso de autenticación ya no se produce nuevamente debido a la funcionalidad de Single Sign On que ofrece CAS, indica que usuario se ha autenticado, en este caso que tiene el rol de supervisor y el sitio al que debe acceder.

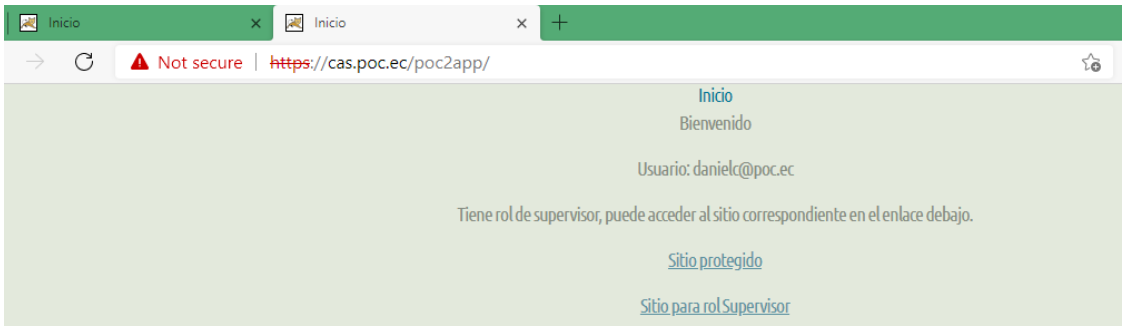


Figura 79 Verificación del servicio de autenticación única para las aplicaciones desarrolladas y protegidas con CAS

Al finalizar la autenticación para la aplicación poc1app, vemos que indica el usuario que se ha autenticado, los roles que tiene y a que funcionalidades de cada módulo tiene permiso.

Figura 80 Acceso a la aplicación con usuario y roles asignado

Siguiendo con la autorización en el módulo de capacitaciones, sobre la aplicación poc1app puede verse que el usuario con rol supervisor, tiene acceso al ingresar en la gestión de actividades.

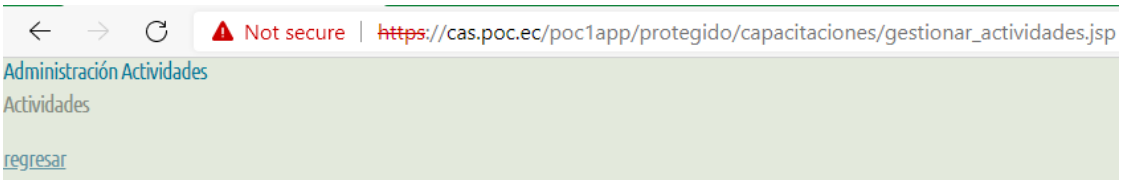


Figura 81 Acceso autorizado al módulo correspondiente para el usuario autenticado

Si se intenta un acceso a la gestión de usuarios en el módulo de capacitaciones, sobre la aplicación poc1app se presenta una página que indica la denegación a esa funcionalidad.

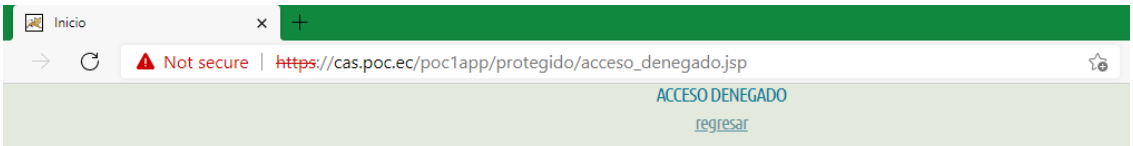


Figura 82 Acceso denegado al módulo correspondiente para el usuario autenticado

5. Conclusiones

La realización de este trabajo ha tenido actividades en las que para conseguir la funcionalidad necesaria de cada componente debió hacerse uso prácticamente del tiempo total de la planificación establecida en su investigación y análisis, con el fin de cumplir con los plazos de entrega.

La metodología utilizada ha sido determinante al momento de realizar avances y pruebas, sobre todo en la autenticación con certificados X.509 y aún más con el control de acceso granular, para alcanzar el objetivo principal y los secundarios.

