Multilevel Multifactor Single Sign-On

Author: **Angel Linares Zapater**

Master's Degree in Information and Communication Technology Security

Authentication and Authorization Systems Area

Director: **Antoni González Ciria (ANCERT)**

Supervising Professor: **Víctor Garcia Font (UOC)**

December 31st, 2019

Title:	*Multilevel Multifactor Single Sign-On*
Author:	*Angel Linares Zapater*
Director:	*Antoni González Ciria*
Supervising professor:	*Víctor Garcia Font*
Date:	*12/2019*
Degree or program:	*Master’s Degree in Information and Communication Technology Security*
***A*rea:**	*Authentication and Authorization Systems*
Language:	*English*
Keywords:	*login attribute retrieval, multilevel authorization, multifactor authentication*

ABSTRACT:

The single sign-on mechanism (SSO) is a well-known technique to protect access to a set of resources that require prior authentication and, possibly, additional authorization. The idea behind the single sign-on procedure consists of requesting the user to provide access credentials once and, thereafter, granting access to any of the resources. Besides the practical benefits the user gets, this mechanism implies several limitations and drawbacks, like granting an "all or nothing" access to all the resources, or making a simple attack to just one user-name and password pair very dangerous because, when successful, it can provide full access to all the protected resources.

In this work we first design an SSO solution following a classical architecture consisting of a central authentication server and a user directory, dedicated to protect a set of internal applications behind a perimeter network.

We explore how to define more complex security policies for a set of resources that could allow setting multiple levels of authorization for the resources so that some users may only be able to access a subset of the resources depending on their authorization level.

Also, we test the use of multiple factors of authentication to access the resources in order to enhance their security by requesting different types of user credentials (e.g., user-name/password pairs or digital certificates).

In this solution we also demonstrate how passing information to the client applications about login procedure attributes allows implementing more complex authorization techniques to protect access to these resources. Finally, we implement a basic case of a single sign-off mechanism.

RESUM:

El mecanisme d’inici de sessió únic (SSO) és una coneguda tècnica per a protegir l’accés a un conjunt de recursos que requereixen autenticació i, possiblement, autorització addicional. La idea darrere d’un procediment d’inici de sessió únic consisteix en demanar a l’usuari que proporcioni les credencials d’accés una sola vegada però que se li concedeixi l’accés a tots els recursos a partir d’aquí. A més dels avantatges pràctics que n’obté l’usuari, aquest mecanisme comporta diverses limitacions i inconvenients, com ara un accés “tot o res” a tots els recursos o fer que un atac simple a només un nom d’usuari i contrasenya sigui molt perillós perquè, de tenir èxit, proporciona accés complet a tots els recursos protegits.

En aquest treball primer dissenyem una solució SSO tot seguint una arquitectura clàssica formada per un servidor d’autenticació central, juntament amb un directori d’usuaris, dedicat a protegir un conjunt d’aplicacions internes darrere d’una xarxa perimetral.

A continuació explorarem com definir polítiques de seguretat més complexes per a un conjunt de recursos que permetin definir diversos nivells d’autorització dels recursos de manera que alguns usuaris només puguin accedir a un subconjunt dels recursos en funció del seu nivell d’autorització.

A més, posem a prova l’ús de diversos factors d’autenticació per a accedir als recursos per millorar la seva seguretat sol·licitant diferents tipus de credencials d’usuari (per exemple, nom d’usuari/contrasenya o certificats digitals).

En aquesta solució també demostrem com l’enviament d'informació sobre atributs del procediment d'inici de sessió a les aplicacions client permet implementar tècniques d'autorització més complexes per a protegir l'accés a aquests recursos. Finalment, implementem un cas bàsic de mecanisme de desconnexió única.

RESUMEN:

El mecanismo de inicio de sesión único (SSO) es una técnica bien conocida para proteger el acceso a un conjunto de recursos que requieren autenticación y, posiblemente, autorización adicional. La idea detrás de un procedimiento de inicio de sesión único consiste en solicitar al usuario que proporcione credenciales de acceso una vez pero que otorgue acceso a todos los recursos a partir de entonces. Aparte de los beneficios prácticos que obtiene el usuario este mecanismo implica varias limitaciones y desventajas como otorgar un acceso de "todo o nada" a todos los recursos o hacer que un ataque simple a un solo nombre de usuario y contraseña sea muy peligroso porque, de tener éxito, proporciona acceso completo a todos los recursos protegidos.

En este trabajo, primero diseñamos una solución SSO siguiendo una arquitectura clásica consistente en un servidor de autenticación central, junto con un directorio de usuarios, dedicado a proteger un conjunto de aplicaciones internas detrás de una red perimetral.

Luego exploramos cómo definir políticas de seguridad más complejas para un conjunto de recursos que permitan definir múltiples niveles de autorización para estos recursos, de modo que algunos usuarios solo puedan acceder a un subconjunto de los recursos según su nivel de autorización.

También probamos el uso de múltiples factores de autenticación para acceder a los recursos con el fin de mejorar su seguridad al solicitar diferentes tipos de credenciales de usuario (por ejemplo, nombre de usuario/contraseña o certificados digitales).

En esta solución además demostramos cómo el pasar información sobre atributos del procedimiento de inicio de sesión a las aplicaciones cliente permite implementar técnicas de autorización más complejas para proteger el acceso a estos recursos. Finalmente, implementamos un caso básico de mecanismo de cierre de sesión único.

Contents

Table of Figures

1 Introduction

1.1 Context and Justification

This project is based on a proposal by **ANCERT**[1] to implement a ***single sign-on*** **(SSO)** solution capable of authenticating users to an application server by using either passwords or digital certificates. Depending on the type of credentials used to authenticate, the user will be granted access to a different set of applications among those installed in the application server. In addition to that, the applications should be able to retrieve some information about the logged-in user from the authentication server and the user repository*(*)* [2].

It is also a requirement in the proposal that the implemented solution must be based on open source projects such as **Apereo CAS** or **OpenAM**. Also, it is required that the authentication and application servers must be protected from direct access from the Internet by the use of *proxy servers(*)* located in a *perimeter network(*)*. Indeed, to enhance security it is a common architectural pattern to protect the servers in the internal network from the Internet by using an intermediate network or *demilitarized zone(*) (DMZ)*. Thus, the connections attempted from the external network to the protected servers in the internal network are performed through *proxy servers*.

As we will further analyze, this is a common scenario for corporate environments where a set of resource servers, in this case application servers, need to be accessible for the corporation users, either internal (e.g., employees) or external (e.g., customers or providers).

In a very simplistic scenario, these internal applications would require every user to authenticate separately to each one of the applications. To avoid these repetitive login steps, a *single sign-on* mechanism is implemented in such a way that, after a first successful authentication to any of the applications, the following accesses to any other application are automatically granted to the user without prompting again for the user's credentials.

One side effect of this basic SSO scheme is that the user gets access to **all** the applications protected by the authentication server. While this is sometimes the required behavior, in some scenarios not all the protected applications in the same environment should be accessible for a given user but just a smaller subset of them according to some corporate policy. In this case, an ordered set of access privileges can be defined so that a user with an access level of, say, 2 cannot use resources with access level 3, but access is effectively granted to resources at level 1 just with one login operation.

Although this **multilevel**[3] **authorization(*)** scheme for SSO provides enhanced capabilities to configure and manage access to a set of applications, it may imply new, but not always evident, consequences that we will explore in the theoretical study in this work. Furthermore, this layered approach should be not confused with **multifactor authentication(*)**, where more than one kind of credentials are requested from the user within the same authentication operation (e.g., a pair of user/password plus a one-time sent SMS code). We will explore the use of passwords and digital certificates as authentication factors.

The final result of this project is, therefore, the **implementation of an SSO solution** for a set of applications, protected with passwords and/or certificates under a multilevel access policy. Also, an exploratory theoretical analysis of the **authentication function** and its implications will be presented.

[1] Agencia Notarial de Certificación: https://www.ancert.com/

[2] Terms marked with (*) are explained in the glossary chapter.

[3] *Multilevel* and *multifactor* are sometimes used interchangeably in different contexts. We will use the term *multilevel* to refer to authorization permissions and *multifactor* to refer to types of user credentials.

2 Single Sign-On Authentication

2.1 Motivation

A common practice to access a secured resource, like a corporate application in a networked environment, is to request the user to provide some kind of credentials in other to verify that the user is who he claims to be. This process is known as **authentication**. This authentication process can be performed by the very final system being accessed. The credentials validation may consist of simply verifying a valid match of *user-name* and *password* or may consist of a more sophisticated technique like a presenting and verifying a digital certificate or entering an SMS code just created and sent for this authentication attempt.

In any case, in some scenarios this same user may have to authenticate against several systems in an internal network in order to perform her expected daily work. For example, this would be the case of a corporate employee who needs to log in to the local workstation, access the corporate e-mail server, work with one or more enterprise applications and access some external services, like *Google Drive*, *Dropbox*, etc. The user is, therefore, forced to enter several user-name/passwords, one for each system she needs to access. Moreover, when more sophisticated authentication schemes are introduced, for example by requiring a *two-factor authentication*[4], the number of options that the user has to remember grows rapidly. These repetitive, annoying actions may have the unwanted side effect of leading the user to simplify these procedures by setting, for example, the same user/password for all the systems, a practice that decreases the security of the whole system.

2.2 Basic SSO authentication

The ***Single Sign-On (SSO)*** authentication is a well known, broadly used authentication mechanism for these environments, where there exists the need to simplify multiple logging operations to several applications.

In this approach, when the user tries to access any of the applications for the first time, the final application redirects the user to a central server that performs the actual authentication on behalf of this application and also records the result of the authentication. Then, when the user tries to access a second application, the same redirection to the central server is performed but this time the authentication server is able to determine that this user has been successfully logged in and grants this user access to the second application without prompting for credentials. This process happens behind the scenes and the user experiences immediate access to the second application without having to enter again the credentials.

This basic mechanism can be implemented in several ways depending on the need to support different authentication schemes (e.g., user-names/passwords, digital certificates in the browser, cryptographic pen-drives, SMS codes, etc.) or the need for policies defining combinations of authorizations to access some of the internal resources.

Indeed, it is important to notice that, in this simplified scenario, this authentication process (verifying who the user is) actually leads to an implicit authorization to use all the applications protected by the same authentication server. However, granting the user any rights to use a given resource, the **authorization**, is a different process than authenticating the user.

Finally, a common practice is to provide the internal application with additional information about the users and their expected roles concerning the application. This allows the application to customize the user experience and even more to authorize, or not, some features in the application.

[4] The *two-factor authentication (2FA)* is a current trend in web applications, where the user is required to enter a random code received via SMS just after having entered the correct user-name and password.

2.2.1 SSO trade-offs

Selecting a Single Sign-On as an authentication mechanism for a set of resources is not free from some trade-offs. The final decision about implementing an SSO solution should, therefore, be made by considering both the benefits and the disadvantages of this authentication mechanism, especially in a scenario with multiple levels of authorization.

As a high level summary, the advantages of using an SSO mechanism are:

- **One-time login**: the user logs in only once (for a given period of time) no matter how many applications under the control of a given SSO solution will use.
- **Centralized administration**: the management of the user repository and the authentication policies are moved from each application to a central server, unifying the configuration of the application authentication and authorization processes.
- **Reduced surface attack**: because the applications do not store authentication data such as the passwords of each user that are centralized in the user's repository, the places where a leak of that information can happen are reduced.

And the disadvantages that an SSO implementation can imply are:

- **Lack of user information: under this scheme,** the applications lose the information about the users because in some basic implementations this data is stored in the user repository and the SSO server just provides the application with a binary result, i.e., whether the user was authenticated or not.
- **Unlimited access**: the user logged in one system has access to all the resources or applications under the same SSO server. This effect can be modulated with the use of access policies, like the multilevel authorization that we will explore.
- **Single point of failure**: once the authentication is centralized a very high dependency of the central authorization server availability is introduced. If the server fails, no application can be used. Fault tolerance, high availability techniques should be put in place to deal with this risk.

We will explore these challenges and alternatives when defining the final architecture and design of the solution to implement.

2.3 Available SSO solutions

The SSO approach has been in use for a long time in the industry. This means, in practical terms, that a great number of solutions are already available in the market[5].

These solutions can be divided into two broad categories:

- **commercial solutions:** developed by a given manufacturer with their own proprietary implementation choices and closed source code, but possibly optimized for a given technology or product line.
- **open source solutions**: developed by a broad community of users with the source code available and, usually, under non-restrictive licenses.

All the actual alternatives usually provide a broader set of functionality besides the SSO itself, such as access management[6], federation of systems. legacy technology compatibility, etc.

2.3.1 Open Source solutions

Among the open source solutions, the following can be considered the main choices for a new SSO project:

- **OpenAM:** This product was initially developed by *Sun Microsystems* as **OpenSSO**. After this company was acquired by *Oracle Corporation*, the project was forked by *ForgeRock.* When this company closed the source code, a new fork[(*)] of the codebase was created

[5] See, for example, the Wikipedia specific article about the list of SSO implementations: https://en.wikipedia.org/wiki/List_of_single_sign-on_implementations

[6] To avoid confusion between *authentication* and authorization, some literature, especially commercial copy, may use the terms *identity management* and *access management*, respectively.

and is now maintained by *the Open ID Platform* community[7] under the name **OpenAM**. The SSO solution is part of a broader portfolio of identity management solutions including directory services, access management, etc.

- **Keycloack:** As part of the *JBoss* portfolio, **Keycloack[8]** is the *Red Hat* solution for Identity and access management. Like other broad solutions, it provides SSO capabilities as one of the features of the product. The recent acquisition of Red Hat by IBM poses some doubts about the future roadmap of this portfolio of solutions.
- **Apereo CAS:** This solution started out as an internal solution at *Yale University.* Later on, the project joined the *Java in Administration Special Interest Group (JASIG).* In 2012, *JASIG* and the *Sakai Foundation* joined into the *Apereo Foundation*[9] where the solution is maintained as **Apereo CAS.** This solution initially implemented its own authentication protocol known as *CAS* (now in version 3.0) but later incorporated support for other standard, open protocols, like *OAuth*[10], which makes this product a flexible option for SSO implementations.

2.3.2 Commercial solutions

Most of the main software companies offer an SSO solution, usually integrated with products of broader capabilities, like access control, system federation, web services or API security, etc.

These products are usually tightly integrated with each manufacturer’s product line although some may just be rebranded, existing open source solutions. Examples are: *Microsoft Active Directory Federation Services*[11], *Oracle IAM*[12], *IBM Enterprise Identity Mapping*[13], etc.

However, many independent vendors have started to offer SSO solutions in the cloud under a *software as a service*(*) *(SaaS)* model. Two of the main cloud providers are:

- **Okta:** The provider **Okta**[14] offers access management solutions in the cloud, which also provide SSO capabilities. This offer is mainly directed towards corporate users to integrate access to all their applications in one place.
- **Auth0:** This authentication and authorization offering from **Auth0[15]** is directed to application developers but can also escalate to enterprises.

Selecting a commercial solution may depend on several factors, the main being having an already installed base of applications and infrastructure services from a given manufacturer (e.g., a Windows Server based company will probably choose Microsoft ADFS).

Nonetheless, it is not so uncommon that companies now choose to install open source based solutions but integrated with their proprietary applications in an effort to migrate to an open source model and avoid *vendor lock-in*(*).

[7] https://www.openidentityplatform.org/
[8] https://www.keycloak.org/
[9] https://www.apereo.org/
[10] https://oauth.net/
[11] https://docs.microsoft.com/en-us/windows-server/identity/active-directory-federation-services
[12] https://www.oracle.com/cloud/security/cloud-services/identity-access-cloud.html
[13] https://www.ibm.com/support/knowledgecenter/ssw_ibm_i_71/rzalv/rzalvmst.htm
[14] https://www.okta.com/
[15] https://auth0.com/

3 Project Definition

3.1 Purpose

The purpose of this project is to study the practical application of the single sign-on mechanism when a policy of access based on levels must be in place for a set of resources, in our case, an application server with several software applications installed.

By *access levels* we must understand that a user having access to, say, level 3 applications, is also granted access to applications of a lower level. Contrarily, a user given access to applications at level 2 has no access to any application of a higher level.

Specifically, the main goal is to implement a solution as described in the high-level diagram and the description below:

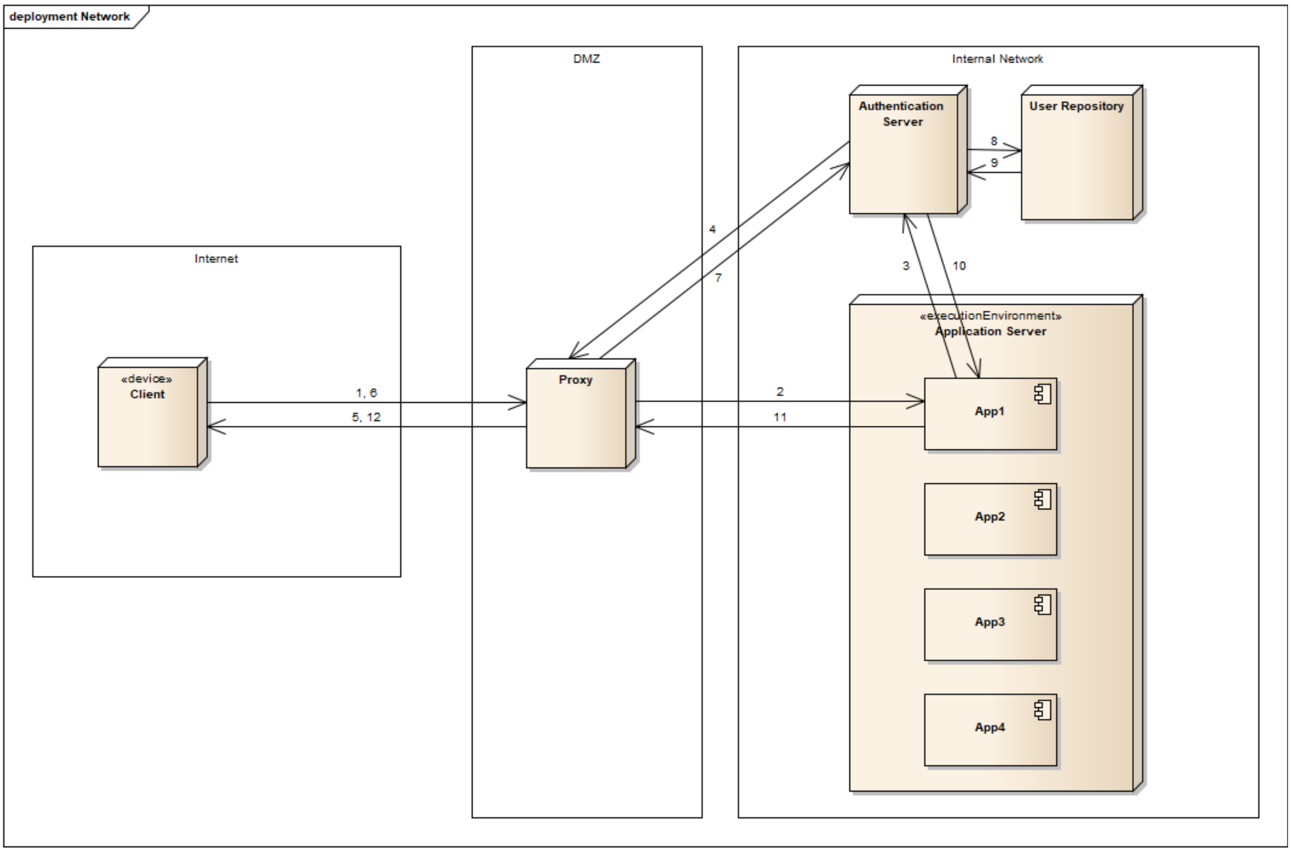


Figure 1: Solution high-level diagram

In this scenario, a user accessing from a client device some application, e.g., App1, connects to the application via a proxy (1,2) in the DMZ.

The application redirects (3) this access attempt to an authentication server. This server now requests the user to provide valid credentials, on behalf of App1 and always via a proxy (4,5).

The provided credentials (6,7) are checked by the authentication server against the user repository (8,9) and, upon successful authentication, the information about the permissions, and probably some additional user information, are sent back to the application (10) to provide the expected functionality to the user (11,12).

After the first successful login, if the user wants to access another internal application, like App2, the same flow is repeated with the exception of the authentication server prompting again the user for new access credentials (4,5,6,7).

3.2 Objectives

The main goals for this project are:

- **(O1)** To define the architecture and the detailed design of a solution capable of solving the challenges described before.
- **(O2)** To implement a solution, as specified in the design, consisting of at least:
    - **(O2.1)** a centralized authentication server along with a common user repository (see *Technological restrictions* below),
    - **(O2.2)** an application server with several custom applications installed.
    - **(O2.3)** each of these applications will have different, increasing access levels and will require different authentication techniques, specifically:
        - App1 will be accessible to any user in the directory
        - App2 will be accessible by users with a valid user-name and password
        - App3 will be accessible by users with a valid user-name and password or with a valid digital certificate
        - App4 will be accessible by users only with a valid digital certificate
        - users accessing any application will be granted SSO access to lower lever applications (e.g., access to App3 provides SSO authentication to App2 and App1, but not to App4).
- **(O3)** To allow the protected applications to retrieve some information about the user and the login operation.
- **(O4)** To implement a *demilitarized zone* (DMZ) to protect these servers from direct access from the Internet.
- **(O5)** To provide the tools to issue and verify digital certificates for those users that may be required to access some of the applications with the type of credential.

Additional goals are:

- **(O6)** To characterize, from a theoretical point of view, an *SSO policy,* to allow a more formal specification of the behavior that the final solution must implement.
- **(O7)** To analyze further implications, both theoretical and practical, about the different scenarios that a more complex authentication function may provide in comparison to just a simpler user-name/password authentication scheme (e.g., should users successfully logged in App4 access App2 according to objective O2.3?)

3.3 Other requirements and constraints

3.3.1 Solution constraints

- **(R1)** For the open source based SSO solution, we choose to use **Apereo CAS**.
- **(R2)** For the user repository, the chosen solution must implement the **LDAP** protocol.
- All other technological choices for the application server, user repository, etc. are delayed to a later phase.

3.3.2 Cultural constraints

- **(R3)** All the documentation of the project will be written in American English. Also, all the computer code, user interface and other project deliverables for the implemented solution will be in American English.

3.4 Methodology

3.4.1 Practical approach

The nature of this project is practical, the main goal being to implement a *proof of concept* for the specified SSO scenario.

Nonetheless, an analysis of the authentication and authorization processes considered as functions with credentials as inputs and permissions as outputs will take a more theoretical, qualitative approach.

The main strategy to implement the SSO solution is to select and use existing open source products for each component of the final architecture (i.e., authentication server, user repository, etc.) and configure and integrate all of them into a final product that satisfies the specified goals.

3.4.2 Project management

Given the time constraints and compulsory partial deliverables of the project, we will apply a predictive project management[(*)] method. Despite this deterministic approach, successive iterations to refine or rework some deliverables throughout the development may take place.

4 Project Plan

4.1 Project Scope

In this project we will explore different mechanisms to configure and manage these authentication/authorization challenges for an SSO solution.

On one hand, we will use both user-names/passwords and digital certificates to require some form of **multiple factor authentication**. On the other, we will also configure a **multiple level authorization** scheme such that successfully logging in to an application of a given level will grant access to applications with equal or lower levels, but not to applications with higher levels.

Finally, we will explore mechanisms for the internal applications to get some information about the successfully logged in user from the user repository.

4.1.1 Work Breakdown Structure

The scope of the project can be broken down into the following main work packages:

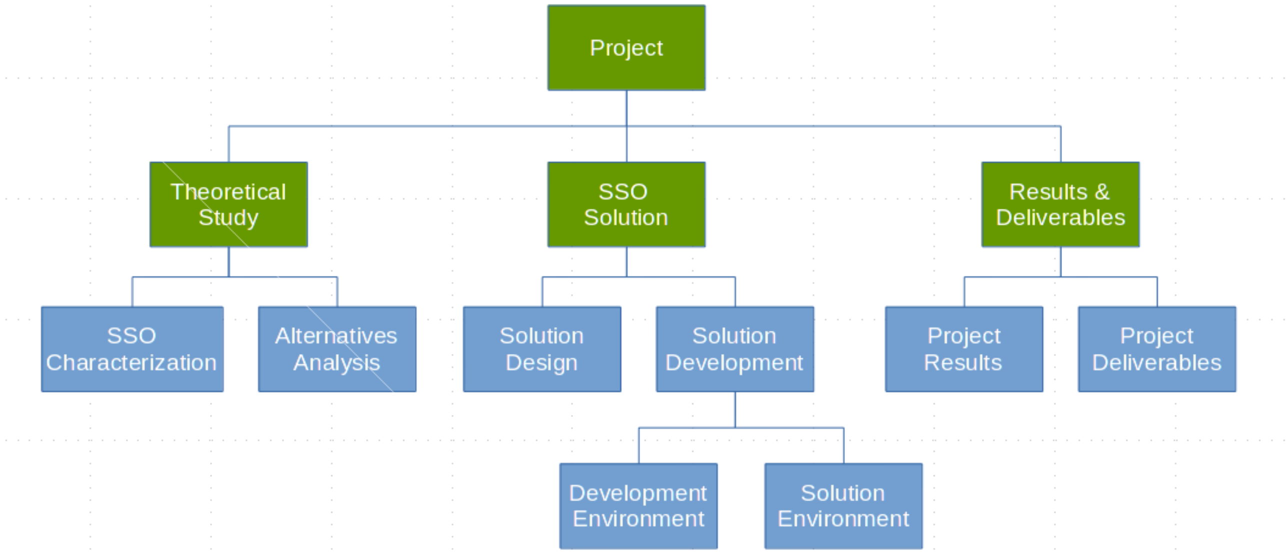


Figure 2: Work breakdown structure for the project

The work of the project will consist of a theoretical study and an actual SSO solution implementation plus a final analysis of the results obtained with this implementation. Also, included in the total work of the project are the required formal documents for the thesis submission.

The **Theoretical Study** will characterize the SSO policy specification from a general perspective and analyze the different alternatives available for the later implementation of the components of the solution.

It will consist of:

- **SSO characterization**: The authentication/authorization process can be modeled by means of what we will call *authentication and authorization functions* taking the provided credentials as inputs and the resulting authorized resources as outputs. We will study how far this model can be developed and how it applies to the specific case under discussion.
- **Alternatives Analysis**: An analysis of the different, available alternatives for the actual implementation of the practical case. We will focus on open source alternatives, as required by the original proposal, and justify the selection of each of them.

Also, a working **SSO Solution** will be implemented to test and validate the hypothesis and choices made in the theoretical study.

Two work packages are defined:

- **Solution design**: First, a detailed architecture and design for the SSO solution will be developed plus the necessary work environment and ancillary tools to develop and test its functionality. This will include both the centralized SSO part of the solution and the application server with custom applications that will use it.
- **Solution development**: Second, the actual solution will be implemented following the requirements and choices made before. The development of the solution will require the implementation of:
    - **a development environment:** to build the final applications and the SSO solution, as well as the necessary tools to provide digital certificates to the users.
    - **the actual implementation:** of a computer network with, at least, an SSO server plus a user repository, an application server with all the final applications protected with the SSO solution.

The **Results & Documentation** part of the work includes two major work packages that will be completed after the implementation and testing of the solution.

These are:

- **a discussion of the actual results** obtained from the solution implementation and possible lessons learned in the process,
- **the compulsory deliverables** for the thesis submission, namely, the thesis document or memory, and a presentation/demonstration video of the final solution.

4.1.2 Out of scope

It is not included in the scope of this project:

- using authentication credentials other than user-names and passwords pairs or digital certificates.
- using user repository implementations other than an LDAP based server.
- using client devices other than internet browsers to connect via secure HTTP protocol to the application server and to eventually provide a digital certificate.
- dealing with load balancing and high availability features on the final implementation

4.2 Project Schedule

The work of the project is distributed in a period of 16 weeks, corresponding to an academic semester. In the following high-level schedule the main activities and deliverables are shown along with the main milestones.