- Se ha logrado la autenticación única con certificado digital, para las aplicaciones habilitadas dentro del servicio de CAS.
- La investigación bibliográfica ha sido la correspondiente para tener una implementación correcta y con resultados precisos del servicio de autenticación única y autorización sobre las aplicaciones protegidas.
- La arquitectura establecida para el ambiente de prueba de concepto ha permitido un funcionamiento correcto en alta disponibilidad, para los componentes implementados y las aplicaciones protegidas.
- Los conocimientos obtenidos en las asignaturas del Master de Ciberseguridad y Privacidad ha sido aplicados y de evidente utilidad para la implementación del proyecto.
- En la finalización de las principales fases del proyecto las pruebas de funcionalidad han sido satisfactorias, para cada uno de los componentes establecidos en su momento de manera integral al terminar las implementaciones y elaboraciones pretendidas.

Aunque se han logrado todos los objetivos establecidos, todo trabajo es susceptible de mejorar, debido a esto se establece líneas de trabajo futuro, para su implementación:

- Desarrollar un módulo para CAS que permita el auto registro de usuarios, para un tipo de autenticación específica como la realizada con certificados X.509 y con roles ligados a una rama correspondiente dentro del árbol de directorio.
- Realizar una auditoría de las versiones de software utilizadas para cada componente en la implementación de este proyecto, analizar la posibilidad de actualizarlas e implementarlas.
- Implementación de doble factor de autenticación, para mejorar su seguridad y confianza al acceder sobre las aplicaciones protegidas.

6. Glosario

Autenticación
El proceso de probar su identidad y/o determinar la validez de un conjunto de credenciales presentadas al sistema.

Autorización
Mecanismo por el cual se le asignan niveles de permiso a un usuario, con el fin de limitar las funcionalidades a las que tiene acceso dentro de una aplicación.

Credenciales
Un conjunto de datos (por ejemplo, un nombre de usuario y contraseña o certificado digital) que define un usuario para el sistema.

Certificado digital
Archivo que contiene la información de una persona, donde se establece un anclaje de confianza en la entidad que certifica los datos registrados, está enlazado a una llave criptográfica privada para firma digital o des cifrado de datos para uso exclusivo del usuario.

CAS
Un producto que gestiona centralmente el acceso a múltiples aplicaciones, proporciona a los usuarios con inicio de sesión único, para las aplicaciones web en las que se ha autenticado.

Single sign-on – autenticación única
Un método de inicio de sesión que permite a los usuarios iniciar sesión en más de una aplicación, sin tener que autenticarse cada vez que quieren utilizar un recurso.

Nodo
Una máquina, instancia de servicio, grupo de instancias de servicio o grupo de máquinas en un escenario de conmutación por error o cluster.

Cluster
Un grupo de nodos en un escenario de failover con servidores en activo-activo, utilizado en un ambiente con alta disponibilidad. Los servicios también pueden estar en modo activo-pasivo, para establecer un nivel de disponibilidad menor.

Recurso
Cualquier información basada en la Web (aplicaciones, páginas HTML y etc.) que se desea proteger, para que nadie más que el usuario autenticado y con el rol correcto pueda accederlo.

Recurso protegido
Un recurso web que requiere que los usuarios inicien sesión y tengan derechos apropiados para acceder a él.

LDAP
Protocolo ligero de acceso a directorios. Un acceso al directorio con el protocolo (DAP) especificado por Internet Engineering Task Force (IETF) RFC 1487.

Atributo
Dato que describe un aspecto de una entrada o registro de directorio. Las entradas son los bloques de construcción del directorio o árbol. Cada atributo consta de un tipo y al menos un valor. Por ejemplo, un tipo de atributo es "cn", y su valor de atributo podría ser "Daniel Cubero".

Spring Security
Es un framework que por medio de su configuración, permite establecer la autorización a diferentes aplicaciones una vez que el usuario ha sido autenticado.

Rol
Una designación la cual permite que grupos de personas tengan privilegios no disponibles para otros usuarios.