	Start	End	Main project activities and deliverables	Academic milestones
1	Sept 18	**Sept 22**	Project planning	
2	Sept 23	**Sept 29**		
3	Sept 30	**Oct 6**	Theoretical SSO modeling	**Oct 1st** - Project Plan
4	Oct 7	**Oct 13**	Alternatives analysis	
5	Oct 14	**Oct 20**	Architecture of the solution	
6	Oct 21	**Oct 27**	Design of the solution	
7	Oct 28	**Nov 3**	Development environment and ancillary tools	Oct 29th - PAC2
8	Nov 4	**Nov 10**	Network setup and certificates infrastructure	
9	Nov 11	**Nov 17**	SSO server implementation and user repository	
10	Nov 18	**Nov 24**	Application server implementation	
11	Nov 25	**Dec 1**	Custom applications development	Nov 26th - PAC3
12	Dec 2	**Dec 8**	Solution integration	
13	Dec 9	**Dec 15**	Solution testing and benchmarking	
14	Dec 16	**Dec 22**	Results collection	
15	Dec 23	**Jan 4**	Thesis review and Slides presentation	**Dec 31st** - Final thesis
16	Jan 5	**Jan 12**	Video recording	**Jan 7th** - Video demonstration

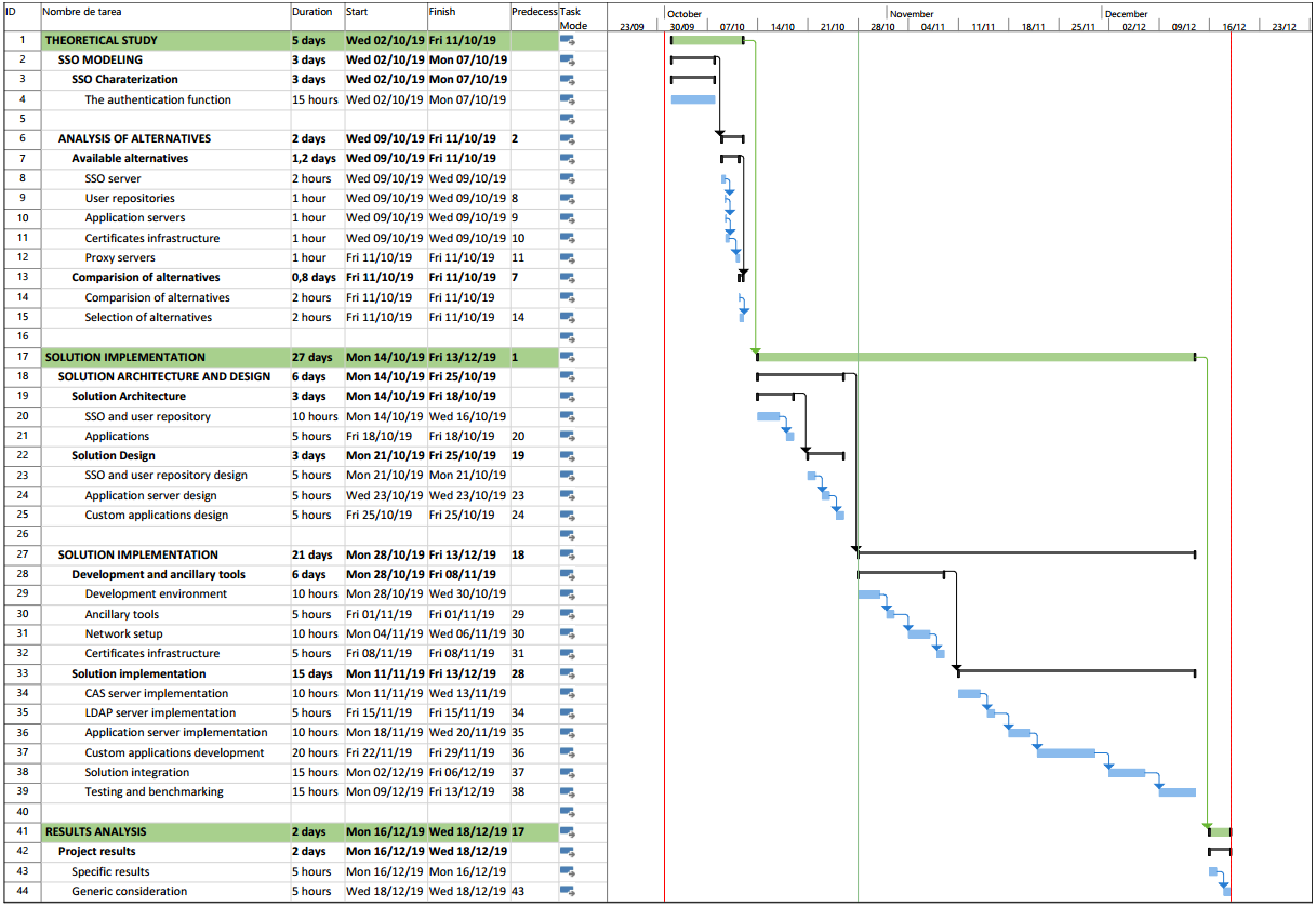
The workload is distributed evenly across the semester at a pace of 15 hours per week, tallying up a total of 225 hours of planned work, which corresponds to the academic estimation of 25 hours per credit for a 9 credits subject.

For the purpose of scheduling, the working weeks have been defined as a set of three weekly sessions of 5 work hours each.

4.2.1 Gantt diagram

The following Gantt diagram provides a more detailed view of the activities to be executed. This diagram will be further detailed as the solution design is developed and will serve as a baseline for monitoring the progress of the project.

In this diagram, the activities purely related to thesis, presentation and video preparation are not included.



ID	Nombre de tarea	Duration	Start	Finish	Predecess	Task Mode
1	**THEORETICAL STUDY**	**5 days**	**Wed 02/10/19**	**Fri 11/10/19**		
2	**SSO MODELING**	**3 days**	**Wed 02/10/19**	**Mon 07/10/19**		
3	**SSO Charaterization**	**3 days**	**Wed 02/10/19**	**Mon 07/10/19**		
4	The authentication function	15 hours	Wed 02/10/19	Mon 07/10/19		
5						
6	**ANALYSIS OF ALTERNATIVES**	**2 days**	**Wed 09/10/19**	**Fri 11/10/19**	**2**	
7	**Available alternatives**	**1,2 days**	**Wed 09/10/19**	**Fri 11/10/19**		
8	SSO server	2 hours	Wed 09/10/19	Wed 09/10/19		
9	User repositories	1 hour	Wed 09/10/19	Wed 09/10/19	8	
10	Application servers	1 hour	Wed 09/10/19	Wed 09/10/19	9	
11	Certificates infrastructure	1 hour	Wed 09/10/19	Wed 09/10/19	10	
12	Proxy servers	1 hour	Fri 11/10/19	Fri 11/10/19	11	
13	**Comparision of alternatives**	**0,8 days**	**Fri 11/10/19**	**Fri 11/10/19**	**7**	
14	Comparision of alternatives	2 hours	Fri 11/10/19	Fri 11/10/19		
15	Selection of alternatives	2 hours	Fri 11/10/19	Fri 11/10/19	14	
16						
17	**SOLUTION IMPLEMENTATION**	**27 days**	**Mon 14/10/19**	**Fri 13/12/19**	**1**	
18	**SOLUTION ARCHITECTURE AND DESIGN**	**6 days**	**Mon 14/10/19**	**Fri 25/10/19**		
19	**Solution Architecture**	**3 days**	**Mon 14/10/19**	**Fri 18/10/19**		
20	SSO and user repository	10 hours	Mon 14/10/19	Wed 16/10/19		
21	Applications	5 hours	Fri 18/10/19	Fri 18/10/19	20	
22	**Solution Design**	**3 days**	**Mon 21/10/19**	**Fri 25/10/19**	**19**	
23	SSO and user repository design	5 hours	Mon 21/10/19	Mon 21/10/19		
24	Application server design	5 hours	Wed 23/10/19	Wed 23/10/19	23	
25	Custom applications design	5 hours	Fri 25/10/19	Fri 25/10/19	24	
26						
27	**SOLUTION IMPLEMENTATION**	**21 days**	**Mon 28/10/19**	**Fri 13/12/19**	**18**	
28	**Development and ancillary tools**	**6 days**	**Mon 28/10/19**	**Fri 08/11/19**		
29	Development environment	10 hours	Mon 28/10/19	Wed 30/10/19		
30	Ancillary tools	5 hours	Fri 01/11/19	Fri 01/11/19	29	
31	Network setup	10 hours	Mon 04/11/19	Wed 06/11/19	30	
32	Certificates infrastructure	5 hours	Fri 08/11/19	Fri 08/11/19	31	
33	**Solution implementation**	**15 days**	**Mon 11/11/19**	**Fri 13/12/19**	**28**	
34	CAS server implementation	10 hours	Mon 11/11/19	Wed 13/11/19		
35	LDAP server implementation	5 hours	Fri 15/11/19	Fri 15/11/19	34	
36	Application server implementation	10 hours	Mon 18/11/19	Wed 20/11/19	35	
37	Custom applications development	20 hours	Fri 22/11/19	Fri 29/11/19	36	
38	Solution integration	15 hours	Mon 02/12/19	Fri 06/12/19	37	
39	Testing and benchmarking	15 hours	Mon 09/12/19	Fri 13/12/19	38	
40						
41	**RESULTS ANALYSIS**	**2 days**	**Mon 16/12/19**	**Wed 18/12/19**	**17**	
42	**Project results**	**2 days**	**Mon 16/12/19**	**Wed 18/12/19**		
43	Specific results	5 hours	Mon 16/12/19	Mon 16/12/19		
44	Generic consideration	5 hours	Wed 18/12/19	Wed 18/12/19	43	

Figure 3: Gantt diagram for the project

5 Theoretical Study

5.1 SSO characterization

In an SSO paradigm, a user providing some credentials may be authorized, upon successful authentication, to access the protected resources. We can see that there are two sets involved in this process, namely, the credentials provided (one or more) by any user and the resources authorized (one, more than one, or none at all). Therefore, an SSO operation implies both an authentication of the user and an authorization to some resources.

In order to characterize the SSO process, we will take the approach of starting by identifying these two operations with simple mathematical functions and then examine how far this model can be taken, by expanding the parameters and results of those functions, with regard to the specification of any given SSO policy.

5.1.1 The authentication process

The authentication process can be modeled by means of what we will call an **authentication function**. Indeed, we can see the authentication process as a mathematical function that takes the credentials provided by the user as inputs and then yields a true/false result as the output for that user authentication attempt (i.e., telling whether the user has been authenticated or not).

Thus, we can define a function **A**:

$$A:(user\ identifier\ ,credentials) \rightarrow user\ authenticated$$

$$(u,c) \rightarrow \{0,1\}$$

which takes a user *identifier* and the provided *user credentials* and returns a Boolean value to indicate whether the *user authentication* was successful or not.

To define a complete SSO policy we must be more specific and enumerate the sets of valid values for the domain of the function, both for the set of valid user identifiers and the set of corresponding credentials. Thus, for the user identifier, a syntactically correct e-mail address or a user-name formation rule may be defined, for example. And for the credentials, some string of characters following some formation rules are usually specified by the application designers. These set of definitions are specific to the concrete solution we want to implement in each real case.

Multifactor authentication

To achieve an increased level of security, it is a good practice to require more than one kind of credentials in the same authentication operation. This idea is based on the concept of *factors of authentication*[16]*.* These factors may be:

- **something the user knows**: a password, a PIN, etc.
- **something possessed only by the user**: a card, a USB key, etc.
- **something inherent only to the user**: the fingerprint, the retina image or some other biometric element

So, for the purpose of enhancing the authentication function, this enhanced case corresponds to a mathematical function of several variables. Thus, we can redefine the function **A** as:

$$A:(\text{user identifier}, \text{factor1}, \text{factor2}, \ldots) \rightarrow user\ authenticated$$

$$(u, f_{1,} f_{2,} \ldots) \rightarrow \{0,1\}$$

Again, the exact specification of the number and the type of factors is up to the application designers. This set of information and the validation rules have to be implemented in the actual authentication component of the SSO solution.

[16] Two factor authentication is an increasing trend especially in cloud based applications. Indeed, applications with higher security requirements may rely on requesting three or more authentication factors. See https://en.wikipedia.org/wiki/Authentication#Authentication_factors for a first discussion.

Multilevel authentication

As commented previously, in an SSO solution, having a binary response to an authentication challenge has the side effect of authorizing the authenticated user to access all the protected resources in the same authentication server.

To deal with this (not always desired) behavior, we can now enhance the authentication function on the side of the image. Indeed, nothing prevents us from defining a different image set of values other than the Boolean set {0,1} or {false, true}.

A first enhancement can be to define the image of the authentication function **A** in some subset of the natural numbers:

$$A:(\text{user identifier}, \text{factor1}, \text{factor2}, ...) \rightarrow \textit{user level}$$

$$(u, f_1, f_2, \ldots) \rightarrow \{0,1,2,\ldots\}$$

In this case, the result of the authentication process is a value that can be interpreted as the *user level of authentication*, where a 0 would mean "the user is not authenticated" and any other value meaning "the user is authenticated at level N".

Again, the concrete specification of the image set for a given solution is left to the application designers so the authentication server can be implemented accordingly.

5.1.2 The authorization process

At this point, it is important to note that in this definition of the authentication function there is no information about the authorized resources that a user, with a given authentication level, may be granted. We need to define which resources are accessible for a given authentication level.

Having two different function keeps separate the authentication process from the authorization one and prevents the side effect of having an implicit authorization by getting successfully authenticated, as commented in section 2.2

Access level

To specify the authorization process to access some protected resources, we can define an **authorization function** that yields, for some user and a given authorization level[17], the set of resources accessible by this user. Thus let’s define the function **B**:

$$B:(\text{user identifier}, \text{authentication level}) \rightarrow \textit{authorized resources}$$

$$(u, l) \rightarrow \{r_1, r_2, \ldots\}$$

This function gets the user’s identifier and the user’s access level and returns the set of resources effectively granted for that user to access at that level.

The set of potential resources accessible is specific to each solution and must be implemented by the SSO solution, as well as the exact combination of users, levels and resources authorized. Notice that having an authentication level of 0 does not necessarily mean a null set of resources; some resources could be authorized to any user, even anonymous users for that access level in some scenarios.

Extra authorization requirements

Under some security requirements, the authorization to access some resources may be further restricted with extra requirements besides getting an access level. This is the case of out App4, that requires the user not only having an access level of 40 but also having logged in using a digital certificate.

To model these additional parameters let’s expand the function **B** to include any additional restriction in the domain of the function:

$$B:(\text{user identifier}, \text{acess level}, \text{restriction}_{1,} \text{restriction}_{2,} \ldots) \rightarrow \textit{authorized resources}$$

$$(u, l, q_{1,} q_{2,} \ldots) \rightarrow \{r_1, r_2, \ldots\}$$

[17] We refer to the result of the authentication function as *authentication level,* but also as *authorization level* when later used in the authentication function, or just to simplify as the *access level*. In this model there is a one-to-one identification of the authentication level and the authorization level.

The number of extra restrictions, again, is up to the designers of the final solution to decide.

Finally, it is important to realize that the complexity of the actual implementation needed to check some of these restrictions, as well as their potential variety, is what makes the authentication and authorization processes difficult to formalize as a generic framework.

5.1.3 SSO Policies

For our purposes, we can now model the complete SSO process as a combination of these two functions, that is:

$$P:(\mathrm{user}, \mathrm{factor1}, \mathrm{factor2}, \ldots) \rightarrow \mathit{authorized\ resources}$$

$$(u,(u,f_1,f_2,\ldots),(q_1,q_2,\ldots)) \rightarrow \{r_1,r_2,\ldots\}$$

We can define this "combination of functions" as an **SSO policy**, in the sense that it specifies for a given user and the provided authentication factors which resources are actually allowed to be accessed when the authentication of that user is successful.

Internally, the policy implementation relies on the intermediate "access level" obtained from the authentication function and, taking into account any other authorization restrictions, yields the final list of resources.

Therefore, to specify the SSO policy, for any given implementation of an SSO scheme, will need to define:

- The set of valid user identifiers
- The set of authentication factors allowed for each user
- A table of valid user identifier/credential combinations and their specific validation rules plus the access level obtained when authenticated
- The set of protected resources
- A table of resource/access level required to grant permissions[18], plus any other extra restriction imposed to any resource with the corresponding validation rule.

We will use this theoretical model to define the multilevel multifactor SSO policy for the project in section 6.2.5 (page 28).

5.2 Analysis of Alternative Products

5.2.1 Available alternatives

SSO server

As discussed in section 2.3.1, many SSO products are available in the market to set up and operate an SSO solution. Among the open source solutions, the two most prominent, that we take into consideration, are **Apereo CAS** and **OpenAM**.

User repositories

The need for a central repository of user information in a software application has led to a number of standardized solutions, commonly referred to as *directories*, and a set of protocols to interact with these tools. A directory may not only include user information but also may store information about resources in the network (e.g., shared printers, meeting rooms), etc.

Several proprietary directories have been developed by different vendors, e.g. **Microsoft Active Directory**, but on the open source side, the Lightweight Directory Access Protocol, LDAP has become the *de facto* industry standard for directory system access. Also, the recent trend in cloud computing has increased the offering of *SaaS* solutions for user identification and management, like **Amazon IAM**, for example.

These are the main alternatives for an open source LDAP server:

[18] Note that in this model, the user gets "full permissions" for a given resource. To specify more fine-grained permissions (e.g only to read, write, etc.) we will need other enhanced models. See Future work on page 52.

- **OpenLDAP[19]**: an open source implementation for the LDAP protocol base directory and tools.
- **389 Directory Server[20]:** another LDAP directory server for Linux machines with enterprise-class features.
- **ApacheDS[21]:** is an LDAP and Kerberos server written in Java and maintained at the Apache Foundation. As a Java application can be easily embedded in Java applications allowing them to use the LDAP functionality without the need for an external LDAP. It also features interesting options like stored procedures to help administer the directory in a more efficient way.
- **OpenDJ[22]:** is another project that started out as a fork of the *Sun Microsystems* OpenDS. It can be seen as another member of the family of OpenAM.

To manage the LDAP repositories there is a great number of available client tools, among others:

- **JXplore[23]:** a Java tool capable of connecting to an LDAP server to manage the directory schema and entries (e.g.. groups of user, user information, etc.)
- **Apache Directory Studio[24]:** is a directory tooling platform particularly designed to manage the ApacheDS server, written in Java as an Eclipse tool.

Application servers

In order to run applications in a Java Enterprise Edition (Java2 EE), now know as **Jakarta EE[25]** at the Eclipse Foundation, an execution environment or container is required. For the applications needed in the SSO solution to develop, only the *servlet*(*) specification[26] is needed.

The main open source alternatives for a servlet container are:

- **Apache Tomcat[27]:** a web server and servlet container, developed in Java, widely used and proven by the Java community. Alternatively, when a full Jakarta EE application server is needed another option is Apache Geronimo[28].
- **Eclipse Jetty[29]:** is a servlet container from the Eclipse Foundation especially well suited to be embedded in other applications which, this way, can deploy a servlet container without the need of an external server.
- **Redhat Wildfly[30]:** formerly known as JBoss AS, is a light-weight but full-featured Jakarta EE 8 application server, thus providing all the specification capabilities.

Proxy

The proxy component in the DMZ has the purpose of filtering all the incoming requests from client devices and forwarding to the corresponding application.

This component can also be assigned responsibilities for load balancing and fault tolerance. None of these features are requirements for our SSO solution.

The main open source products for implementing a proxy server are:

- **HAproxy[31]:** a specialized solution for high availability, load balancing and proxying HTTP request.
- **Apache httpd:** the well known open source web server that also has proxying capabilities. This option can be interesting in deployments where *Apache httpd* is already installed or to take advantage of previous training with the product and Java ecosystem in some corporations.

[19] http://www.openldap.org/
[20] http://www.port389.org/
[21] http://directory.apache.org/apacheds/
[22] https://forgerock.github.io/opendj-community-edition/
[23] http://jxplorer.org/
[24] http://directory.apache.org/studio/
[25] https://jakarta.ee/
[26] https://javaee.github.io/servlet-spec/
[27] https://tomcat.apache.org/
[28] http://geronimo.apache.org/
[29] https://www.eclipse.org/jetty/
[30] https://wildfly.org/
[31] www.haproxy.org

- **Nginx:** another well known, very fast web server that also includes proxy server features.

Digital certificates

The SSO solution requires digital certificates for two main purposes. The first is to allow users to provide enhanced credentials in the SSO dialog with respect to the pair user-name/password. The second is to secure the communications between the user's client device and the servers in the internal network (in fact, irrespective of any actual specific application needs, at lease the password has to be protected when traversing the network!).

The current standard for digital certificates in the web and internet applications is the X.509 specification[32]. The most common tools for creating digital certificates are:

- **OpenSSL[33]:** a suite of tools to work with the SSL/TLS protocols and also to create and manage digital certificates.
- **LibreSSL[34]:** an alternative project, forked from OpenSSL, to modernize and further secure the toolkit.
- **Java keytool[35]:** the Java runtime provides a tool to manage X.509 certificates and certificate databases (a.k.a. *keystores*) for Java applications.

5.2.2 Comparison of alternatives

SSO servers

Both products, Apereo CAS and OpenAM, provide similar functionality in terms of supported authentication and authorization protocols, SSO features, encrypted communications, etc. Apereo CAS provides its own protocol (now in version CAS 3.0) but also supports other more standardized protocols, like OAuth 2.0 or WS Federation, so there is no special reason to select one over the others.

We choose Apereo CAS due to what seems a better (or at least broader) documentation and wider support from stable organizations like Universities that are actually using the product in real deployments.

Also, for long term projects, the recurrent changes in leadership and the multiple forks of the OpenAM code base contrast with the steady level of support for Apereo CAS, either under the JAAS group or the Apereo Foundation. Moreover, OpenAM is distributed under CDDL[(*)], a less common license than Apereo CAS, under Apache 2.0 license. This latter license model has a broader acceptance than the former, a choice factor that can be decisive in some projects.

User repositories

In the realm of open source LDAP servers, OpenLDAP is the best known and popular implementation. Nonetheless, the set of tools to administer an OpenLDAP server is mainly console-based and can pose some challenges to less experienced network administrators than other choices like ApacheDS. This later solution not only provides the LDAP server but also accompanies this product with a number of tools, like Apache Directory Studio, which can ease the maintenance of the directory structure and contents.

In this project, the LDAP server is an intentionally-separated component from the SSO server, so we can choose a more straightforward (and not Java based) product like OpenLDAP. Should we had to administer, especially in a programmatic way, a heavily used and evolving user directory in a Java environment, ApacheDS could be an appropriate choice.

Application servers

As the SSO server chosen requires a servlet container but does not need a complete Jakarta EE application server, using Apache Tomcat seems the more common option for this scenario. It is a well known product with broad support in the Java community. Indeed, the installation of Apereo CAS can be implemented either by using an external Tomcat server or by embedding a Tomcat server in the very same SSO application.

[32] https://tools.ietf.org/html/rfc5280
[33] https://www.openssl.org/
[34] http://www.libressl.org/
[35] https://docs.oracle.com/javase/8/docs/technotes/tools/unix/keytool.html

Is it important to note, however, that Apereo CAS makes no special assumption about the servlet container used to run the SSO server. By having selected such an agnostic SSO solution, a future migration to, say, Wildfly can be undertaken with minimal effort, either migrating from an external Tomcat or from an embedded one.

Application frameworks

The Apereo CAS product is, indeed, a Spring Boot[36] Java application itself. Although the knowledge of this Java framework is not needed for the deployment of the SSO server, in this specific project we choose to develop the simple, custom applications also as Spring-based Java applications. The Spring suite of frameworks will provide convenient tools to secure and connect these applications against any SSO server, and obviously against the Apereo CAS server, being both sides built on top of the Spring libraries.

Proxies

Among the three proxy solutions seen in the previous section, namely Apache httpd, Nginx and HAproxy, the latter is a highly specialized solution while the other two are mainly web servers with additional proxy capabilities.

Given that we are not interested in testing high-availability and load balancing features in this specific project, using a web server can be an acceptable solution for the proxy component in the architecture. And, as the rest of the main components selected before gravitate around the Java ecosystem and the Apache Foundation solutions, we choose to select Apache httpd as the proxy in the DMZ[37].

However, there is the risk that in the final implementation the requirement of end-to-end encryption could not work completely well with this product in the DMZ. We can keep the alternative of HAproxy as a backup solution to change this component implementation if required by some unforeseen issues while dealing with digital certificates and encrypted traffic.

Digital certificates

For the creation of X.509 digital certificates we will use the Java *keytool* when convenient (e.g., to secure the Java based Tomcat server), but we will use OpenSSL when a generic certificate is needed (e.g., providing a certificate to a user to log in some application).

Using LibreSSL, instead of OpenSSL, is a choice of preferences mainly due to licensing reasons. For our project, there are no further implications and we will stick to OpenSSL.

[36] https://spring.io/projects/spring-boot

[37] In a more complex project, this option also has the benefits of serving the static web pages of the solution from this server, saving the application server resources for heavier application workloads. Nevertheless, Nginx could also perform the same role in the architecture.

6 Solution Architecture and Design

6.1 Solution Architecture

As hinted in the high-level diagram on page 12, we can identify three main subsystems for the solution architecture. On the one hand, there are the actual resources to be protected in the internal network plus the proxy components in the perimeter network devoted to isolating these resources from the Internet. And, on the other hand, there is an additional, accompanying subsystem with two new components that implement the SSO features as specified for the project.

The following diagram represents a two-layered architecture, where the SSO subsystem enhances the protected resources in the internal network with the new SSO functionality:

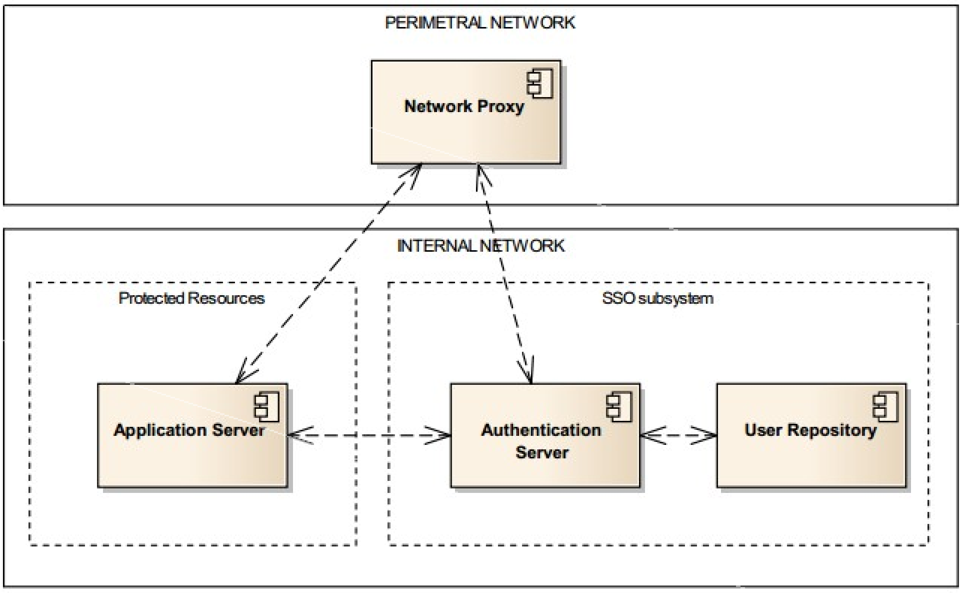


Figure 4: Component Architecture of the SSO solution

6.1.1 Protected Resources

Custom Applications

We identify the set of protected resources in this project as a number of *ad-hoc* client applications accessible from the external network under the SSO policy defined in the objectives of the project (p. 13).

These applications are simple enough just to allow us to test that the security requirements are met by the final solution. Thus, these simple applications will be only capable of displaying information to identify the actual resource being accessed and providing some contextual information about the security applied in the connection.

Obviously, in any real development, the resources in the internal network may vary in every specific case. In this project, however, we will use an application server to host all these applications.

Application server

All these "protected" applications will be running in the same execution environment, an application server (just a servlet container, indeed). This approach resembles the typical Java-

Enterprise-based installation in a corporate environment and allows sharing some security configurations, set in one place, for all the applications.

Nonetheless, the authentication dialog with the SSO server is expected to be performed by each application on its own without any intervention from the application server.

Finally, it is important to note that in a real deployment, the internal network could also host some other servers and services, like database servers, etc. that are not included in this project due to the simplicity of the applications tested, just focused on the SSO proof of concept.

6.1.2 DMZ

Perimeter Network

Other extra components to account for in the final solution are the proxies in the DMZ, as required by the proposal. This is an extra layer in the architecture that is not necessarily linked to the SSO functionality itself. However, it is a usual practice in any corporate environment accessible from the Internet to implement such type of isolation between the external network and the internal network.

Indeed, having this intermediate component in the architecture can pose some additional challenges to the SSO implementation. This is especially relevant when dealing with secure connections from the applications. which we intend to be secured end-to-end, that is, from the user's client device to both the application and SSO servers, and could be affected by the presence of this intermediate element.

6.1.3 SSO subsystem

Authentication server

We define a unique, central user authentication server for all the resources in the internal network. This centralized option provides a lot of benefits with respect to other possible architectural patterns, like some form of a distributed authentication solution. It also simplifies the selection of the *commercial off the shelf*(*) product from the actual market offering for SSO servers.

Note that the authentication server also requires access to the Internet via the proxies in the DMZ because the actual authentication dialog with the user's client (e.g., the user's browser or a user's mobile app) is delegated to the SSO server and not performed by the applications themselves when a new authentication is required.

User repository

Although some form of user repository functionality can be implemented by the SSO server itself, it is a good practice to separate, at least at an architectural level, this concern (i.e., managing the set of users) from the actual SSO functionality (to authenticate the user and keep track of the valid sessions). In practice, indeed, the user management capabilities provided by the main available SSO components can be very limited with respect to a proper user repository product.

By defining a different sub-component for the user repository we gain a lot of flexibility for the final solution, but at the cost of greater workload to install and configure the new component. However having a separated component will allow us to further test one of the requirements for the solution, namely, to let the applications receive some information about the logged in user from the SSO server.

As a final note, the user repository could also be one of the existing servers in the internal network dedicated to providing other identity management services to other network components. In this architecture, no direct access from the applications to the user repository is allowed.

6.2 Solution Design

6.2.1 The Apps

The "secured" custom applications are very simple applications that contain the minimum functionality to test the SSO solution and interact with the authentication server to retrieve and display some information about the connection made and the privileges granted to the user.

Basic use cases

A set of very simple use cases for these applications can be specified with the following scenarios:

APP	Show App information
Actor	user
Preconditions	the user is logged in the App
Scenario	1. the application shows information about: - the user details received from the SSO server - the available details about the connection
Alternatives	1.1.a if the user is not logged execute the *Log in* use case 1.2.a if the user is already logged, according to the SSO server, continue.
Postconditions	none

APP	Log in
Actor	user
Preconditions	the user is not logged in the App
Scenario	1. the user enters the required credentials 2. the SSO server validates the credentials for the user and app 3. the SSO server keeps track of the logged session for this user
Alternatives	2.1.a if the authentication fails the App shows an error message and restarts
Postconditions	the user is registered by the SSO server as logged in the App

APP	Log out
Actor	user
Preconditions	the user is logged in the App
Scenario	1. the user selects the logout option 2. the SSO server clears the session
Alternatives	
Postconditions	the user is no longer registered in the SSO server as logged-in

It is important to note that step 3 in the *Log In* use case, "to keep track of the logged in session", may be implemented in very different ways depending on the SSO server used in the solution.

Furthermore, not all the SSO servers support a "multiple log out from all the applications" feature when the user logs out from one application. This kind of extra features, far from being required upfront in a project, are usually left upon the set of features provided by the chosen SSO product.

6.2.2 The APP Server

The application server will be an Apache Tomcat *servlet* container as selected above (p. Error: Reference source not found). This choice is appropriate in his scenario where the SSO clients are Spring applications and can leverage the needed tools from the Java Enterprise ecosystem. Other different scenarios could be designed to test different kinds of clients against the same Apereo CAS server, depending on the needs of the final solution. Examples of these clients are Python applications, Android or iOS apps, etc.

In any case, the presence of the servlet container should not interfere with the applications interacting directly with the SSO server.

6.2.3 The CAS Server

The Apereo CAS server, as an application itself, happens to be a Spring Boot application that, therefore, requires a *servlet* container where to run. We will use a second Tomcat *servlet* container for the CAS application, in a different machine that the Tomcat server where we will install the custom client applications[38].

This CAS server will be configured to connect to an external LDAP server. The user repository will contain the necessary information to authenticate the user and, in our multilevel authentication scenario, return the authentication level and the corresponding permissions to access some of the applications depending on the authentication granted.

6.2.4 The LDAP Server

The user repository in this solution is an LDAP server, specifically an OpenLDAP server.

In a real environment this LDAP server could be already deployed in the network, or as in our case be deployed only for SSO purposes. Having the LDAP functionality separated from the SSO server allows this flexibility in implementing an SSO solution.

This LDAP server will implement the user repository containing not only the directory information but also relevant meta-data about the SSO policy for the set of applications.

6.2.5 The SSO policy

The specification of the SSO policy should correspond to objective **O2.3** of the project as defined on page 13.

Authentication

We can first define the authentication function, by extension, including the allowed credential factors for each user:

User	credential type	authentication level
user10	password	level 10
user20	password	level 20
user30	password	level 30
user31	certificate	level 30
user40	password	level 40
user41	certificate	level 40

For each enumerated user, if the authentication using the credential type shown is valid, the SSO grants the authentication level indicated in the third column.

To accept a given credential as valid, we specify that:

[38] The option to build the CAS server with a Tomcat server embedded is also possible with Apereo CAS.

- for passwords, we will require them to match those defined in the corresponding user directory entry.
- for certificates, we will require the certificate to be signed by out CA authority$^{(*)}$ and the user-name included in the certificate to be found in the user directory.

Authorization

We can now specify the authentication level required to access each application, as well as any other additional restriction imposed in this project (like requiring to log in using some type of credential).

App	credential type	access level
App1	any	level 10
App2	any	level 20
App3	any	level 30
App4	certificate	level 40

We have to keep in mind that the semantics of this "access levels" implies that a user having a greater authentication level can access applications of a lower level:

The resulting policy

Combining the two tables and the rule of increasing authentication levels we can obtain the complete SSO policy for all users, provided that a user gets correctly authenticated using the kind of credentials required. Explicitly:

User	applications authorized after a single sign-on
user10	App1
user20	App1, App2
user30	App1, App2, App3
user31	App1, App2, App3
user40	App1, App2, App3
user41	App1, App2, App3, App4

This SSO policy, for this specific set of users and apps, should match the objectives set initially for the project.

Some remarks to be noticed are:

- all users can access App1, if correctly logged in,
- user20 gets access to App1 when logging in to App2 as expected from an SSO scenario,
- user30 and user31, are required to use different credential types but obtain the same effective authorization[39]
- user40, although getting an access level of 40, cannot access App4 due to the additional requirement to access using a certificate. However, this user gets access to the rest of the applications where the access level is sufficient.

In a real context, the complete SSO requirements would probably be given in the form of the final table and we should have to work backward to find a combination of access levels and credentials to match the expected policy.

[39] A hypothetical *user32,* able to authenticate either by password OR certificate, can be tested in the final solution. Also, requiring some *user33* to use a password AND a certificate to log in could be defined.

6.2.6 Ancillary components

X.509 certificate infrastructure

To secure the communication between components in the solution we will create and distribute X.509 digital certificates. These certificates will be created either using the Java tools (for Java applications) or using the OpenSSL suite for more general purposes.

A certificate to identify each server is created, plus a certificate for the users who need to log in using this kind of credential to any App that requires such an authentication mechanism.

6.2.7 The network setup

The development environment

All the systems deployed in the SSO solution will be created as virtual machines running on a development machine on the network **192.168.1.0/24.**

This network will act as the "external network" of the project and provide access to the Internet, via the proxies in the "perimeter network", to the "internal network" servers.

The perimeter and internal networks

To build the internal network for the solution we select a DNS name (**mistic.lan**) for the machines and a TCP/IP class C address range (**192.168.2.0/24**) different from the external network (**192.168.1.0/24**).

The following diagram summarizes the network setup for the project:

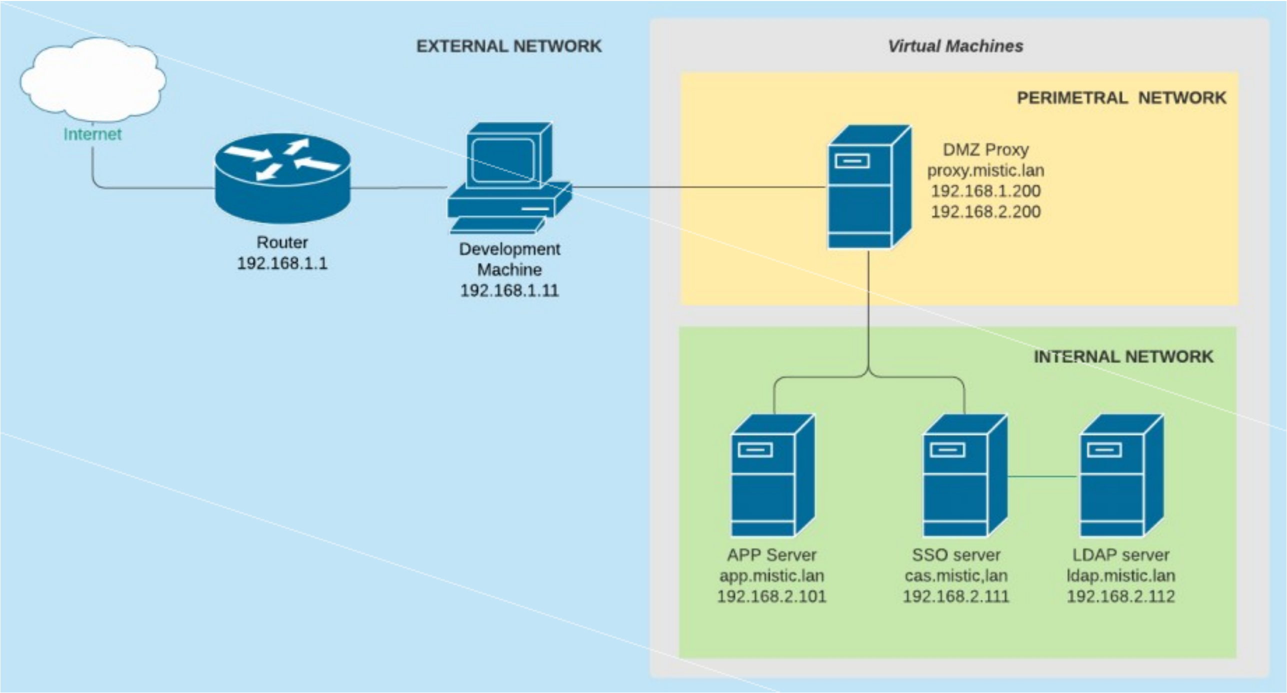


Figure 5: Network diagram for the project

Notice the proxy machine **proxy.mistic.lan** having two internet addresses, one for each network adapter, and acting therefore as a *TCP/IP bridge* between both networks.

7 Solution Implementation

7.1 The development environment

7.1.1 Host environment

The local network at range 192.168.1.x represents the "external network" of the solution, which provides access to the Internet via the router 192.168.1.1. In this host environment, where the guest virtual environment will be built, the machine at IP address 192.168.1.11 will act as a client device accessing the internal applications from this external network (or from the internet, for that matter).

This machine will also work as an application development system, and as a systems administration machine to create virtual machines, manage and deploy digital certificates, etc.

In this machine, we can set the **/etc/hosts** file to have the DNS names needed to access the internal network, which by design is just the PROXY server named **www.mistic.lan**.

```
127.0.0.1 localhost

192.168.1.200 www.mistic.lan
```

7.1.2 Virtual environment

To provision the "internal network" machines we create a dedicated **Oracle Virtualbox 6.0.x** environment in the development machine. Then we prepare a set of Virtualbox-based virtual servers in a different network range.

All these machines have their network adapters in the internal address range 192.168.2.x except for the DMZ Proxy that has also an additional, second adapter on the internal network 192.168.1.x, as detailed in *The network setup* diagram (p. 30). The DMZ Proxy, with interfaces both in the external and the internal networks, provides bridging capabilities between both networks, allowing all the internal network machines to connect to external network clients but in a supervised, protected way.

Debian 10 virtual machine base image

We create, for convenience, a virtual machine **server100** (192.168.1.100) with **Debian 10 x64** as a base image to clone later the actual servers for the solution. In this base machine, we install the most basic software packages with no graphical interface using the *Expert Installation* mode (namely, the S*tandard System Utilities* and the *SSH server* packages from the Debian installation menu). We upgrade the system and install some tools for better administration (like *Midnight Commander*, *Emacs*, etc.) and change the default configuration of the SSH server to allow the root user[40] to log in:

```
$ ssh root@192.168.1.100
root@192.168.1.100's password:
Linux Debian10base 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u1 (2019-09-20) x86_64
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.

root@Debian10base:~#  uname -a
Linux Debian10base 4.19.0-6-amd64 #1 SMP Debian 4.19.67-2+deb10u1 (2019-09-20) x86_64 GNU/Linux
```

This base image has only one network interface, at address 192.168.1.100, which will allow us to connect to this machine to further customize it after each cloning operation.

[40] Allowing the *root* user to *ssh* with user-name and password into a system is considered a bad security practice. We set this option for easier development and management of this installation. In a production environment, however, more secure configurations should be considered.

Development using the internal network machines

In this setup, reaching the internal network machines from the development machine for testing purposes may require to add an additional network route[41] in the development PC. To be able to reach and connect to the internal servers, we can add the additional route:

```
# sudo ip route add 192.168.2.0/24 via 192.168.1.200
```

This will allow us to connect from the development machine at IP 192.168.1.11 to the machines in the range 192.168.2.0/24. To remove the temporary route, just issue the command:

```
# sudo ip route del 192.168.2.0/24
```

We can, also, set for temporary development and management purposes some other internal network machines names in the /etc/hosts file:

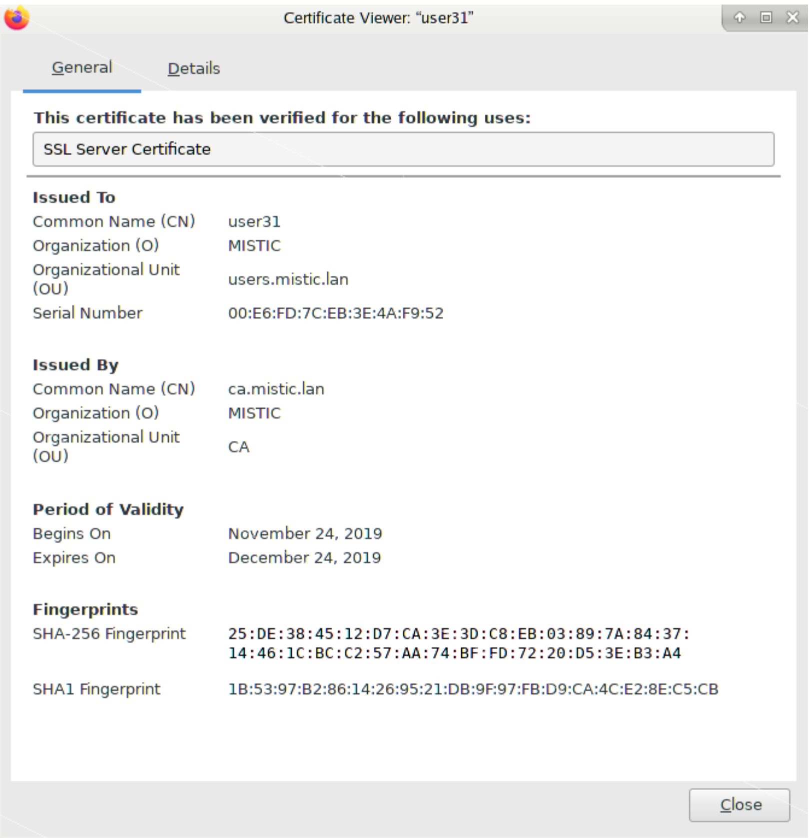
```
192.168.2.200 proxy
192.168.2.101 app
192.168.2.111 cas
192.168.2.112 ldap
```

7.1.3 Certificates infrastructure

We create all the needed digital certificates for the servers to authenticate themselves and for the users who are required to use this kind of credential instead of user-name and password (see Creating the certificates infrastructure for MISTIC.lan p. 56).

Certificates at the user's browser

For those users who need to authenticate by presenting a digital certificate, the corresponding certificate has to be installed in their browsers. In the Mozilla Firefox browser, for example, a user's certificate can be imported in the *Certificate Manager* found in the *Preferences/Privacy & Security/View Certificates* section:



This certificate has been verified for the following uses:

SSL Server Certificate

Issued To

Common Name (CN)	user31
Organization (O)	MISTIC
Organizational Unit (OU)	users.mistic.lan
Serial Number	00:E6:FD:7C:EB:3E:4A:F9:52

Issued By

Common Name (CN)	ca.mistic.lan
Organization (O)	MISTIC
Organizational Unit (OU)	CA

Period of Validity

Begins On	November 24, 2019
Expires On	December 24, 2019

Fingerprints

SHA-256 Fingerprint	25:DE:38:45:12:D7:CA:3E:3D:C8:EB:03:89:7A:84:37: 14:46:1C:BC:C2:57:AA:74:BF:FD:72:20:D5:3E:B3:A4
SHA1 Fingerprint	1B:53:97:B2:86:14:26:95:21:DB:9F:97:FB:D9:CA:4C:E2:8E:C5:CB

[41] This type of convenient network configurations, useful at development and deployment time, for example to ssh into the internal servers, are not required in the production environment and should be removed and replaced with more secure administration alternatives when needed.

CA root certificate at the user’s browser

The development machine browser does not include the root certificate created for our CA. We need to install this root certificate if we want to avoid annoying messages from the browser warning that the servers visited in the internal network are presenting certificates not recognized by any public CA. In a real environment, where the final certificates are usually signed by a globally trusted CA, this warning will not show when the client connects to the production servers.

Certificates for the PROXY and LDAP server

We create the certificates for both the PROXY and the LDAP servers (see Creating MISTIC-signed X.509 certificates , p. 57).

After provisioning these two servers we will install the corresponding public and private key pair in each server.

```
# scp www.mistic.lan.{crt,key} root@proxy:/srv/www/mistic
root@proxy.mistic.lan's password:
www.mistic.lan.crt                                      100% 1996     2.8MB/s   00:00
www.mistic.lan.key                                      100% 3272     7.4MB/s   00:00

# scp ldap.mistic.lan.{crt,key} root@ldap:/etc/ldap/certs
root@ldap.mistic.lan's password:
ldap.mistic.lan.crt                                     100% 1996     2.8MB/s   00:00
ldap.mistic.lan.key                                     100% 3272     7.4MB/s   00:00
```

and also will copy the root CA certificate in the Proxy and LDAP servers for later validation of the certificates received from clients:

```
# scp ../certs/ca.mistic.lan.crt root@proxy:/srv/www/mistic
root@ldap's password:
ca.mistic.lan.crt                                        100% 2126    2.1MB/s   00:00

# scp ../certs/ca.mistic.lan.crt root@ldap:/etc/ldap/certs
root@ldap's password:
ca.mistic.lan.crt                                        100% 2126    2.1MB/s   00:00
```

Certificates for the APP and CAS server

For the servers running Tomcat servlet containers, namely APP and CAS, we create their certificates using the Java keytool utility instead of using certificates created with openssl (see Creating Java keystores for the Tomcat servers p. 58).

These Java keystores will be installed in the **/opt/tomcat/conf** directory at the corresponding Tomcat server machine (APP or CAS):

```
$ scp app.mistic.lan.jks root@app:/opt/tomcat/conf
root@app's password:
app.mistic.lan.jks                                      100% 2615     3.9MB/s   00:00
```

Also, in the case of the CAS server, the keystore for the CA certificate has to be copied:

```
$ scp ca.mistic.lan.jks root@cas.mistic.lan:/opt/tomcat/conf
root@cas.mistic.lan's password:
ca.mistic.lan.jks                                       100% 1834     1.7MB/s   00:00
```

This choice of tool to create certificates eases the configuration of the Tomcat servlet containers. Nevertheless, Tomcat supports the use of other formats of certificate containers like PKCS#12 via the JSSE facility[42].

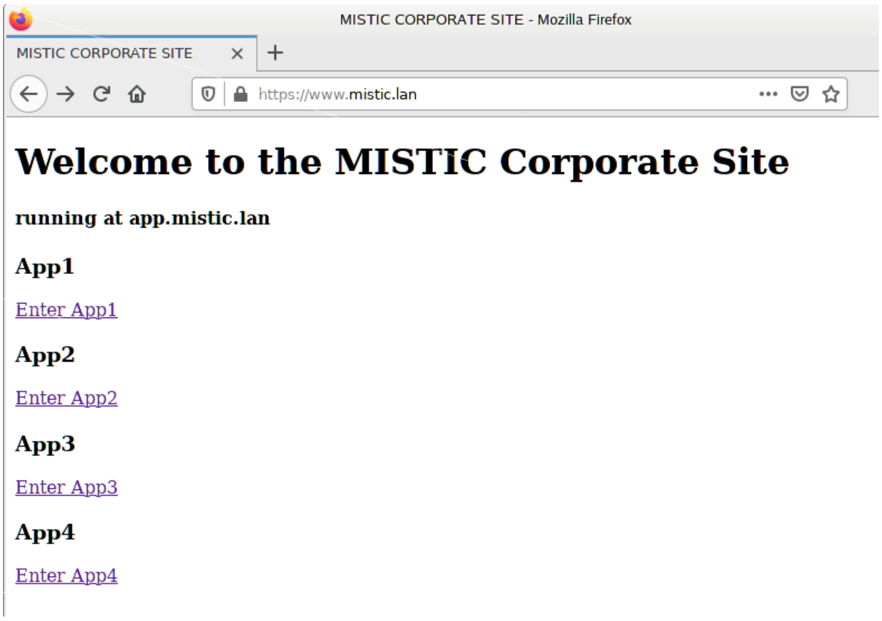
It is important to note that, using *keytool* in this way, implies that these certificates are not signed by the MISTIC CA authority but self-signed during their creation process. For internal connections, there is no need to validate these certificates as issued by the MISTIC CA. In fact, the communications with external users on the Internet are handled by the proxy server in the DMZ which checks against the CA, but the internal connections originated at the PROXY and the APP server can be secured with self-signed certificates.

[42] See https://tomcat.apache.org/tomcat-8.0-doc/ssl-howto.html

7.1.4 Custom applications

The Home Page

To have all the applications grouped for the external users in one place, we create a simple home web page and install it in the APP server Tomcat’s ROOT directory just replacing the default Tomcat home web page[43]. This way, users connecting to the main URL https://www.mistic.lan will get this greeting page (after convenient redirection by the DMZ proxy) with the menu of the available applications:



Welcome to the MISTIC Corporate Site

running at app.mistic.lan

App1

Enter App1

App2

Enter App2

App3

Enter App3

App4

Enter App4

Developing the CAS client applications

According to the project objectives for the applications (see Objectives, p. 13) these applications must implement some kind of log in procedure and session management. We create them as very simple Spring Boot applications (see Applications source code, p. 77) and configure them as CAS clients.

Securing the application as CAS clients

The custom applications require the user to provide a user-name and password or a digital certificate that will be validated in the single sign-on process. To secure these applications we add the Spring Security CAS client module[44] dependency in the application *pom.xml* file:

```
       <dependency>
          <groupId>org.jasig.cas.client</groupId>
          <artifactId>cas-client-support-springboot</artifactId>
          <version>3.6.1</version>
       </dependency>
```

Then we add the necessary code to configure[45] the connection between the application, as a CAS client, and the actual CAS server.

Finally, in the **application.properties** file, we add the entries:

```
cas.server-url-prefix=https://www.mistic.lan/cas
cas.server-login-url=https://www.mistic.lan/cas/login
cas.client-host-url=https://www.mistic.lan
```

[43] Installing this page in the ROOT directory or in a dedicated subdirectory is a matter of preference. When installed as an application in a subdirectory, like *home*, users will have to use longer *URLs: https://www.mistic.lan/home*

[44] This process sometimes is referred to as *casifying* the application. See https://docs.spring.io/spring-security/site/docs/3.0.x/reference/cas.html

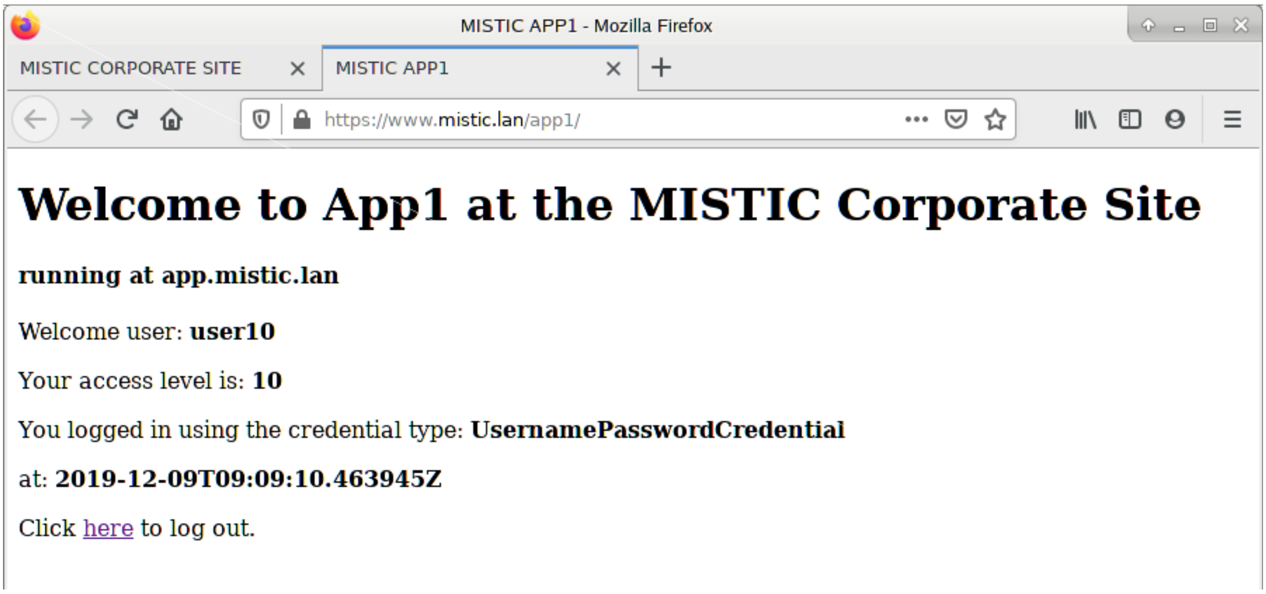
[45] See https://github.com/apereo/java-cas-client, at section `Spring Boot AutoConfiguration`.

These are the URLs that the CAS client will issue and the DMZ proxy will translate into internal network addresses.

Note that the destination URLs are formed as seen by the CAS client (i.e. the user's browser) from outside the internal network, that is, using the name **www.mistic.lan** (or 192.168.1.200 IP address) instead of names like **proxy.mistic.lan** or **cas.mistic.lan** (or any reference to the internal 192.168.2.x addresses). From the point of view of any user in the external network, the internal addresses (or their DNS names) are completely unknown and there is only one point of interaction, namely, the PROXY server.

Logging in to the App and the SSO server

When the user connects to an application, the browser presents the web page returned by each application from the APP server:



Welcome to App1 at the MISTIC Corporate Site

running at app.mistic.lan

Welcome user: **user10**

Your access level is: **10**

You logged in using the credential type: **UsernamePasswordCredential**

at: **2019-12-09T09:09:10.463945Z**

Click here to log out.

Of course, before accessing this web page, the CAS server takes care of the authentication process and checks that the user and the credentials provided are correct according to the SSO policy we have implemented.

Authorizations at the application level

Notice that the application can show some information about the login operation that travels back from the CAS server (and the LDAP server) to the CAS client, so the app can show (or work with, as we will see with App4) this information and other attributes retrieved from the CAS and LDAP servers.

So, in addition to be able to implement a common mechanism of authentication via an LDAP server or defining authorization levels via services at the CAS server, we can take advantage of the information received from the CAS server at the application code to implement even further conditions for authorizing the use of the application or some of its features.

In this project, for example, App4 has an extra authorization requirement: this application is only available for users not only having an access level of 40 but also accessing the application with X.509 certificate credentials.

To implement this requirement the App4 code handling the main request can check for the type of credentials used, as received from the CAS server in the login procedure:

```
// Check for a correct credential type
if (attributeName.equals("credentialType") && attributeValue.equals("X509CertificateCredential"))
{
  return "unauthorized";
}
```

In this case, the access to App4 can be denied when necessary and an **unauthorized.html** page showing a message is returned to automatically redirect the user to the logout page.

Single sign-off

Besides being able to log in to an SSO session, the users should be able to log out not only from one application but also from the SSO session[46] and all the logged in applications. This process is also known as *single sign-off* or *single log out*, in analogy to the concept of *single sign-on*.