Alta disponibilidad
Implementación de nodos de sistemas informáticos que permite mantener su servicio operacional, durante un alto requerimiento del mismo o cuando suceda algún fallo en uno de ellos.

7. Bibliografía

[1] Valoración de sistemas de autenticación única / Software Libre
https://github.com/OpenIdentityPlatform/OpenAM/wiki/Documentation, consultado el 3 de mayo del 2021.
https://www.gluu.org/docs/, consultado el 3 de mayo del 2021.
https://apereo.github.io/cas/6.3.x/index.html, consultado el 3 de mayo del 2021.
https://www.keycloak.org/documentation, consultado el 3 de mayo del 2021.
https://wiki.shibboleth.net/confluence/display/CONCEPT, consultado el 4 de mayo del 2021.
https://lemonldap-ng.org/documentation/2.0/start, consultado el 4 de mayo del 2021.

[2] Valoración de sistemas de autenticación única
https://apereo.github.io/cas/6.3.x/installation/X509-Authentication.html, consultado el 5 de mayo del 2021.

[3] Uso de Spring Framework sobre CAS
https://apereo.github.io/cas/6.3.x/planning/Architecture.html#spring-framework, consultado el 6 de mayo del 2021.

[4] Protocolos soportados por el servidor CAS
https://apereo.github.io/cas/6.3.x/planning/Architecture.html#supported-protocols, consultado el 6 de mayo del 2021.

[5] Integración de CAS con Spring Security
https://github.com/apereo/java-cas-client#spring-security-integration-1, consultado el 6 de mayo del 2021.

[6] Protocolo de autenticación CAS
https://apereo.github.io/cas/6.3.x/protocol/CAS-Protocol-V2-Specification.html, consultado el 8 de mayo del 2021.

[7] Controladores y métodos de autenticación soportados por CAS
https://apereo.github.io/cas/6.3.x/installation/Configuring-Authentication-Components.html#authentication-handlers, consultado el 10 de mayo del 2021.

[8] Autenticación con certificado digital estándar X.509
https://apereo.github.io/cas/6.3.x/installation/X509-Authentication.html, consultado el 10 de mayo del 2021.

[9] Resolución del atributo para el nombre de login de usuario
https://apereo.github.io/cas/6.3.x/configuration/Configuration-Properties.html#principal-resolution-1, consulado el 11 de mayo del 2021.

[10] Arquitectura de componentes del servicio de CAS
https://apereo.github.io/cas/6.3.x/planning/Architecture.html, consultado el 11 de mayo del 2021.

[11] Requisitos de hardware para el despliegue de CAS https://apereo.github.io/cas/6.3.x/planning/Installation-Requirements.html#hardware, consultado el 12 de mayo del 2021.

[12] Guillermo Navarro Arribas. (2019). Arquitecturas con redes perimetrales. Sistemas de Cortafuegos (pp. 19) https://materials.campus.uoc.edu/daisy/Materials/PID_00212165/pdf/PID_00191697.pdf

[13] Guillermo Navarro Arribas. (2019). Arquitecturas con redes perimetrales. Sistemas de Cortafuegos (pp. 20) https://materials.campus.uoc.edu/daisy/Materials/PID_00212165/pdf/PID_00191697.pdf

[14] Guillermo Navarro Arribas. (2019). Arquitecturas con redes perimetrales. Sistemas de Cortafuegos (pp. 20) https://materials.campus.uoc.edu/daisy/Materials/PID_00212165/pdf/PID_00191697.pdf

[15] Hardening básico de Linux https://www.incibe-cert.es/seminarios-web/hardening-basico-linux, consultado el 14 de mayo del 2021.

[16] Herramienta Lynis para auditoría y hardening de sistemas Linux https://cisofy.com/lynis/, consultado el 15 de mayo del 2021.

[17] Registro y distribución de tickets para CAS https://apereo.github.io/cas/6.3.x/ticketing/Configuring-Ticketing-Components.html#ticket-registry, consultado el 17 de mayo del 2021.

[18] Registro de tickets por default https://apereo.github.io/cas/6.3.x/ticketing/Default-Ticket-Registry.html, consultado el 17 de mayo del 2021.