When the user clicks the logout link in the application page, the CAS server is notified by calling the URL **https://www.mistic.lan/cas/logout** (specifically, the user is first redirected to the application logout.html page which redirects to this URL. See this redirection at any logout.html page in the custom applications code, p. 79).

```
       <meta http-equiv="Refresh" content="0; url=https://www.mistic.lan/cas/logout" />
```

After closing the SSO session, the CAS server notifies all the applications that were active in the same SSO session for them to close their respective application sessions. This behavior is defined in the CAS service configuration file, defining a call to a logout URL, so this call is automatically done by the CAS server:

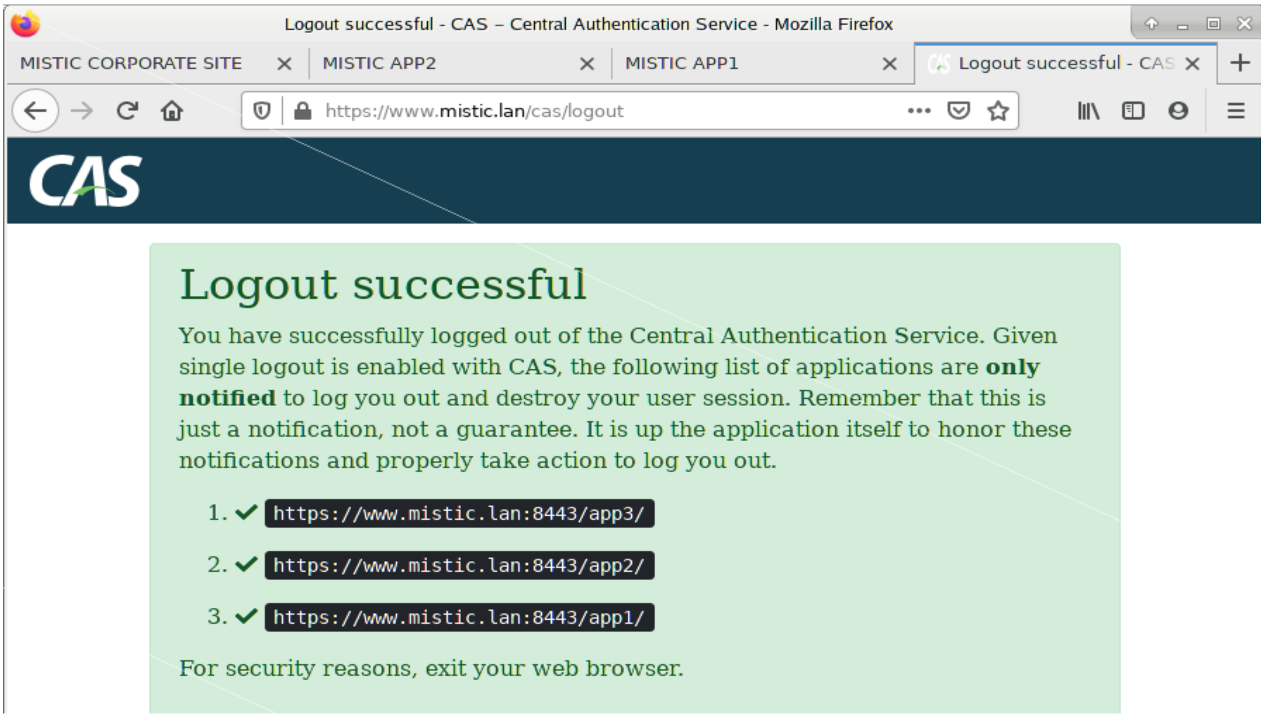
```
 "logoutType" : "FRONT_CHANNEL",
 "logoutUrl" : "https://www.mistic.lan/app3/logout_cas"
```

The specific semantics of a single logout action, if allowed in a given set of applications, depends on the requirements for that application. This is why the CAS server "just" notifies of a "log out" and then the applications must implement the behavior for them specified.

This feature can be enabled or not in an SSO solution depending on the requirements of the domain application because the user may experience that behavior as unexpected logouts. We implemented, in the CAS services configuration, that all the application sessions should be terminated after one logout in any of them, and we programmed the application controllers to accept a request to **/logout_cas** for closing the application session.

```
   @RequestMapping("/logout_cas")
   public String logout_CAS()
   {
       session.invalidate();

       return "logout";
   }
```

In this example below, after the user logs out from App3 the server notifies App1 and App2:



Logout successful - CAS – Central Authentication Service - Mozilla Firefox
MISTIC CORPORATE SITE
MISTIC APP2
MISTIC APP1
Logout successful - CAS
https://www.mistic.lan/cas/logout
CAS

Logout successful

You have successfully logged out of the Central Authentication Service. Given single logout is enabled with CAS, the following list of applications are **only notified** to log you out and destroy your user session. Remember that this is just a notification, not a guarantee. It is up the application itself to honor these notifications and properly take action to log you out.

1. ✓ `https://www.mistic.lan:8443/app3/`
2. ✓ `https://www.mistic.lan:8443/app2/`
3. ✓ `https://www.mistic.lan:8443/app1/`

For security reasons, exit your web browser.

[46] See https://apereo.github.io/cas/6.1.x/installation/Logout-Single-Signout.html for a discussion.

7.2 The production environment

For the production environment, we provide and configure the servers in the internal network that implement the SSO solution.

7.2.1 The DMZ Proxy

Provisioning the server

Using the cloning feature of Virtualbox, we clone the base image server to create a new server as the Proxy at the DMZ. Once cloned, we reconfigure the machine to meet the corresponding settings as server **proxy.mistic.lan** (see Creating a new machine from the base image, p. 54).

Installing the proxy component

We install the *Apache httpd* software component with the proxy modules enabled to allow access to the internal servers via this machine (see Installing Apache httpd as a reverse proxy, p. 60).

Once properly configured, the DMZ proxy will transparently redirect all the external requests to the address **https://www.mistic.lan** to the internal servers either at the APP server address **https://app.mistic.lan** or at CAS server **https://cas.mistic.lan** as needed.

7.2.2 The APP server

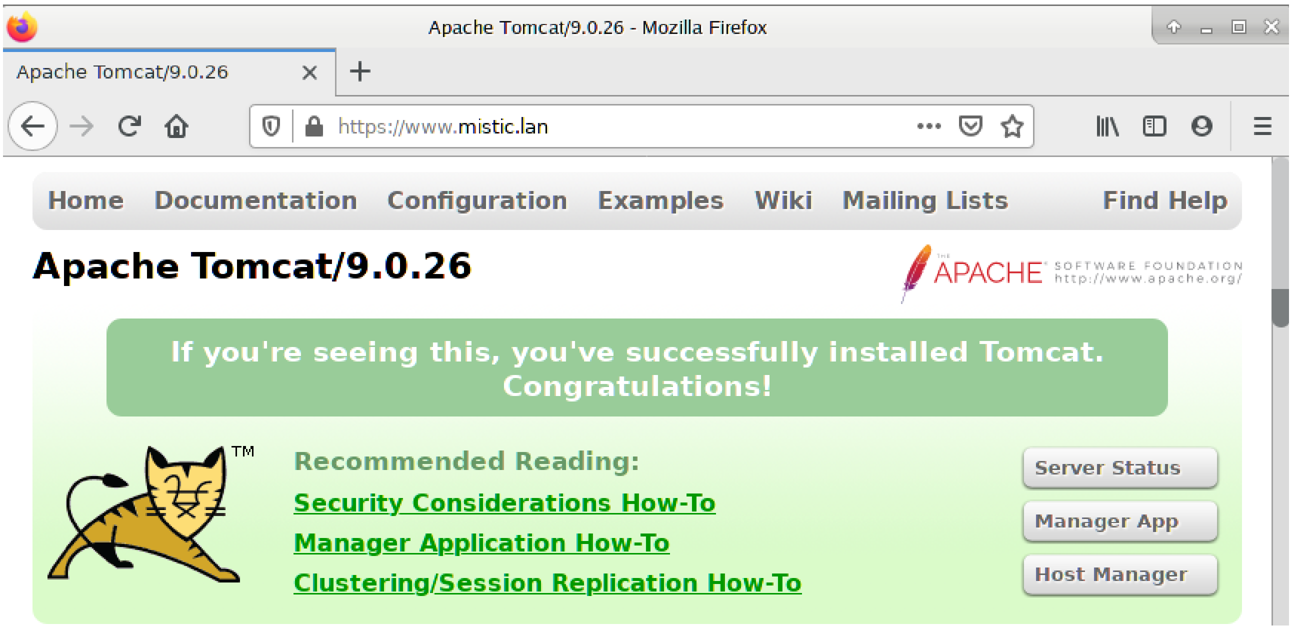
Provisioning the server

Using again the cloning features, we clone a new machine for the APP server. Once cloned we reconfigure the machine to meet its corresponding settings as the server **app.mistic.lan**.

Installing the application server

In this APP server, we first install Amazon Corretto 11 to have the Java runtime ready (see Installing Amazon Corretto 11, p.59), and then install a Tomcat server (see Installing Apache Tomcat, p.62) where the internal applications will run.

To test the accessing the APP server via the Proxy server from the external network, just requesting the address of the **www.mistic.lan** from the client browser we actually obtain the default Tomcat pages at **app.mistic.lan**:



Apache Tomcat/9.0.26 - Mozilla Firefox

Apache Tomcat/9.0.26

https://www.mistic.lan

Apache Tomcat/9.0.26

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:

Security Considerations How-To

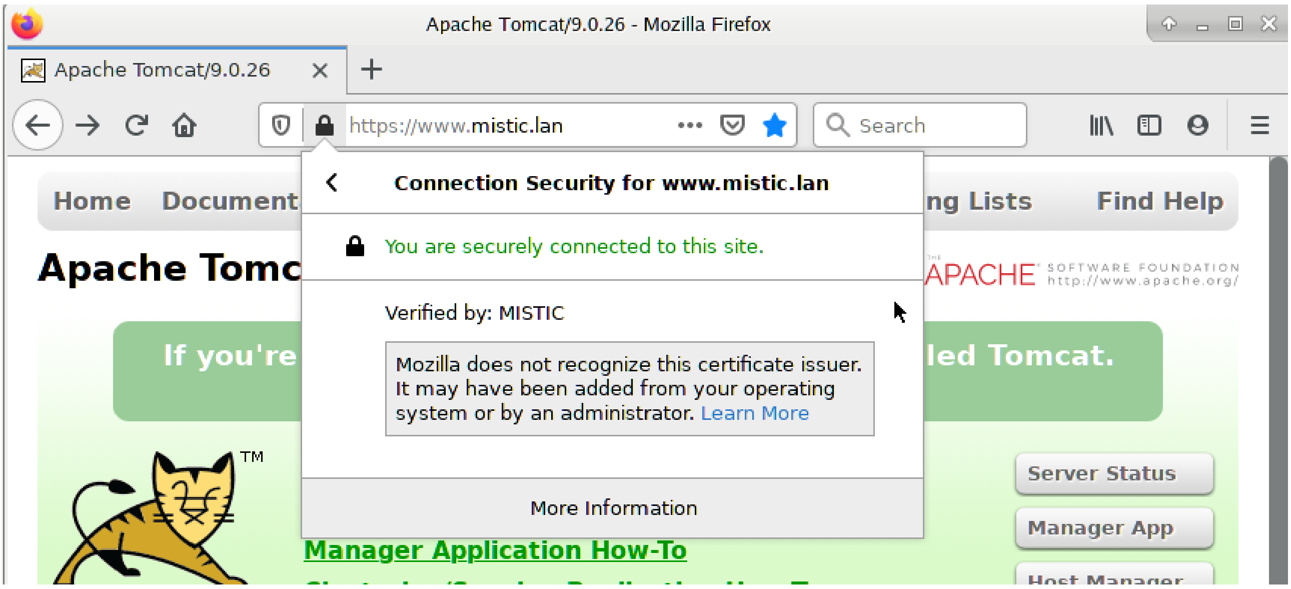
Manager Application How-To

Clustering/Session Replication How-To

We can also check that the connection to the internal server working as expected passing through the proxy :

```
$ traceroute app.mistic.lan
traceroute to app.mistic.lan (192.168.2.101), 30 hops max, 60 byte packets
 1  www.mistic.lan (192.168.1.200)  0.162 ms  0.156 ms  0.155 ms
 2  app.mistic.lan (192.168.2.101)  0.362 ms  0.369 ms  0.371 ms
```

and that the connections are secured with the MISTIC certificates and the MISTIC CA:



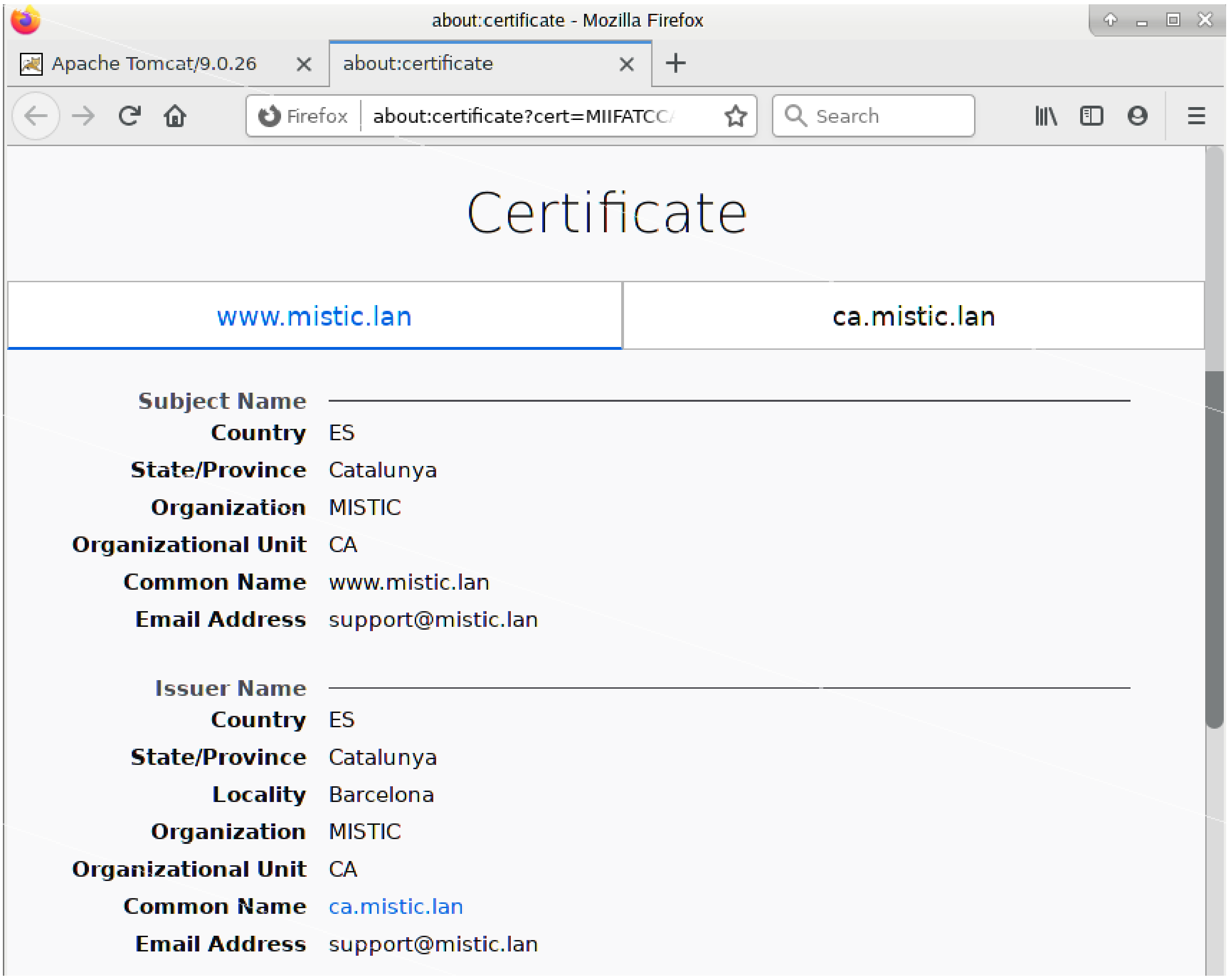
Connection Security for www.mistic.lan

You are securely connected to this site.

Verified by: MISTIC

Mozilla does not recognize this certificate issuer. It may have been added from your operating system or by an administrator. Learn More

Requesting "More Information" about the certificate, the user can check the issuer of the certificate and the corresponding Certification Authority:

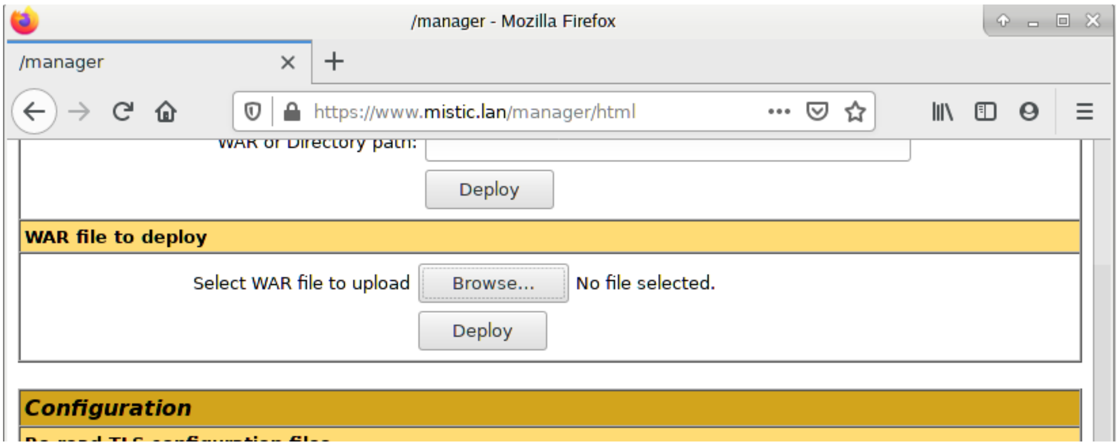


Subject Name	
Country	ES
State/Province	Catalunya
Organization	MISTIC
Organizational Unit	CA
Common Name	www.mistic.lan
Email Address	support@mistic.lan
Issuer Name	
Country	ES
State/Province	Catalunya
Locality	Barcelona
Organization	MISTIC
Organizational Unit	CA
Common Name	ca.mistic.lan
Email Address	support@mistic.lan

Installing the custom applications

In the APP server, we can deploy the custom applications **App1**, **App2**, **App3** and **App4** from the development environment.

To do so, we have access to the Tomcat's Manager application installed by default at the APP Tomcat server and use this interface to deploy the WAR files of the custom applications to the APP server:


/manager - Mozilla Firefox
/manager
https://www.mistic.lan/manager/html
Deploy
Browse...
Deploy

WAR or Directory path:

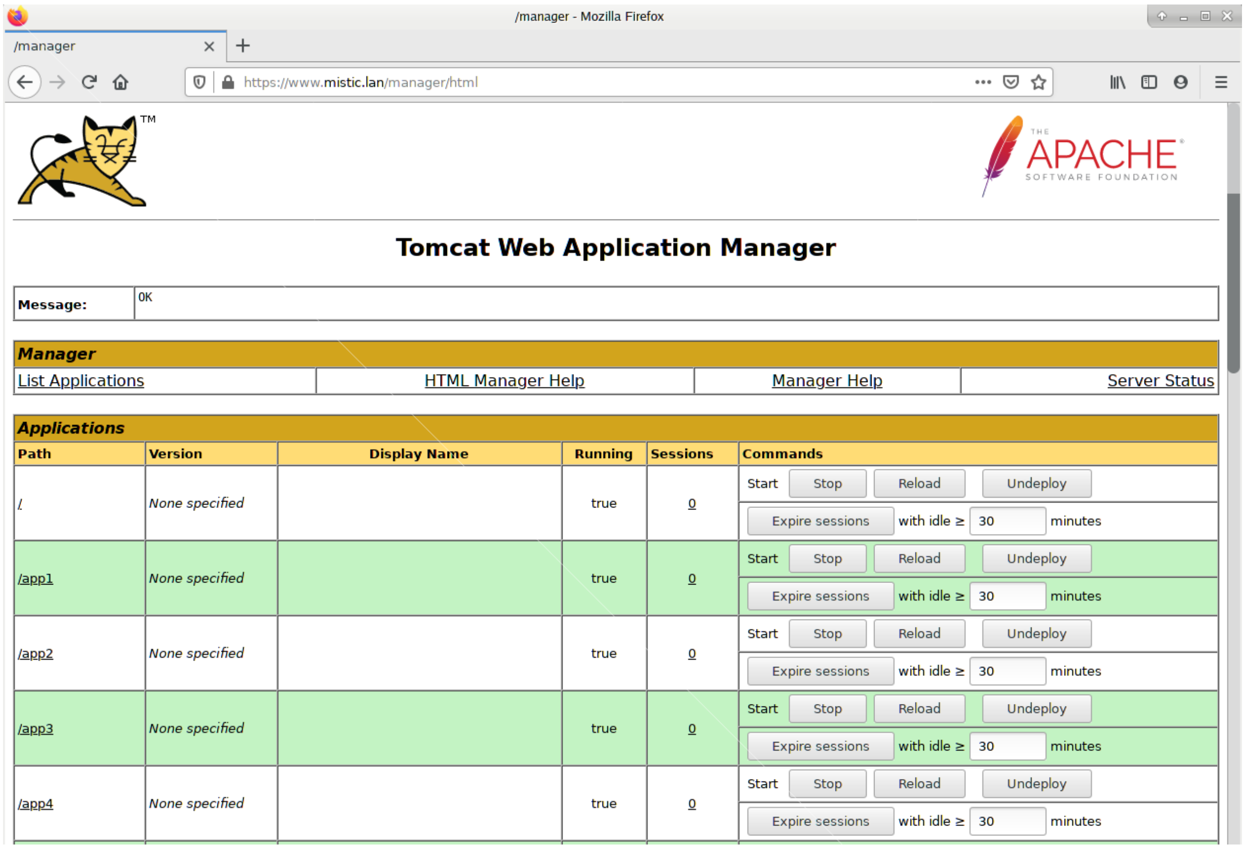
WAR file to deploy

Select WAR file to upload No file selected.

Configuration

or we can simply copy the applications into the **/opt/tomcat/webapps** directory in the APP server from the development machine, or even use the Eclipse IDE capabilities to deploy the WAR after a successful built.

Once all the custom applications are installed and deployed, the Tomcat Manager will show them up and running at **https://www.mistic.lan/manager**[47]:


/manager - Mozilla Firefox
/manager
https://www.mistic.lan/manager/html
TM
THE APACHE SOFTWARE FOUNDATION

Tomcat Web Application Manager

Message:	OK

Manager

List Applications	HTML Manager Help	Manager Help	Server Status

Applications

Path	Version	Display Name	Running	Sessions	Commands
/	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/app1	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/app2	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/app3	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes
/app4	None specified		true	0	Start Stop Reload Undeploy Expire sessions with idle ≥ 30 minutes

Finally, we install the Home Page at the *ROOT* directory at */opt/tomcat/webapps* (the original Tomcat server home page can be saved by renaming the ROOT directory as *tomcat*, for example, so it is still accessible at www.mistic.lan/tomcat). The MISTIC main page can be now accessed from **https://www.mistic.lan**.

[47] Although useful for development purposes, these administration applications should be removed from external access to better secure the installation on production.

7.2.3 The LDAP server

Provisioning the server

We clone a new virtual machine for the LDAP server and configure it with the necessary settings to act as the **ldap.mistic.lan** server (see Creating a new machine from the base image p.54).

Installing the LDAP service

After installing the directory service (see Installing OpenLDAP, p. 65) we are ready to configure the MISTIC organization scheme to set up the right users and their attributes for the applications.

Configuring the MISTIC directory

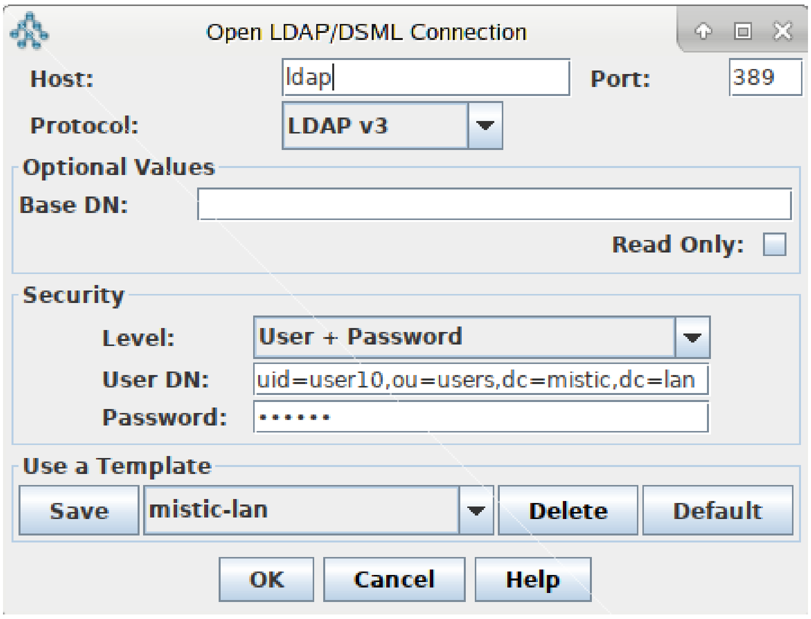
We define the list of users and organizational units for the MISTIC corporation in an LDIF(*) file (see **lan.mistic.ldif** at LDAP server configuration files, p.75).) with all the needed directory entries for users and their attributes. Then we can load the scheme into the LDAP directory using the command *ldapadd.* At the LDAP server, with the file available, we run the command:

```
root@ldap# ldapadd -v -D cn=admin,dc=mistic,dc=lan -x -w mistic -c -f lan.mistic.ldif
```

The *ldapadd* command connects to the LDAP server as user *admin.mistic.lan* and password *mistic* and loads the contents of the file **lan.mistic.ldif**. Then, we can use the command *slapcat* to check the scheme:

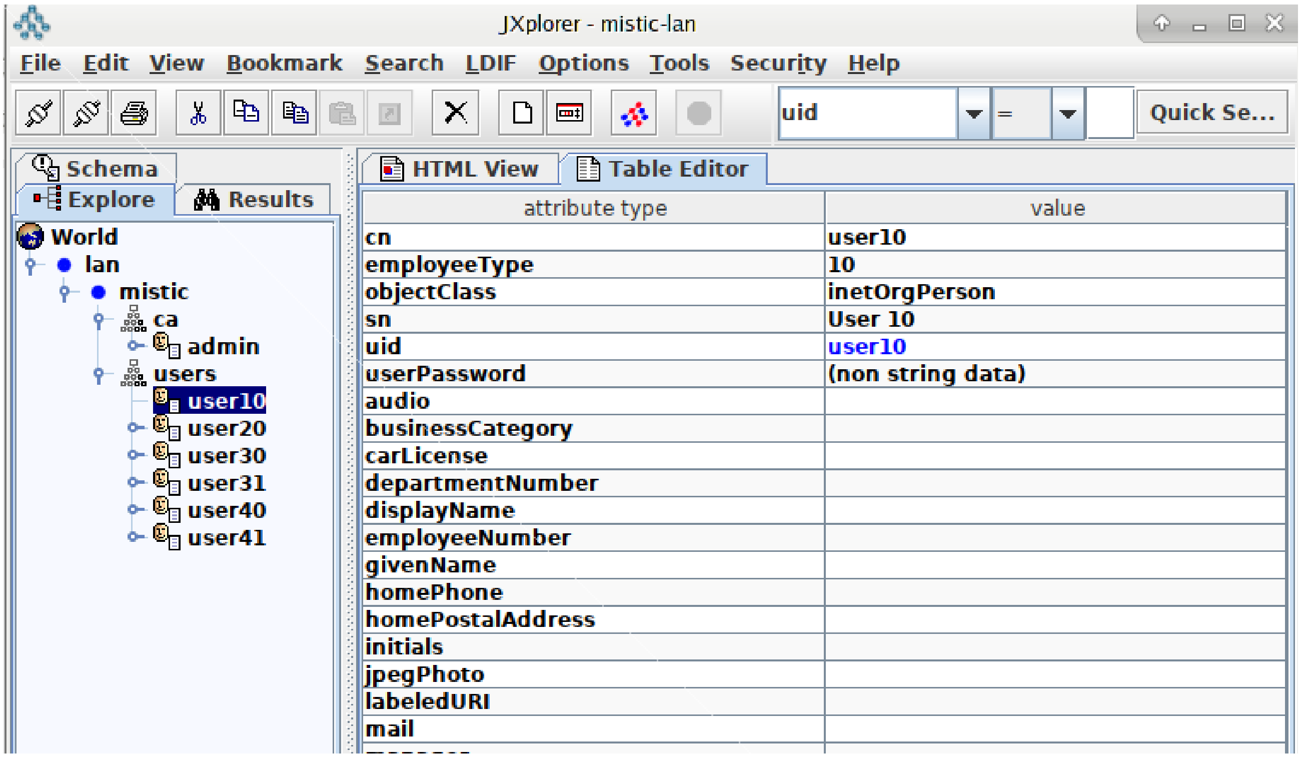
```
root@ldap# slapcat
dn: dc=mistic,dc=lan
objectClass: top
objectClass: dcObject
objectClass: organization
o: mistic.lan
dc: mistic
structuralObjectClass: organization
entryUUID: 0acbdb7a-b4f4-1039-944d-c9cba5f02b93
creatorsName: cn=admin,dc=mistic,dc=lan
createTimestamp: 20191217083619Z
entryCSN: 20191217083619.365920Z#000000#000#000000
modifiersName: cn=admin,dc=mistic,dc=lan
modifyTimestamp: 20191217083619Z

<...some lines are omitted...>

dn: ou=users,dc=mistic,dc=lan
objectClass: organizationalUnit
ou: Users
structuralObjectClass: organizationalUnit
entryUUID: 9c2d9fd2-b4f8-1039-8b53-f74736586912
creatorsName: cn=admin,dc=mistic,dc=lan
createTimestamp: 20191217090901Z
entryCSN: 20191217090901.263213Z#000000#000#000000
modifiersName: cn=admin,dc=mistic,dc=lan
modifyTimestamp: 20191217090901Z

dn: uid=user10,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user10
userPassword:: dXNlcjEw
cn: user10
sn: User 10
employeeType: 10
structuralObjectClass: inetOrgPerson
entryUUID: 9c2fb48e-b4f8-1039-8b54-f74736586912
creatorsName: cn=admin,dc=mistic,dc=lan
createTimestamp: 20191217090901Z
entryCSN: 20191217090901.276851Z#000000#000#000000
modifiersName: cn=admin,dc=mistic,dc=lan
modifyTimestamp: 20191217090901Z

<...the rest of the scheme omitted...>
```

At this point, we can also use any graphical tool for browsing an LDAP directory, like **JXplorer**, to connect and navigate this new LDAP scheme:



Host: ldap
Port: 389
Protocol: LDAP v3
Base DN:
Read Only: □
Level: User + Password
User DN: uid=user10,ou=users,dc=mistic,dc=lan
Password: ••••••
mistic-lan

Once connected to the LDAP directory, we can navigate through the MISTIC organization and users there defined:



World
- lan
  - mistic
    - ca
      - admin
    - users
      - user10
      - user20
      - user30
      - user31
      - user40
      - user41

attribute type	value
cn	user10
employeeType	10
objectClass	inetOrgPerson
sn	User 10
uid	user10
userPassword	(non string data)
audio	
businessCategory	
carLicense	
departmentNumber	
displayName	
employeeNumber	
givenName	
homePhone	
homePostalAddress	
initials	
jpegPhoto	
labeledURI	
mail	

7.2.4 The CAS server

The Apereo CAS software product is, indeed, a Spring Boot application that can be built either as a WAR file to be deployed in any Tomcat server or as a JAR file with an embedded Tomcat server. We choose to build a second server with another Tomcat installed, following the procedures shown before but with no custom applications installed but just the Apereo CAS application.

Provisioning the server

After cloning a new virtual machine for the CAS server, we reconfigure it with the appropriate settings as server **cas.mistic.lan** (see Creating a new machine from the base image p.54).

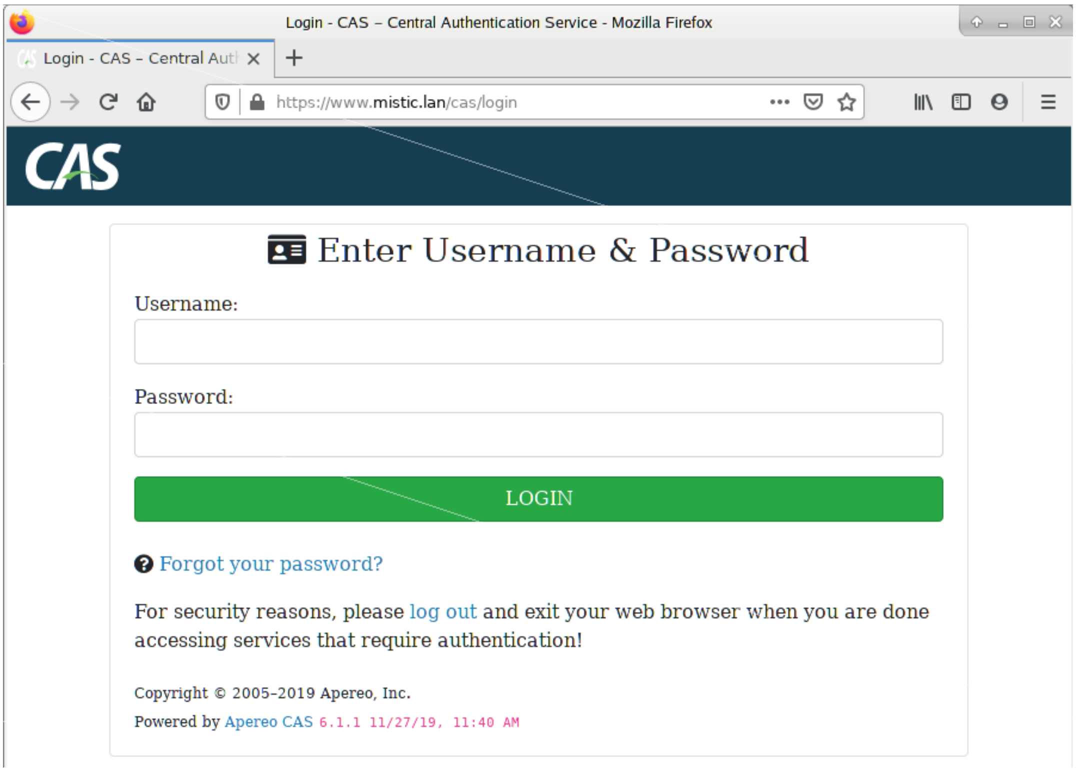
Installing the Tomcat server

In this server, we also install Amazon Corretto 11 (see Installing Amazon Corretto 11, p.59), and a Tomcat server ready to deploy the CAS WAR file (see Installing Apache Tomcat, p.62).

Installing the CAS application

We first build the CAS application from the Maven overlay[48] and then deploy it in the CAS server Tomcat container (see Installing Apereo CAS server p. 67) as a normal Java EE application.

Once the CAS application is installed in the Tomcat servlet container, the service is available for the applications, via the proxy, at the URL **https://www.mistic.lan/cas**.


Login - CAS – Central Authentication Service - Mozilla Firefox
Login - CAS – Central Aut
https://www.mistic.lan/cas/login
CAS

Enter Username & Password

Username:

Password:

LOGIN

Forgot your password?

For security reasons, please log out and exit your web browser when you are done accessing services that require authentication!

Copyright © 2005-2019 Apereo, Inc.

Powered by Apereo CAS 6.1.1 11/27/19, 11:40 AM

Configuring the CAS application

To completely configure the CAS application to work with the LDAP server to validate users, we have to log in to the CAS server to review and change the CAS configuration file at **/etc/cas/config/cas.properties.**

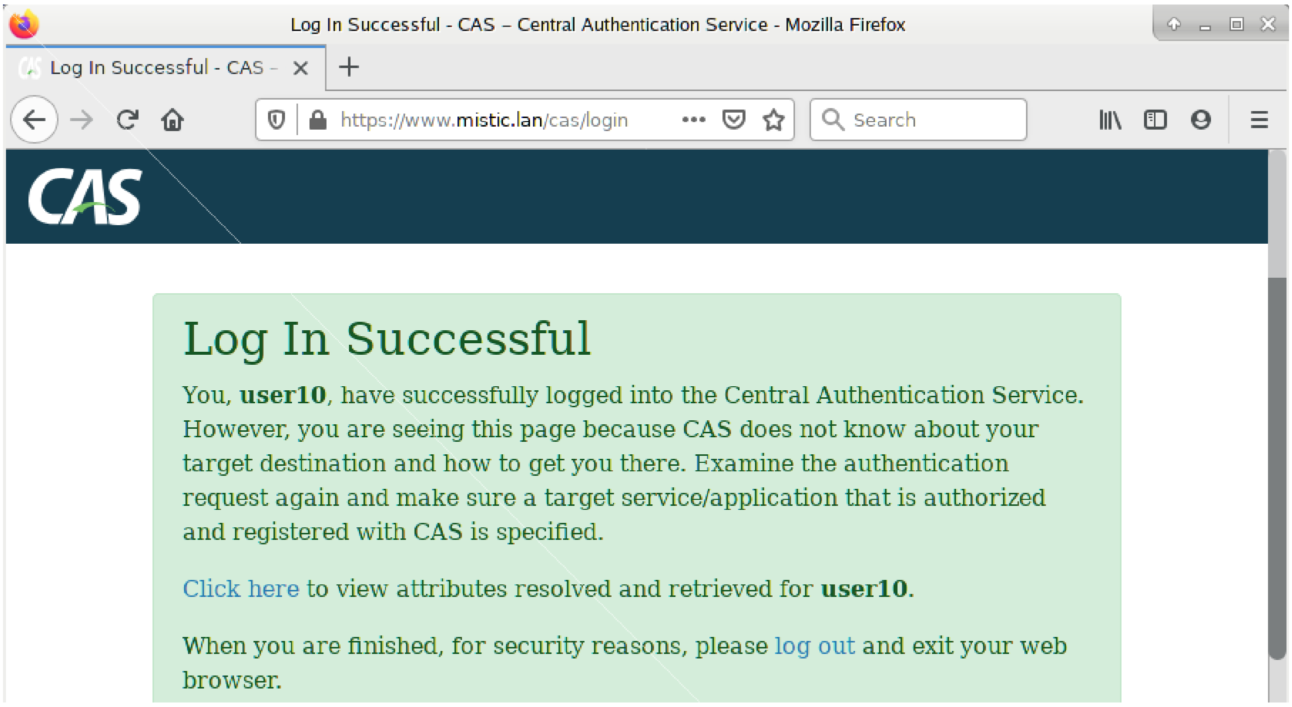
We replace the default values with the configuration needed for the MISTIC project purposes (see CAS server configuration files, p. 72).

Also, in this configuration file, there is a reference to another configuration file for the CAS server logs. As the Tomcat server is run under the *tomcat* user and this user has no permissions to write in the default directory /var/log, we choose to change the **/etc/cas/config/log4j2.xml** file to instead write the CAS server logs in the directory /opt/tomcat/logs:

[48] https://apereo.github.io/cas/6.0.x/installation/WAR-Overlay-Installation.html

```
<Configuration monitorInterval="5" packages="org.apereo.cas.logging">
    <Properties>
        <Property name="baseDir">/opt/tomcat/logs</Property>
```

After these changes are done in the /etc/cas files, we restart the server and accessing to the URL **https://www.mistic.lan/cas** again we can log in with an LDAP user (e.g. using *user10*) and check that the CAS server correctly logs in the user:


Log In Successful - CAS – Central Authentication Service - Mozilla Firefox
Log In Successful - CAS –
https://www.mistic.lan/cas/login
Search
CAS

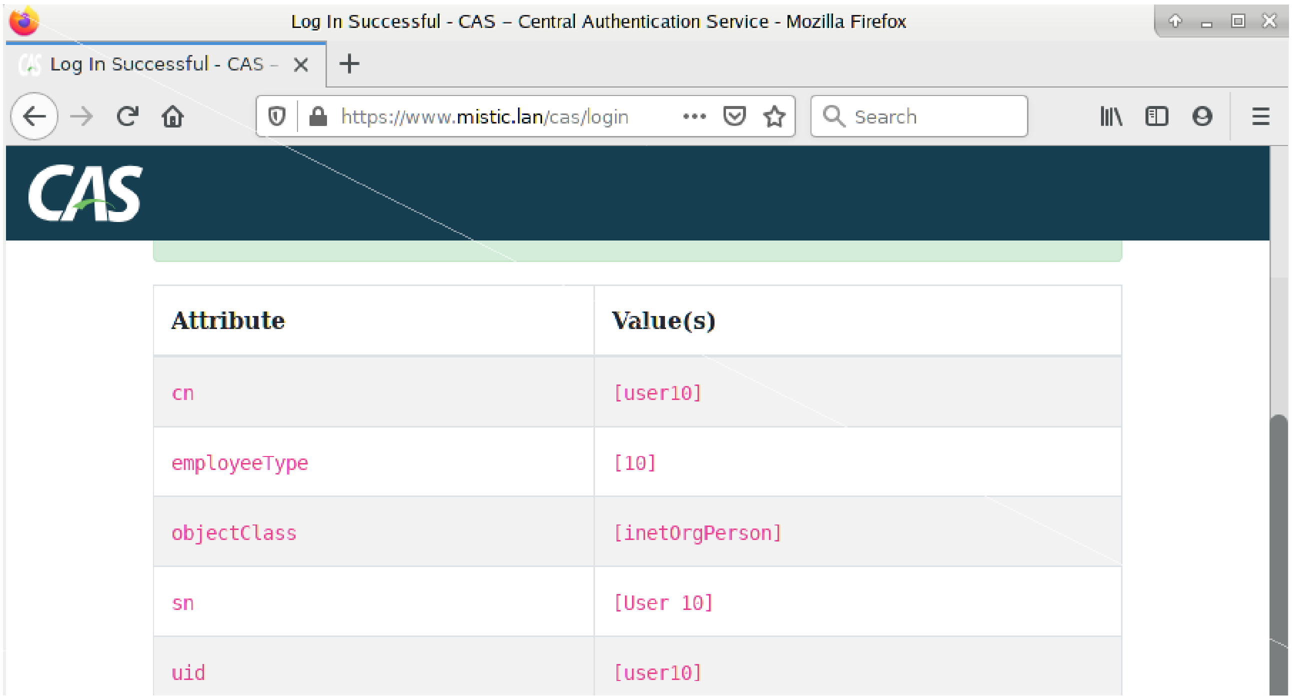
Log In Successful

You, **user10**, have successfully logged into the Central Authentication Service. However, you are seeing this page because CAS does not know about your target destination and how to get you there. Examine the authentication request again and make sure a target service/application that is authorized and registered with CAS is specified.

Click here to view attributes resolved and retrieved for **user10**.

When you are finished, for security reasons, please log out and exit your web browser.

Indeed, we can check the attributes received from the CAS server and retrieved from the LDAP directory, thanks to the configuration defined in the **/etc/cas/config/cas.properties** file:

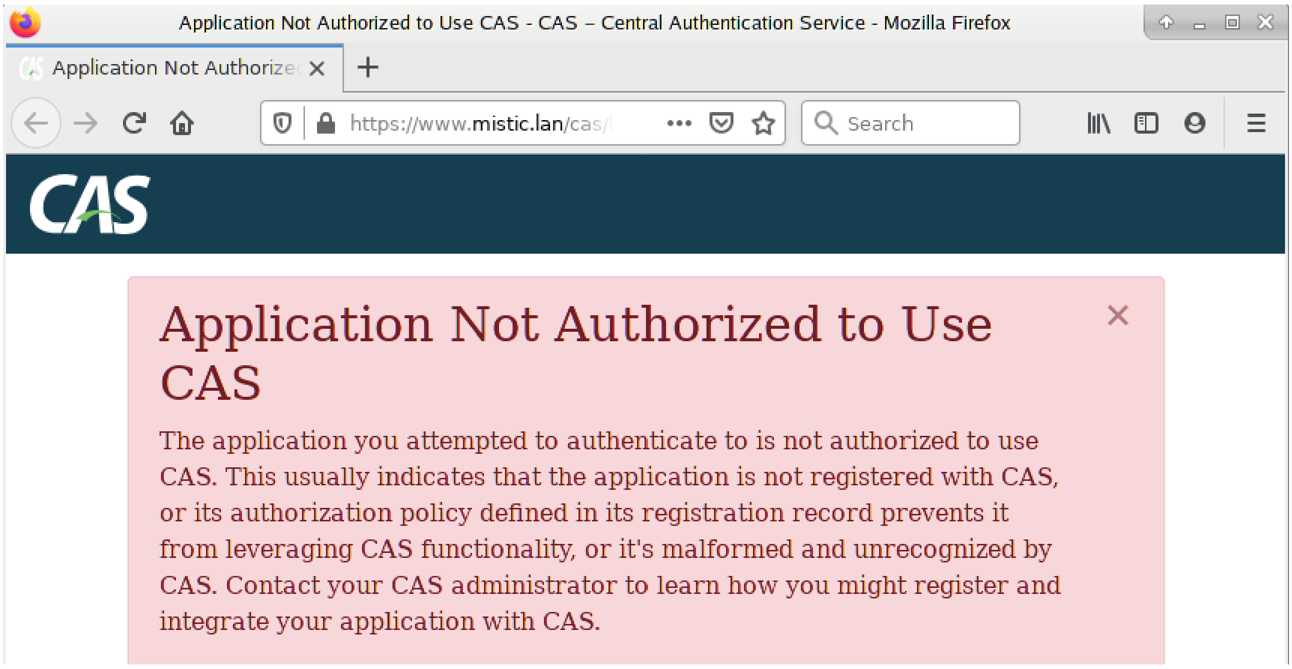

Log In Successful - CAS – Central Authentication Service - Mozilla Firefox
Log In Successful - CAS –
https://www.mistic.lan/cas/login
Search
CAS

Attribute	Value(s)
cn	[user10]
employeeType	[10]
objectClass	[inetOrgPerson]
sn	[User 10]
uid	[user10]

These are attributes above are retrieved from the LDAP server and sent to the applications, after successful authentication. More specifically, we will use the **employeeType** attribute for further authorization at the application layer, even for those users correctly authenticated.

Configuring CAS services

The Apereo CAS solution allows defining dedicated configurations for the different client applications that can connect to the server, called *services* in the CAS terminology. We use this feature to implement the authorization policy for each application depending on the LDAP attribute **employeeType** retrieved after the user authentication. Otherwise, if we try to connect from the MISTIC menu from the Java CAS applications we get an authorization error, even using the right user credentials:



Application Not Authorized to Use CAS - CAS – Central Authentication Service - Mozilla Firefox
Application Not Authorize
https://www.mistic.lan/cas/
Search
CAS

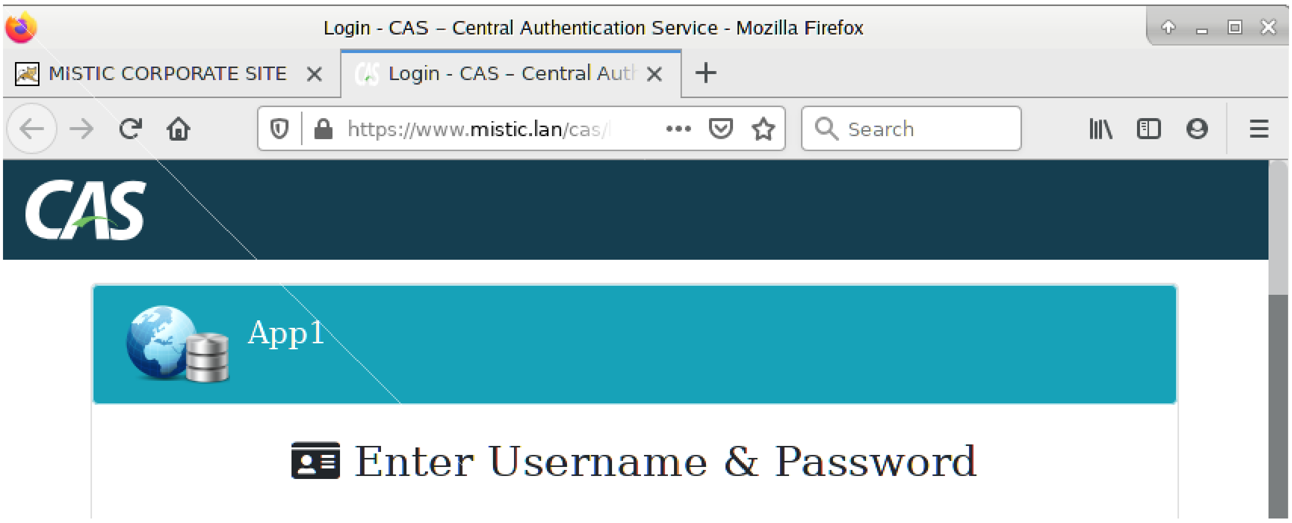
Application Not Authorized to Use CAS

The application you attempted to authenticate to is not authorized to use CAS. This usually indicates that the application is not registered with CAS, or its authorization policy defined in its registration record prevents it from leveraging CAS functionality, or it's malformed and unrecognized by CAS. Contact your CAS administrator to learn how you might register and integrate your application with CAS.

To register the different services (client applications) that the CAS server will go through for further checks and test after user authentication, we copy the service configuration files from CAS server configuration files, p73 in the /etc/cas/services/ directory, one for each of our custom applications.

After restarting the CAS Tomcat server, now we are able to access these applications at the APP server, after getting authorization from the CAS server.

Notice the CAS login form customized to show the CAS service being accessed (e,g. App1):



Login - CAS – Central Authentication Service - Mozilla Firefox
MISTIC CORPORATE SITE
Login - CAS – Central Auth
https://www.mistic.lan/cas/
Search
CAS
App1

Enter Username & Password

7.3 Solution integration

7.3.1 Checking the SSO policy implementation

With all the servers installed and configured, now we can test the whole solution by connecting to the main page and accessing each application using different user credentials. This will test the correct and complete implementation of the SSO policy for the project as defined in The SSO policy (p. 28).

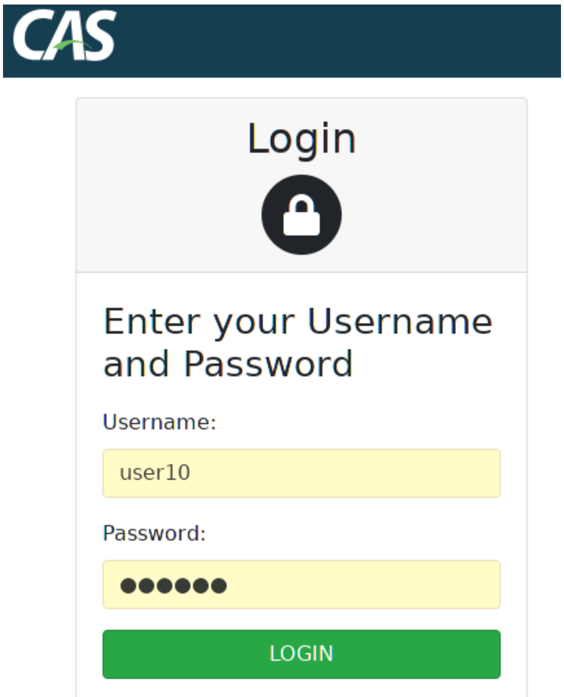
Accessing as a user with user-name and password

Testing as user10

User10 is required to log in using a valid user-name and password and can only access applications at level 10, namely, only App1.

When we connect to the MISTIC main page and click on the App1 link, the browser connects to the App1 at APP server, but being App1 a CAS client, it connects in turn to the CAS server to authenticate the user.

At this point the CAS server prompts the user[49] for user-name and password:



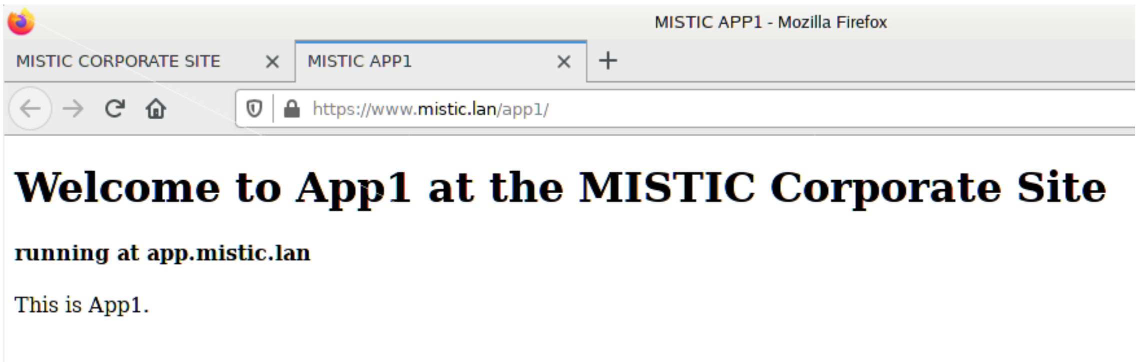
Login

Enter your Username and Password

Username: user10

Password: ●●●●●●

If the credentials are correct (once validated against the LDAP directory), the user gets access and connects to the App1 application:



https://www.mistic.lan/app1/

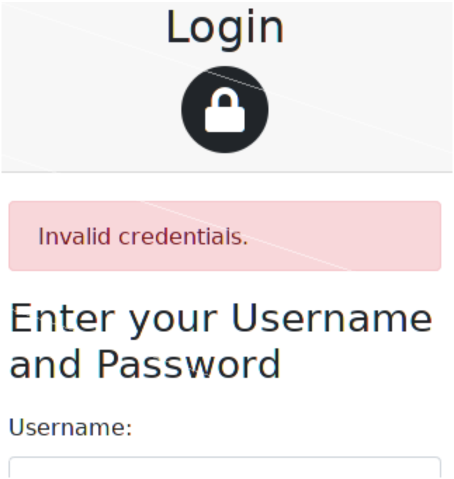
Welcome to App1 at the MISTIC Corporate Site

running at app.mistic.lan

This is App1.

[49] The login page can be customized to use the corporate logo and design for better integration in the corporate environment and seamless user experience. We are using the default pages from CAS version 6.1.x.

However, if the provided credentials are not valid, the CAS server returns an error and prevents the user from using the application:



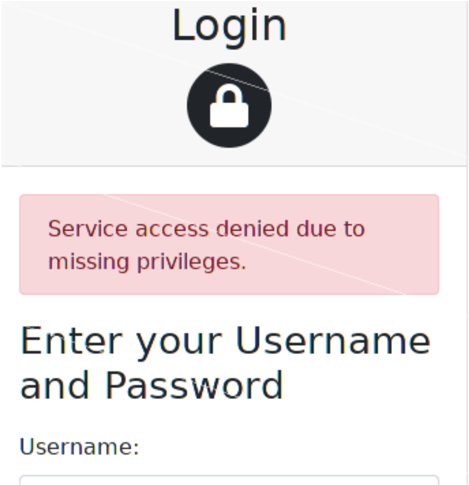
Login

Invalid credentials.

Enter your Username and Password

Username:

If this user10 has logged in correctly, and therefore has initiated a single sign-on session, he or she could enter immediately in any other application defined at level 10. However, if the user attempts to access a higher-level application, as defined in the project policy, the CAS server will not authorize this access. For example, if user10 tries to access App2, the CAS server responds:



Login

Service access denied due to missing privileges.

Enter your Username and Password

Username:

The key point is that the CAS services defined at the CAS server include an "access strategy" based on certain attributes, in our case "a required" attribute and value for "employeType" as retrieved from the LDAP server account after a correct authentication.

Notice that App1 accepts all our specified access levels:

```
"accessStrategy" : {
  "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
  "requiredAttributes" : {
    "@class" : "java.util.HashMap",
    "employeeType" : [ "java.util.HashSet", [ "10" "20" "30" "40" ] ]
  }
}
```

but App2, and respectively all other applications, only accept higher access levels:

```
"accessStrategy" : {
  "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
  "requiredAttributes" : {
    "@class" : "java.util.HashMap",
    "employeeType" : [ "java.util.HashSet", [ "20" "30" "40" ] ]
  }
}
```

This effectively restricts user10 to level 10 applications, but correctly provides at the same time a valid SSO session to use other level 10 applications at the same APP server.

Testing as user20

User20 is required to log in using a valid user-name and password but, having been granted a level access of 20 (by the *employeeType* attribute at the LDAP directory) in case of correct log, in, this user will be able to access applications at level 20, that is only App2, but also without a second log in procedure to applications at lower level, App1 in this case.