[19] Registro de tickets con Ehcache https://apereo.github.io/cas/6.3.x/ticketing/Ehcache-Ticket-Registry.html, consultado el 17 de mayo del 2021.

[20] Registro de tickets con Ignite https://apereo.github.io/cas/6.3.x/ticketing/Ignite-Ticket-Registry.html, consultado el 17 de mayo del 2021.

[21] Registro de tickets con Memcached https://apereo.github.io/cas/6.3.x/ticketing/Memcached-Ticket-Registry.html, consultado el 18 de mayo del 2021.

[22] Registro de tickets con Infinispan https://apereo.github.io/cas/6.3.x/ticketing/Infinispan-Ticket-Registry.html, consultado el 18 de mayo del 2021.

[23] Registro y distribución de tickets con Hazelcast
https://apereo.github.io/cas/6.3.x/ticketing/Hazelcast-Ticket-Registry.html, consultado el 18 de mayo del 2021.

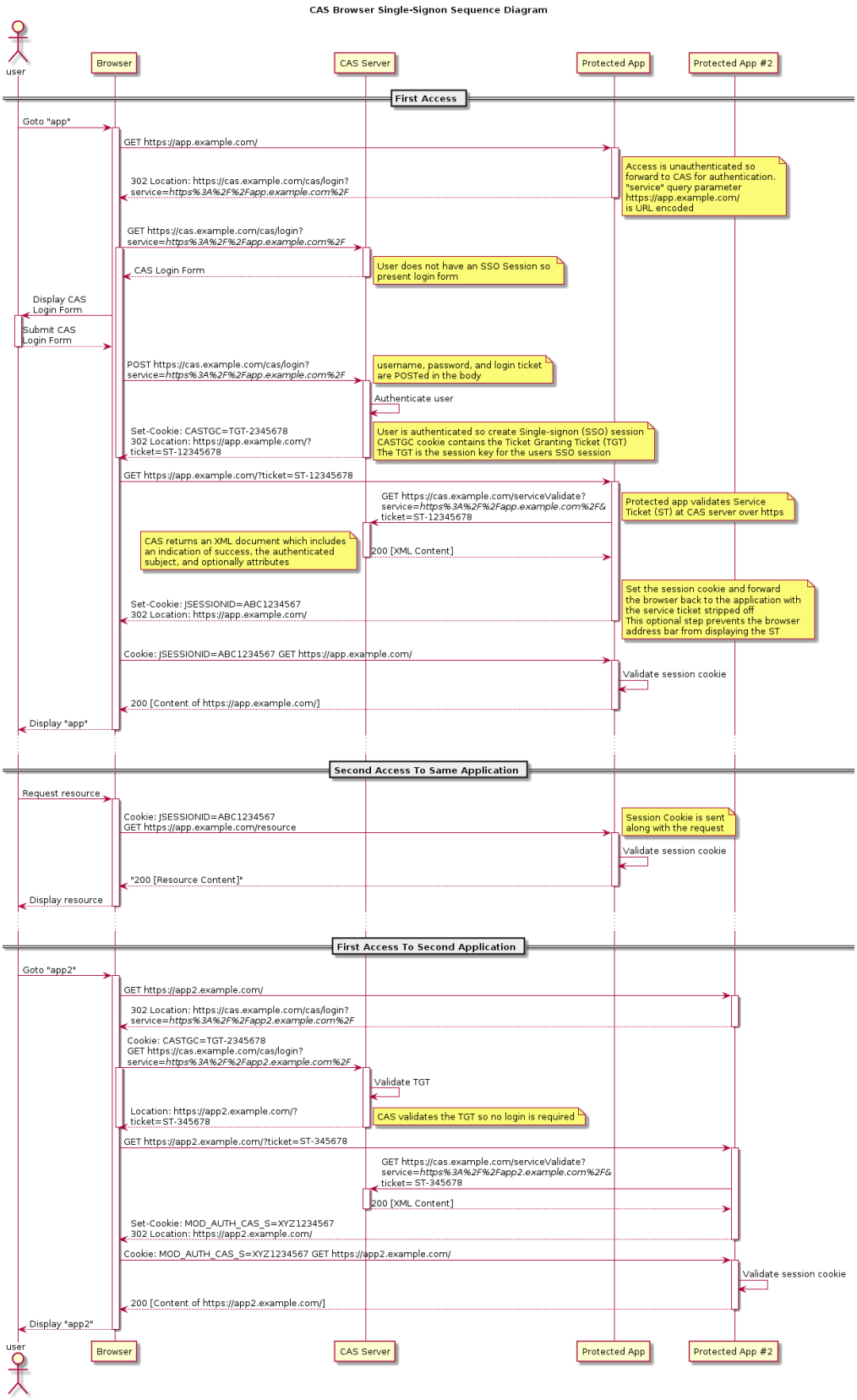
[24] Registro de aplicaciones con JSON a proteger sobre CAS
https://apereo.github.io/cas/6.3.x/services/JSON-Service-Management.html, consultado el 19 de mayo del 2021.