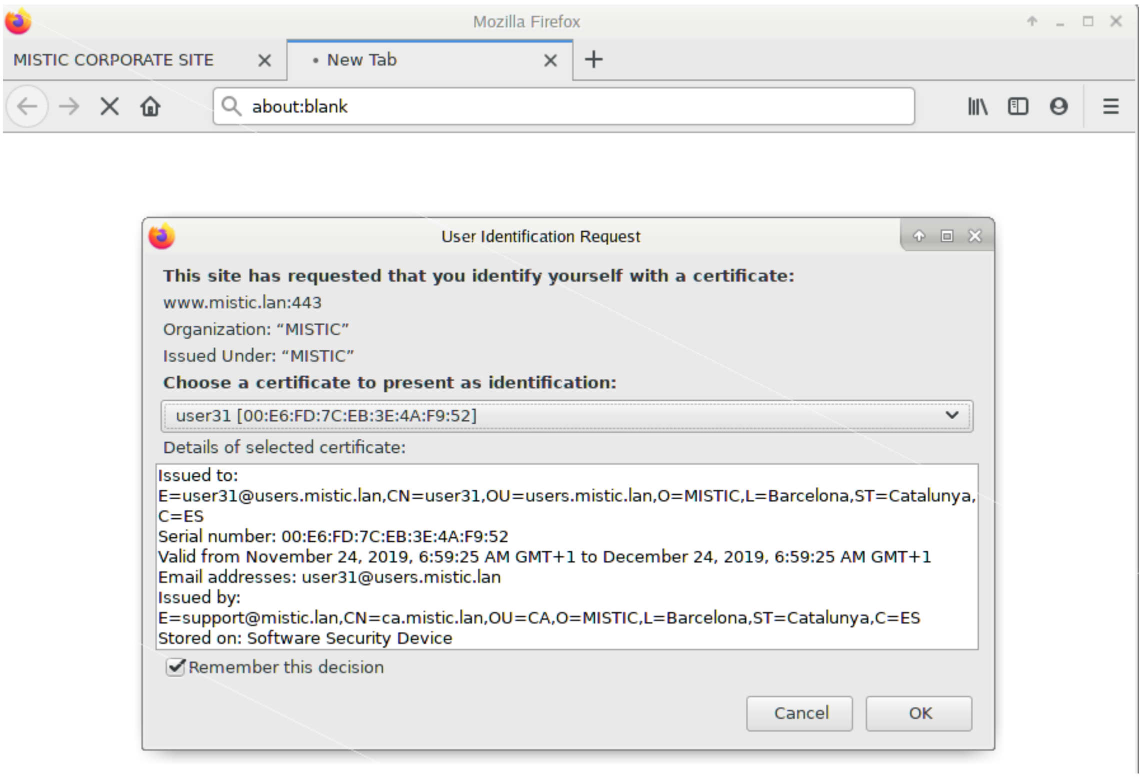
Testing as user30

User30 is required to log in using a valid user-name and password. Like user20, this user is able to access App3, App2 and App1 with one single log in operation in any one of these applications. The access to the rest of the applications is granted with the SSO session obtained in the first correct authentication.

Accessing as a user with a digital certificate

Testing as user31

User31 is required to authenticate using a digital certificate. When this user access App3, the CAS server requests the browser to prompt the user to provide a digital certificate (depending on the browser used, the user can select which certificate to use if more than one):



This site has requested that you identify yourself with a certificate:

www.mistic.lan:443

Organization: “MISTIC”

Issued Under: “MISTIC”

Choose a certificate to present as identification:

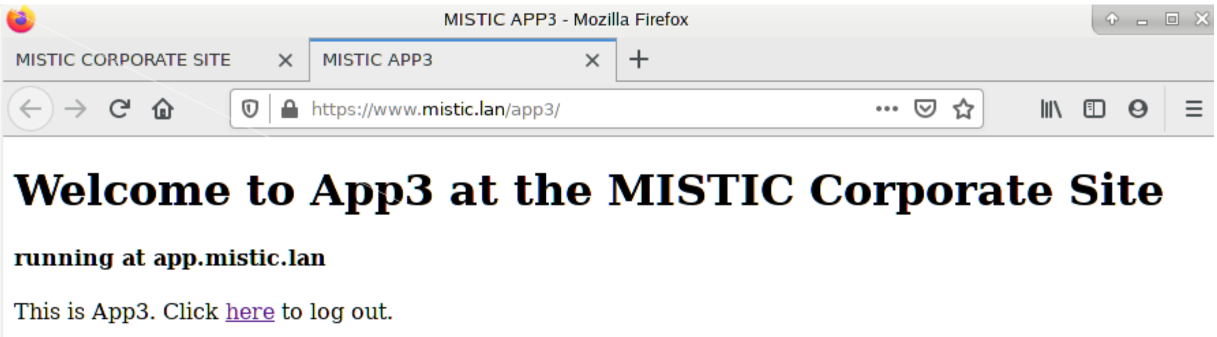
user31 [00:E6:FD:7C:EB:3E:4A:F9:52]

Details of selected certificate:

Issued to:
E=user31@users.mistic.lan,CN=user31,OU=users.mistic.lan,O=MISTIC,L=Barcelona,ST=Catalunya,C=ES
Serial number: 00:E6:FD:7C:EB:3E:4A:F9:52
Valid from November 24, 2019, 6:59:25 AM GMT+1 to December 24, 2019, 6:59:25 AM GMT+1
Email addresses: user31@users.mistic.lan
Issued by:
E=support@mistic.lan,CN=ca.mistic.lan,OU=CA,O=MISTIC,L=Barcelona,ST=Catalunya,C=ES
Stored on: Software Security Device

☑ Remember this decision

If a valid certificate is installed in the browser and the user selects the correct one, then the certificate is validated as signed by a recognized certification authority. If the certificate is accepted, then the CAS server uses the user-name in the certificate to access the LDAP server to validate the user and retrieve the rest of the needed attributes. If everything is correct, the user can access the authorized applications, without entering user-name and password, but with a valid SSO session available:



Welcome to App3 at the MISTIC Corporate Site

running at app.mistic.lan

This is App3. Click here to log out.

Accessing an app that accepts only digital certificates

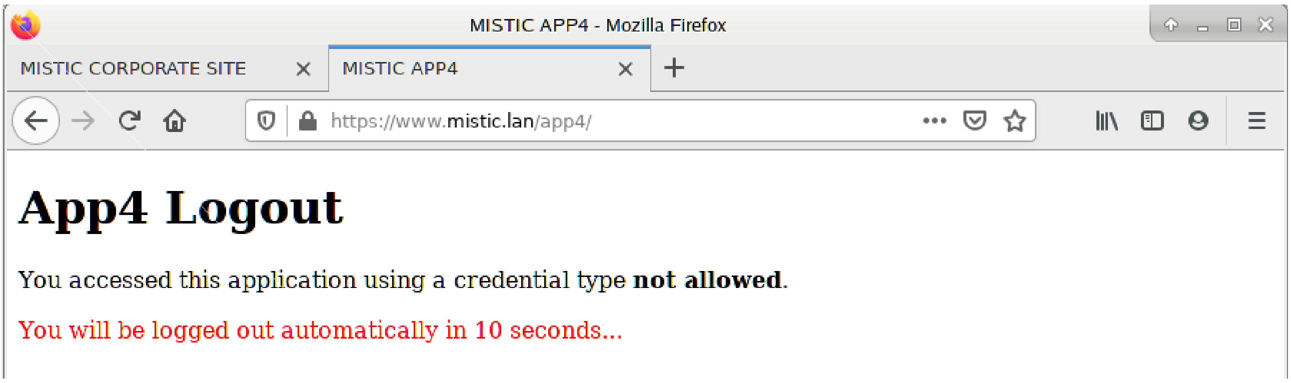
Additional requirements for App4

App4 has a different authorization requirement with respect to the other apps that can be accessed by users either with user-name and password or with digital certificates. For App4, besides requiring an access level of 40, there exists the extra requirement that states that only users presenting a digital certificate can access the app regardless of their access level.

Testing as user40

User40 has been given user-name and password credentials and an access level of 40 in the user directory. However, this is not enough to access App4 because no digital certificate is presented as a credential in the login procedure.

Thus, when user40 logs in App4, even though the CAS server grants a valid SSO session to the user, the application in the APP server detects that the log in credentials were not a digital certificate and, in our implementation, automatically logs out the user from the application:

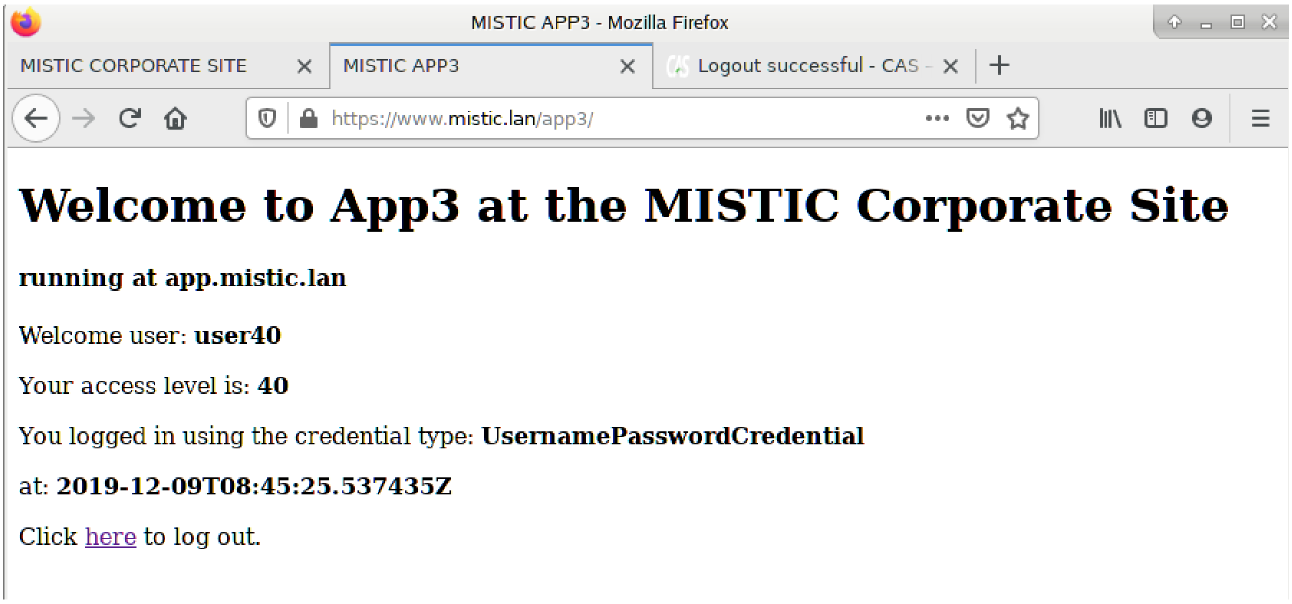


App4 Logout

You accessed this application using a credential type **not allowed**.

You will be logged out automatically in 10 seconds...

Nonetheless, user40 has a valid SSO session and enough access level to enter all the other applications at lower levels:


MISTIC APP3 - Mozilla Firefox
MISTIC CORPORATE SITE
MISTIC APP3
Logout successful - CAS –
https://www.mistic.lan/app3/

Welcome to App3 at the MISTIC Corporate Site

running at app.mistic.lan

Welcome user: **user40**

Your access level is: **40**

You logged in using the credential type: **UsernamePasswordCredential**

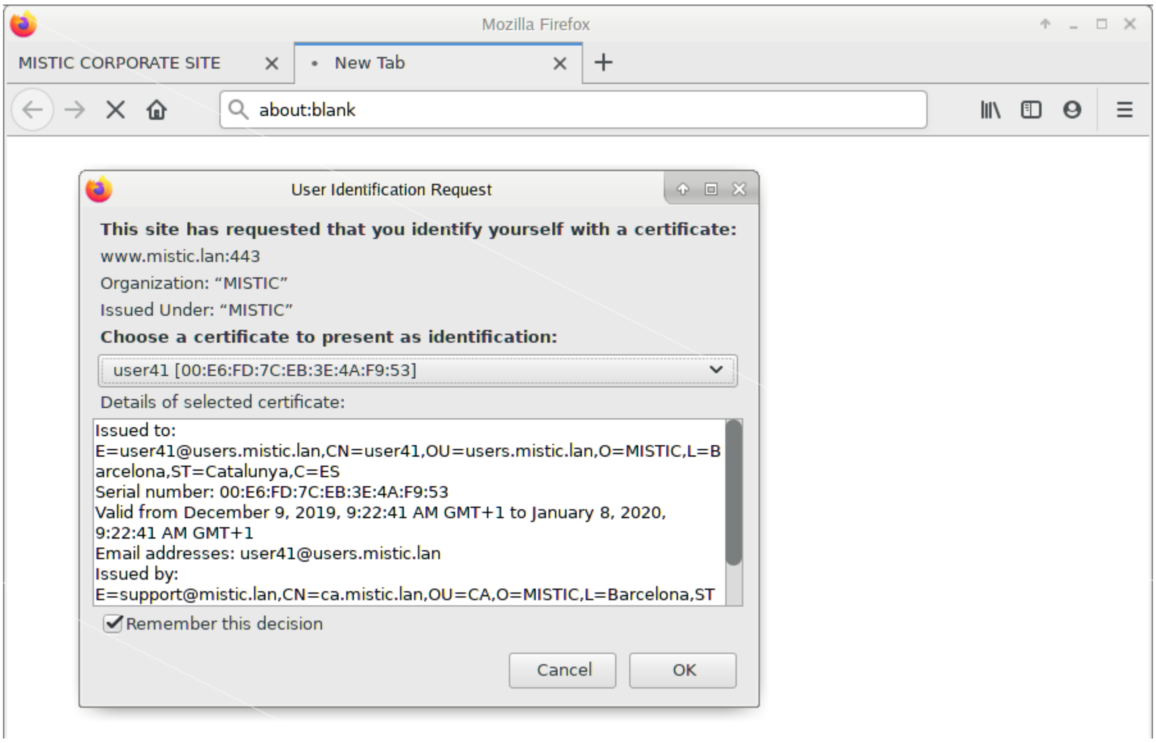
at: **2019-12-09T08:45:25.537435Z**

Click here to log out.

Testing as user41

User41, instead, has been given a digital certificate, plus the access level of 40 in the user directory. This translates in this solution in that user41 can effectively access App4 along with the rest of applications using the digital certificate as user credentials.

For example, when user41 clicks the App4 access link, the correct certificate can be provided:



This site has requested that you identify yourself with a certificate:

www.mistic.lan:443

Organization: “MISTIC”

Issued Under: “MISTIC”

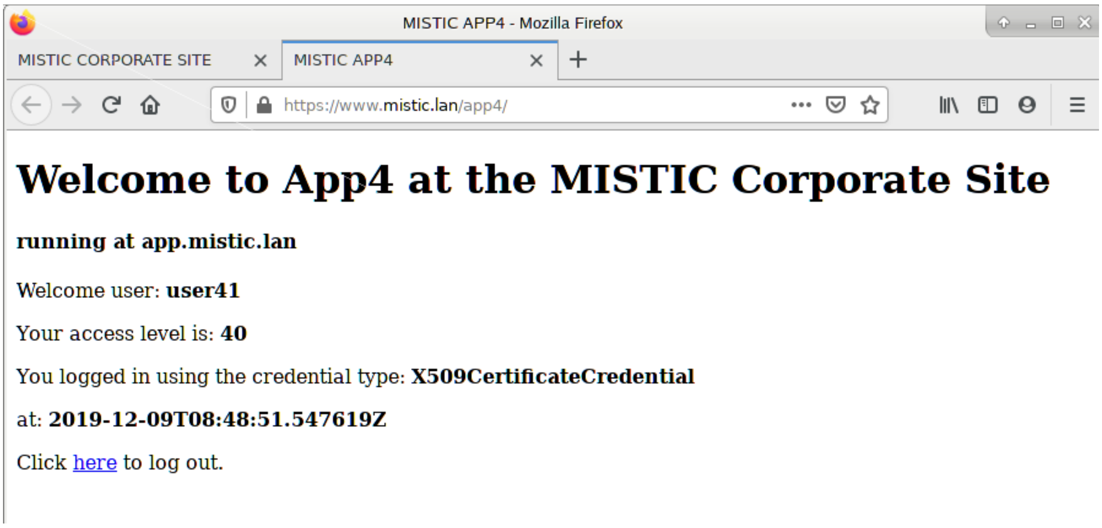
Choose a certificate to present as identification:

user41 [00:E6:FD:7C:EB:3E:4A:F9:53]

Details of selected certificate:

```
Issued to:
E=user41@users.mistic.lan,CN=user41,OU=users.mistic.lan,O=MISTIC,L=B
arcelona,ST=Catalunya,C=ES
Serial number: 00:E6:FD:7C:EB:3E:4A:F9:53
Valid from December 9, 2019, 9:22:41 AM GMT+1 to January 8, 2020,
9:22:41 AM GMT+1
Email addresses: user41@users.mistic.lan
Issued by:
E=support@mistic.lan,CN=ca.mistic.lan,OU=CA,O=MISTIC,L=Barcelona,ST
```

☑ Remember this decision



Welcome to App4 at the MISTIC Corporate Site

running at app.mistic.lan

Welcome user: **user41**

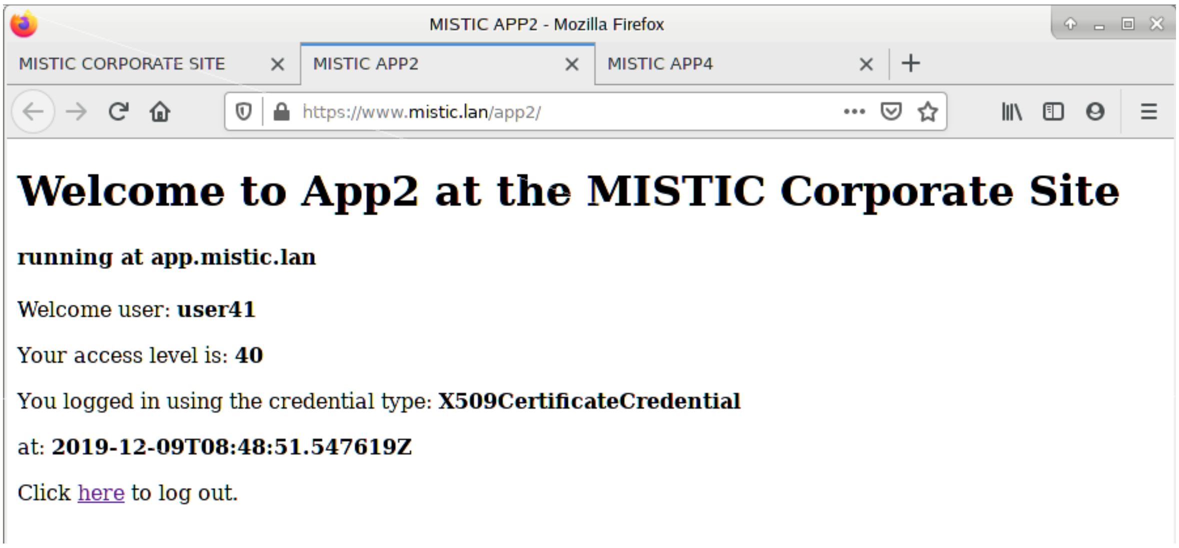
Your access level is: **40**

You logged in using the credential type: **X509CertificateCredential**

at: **2019-12-09T08:48:51.547619Z**

Click here to log out.

and having an SSO session granted, this user can also access other applications with no extra steps:



Welcome to App2 at the MISTIC Corporate Site

running at app.mistic.lan

Welcome user: **user41**

Your access level is: **40**

You logged in using the credential type: **X509CertificateCredential**

at: **2019-12-09T08:48:51.547619Z**

Click here to log out.

8 Project results

8.1 Specific results

Single sign-on

In this project, we have implemented a Single Sign-On solution based on the Apereo CAS product and custom Java client applications.

The CAS product offers a complete set of features to implement an SSO solution, which provided us with the necessary tools to implement the objectives for the project with the exception of the extra, specific validation for App4 to restrict the users to log only using digital certificates.

The CAS server is capable of validating users against several sources of user information. We implemented an OpenLDAP server in this project and we could not only validate the user-name and password but also perform additional lookups in the LDAP database to retrieve more attributes for the user. This was the case of the "access level", that we implemented using the existing *employeeType* field on the *inetOrgPerson* object class provided by OpenLDAP.

Single sign-off

One issue with the SSO sessions, sometimes an unexpected behavior but other times a requirement for the project, is having the possibility to log out from all the applications, where a user is logged in, using just a single log out operation.

Apereo CAS provides a feature to signal to all the applications involved in the same SSO session when a logout is performed at one application. Nonetheless, it is still the responsibility of each application to implement its own sign off procedures when that signal is received.

Multilevel authentication

The SSO scheme, by its very own nature, implies that a user logging in to an application in the set of protected resources gets immediate access to the rest of the applications. This can be sometimes regarded as a drawback, and a much more detailed degree of control over the authorizations to use some applications may be required.

We have implemented a multilevel authorization mechanism that allows defining an SSO policy specifying which "access level" is required to access each application. This provides the capability to define multiple tiers of authorization for a set of applications.

Multifactor authentication

Having the possibility to request more than one type of credential in the same log in operation has become a common trend, especially in web applications on the internet, as a mechanism to increase the security of these applications and prevent identity impersonation attacks[50].

We have implemented in our solution two types of credentials, namely user-name/passwords and X.505 digital certificates, for user authentication. And we have defined in our SSO policy when a user should access using which type.

However, we have not explored all the possible cases, like requiring a user to provide both types of credentials in the same login operation. We have considered an authentication successful with just one type or the other is valid.

Nonetheless, the current trend in the industry is to implement forms of multifactor authentication requiring more than one credential type in the same authentication procedure. This is the case of the so-called *Two Factor Authentication* (**2FA**) where a user, after a successful user-name and password validation. is temporarily provided with a second token, via SMS for example, to ensure that the user logging in is really the owner of the first credentials. Other mechanisms are been deployed by the main industry actors, like replacing the SMS code[51] with a dedicated authentication application installed in the user’s smartphone.

[50] See a case of building an impersonation attack on SSO systems at Mayer (2014).

[51] The 2FA solution is already regarded as insecure as more and more successful attacks are being discovered. See, for example, Grimes (2019)

Traffic encryption

All the network connections have been secured even in the internal network, with the exception of the connection from the PROXY to the CAS server. This limitation is imposed by the need to pass information from the user's certificate (which is actually validated by the PROXY server) to the CAS server so it can obtain the user identifier and then perform the needed lookups and validations on the LDAP server. This security feature is important in the increasing trend to move corporate workloads to the cloud, where the "internal network" connecting different servers should no longer be considered secure, especially when multiple vendors are involved or in hybrid cloud environments.

8.2 Generic results and remarks

Choosing an SSO server

The choice of Apereo CAS as the base product for the SSO solution has revealed as a good alternative. However, Apereo CAS is a fast-moving platform (during the development of the project we had the opportunity to change from version 6.0.x to 6.1.x, for example).

This implies that implementing a long term SSO solution based on Apereo CAS (or any other modern product, for that matter) should require provisioning the necessary human resources to keep the solution not just updated with the latest product patches but to be able to upgrade to newer releases at the pace the vendor releases more versions and features.

Choosing an LDAP server

Although an LDAP server seems a more mature and stable product in the industry, other aspects may be considered when choosing the LDAP server. The choice of implementing an LDAP server just dedicated to the SSO solution or, instead, using an existing user directory (say, a Microsoft Active Directory in the corporate network) can have architectural and security implications especially about where the SSO policy is actually defined and by who.

SSO clients dependency

Besides any internally developed applications, other third-party or closed-source applications could be required to be put under the same SSO policy. As long as not all the features required to fully implement an SSO policy can be assigned to, and therefore implemented in, the SSO server (in our case, limiting the access to App4 to digital certificates requires dedicated code in the Java client), defining some complex SSO policies can be difficult to implement in some environments.

We also have seen that even the latest Internet browsers (like Mozilla Firefox) may have some degree of incompatibility when interacting with the SSO solution (e.g. the TLS v1.3 and Tomcat post handshake problems).

Also, not all Internet browsers may behave the same way when dealing with the user's digital certificates (Firefox has to be closed and restarted to forget the last certificate used) which results in an inconsistent experience for lay users and different behaviors among the variety of user platforms.

Automation and centralization

Centralizing the point of authentication, and even authorization, in an SSO solution has obvious implications with respect to the availability of the solution, now with a single point of failure(*). And despite the initial strategy to concentrate efforts and resources around the SSO solution, the diversity of potential clients (say, in a BYOD(*) environment) makes almost impossible to avoid the nuances of dealing with multiple types of client devices.

Finally, although the automated deployment of clients and unattended installation of certificates in final user machines can ease the workloads of deploying and maintaining a complete SSO solution, all these extra requirements should be carefully considered when deciding to move towards an SSO solution in a given organization.

8.3 Future work

8.3.1 Theoretical approach

SSO policy specifications

Our very basic theoretical approach to the "SSO policy" has allowed us to design a decoupled authentication/authorization model based on an indirection[52] via the notion of "access level". In order for this approach to evolve to a more consistent theoretical framework, several improvements should be developed.

Nonetheless, the advantages of having some kind of formal specification for a (complex) authentication and authorization scheme seem promising at least at the degree of being able to define unambiguously the expected behavior of any actual SSO implementation.

Algebraic properties

The notion of multilevel authorization assumes a tiered scheme; that is, a layer of authorized resources one on top of another. This is due to the inherent semantics associated with using natural numbers (and its relation of order) as access levels, thus resulting in increasing (and including) subsets of authorized resources. Indeed, our specific project relies on an *ordered set* of just four elements {10, 20. 30, 40} more than in the whole set of natural numbers.

Whether this choice of using natural numbers as the image set of the authentication function reveals as an unnecessary limitation and the model is open to other looser or wider algebraic structures to gain further flexibility can be explored[53].

Validation expressions

The definition of how the multiple factors of authentication are combined to validate a user’s login is not clearly specified in this model. For example, in our case, whether two credentials are required in an OR-type or AND-type combination is not specified. In the case of the default set up of Apereo CAS, the different authentication solvers are called one after another resulting in a valid authentication when one method succeeds (thus implementing an OR-type check).

More sophisticated definitions for "expressions", using some formal language grammar, could be added to the model to enhance the specification on how to combine multiple factors of authentication.

Fine-grained permissions

In this model, once the user gets access to a resource, no additional information about the allowed use of the resource is specified. An example of such more detailed permissions is to define whether the user can modify the resource or just read it, or whether some features of a given application (e.g. access some menu options) are authorized.

This kind of permission management is usually left to the application to implement[54], but exploring how to move some of these permission assignments to the SSO policy could be interesting, for example, in the actual trend of publishing open web services under public APIs to allow accessing internal database, or to provide some form of RBAC[(*)]-like capabilities to the model.

SLO specifications

Finally, note that the defined policy provides no formal information about the requirements of the single sign-off procedure. In this project, we just explored one way Apereo CAS allows implementing a given form of single log out, which signals to all the now-in-use applications that a logout operation has been performed in some other application.

[52] An aphorism attributed to David Wheeler states that "All problems in computer science can be solved by another level of indirection". The opposite idea attributed to Jim Grat, says "There is no performance problem that can not be solved by eliminating a level of indirection", according to Coulouris (2012) ch 6.1.

[53] See Gollmann (2002), ch. 5.8 for the use of lattices. For a historical perspective on first attempts of formalization, see Bell (1973) or Demming (1976).

[54] This is a present day, common requirement and most modern protocols support it to some extent. See, for example, https://tools.ietf.org/html/rfc6749#page-10 about access tokens in Oauth 2.0 and also https://tools.ietf.org/html/rfc6750 for widely used bearer tokens.

8.3.2 Practical implementation

Load balancing and fault tolerance

The use of *Apache httpd* as a proxy server can be improved with load balancing and failure tolerance capabilities, which should be basic requirements for a complete corporate solution.

Alternatively, in order to deploy these types of features, this proxy/web server component can be replaced with a specialized proxy component. For example, moving from such a "generic" product like *Apache httpd* to a more "specialized" one, like *HAproxy,* could provide a different approximation to load balancing and fault tolerance features.

Service ports selection

In this particular case, we replicated the entries for both ports 443 and 8443 at the PROXY server configuration, due to the fact that CAS expects the Tomcat server to work on port 8443 while we set up our APP server to work on the standard port 443.

A careful selection of service ports at design time, obviously conditioned by the availability of free port numbers in the actual installation should be considered upfront. Or, alternatively, a redesign of the whole solution to only use non-standard, non-reserved ports could avoid potential collisions with already-in-use port numbers and could, thus, simplify the connection configurations between all the implicated servers.

LDAP schemes

The LDAP scheme used is the very basic and standard one provided by OpenLDAP. For an enhanced solution, a dedicated LDAP scheme with new, specialized attributes for the SSO policy could be considered, especially in cases where the user directory to use is an existing one in the installation or the solution has to be integrated with other proprietary products.

Securing internal communications

On another side, the debate whether all network connections should be encrypted or not in the "internal network" remains alive in the industry, with a tendency to go for the all-encrypted approach[55]. We tried this latter approach but had to settle, though, to use AJP, a not-well documented and non-encrypted protocol[56], to connect the APP server to the CAS server to be able to pass details of the user certificate to the CAS server. Further work on this issue could provide a huge improvement because this single fact (having to rely on AJP) prevents the deployment of an "all-encrypted" standardized solution.