[25] Configuración del archivo web.xml para integrar con Spring Security
https://github.com/apereo/java-cas-client, consultado el día 20 de mayo del 2021.

[26] Configuración principal de Spring Security archivo applicationContext-security.xml
https://github.com/apereo/java-cas-client, consultado del día 21 de mayo del 2021.

8. ANEXOS

DIAGRAMA DE FLUJO WEB - AUTENTICACION UNICA



INSTALACIÓN Y CONFIGURACIÓN DE SCRITPS NECESARIOS PARA EL MONITOREO DEL SERVICIO DE OPENLDAP

Instalación de xinetd, para la respuesta html al monitoreo del servicio LDAP realizado desde haproxy.

```
daniel@cas1:/opt$ sudo apt-get install xinetd
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following packages were automatically installed and are no longer required:
  analog apache2-bin apache2-data apache2-doc apache2-utils libapr1 libaprutil1
  libaprutil1-dbd-sqlite3 libaprutil1-ldap libbrotli1 libgd3 libjansson4
  liblua5.2-0 ssl-cert
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  update-inetd
The following NEW packages will be installed:
  update-inetd xinetd
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
```

Agregar el script ldapchk en el path /etc/xinetd.d, cambiar el usuario a openldap en la configuración del archivo ldapchk.

```
service ldapchk
{
        flags           = REUSE
        socket_type     = stream
        port            = 49202
        wait            = no
        user            = openldap
        server          = /opt/scripts_ldapchk/ldapchk.sh
        log_on_failure += USERID
        disable         = no
        only_from       = 0.0.0.0/0
        per_source      = UNLIMITED
}
```

Crear la carpeta scripts_ldapchk en el path /opt, cambiar la propiedad de la carpeta para el usuario openldap y poner los permisos en 700 y creación del script de respuesta html.

```
daniel@cas1:/opt$ sudo mkdir scripts_ldapchk
daniel@cas1:/opt$ cd scripts_ldapchk/
daniel@cas1:/opt/scripts_ldapchk$ sudo vim ldapchk.sh
```

Actualización de script de respuesta html para monitoreo de servicio de LDAP en el puerto 389

```
#!/bin/bash
LDAP_SERVER=10.1.2.10
LDAP_PORT=389
LDAP_PASSWD="Password.123"
LDAP_DN="cn=admin,dc=poc,dc=ec"
output=`/usr/bin/ldapsearch -x -h ${LDAP_SERVER} -p ${LDAP_PORT}
 -D "${LDAP_DN}" -w ${LDAP_PASSWD}`
if [[ "$output" == *numResponses* ]]
then
 # LDAP is fine, return http 200
 /bin/echo -e "HTTP/1.1 200 OK\r\n"
 /bin/echo -e "Content-Type: Content-Type: text/plain\r\n"
 /bin/echo -e "\r\n"
 /bin/echo -e "LDAP is running.\r\n"
 /bin/echo -e "\r\n"
else
 # LDAP not fine, return http 503
 /bin/echo -e "HTTP/1.1 503 Service Unavailable\r\n"
 /bin/echo -e "Content-Type: Content-Type: text/plain\r\n"
 /bin/echo -e "\r\n"
 /bin/echo -e "LDAP is *down*.\r\n"
 /bin/echo -e "\r\n"
fi
```

Ejecutar el script install_xinetd para agregar xinetd al archivo de sistema /etc/services

```
daniel@cas1:/opt/scripts_ldapchk$ ls -lth
total 8.0K
-rwxr-xr-x 1 root     root     237 Mar 26 00:29 install_xinetd
-rwxr-xr-x 1 openldap openldap 688 Mar 26 00:19 ldapchk.sh
daniel@cas1:/opt/scripts_ldapchk$ sudo ./install_xinetd
```

Ejecución de systemctl enable xinetd, para habilitar el servicio en los niveles de ejecución necesarios.

```
daniel@cas1:/opt/scripts_ldapchk$ sudo systemctl enable xinetd
xinetd.service is not a native service, redirecting to systemd-s
ysv-install.
Executing: /lib/systemd/systemd-sysv-install enable xinetd
```

Permisos en firewall UFW, para acceso al puerto 49202 del servicio xinetd

```
daniel@cas1:/opt/scripts_ldapchk$ sudo ufw allow 49202
Rule added
Rule added (v6)
```

Inicio del servicio xinetd con el puerto 49202, habilitado para consultas de verificación del servicio LDAP desde haproxy.

```
daniel@cas1:/opt/scripts_ldapchk$ sudo systemctl start xinetd
daniel@cas1:/opt/scripts_ldapchk$ sudo systemctl status xinetd
● xinetd.service - LSB: Starts or stops the xinetd daemon.
   Loaded: loaded (/etc/init.d/xinetd; generated)
   Active: active (running) since Thu 2021-03-25 23:40:50 -05; 5
```

ARCHIVO WEB.XML APLICACIÓN POC1APP

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

<web-app xmlns="http://xmlns.jcp.org/xml/ns/javaee"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
                      http://xmlns.jcp.org/xml/ns/javaee/web-app_3_1.xsd"
  version="3.1"
  metadata-complete="true">

    <context-param>
        <param-name>contextConfigLocation</param-name>
        <param-value>
            /WEB-INF/applicationContext-security.xml
        </param-value>
    </context-param>

    <filter>
        <filter-name>springSecurityFilterChain</filter-name>
        <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
    </filter>

    <filter-mapping>
        <filter-name>springSecurityFilterChain</filter-name>
        <url-pattern>/*</url-pattern>
    </filter-mapping>

    <!--
        Loads the root application context of this web app at startup.
    -->
    <listener>
        <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>

    <!--
        -Publishes events for session creation and destruction through the application
                - context. Optional unless concurrent session control is being used.
    -->
    <listener>
      <listener-class>org.springframework.security.web.session.HttpSessionEventPublisher</listener-class>
    </listener>

    <description>
                APP POC 1
    </description>
    <display-name>APP POC 1</display-name>