Also on the side of the Tomcat servers, we used Java keystores to ease the installation of both APP and CAS servers. The Tomcat’s Catalina engine already supports the use of X.509 certificates without the need of using Java based keystores. This choice would be at the price of making Tomcat dependent of an external suite like OpenSSL installed on the same server, but would provide the benefits of relying on just one security framework to handle all the digital certificates.

Cloud deployment

Finally, in an increasing tendency of the industry to move to cloud based deployments, refactoring this solution for a container based infrastructure, like Docker or LXC/LXD, could be a needed improvement for real deployments of corporate SSO solutions in the cloud. Managing the set of deployed servers with tools like Kubernetes would be the final approach to a cloud deployable, production-grade SSO solution.

[55] Insider attacks can be performed by internal employees but also by cloud provider personnel and related secondary actors. See a review at Duncan (2015).

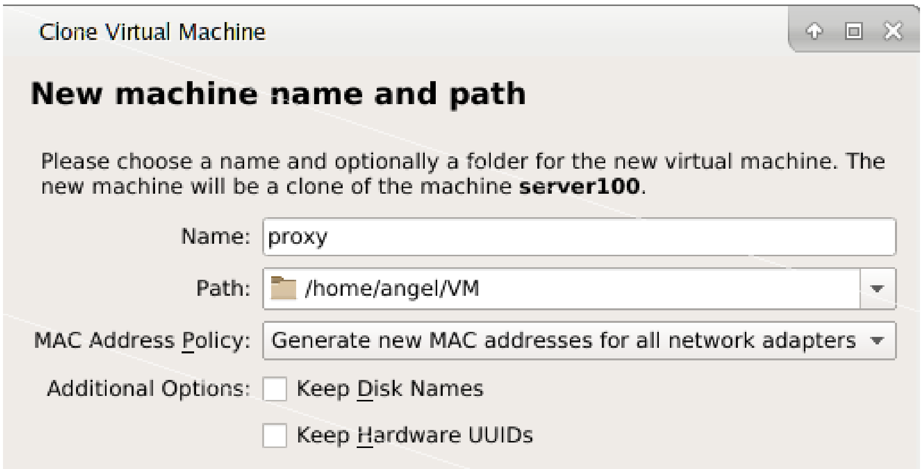
[56] Some documentation is available at https://tomcat.apache.org/connectors-doc/ajp/ajpv13a.html

9 Appendix 1: Installation Procedures

9.1 Creating a new machine from the base image

Cloning the base image

Using the cloning feature of Virtualbox create a new machine from the base image **server100** while assigning a new MAC address to the network adapter to avoid collisions with other cloned machines.



Clone Virtual Machine

New machine name and path

Please choose a name and optionally a folder for the new virtual machine. The new machine will be a clone of the machine **server100**.

Name: proxy

Path: /home/angel/VM

MAC Address Policy: Generate new MAC addresses for all network adapters

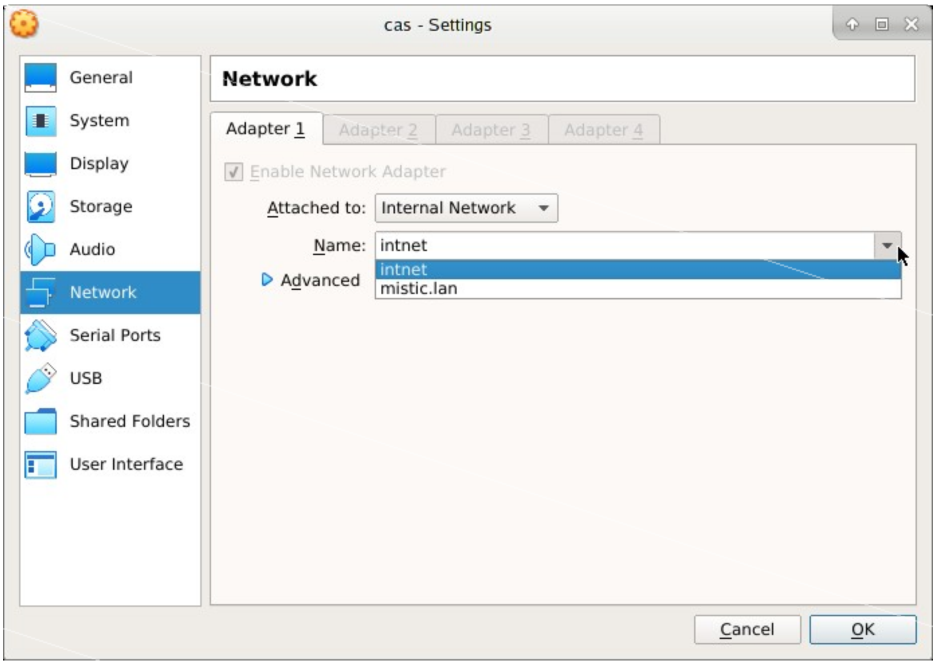
Additional Options: □ Keep Disk Names

□ Keep Hardware UUIDs

Configuring the network adapters

Create a VirtualBox internal network

Then configure the network of the machines in the internal network to use the same name for a Virtualbox network, e.g. **mistic.lan**, so they can connect to each other while remaining isolated from other VMs or the host machine:



cas - Settings

Network

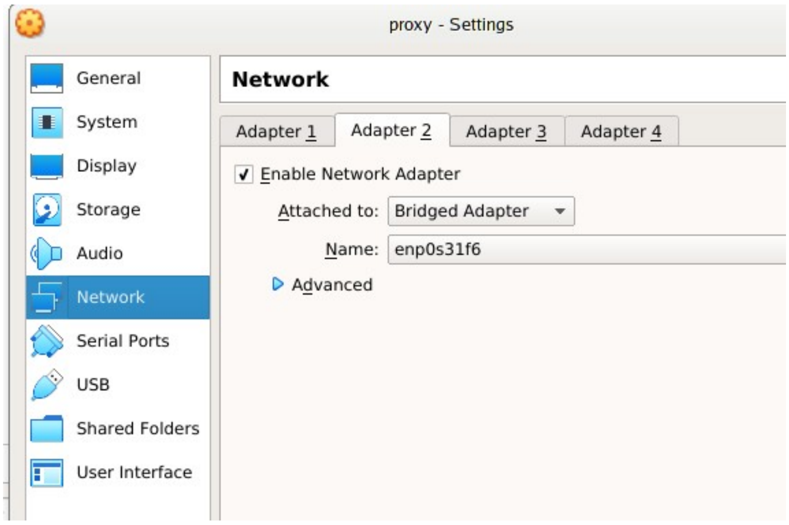
☑ Enable Network Adapter

Attached to: Internal Network

Name: intnet

Configuring a second adapter for the proxy

Besides the configuration for the internal network, for the specific case of the DMZ proxy we need to add a second network adapter connected to the external network via the *VirtualBox bridged adapter*:



☑ Enable Network Adapter

Attached to: Bridged Adapter

Name: enp0s31f6

This is not needed for the rest of the servers.

Customizing the cloned servers

To customize each server, set the correct host name (/etc/hostname), known hosts in the network and interfaces. (see Appendix 2: Source code and configuration files p. 70)

Bridging the networks at the DMZ server

Only in the case of the DMZ Proxy, the network adapters in this server are configured with 2 IP addresses (192.168.2.200 and 192.168.1.200) to bridge the external and the internal network (see PROXY server configuration files, p.70):

```
# ip addr
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1000
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
2: enp0s3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 08:00:27:b5:6c:48 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.200/24 brd 192.168.2.255 scope global enp0s3
       valid_lft forever preferred_lft forever
    inet6 fe80::a00:27ff:feb5:6c48/64 scope link
       valid_lft forever preferred_lft forever
3: enp0s8: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen
1000
    link/ether 08:00:27:09:d0:08 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.200/24 brd 192.168.1.255 scope global enp0s8
       valid_lft forever preferred_lft forever
    inet6 2a01:c50e:21b2:7700:a00:27ff:fe09:d008/64 scope global dynamic mngtmpaddr
       valid_lft 1779sec preferred_lft 579sec
    inet6 fe80::a00:27ff:fe09:d008/64 scope link
       valid_lft forever preferred_lft forever
```

Also, depending on the specific version of the operating system, it may be necessary to enable IP forwarding (setting *net.ipv4.ip_forward=1* in the file /etc/sysctl.conf) to allow TCP/IP packets to pass the bridge between the two network adapters.

9.2 Creating the certificates infrastructure for MISTIC.lan

9.2.1 Creating a Certification Authority certificate

Preparing the development machine

To create and work as a new Certification Authority (CA), the OpenSSL toolkit in the issuing machine needs to be properly configured.

As *root* user, create a directory /root/CA as the working environment for this CA and change the current directory. Then create a set of needed subdirectories and files:

```
root@PCALZ11phy:~ # mkdir CA
root@PCALZ11phy:~ # cd CA

root@PCALZ11phy:~/CA # mkdir certs crl newcerts private
root@PCALZ11phy:~/CA # chmod 700 private
root@PCALZ11phy:~/CA # touch index.txt
root@PCALZ11phy:~/CA # echo 1000 > serial
```

and modify the OpenSSL configuration file (/etc/ssl/openssl.conf) to use this new configuration:

```
[ CA_default ]
dir             = /root/CA
certificate     = $dir/certs/ca.mistic.lan.crt
private_key     = $dir/private/ca.mistic.lan.key
```

Create the private key for the CA

Now, in order to create a Certification Authority, first create the private RSA key:

```
root@PCALZ11phy:~/CA # openssl genrsa -out private/ca.mistic.lan.key 4096
Generating RSA private key, 4096 bit long modulus
...........................................................++++
.......................++++
e is 65537 (0x010001)

root@PCALZ11phy:~/CA # chmod 400 private/ca.mistic.lan.key
```

Creating the root certificate for the CA

Then create the root certificate for the CA and provide the requested information:

```
root@PCALZ11phy:~/CA # openssl req -new -x509 -extensions v3_ca \
 -sha256 -days 3650 -verbose \
 -key private/ca.mistic.lan.key -out certs/ca.mistic.lan.crt

Using configuration from /etc/ssl/openssl.cnf
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) [AU]:ES
State or Province Name (full name) [Some-State]:Catalunya
Locality Name (eg, city) []:Barcelona
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MISTIC
Organizational Unit Name (eg, section) []:CA
Common Name (e.g. server FQDN or YOUR name) []:ca.mistic.lan
Email Address []:support@mistic.lan
```

For security reasons, is a good practice to create a new intermediate CA which will sign the final certificates. In this project we will sign the certificates using the root CA, for simplicity.

9.2.2 Creating MISTIC-signed X.509 certificates

To create a new certificate (e.g. www.mistic.lan for a server or user31.users.mistic.lan for a user) signed by the previously created CA, first create a new RSA pair of keys:

```
root@PCALZ11phy:~/CA # openssl genrsa -out newcerts/www.mistic.lan.key 2048
Generating RSA private key, 2048 bit long modulus
..................++++
........................................++++
e is 65537 (0x010001)

root@PCALZ11phy:~/CA # chmod 400 newcerts/www.mistic.lan.key
```

Then create a certificate signing request (CSR) using the previous keys and enter the data for the new certificate:

```
root@PCALZ11phy:~/CA # openssl req -new -sha256 \
  -key newcerts/www.mistic.lan.key -out newcerts/www.mistic.lan.csr
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) [AU]:ES
State or Province Name (full name) [Some-State]:Catalunya
Locality Name (eg, city) []:Barcelona
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MISTIC
Organizational Unit Name (eg, section) []:CA
Common Name (e.g. server FQDN or YOUR name) []:www.mistic.lan
Email Address []:support@mistic.lan

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

In the case of certificate for a user in the organization, special care must be taken in the organizational information provided in the certificate because this piece of information will be used by the CAS server to validate the user against the LDAP directory:

```
root@PCALZ11phy:~/CA # openssl req -new -sha256 \
   -key newcerts/user31.users.mistic.lan.key -out newcerts/user31.users.mistic.lan.csr
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) [AU]:ES
State or Province Name (full name) [Some-State]:Catalunya
Locality Name (eg, city) []:Barcelona
Organization Name (eg, company) [Internet Widgits Pty Ltd]:MISTIC
Organizational Unit Name (eg, section) []:users.mistic.lan
Common Name (e.g. server FQDN or YOUR name) []:user31
Email Address []:user31@users.mistic.lan
Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
```

Finally, sign these CSR requests using the CA certificate to create the signed certificates:

```
# openssl x509 -sha256 -days 365 -req -in newcerts/user31.users.mistic.lan.csr \
  -CA certs/ca.mistic.lan.crt -CAkey private/ca.mistic.lan.key -CAcreateserial \
  -out newcerts/user31.users.mistic.lan.crt
Signature ok
subject=C = ES, ST = Catalunya, L = Barcelona, O = MISTIC, OU = users.mistic.lan, CN = user31,
emailAddress = user31@users.mistic.lan
Getting CA Private Key
```

Also, in the case of user certificates, to import them later into the user’s browsers it may be convenient to distribute them in PKCS12 format:

```
# openssl pkcs12 -export -in newcerts/user31.users.mistic.lan.crt \
  -inkey newcerts/user31.users.mistic.lan.key -out newcerts/user31.users.mistic.lan.p12
Enter Export Password:
Verifying - Enter Export Password:
```

The password included in this PKCS12 file is for security reasons to protect the processes of importing and exporting the certificate at the user’s machine.

9.2.3 Creating Java keystores for the Tomcat servers

The Tomcat certificates keystore

The Tomcat servers, both APP and CAS, need a keystore to hold their own certificate when working with the SSL/TLS connector to work with other machines.

Create RSA keys and the save them in a Java keystore (one for each server, that is **app.mistic.lan.jks** and **cas.mistic.lan.jks**) using the *Java keytool* utility:

```
$ keytool -genkeypair -alias cas.mistic.lan -keyalg RSA -keystore cas.mistic.lan.jks
Enter keystore password:
Re-enter new password:
What is your first and last name?
  [Unknown]:  cas.mistic.lan
What is the name of your organizational unit?
  [Unknown]:  CA
What is the name of your organization?
  [Unknown]:  MISTIC
What is the name of your City or Locality?
  [Unknown]:  Barcelona
What is the name of your State or Province?
  [Unknown]:  Catalunya
What is the two-letter country code for this unit?
  [Unknown]:  ES
Is CN=cas.mistic.lan, OU=CA, O=MISTIC, L=Barcelona, ST=Catalunya, C=ES correct?
  [no]:  yes
```

We use the password *mistic* and set the proper organizational values.

The MISTIC CA keystore

Besides the keystore for Tomcat server certificates, we need to create a second keystore to hold the trusted certificates, like our CA, that will be used to validate the received X.509 certificates (e.g., when an application at the APP server connects to the CAS server).

We import the *ca.mistic.lan.crt* certificate into a new Java keystore using the *Java keytool* utility:

```
$ keytool -import -alias ca.mistic.lan -file ca.mistic.lan.crt -keystore ca.mistic.lan.jks
Enter keystore password:
Re-enter new password:
Owner: EMAILADDRESS=support@mistic.lan, CN=ca.mistic.lan, OU=CA, O=MISTIC, L=Barcelona,
ST=Catalunya, C=ES
Issuer: EMAILADDRESS=support@mistic.lan, CN=ca.mistic.lan, OU=CA, O=MISTIC, L=Barcelona,
ST=Catalunya, C=ES
Serial number: 95a81684197c6a69
Valid from: Fri Nov 08 11:44:03 CET 2019 until: Mon Nov 05 11:44:03 CET 2029
Certificate fingerprints:
        SHA1: 94:4C:81:DF:30:15:64:DC:8D:E6:56:6B:E9:D4:DA:DC:E7:37:28:B7
        SHA256:
B6:61:37:D1:06:E0:BA:E3:AA:9E:07:A2:05:CB:9B:AF:53:91:E4:D4:E0:73:44:25:DF:FD:6C:AB:5C:70:E9:D3
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: DF B2 61 40 B7 4E 68 13   F4 85 9D 29 AF 1E 35 EA  ..a@.Nh....)..5.
0010: 6A AB CF DA                                        j...
]
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: DF B2 61 40 B7 4E 68 13   F4 85 9D 29 AF 1E 35 EA  ..a@.Nh....)..5.
0010: 6A AB CF DA                                        j...
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

We use the password *mistic* to access this keystore.

9.3 Installing Amazon Corretto 11

Download the package from Amazon.com in the development machine and copy the tarball in the desired server (e.g. app.mistic.lan or cas.mistic.lan):

```
$ scp amazon-corretto-11.0.4.11.1-linux-x64.tar.gz root@app:/opt
root@app's password:

amazon-corretto-11.0.4.11.1-linux-x64.tar.gz                           100%  183MB 132.5MB/s   00:01
```

Now log in this machine and uncompress the tarball in the /opt directory and rename it:

```
root@app# cd /opt
root@app:/opt# tar -xvf amazon-corretto-11.0.4.11.1-linux-x64
root@app:/opt# mv amazon-corretto-11.0.4.11.1-linux-x64 corretto-11
```

Create the file */etc/profile.d/corretto-11.sh* with the **JAVA_HOME** and **PATH** variables:

```
root@app# more /etc/profile.d/corretto-11.sh
export JAVA_HOME=/opt/corretto-11
export PATH=$PATH:$JAVA_HOME/bin
```

Log out and log in again to the server (or just source the new /etc/profile.d file), to check the installation of the Java runtime:

```
root@app# source /etc/profile.d/corretto-11.sh
root@app# java -version
openjdk version "11.0.4" 2019-07-16 LTS
OpenJDK Runtime Environment Corretto-11.0.4.11.1 (build 11.0.4+11-LTS)
OpenJDK 64-Bit Server VM Corretto-11.0.4.11.1 (build 11.0.4+11-LTS, mixed mode)
```

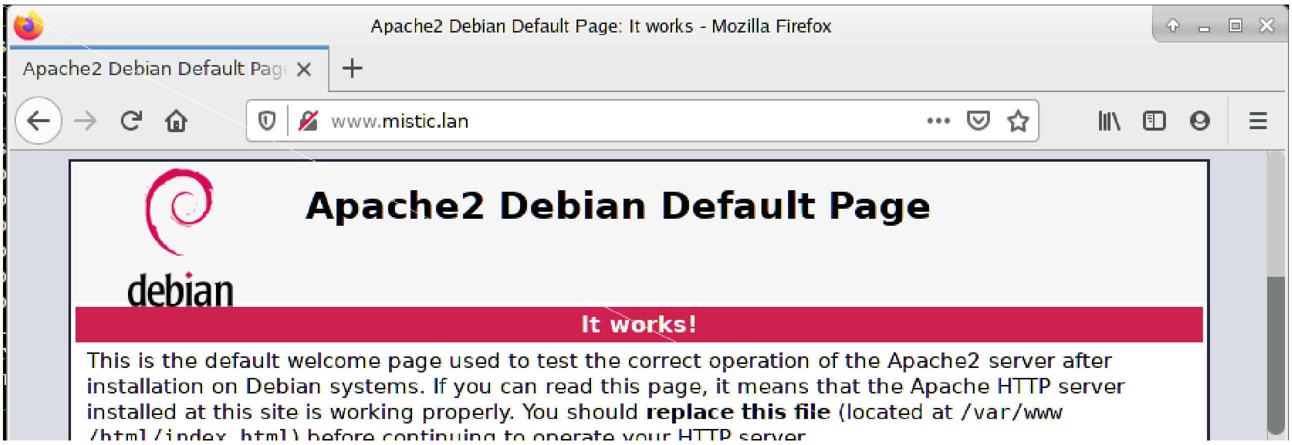
9.4 Installing Apache httpd as a reverse proxy

Installing the web server

Install the **Apache http webserver** package from the Debian repositories:

```
root@proxy:~# apt-get install apache2
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3 libaprutil1-
ldap libbrotli1 libcurl4 libjansson4 liblua5.2-0 ssl-cert
Suggested packages:
  apache2-doc apache2-suexec-pristine | apache2-suexec-custom www-browser openssl-blacklist
The following NEW packages will be installed:
  apache2 apache2-bin apache2-data apache2-utils libapr1 libaprutil1 libaprutil1-dbd-sqlite3
libaprutil1-ldap libbrotli1 libcurl4 libjansson4 liblua5.2-0 ssl-cert
0 upgraded, 13 newly installed, 0 to remove and 0 not upgraded.
Need to get 2,959 kB of archives.
After this operation, 9,663 kB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Test the webserver by connecting to **http://www.mistic.lan** from the development machine browser:


Apache2 Debian Default Page: It works - Mozilla Firefox
Apache2 Debian Default Page
www.mistic.lan
debian

Apache2 Debian Default Page

It works!

This is the default welcome page used to test the correct operation of the Apache2 server after installation on Debian systems. If you can read this page, it means that the Apache HTTP server installed at this site is working properly. You should **replace this file** (located at /var/www/html/index.html) before continuing to operate your HTTP server.

Enabling SSL/TLS

Enable the *Apache httpd SSL* modules:

```
root@proxy:~# a2enmod ssl
Considering dependency setenvif for ssl:
Module setenvif already enabled
Considering dependency mime for ssl:
Module mime already enabled
Considering dependency socache_shmcb for ssl:
Enabling module socache_shmcb.
Enabling module ssl.
See /usr/share/doc/apache2/README.Debian.gz on how to configure SSL and create self-signed
certificates.
```

In this module, disable TLSv1.3 because some common browsers like Firefox do not yet implement post-handsake authentication[57]. This would prevent a user entering an application that does not require a certificate to be able to provide a certificate in the same browser session (the user should close the browser application and start again with a clean session). In the file **/etc/apache/mods-enabled/ssl.conf** add an exception for protocol TLS v1.3:

```
        #   The protocols to enable.
        #   Available values: all, SSLv3, TLSv1, TLSv1.1, TLSv1.2
        #   SSL v2  is no longer supported
        SSLProtocol all -SSLv3 -TLSv1.3
```

[57] See, for example, https://bz.apache.org/bugzilla/show_bug.cgi?id=62975

Setting up the reverse proxy

Creating a website configuration

For the proxy server to perform the expected redirections, create a website configuration file **/etc/apache2/sites-available/www.mistic.lan.conf** with the contents shown at PROXY server configuration files, p. 70. This file defines a new website at the webserver, but note that this website only acts only as a proxy and does not serve any HTML document.

In this configuration all HTTP requests (port 80) are redirected to HTTPS (port 443):

```
<VirtualHost *:80>
     ServerName www.mistic.lan
     Redirect / https://www.mistic.lan/
</VirtualHost>
```

and a VirtualHost for port 8443 is created[58] just to redirect all connections to port 443. This is because the CAS server expects the work on the former but our APP server works on the latter.

Note that all proxy redirections to the APP server are done under HTTPS protocol[59] and are redirected to **app.mistic.lan**, except the URLs starting with **/cas** that are derived to the CAS application at **cas.mistic.lan/cas** using the AJP protocol at port 8009:

```
     ProxyPassReverse /cas ajp://cas.mistic.lan:8009/cas

     ProxyPassReverse /    https://app.mistic.lan:8443/
```

Finally, create the home directory for the website at */srv/www/mistic* with permissions for the Apache httpd user to write there the website logs:

```
root@proxy:~# mkdir -p /srv/www/mistic/logs
root@proxy:~# chown www-data /srv/www/mistic
```

Install also there, with the proper restricted permissions, the certificates created for both the Proxy server and CA authority as the files indicated in the SSL configuration section for the website:

```
     SSLEngine on
     SSLCertificateFile    /srv/www/mistic/www.mistic.lan.crt
     SSLCertificateKeyFile /srv/www/mistic/www.mistic.lan.key
     SSLCACertificateFile  /srv/www/mistic/ca.mistic.lan.crt
```

Configuring the reverse proxy

For the webserver to act as a reverse proxy, enable the *Apache proxy_httpd* and *proxy_ajp* modules and restart the web server :

```
root@proxy:~# a2enmod proxy_http
Considering dependency proxy for proxy_http:
Enabling module proxy.
Enabling module proxy_http.
root@proxy:~# a2enmod proxy_ajp
Considering dependency proxy for proxy_ajp:
Module proxy already enabled
Enabling module proxy_ajp.
root@proxy:~# systemctl apache2 restart
```

The first module enables the reverse proxy for HTTP/HTTPS connections while the second one enables the special Apache protocol AJP used to be able to pass to the CAS server the certificates received from the client.

Test again the web server with TLS/SSL by connecting to **https://www.mistic.lan** (instead of http://) from the development machine.

[58] Listening to this port 8443 has to be enabled in the **/etc/apache/ports.conf** file.

[59] Although in a protected environment the connections inside the internal network may be considered secure, there is no warranty that any server will not moved in the future to another location (like a cloud base data center) that could then require to configure a secured connection.

9.5 Installing Apache Tomcat

Installing Apache Tomcat

Download the package from apache.org at the development machine and copy the tarball in the desired machine:

```
$ scp apache-tomcat-9.0.26.tar.gz root@app:/opt
root@app's password:

apache-tomcat-9.0.26.tar.gz                                        100%   12MB 228.0MB/s   00:00
```

Uncompress the tarball at the remote server and rename the new extracted directory.