    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>

</web-app>
```

ARCHIVO APLICATIONCONTEXT-SECURITY.XML APLICACIÓN POC1APP

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

<beans xmlns:security="http://www.springframework.org/schema/security"
    xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                    http://www.springframework.org/schema/security
http://www.springframework.org/schema/security/spring-security-3.1.xsd">

  <!--
                    Enable security, let the casAuthenticationEntryPoint handle all intercepted urls.
                    The CAS_FILTER needs to be in the right position within the filter chain.
            -->
            <security:http entry-point-ref="casAuthenticationEntryPoint" auto-config="true">

                    <security:intercept-url pattern="/protegido/eventos/**" access="ROLE_ORGANIZADOR,
ROLE_SUPERVISOR"></security:intercept-url>
                    <security:intercept-url pattern="/protegido/capacitaciones/**"
access="ROLE_CAPACITADOR, ROLE_SUPERVISOR"></security:intercept-url>
                    <security:intercept-url pattern="/protegido/talleres/**" access="ROLE_OPERADOR,
ROLE_INSTRUCTOR, ROLE_SUPERVISOR"></security:intercept-url>
                    <security:intercept-url pattern="/protegido/**" access="ROLE_CAPACITADOR,
ROLE_SUPERVISOR, ROLE_OPERADOR, ROLE_INSTRUCTOR,
ROLE_ORGANIZADOR"></security:intercept-url>

                    <security:custom-filter position="CAS_FILTER"
ref="casAuthenticationFilter"></security:custom-filter>
                    <security:logout logout-success-
url="https://cas.poc.ec:446/cas/logout?service=https://cas.poc.ec/poc1app"/>
                    <security:custom-filter ref="requestSingleLogoutFilter" before="LOGOUT_FILTER"/>
                    <security:custom-filter ref="singleLogoutFilter" before="CAS_FILTER"/>
            </security:http>

            <!--
                    Required for the casProcessingFilter, so define it explicitly set and
                    specify an Id Even though the authenticationManager is created by
                    default when namespace based config is used.
            -->
            <security:authentication-manager alias="authenticationManager">
                    <security:authentication-provider ref="casAuthenticationProvider"></security:authentication-
provider>
            </security:authentication-manager>

            <!--
                    This section is used to configure CAS. The service is the
                    actual redirect that will be triggered after the CAS login sequence.
            -->
            <bean id="serviceProperties" class="org.springframework.security.cas.ServiceProperties">
                    <property name="service"
value="https://cas.poc.ec/poc1app/j_spring_cas_security_check"></property>
                    <property name="sendRenew" value="false"></property>
            </bean>

      <!--
                    The CAS filter handles the redirect from the CAS server and starts the ticket validation.
            -->
            <bean id="casAuthenticationFilter"
class="org.springframework.security.cas.web.CasAuthenticationFilter">
                    <property name="authenticationManager" ref="authenticationManager"></property>
            </bean>
```

CONTINUACION DEL ARCHIVO APLICATIONCONTEXT-SECURITY.XML APLICACIÓN POC1APP

```
<!--
		The entryPoint intercepts all the CAS authentication requests.
		It redirects to the CAS loginUrl for the CAS login page.
	-->
	<bean id="casAuthenticationEntryPoint"
class="org.springframework.security.cas.web.CasAuthenticationEntryPoint">
		<property name="loginUrl" value="https://cas.poc.ec:446/cas/login"></property>
		<property name="serviceProperties" ref="serviceProperties"></property>
	</bean>

	<!--
		Handles the CAS ticket processing.
	 -->
	<bean id="casAuthenticationProvider"
class="org.springframework.security.cas.authentication.CasAuthenticationProvider">
		<property name="userDetailsService" ref="userService"></property>
		<property name="serviceProperties" ref="serviceProperties"></property>
		<property name="ticketValidator">
			<bean class="org.jasig.cas.client.validation.Cas20ServiceTicketValidator">
				<constructor-arg index="0"
value="https://cas.poc.ec:446/cas"></constructor-arg>
			</bean>
		</property>
		<property name="key" value="cas"></property>
	</bean>

	<!-- This filter handles a Single Logout Request from the CAS Server -->
	  <bean id="singleLogoutFilter" class="org.jasig.cas.client.session.SingleSignOutFilter"/>
	<!-- This filter redirects to the CAS Server to signal Single Logout should be performed -->
	  <bean id="requestSingleLogoutFilter"
class="org.springframework.security.web.authentication.logout.LogoutFilter">
			<constructor-arg value="https://cas.poc.ec:446/cas/logout"/>
			<constructor-arg>
				<bean
class="org.springframework.security.web.authentication.logout.SecurityContextLogoutHandler"/>
			</constructor-arg>
			<property name="filterProcessesUrl" value="/j_spring_cas_security_logout"/>
	  </bean>

	<security:ldap-server id="ldapServerContext" manager-dn="cn=admin,dc=poc,dc=ec" manager-
password="Password.123" url="ldap://cas-apps.poc.ec:1389" root="dc=poc,dc=ec" port="1389" />

	<security:ldap-user-service id="userService"
		server-ref="ldapServerContext"
		user-search-base="ou=usuarios, o=cas-apps, dc=poc,dc=ec"
		user-search-filter="(mail={0})"
		group-search-base="ou=roles, o=cas-apps, dc=poc,dc=ec"
		group-role-attribute="ou"
		group-search-filter="(member={0})"
		role-prefix="ROLE_" />

</beans>
```