```
root@app# cd /opt
root@app# tar -xvf apache-tomcat-9.0.26.tar.gz
root@app# mv apache-tomcat-9.0.26 tomcat
```

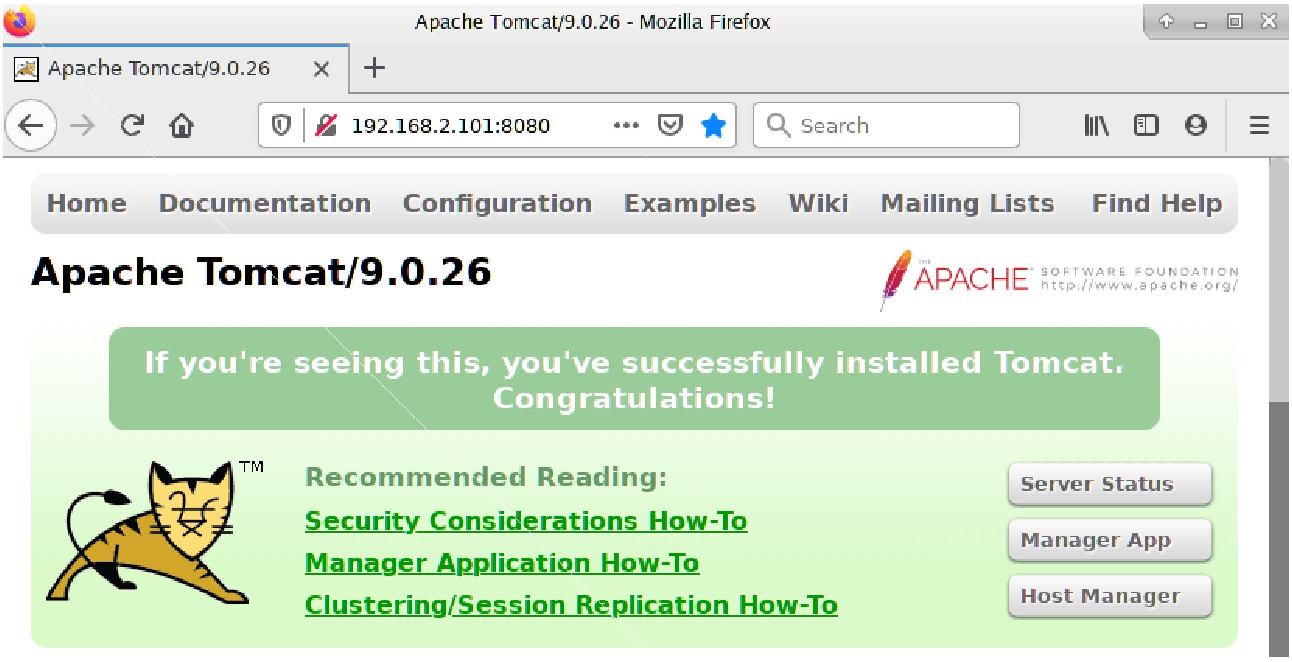
Create a dedicated user for the Tomcat server and change the owner of the Tomcat directory:

```
root@app# useradd -r -m -U -d /opt/tomcat -s /bin/false tomcat
root@app# chown -R tomcat tomcat
```

Create a systemd unit file as shown below and then enable and start the new service:

```
root@app# more /etc/systemd/system/tomcat.service
[Unit]
Description=Tomcat 9 Servlet Container
After=network.target

[Service]
Type=forking

User=tomcat

Environment="JAVA_HOME=/opt/corretto-11"
Environment="JAVA_OPTS=-Djava.security.egd=file:///dev/urandom"

Environment="CATALINA_BASE=/opt/tomcat"
Environment="CATALINA_HOME=/opt/tomcat"
Environment="CATALINA_PID=/opt/tomcat/temp/tomcat.pid"
Environment="CATALINA_OPTS=-Xms512M -Xmx1024M -server"

ExecStart=/opt/tomcat/bin/startup.sh
ExecStop=/opt/tomcat/bin/shutdown.sh

[Install]
WantedBy=multi-user.target

root@app# systemctl enable tomcat.service
Created symlink /etc/systemd/system/multi-user.target.wants/tomcat.service →
/etc/systemd/system/tomcat.service.

root@app# systemctl start tomcat
root@app# systemctl status tomcat
● tomcat.service - Tomcat 9 Servlet Container
   Loaded: loaded (/etc/systemd/system/tomcat.service; disabled; vendor preset: enabled)
   Active: active (running) since Wed 2019-10-23 10:08:54 UTC; 4min 40s ago
 Main PID: 1202 (java)
    Tasks: 48 (limit: 4701)
   Memory: 165.1M
   CGroup: /system.slice/tomcat.service
           └─1202 /opt/corretto-11/bin/java -Djava.util.logging.config.file=/opt/tomcat-
9/conf/logging.properties -Djava.util.logging.manager=org.apache.juli.ClassLoaderLogManager
-Djava.security.egd=file:///dev

Oct 23 10:08:54 app.mistic.lan systemd[1]: Starting Tomcat 9 Servlet Container...
Oct 23 10:08:54 app.mistic.lan startup.sh[1195]: Tomcat started.
Oct 23 10:08:54 app.mistic.lan systemd[1]: Started Tomcat 9 Servlet Container.
```

The server now should respond to direct requests at port 8080:



Apache Tomcat/9.0.26

If you're seeing this, you've successfully installed Tomcat. Congratulations!

Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To

Configuring Apache Tomcat for remote administration

Optionally, for development purposes but also to be able to administer the application server remotely, modify the configuration file */opt/tomcat/conf/tomcat-users.xml* to provide Tomcat administration roles to a new user, for example named *admin* with password *tomcat*:

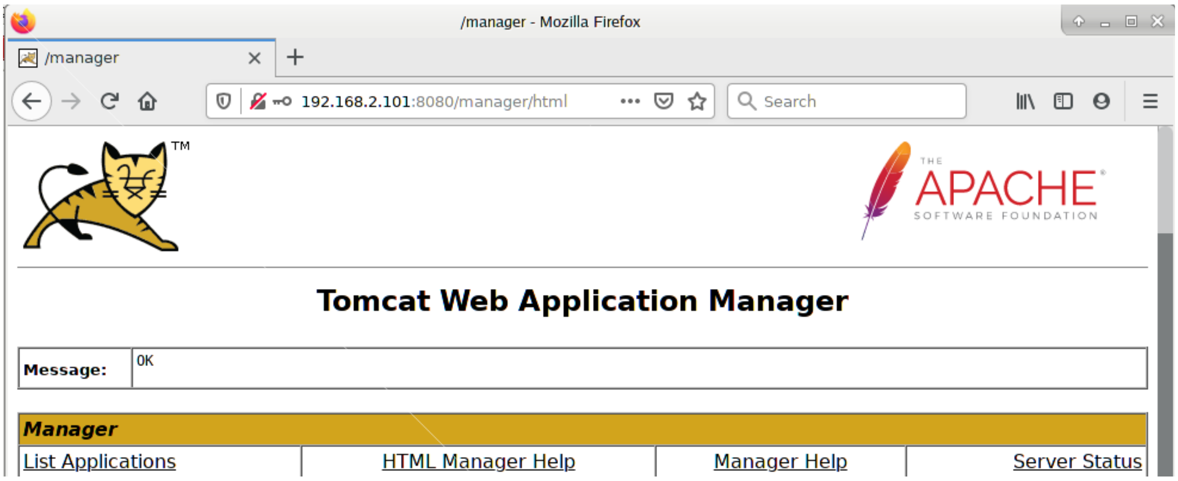
```
<role rolename="admin-gui,manager-gui"/>
<user username="admin" password="tomcat" roles="admin-gui,manager-gui"/>
```

and also modify the file */opt/tomcat/webapps/manager/META-INF/context.xml* to allow remote access from external network machines addresses:

```
Context antiResourceLocking="false" privileged="true" >
  <Valve className="org.apache.catalina.valves.RemoteAddrValve"
         allow="192.168.\d+.\d+|127\.\d+\.\d+\.\d+|::1|0:0:0:0:0:0:0:1" />
```

Restart the Tomcat service now to be able to administer the server remotely at the address **http://192.168.2.101:8080/manager**:

```
# systemctl restart tomcat
```


/manager - Mozilla Firefox
/manager
192.168.2.101:8080/manager/html
Search
TM
THE APACHE SOFTWARE FOUNDATION

Tomcat Web Application Manager

Message:	OK

Manager			
List Applications	HTML Manager Help	Manager Help	Server Status

Configuring Apache Tomcat for SSL connections

Configuring APP server's Apache Tomcat for SSL

Log in to the Tomcat server and enable the Catalina connector for port 8443, by setting the following entry in the file **/opt/tomcat/conf/server.xml** in the Connector section, to point to the right certificate keystores:

```
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" scheme="https" secure="true" SSLEnabled="true"
               keystoreFile="conf/app.mistic.lan.jks" keystorePass="mistic"
               clientAuth="want" sslProtocol="TLS" />
```

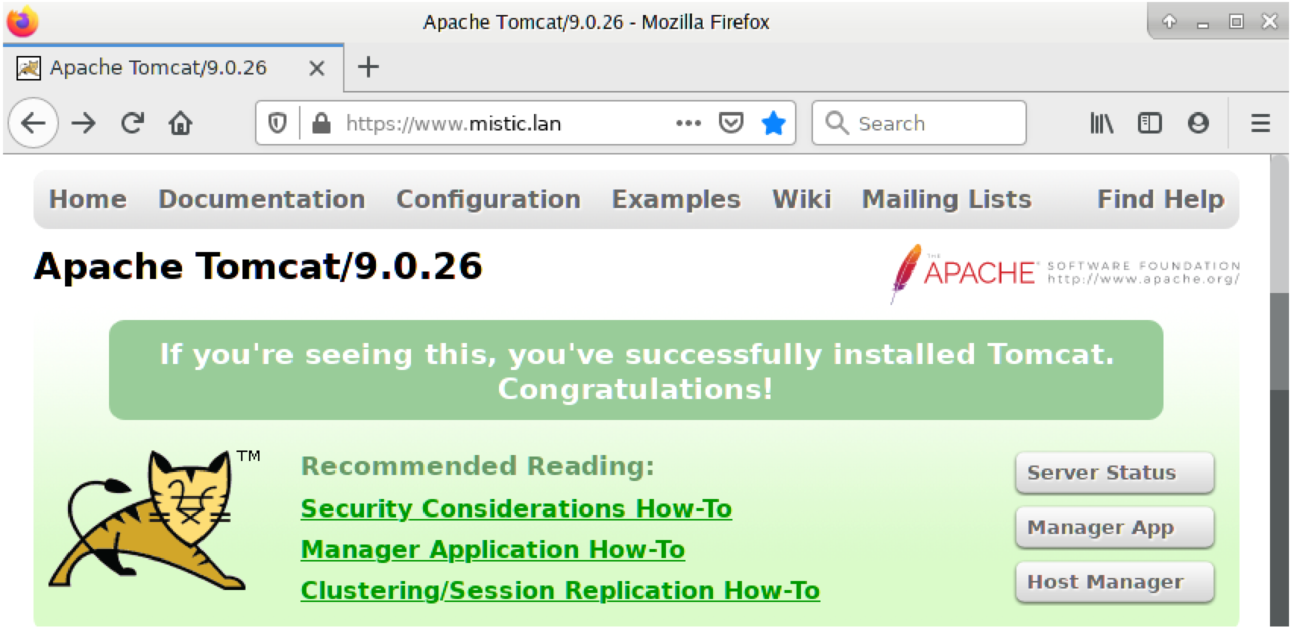
or. for the CAS server, add also the trusted keystore for the CA certificate:

```
    <Connector port="8443" protocol="org.apache.coyote.http11.Http11NioProtocol"
               maxThreads="150" scheme="https" secure="true" SSLEnabled="true"
               keystoreFile="conf/cas.mistic.lan.jks" keystorePass="mistic"
               truststoreFile="conf/ca.mistic.lan.jks" truststorePass="mistic"
               clientAuth="want" sslProtocol="TLS" />
```

and restart the Tomcat server to allow SSL connections:

```
# systemctl restart tomcat
```

Now, for example, the APP server is accessible via the proxy at the URL https://www.mistic.lan:


Apache Tomcat/9.0.26 - Mozilla Firefox
Apache Tomcat/9.0.26
https://www.mistic.lan
Search
Home
Documentation
Configuration
Examples
Wiki
Mailing Lists
Find Help
Apache Tomcat/9.0.26
APACHE SOFTWARE FOUNDATION http://www.apache.org/
If you're seeing this, you've successfully installed Tomcat. Congratulations!
TM
Recommended Reading:
Security Considerations How-To
Manager Application How-To
Clustering/Session Replication How-To
Server Status
Manager App
Host Manager

The DMZ proxy redirects the connection from the user's browser to **www.mistic.lan** to the internal APP server at **app.mistic.lan**, while using a secure connection with certificates signed by the MISTIC CA authority.

Adding the CA root certificate to the Java Runtime keystore at the CAS server

In addition to configuring the Tomcat container in the CAS server to be able to receive TLS/SSL connections from the APP server, the CAS server itself connects to the LDAP server. To allow encrypted connections when the CAS server is a TLS/SSL client, import the MISTIC CA root certificate into the global Java Amazon Corretto 11 keystore (located at /opt/corretto-11/lib/security/cacerts with a standard password *changeit*) to be able to validate any other certificate signed by our CA authority:

```
root@cas:/etc/cas# keytool -import -alias ca.mistic.lan -file ca.mistic.lan.crt -cacerts
Enter keystore password:
Owner: EMAILADDRESS=support@mistic.lan, CN=ca.mistic.lan, OU=CA, O=MISTIC, L=Barcelona,
ST=Catalunya, C=ES
```

```
Issuer: EMAILADDRESS=support@mistic.lan, CN=ca.mistic.lan, OU=CA, O=MISTIC, L=Barcelona,
ST=Catalunya, C=ES
Serial number: 95a81684197c6a69
Valid from: Fri Nov 08 10:44:03 UTC 2019 until: Mon Nov 05 10:44:03 UTC 2029
Certificate fingerprints:
        SHA1: 94:4C:81:DF:30:15:64:DC:8D:E6:56:6B:E9:D4:DA:DC:E7:37:28:B7
        SHA256:
B6:61:37:D1:06:E0:BA:E3:AA:9E:07:A2:05:CB:9B:AF:53:91:E4:D4:E0:73:44:25:DF:FD:6C:AB:5C:70:E9:D3
Signature algorithm name: SHA256withRSA
Subject Public Key Algorithm: 4096-bit RSA key
Version: 3

Extensions:

#1: ObjectId: 2.5.29.35 Criticality=false
AuthorityKeyIdentifier [
KeyIdentifier [
0000: DF B2 61 40 B7 4E 68 13   F4 85 9D 29 AF 1E 35 EA  ..a@.Nh....)..5.
0010: 6A AB CF DA                                        j...
]
]

#2: ObjectId: 2.5.29.19 Criticality=true
BasicConstraints:[
  CA:true
  PathLen:2147483647
]

#3: ObjectId: 2.5.29.14 Criticality=false
SubjectKeyIdentifier [
KeyIdentifier [
0000: DF B2 61 40 B7 4E 68 13   F4 85 9D 29 AF 1E 35 EA  ..a@.Nh....)..5.
0010: 6A AB CF DA                                        j...
]
]

Trust this certificate? [no]:  yes
Certificate was added to keystore
```

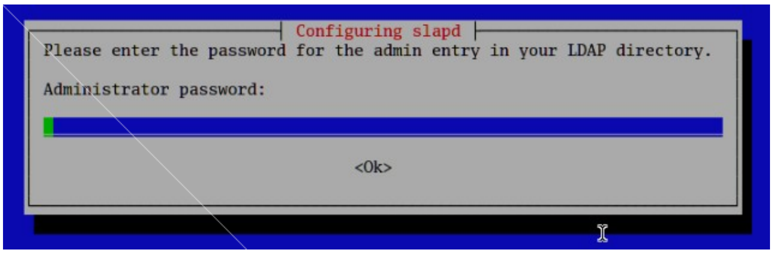
9.6 Installing OpenLDAP

Installing the directory service

Install the OpenLDAP packages for the server and utilities from the Debian repositories[60]:

```
root@ldap# apt-get install slapd ldap-utils
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  libodbc1
Suggested packages:
  libmyodbc odbc-postgresql tdsodbc unixodbc-bin ldap-utils libsasl2-modules-gssapi-mit | libsasl2-
modules-gssapi-heimdal
The following NEW packages will be installed:
  libodbc1 slapd
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 1,659 kB of archives.
After this operation, 16.7 MB of additional disk space will be used.
Do you want to continue? [Y/n]
```

Enter the password *mistic* for the LDAP administrator user:



```
Configuring slapd
Please enter the password for the admin entry in your LDAP directory.

Administrator password:

<Ok>
```

[60] To use the Debian repositories from the Internet, a temporary network adapter can be configured bridged to the external network just to download and install the software and then can be removed from this server.

Once the installation completes:

```
Get:1 http://ftp.cica.es/debian buster/main amd64 libodbc1 amd64 2.3.6-0.1 [223 kB]
Get:2 http://ftp.cica.es/debian buster/main amd64 slapd amd64 2.4.47+dfsg-3+deb10u1 [1,436 kB]
Fetched 1,659 kB in 1s (2,657 kB/s)
Preconfiguring packages ...
Selecting previously unselected package libodbc1:amd64.
(Reading database ... 33644 files and directories currently installed.)
Preparing to unpack .../libodbc1_2.3.6-0.1_amd64.deb ...
Unpacking libodbc1:amd64 (2.3.6-0.1) ...
Selecting previously unselected package slapd.
Preparing to unpack .../slapd_2.4.47+dfsg-3+deb10u1_amd64.deb ...
Unpacking slapd (2.4.47+dfsg-3+deb10u1) ...
Setting up libodbc1:amd64 (2.3.6-0.1) ...
Setting up slapd (2.4.47+dfsg-3+deb10u1) ...
  Creating new user openldap... done.
  Creating initial configuration... done.
  Creating LDAP directory... done.
Processing triggers for systemd (241-7~deb10u1) ...
Processing triggers for man-db (2.8.5-2) ...
Processing triggers for libc-bin (2.28-10) ...
```

check that the service is installed and running:

```
# systemctl status slapd

● slapd.service - LSB: OpenLDAP standalone server (Lightweight Directory Access Protocol)
   Loaded: loaded (/etc/init.d/slapd; generated)
   Active: active (running) since Wed 2019-10-16 07:56:42 UTC; 3min 35s ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 3 (limit: 4701)
   Memory: 3.5M
   CGroup: /system.slice/slapd.service
           └─1933 /usr/sbin/slapd -h ldap:/// ldapi:/// -g openldap -u openldap -F /etc/ldap/slapd.d
```

Now that LDAP is installed, use the Debian command *dpkg-reconfigure* to initialize a new scheme and replace the default one:

```
# dpkg-reconfigure slapd
```

Entering the values for the project when prompted, create a simple organization **mistic.lan** and a group of users **user.mistic.lan.** This organization is just for testing purposes.

Finally. test the LDAP directory with the *slapcat* command:

```
# slapcat
dn: dc=mistic,dc=lan
objectClass: top
objectClass: dcObject
objectClass: organization
o: MISTIC
dc: mistic
structuralObjectClass: organization
entryUUID: 26a3f3ce-a094-1039-8579-fddcb6c02024
creatorsName: cn=admin,dc=mistic,dc=lan
createTimestamp: 20191121101931Z
entryCSN: 20191121101931.138999Z#000000#000#000000
modifiersName: cn=admin,dc=mistic,dc=lan
modifyTimestamp: 20191121101931Z

dn: ou=users,dc=mistic,dc=lan
objectClass: organizationalUnit
ou: application users
ou: users
structuralObjectClass: organizationalUnit
entryUUID: c9dae318-a094-1039-85a2-e916b965a1a1
creatorsName: cn=admin,dc=mistic,dc=lan
createTimestamp: 20191121102404Z
entryCSN: 20191121102404.967672Z#000000#000#000000
modifiersName: cn=admin,dc=mistic,dc=lan
modifyTimestamp: 20191121102404Z
```

Enabling TLS/SSL connections for the LDAP server

To allow the LDAP server to establish TLS/SSL connections with LDAP clients (the CAS server in this case), copy the certificates created for the LDAP server and also the CA authority certificate at the **/etc/ldap/certs** directory.

Then modify the LDAP server configuration by creating an LDIF file (e.g. **/etc/ldap/sdap.d/sldapssl.ldif**) with the values for these certificates locations:

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/certs/ldap.mistic.lan.key
-
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/certs/ldap.mistic.lan.crt
-
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ldap/certs/ca.mistic.lan.crt
```

and issue the command *ldapmodify* to load this new configuration to the LDAP server:

```
root@ldap:/etc/ldap/slap.d# ldapmodify -Y EXTERNAL -H ldapi:/// -f slapdssl.ldif
SASL/EXTERNAL authentication started
SASL username: gidNumber=0+uidNumber=0,cn=peercred,cn=external,cn=auth
SASL SSF: 0
modifying entry "cn=config"
```

Now the LDAP server has its own MISTIC certificate installed to handle TLS/SSL connections plus the MISTIC CA certificate to validate the certificate received from the CAS server when acting as an LDAP client.

To enable the LDAP protocols to be used by the service, edit the file **/etc/default/sldapd** to allow connections via the secured channel:

```
# 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="ldaps:/// ldap:/// ldapi:///"
```

and restart the service and check the the LDAP service is listening to port 636 (ldaps://) as well as port 398 (ldap://):

```
root@ldap# systemctl restart sladp

root@ldap# ss -ntl4
State                 Recv-Q         Send-Q       Local Address:Port        Peer Address:Port
LISTEN                0              128          0.0.0.0:22                0.0.0.0:*
LISTEN                0              20           127.0.0.1:25              0.0.0.0:*
LISTEN                0              128          0.0.0.0:636               0.0.0.0:*
LISTEN                0              128          0.0.0.0:389               0.0.0.0:*
```

9.7 Installing Apereo CAS server

Building and installing the CAS service

Download the CAS overlay template from the GitHub repository for Apereo projects[61] and unzip the ZIP file at the development machine:

```
$ unzip cas-overlay-template-6.1.zip
Archive:  cas-overlay-template-6.1.zip
83e0044cc91250393462396f665c67ef25566c0d
   creating: cas-overlay-template-6.1/
  inflating: cas-overlay-template-6.1/.gitattributes
   creating: cas-overlay-template-6.1/.github/
  inflating: cas-overlay-template-6.1/.github/FUNDING.yml
```

[61] https://github.com/apereo/cas-overlay-template/tree/6.1

```
  inflating: cas-overlay-template-6.1/.gitignore
  inflating: cas-overlay-template-6.1/.mergify.yml
  inflating: cas-overlay-template-6.1/.travis.yml
  inflating: cas-overlay-template-6.1/Dockerfile
  inflating: cas-overlay-template-6.1/LICENSE.txt
  inflating: cas-overlay-template-6.1/README.md
  inflating: cas-overlay-template-6.1/build.gradle
  inflating: cas-overlay-template-6.1/docker-build.sh
  inflating: cas-overlay-template-6.1/docker-compose.yml
  inflating: cas-overlay-template-6.1/docker-push.sh
  inflating: cas-overlay-template-6.1/docker-run.sh
   creating: cas-overlay-template-6.1/etc/
   creating: cas-overlay-template-6.1/etc/cas/
   creating: cas-overlay-template-6.1/etc/cas/config/
  inflating: cas-overlay-template-6.1/etc/cas/config/cas.properties
  inflating: cas-overlay-template-6.1/etc/cas/config/log4j2.xml
   creating: cas-overlay-template-6.1/etc/cas/saml/
  inflating: cas-overlay-template-6.1/etc/cas/saml/.gitkeep
   creating: cas-overlay-template-6.1/etc/cas/services/
 extracting: cas-overlay-template-6.1/etc/cas/services/.donotdel
  inflating: cas-overlay-template-6.1/etc/cas/thekeystore
  inflating: cas-overlay-template-6.1/gradle.properties
   creating: cas-overlay-template-6.1/gradle/
  inflating: cas-overlay-template-6.1/gradle/dockerjib.gradle
  inflating: cas-overlay-template-6.1/gradle/springboot.gradle
  inflating: cas-overlay-template-6.1/gradle/tasks.gradle
   creating: cas-overlay-template-6.1/gradle/wrapper/
  inflating: cas-overlay-template-6.1/gradle/wrapper/gradle-wrapper.jar
  inflating: cas-overlay-template-6.1/gradle/wrapper/gradle-wrapper.properties
  inflating: cas-overlay-template-6.1/gradlew
  inflating: cas-overlay-template-6.1/gradlew.bat
 extracting: cas-overlay-template-6.1/settings.gradle
   creating: cas-overlay-template-6.1/src/
   creating: cas-overlay-template-6.1/src/main/
   creating: cas-overlay-template-6.1/src/main/jib/
   creating: cas-overlay-template-6.1/src/main/jib/docker/
  inflating: cas-overlay-template-6.1/src/main/jib/docker/entrypoint.sh
```

Then change into the new extracted directory to build the actual WAR to deploy:

```
$ cd /opt/cas-overlay-template-6.1
```

First, add the needed dependencies in the **build.gradle** file, to be able to:

- load CAS services from a JSON configuration file,
- add support for LDAP authentication,
- add support for X.509 authentication

```
dependencies {
    // Other CAS dependencies/modules may be listed here...
    compile "org.apereo.cas:cas-server-support-json-service-registry:${casServerVersion}"
    compile "org.apereo.cas:cas-server-support-ldap:${project.'cas.version'}"
    compile "org.apereo.cas:cas-server-support-x509-webflow:${project.'cas.version'}"
}
```

Once the template is configured for the SSO project, issue the *gradlew build* command to compile and build the WAR file (the gradle tool will be installed if not available):

```
$ gradlew build
Starting a Gradle Daemon (subsequent builds will be faster)
> Task :compileJava NO-SOURCE
> Task :processResources NO-SOURCE
> Task :classes UP-TO-DATE
> Task :extractCasBootWarOverlay UP-TO-DATE
> Task :bootWar UP-TO-DATE
> Task :war SKIPPED
> Task :assemble UP-TO-DATE
> Task :compileTestJava NO-SOURCE
> Task :processTestResources NO-SOURCE
> Task :testClasses UP-TO-DATE
> Task :test NO-SOURCE
> Task :check UP-TO-DATE
> Task :build UP-TO-DATE

Deprecated Gradle features were used in this build, making it incompatible with Gradle 6.0.
```

```
Use '--warning-mode all' to show the individual deprecation warnings.
See https://docs.gradle.org/5.6.3/userguide/command_line_interface.html#sec:command_line_warnings

BUILD SUCCESSFUL in 1s
2 actionable tasks: 2 up-to-date
```

Now copy the default CAS configuration folder ***etc/cas*** in the CAS server **/etc** directory.

```
$ scp -r etc/cas root@cas:/etc
root@cas's password:
log4j2.xml                                                            100% 7217     4.6MB/s   00:00
cas.properties                                                        100%  162   144.0KB/s   00:00
.gitkeep                                                              100%   71    65.7KB/s   00:00
.donotdel                                                             100%    0     0.0KB/s   00:00
thekeystore                                                           100% 2266     1.8MB/s   00:00
```

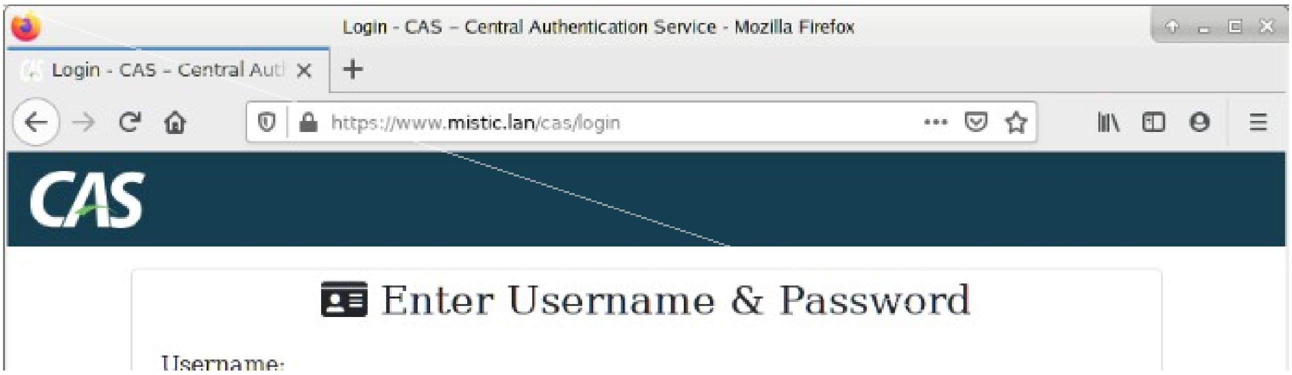
and, finally, deploy the WAR file named **cas.war** in the directory **build/libs.** This is the CAS application ready to be deployed to the CAS Tomcat server as a regular application. Simply copy the WAR file in the **/opt/tomcat/webapps** directory in the CAS server:

```
$ scp build/libs/cas.war root@cas:/opt/tomcat/webapps
root@cas.mistic.lan's password:
cas.war                                                              100%   90MB 160.3MB/s   00:00
```

and the application will be automatically deployed and run by the Tomcat server.

Testing the CAS server

Test the CAS application running in this Tomcat server and check that is accepting connections via the DMZ proxy from the URL **https://www.mistic.lan/cas.** Upon receiving the connection the CAS application redirects the user to the default login page:


Login - CAS – Central Authentication Service - Mozilla Firefox
Login - CAS – Central Aut
https://www.mistic.lan/cas/login
CAS
Enter Username & Password
Username:

10 Appendix 2: Source code and configuration files

10.1 PROXY server configuration files

/etc/hosts

```
127.0.0.1       localhost
192.168.1.200   www.mistic.lan          www

192.168.2.101   app.mistic.lan          app
192.168.2.111   cas.mistic.lan          cas

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

/etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
       address 192.168.2.200/24
allow-hotplug enp0s8
iface enp0s8 inet static
       address 192.168.1.200/24
       gateway 192.168.1.1
       # dns-* options are implemented by the resolvconf package, if installed
       dns-nameservers 1.1.1.1
       dns-search mistic.lan
```

/etc/apache2/sites-available/www.mistic.lan.conf

```
<VirtualHost *:80>
     ServerName www.mistic.lan
     Redirect / https://www.mistic.lan/
</VirtualHost>
<VirtualHost *:8443>
     ServerName www.mistic.lan

     SSLEngine on
     SSLCertificateFile /srv/www/mistic/www.mistic.lan.crt
     SSLCertificateKeyFile /srv/www/mistic/www.mistic.lan.key
     SSLCACertificateFile /srv/www/mistic/ca.mistic.lan.crt

     <Location /app3>
       SSLVerifyClient      optional
       SSLVerifyDepth       2

       SSLOptions +ExportCertData
     </Location>

     <Location /app4>
       SSLVerifyClient      optional
       SSLVerifyDepth       2

       SSLOptions +ExportCertData
     </Location>

     Redirect / https://www.mistic.lan/
```

```
</VirtualHost>

<VirtualHost *:443>
     ServerName www.mistic.lan

     DocumentRoot /srv/www/mistic/html/

     SSLEngine on
     SSLCertificateFile /srv/www/mistic/www.mistic.lan.crt
     SSLCertificateKeyFile /srv/www/mistic/www.mistic.lan.key
     SSLCACertificateFile /srv/www/mistic/ca.mistic.lan.crt

     <Location /app3>
       SSLVerifyClient      optional
       SSLVerifyDepth       2

       SSLOptions +ExportCertData
     </Location>

     <Location /app4>
       SSLVerifyClient      optional
       SSLVerifyDepth       2

       SSLOptions +ExportCertData
     </Location>

     SSLProxyEngine on

     ProxyPass        /cas ajp://cas.mistic.lan:8009/cas
     ProxyPassReverse /cas ajp://cas.mistic.lan:8009/cas

     ProxyPass        / https://app.mistic.lan:8443/
     ProxyPassReverse / https://app.mistic.lan:8443/

     ErrorLog /srv/www/mistic/logs/error.log
     CustomLog /srv/www/mistic/logs/access.log combined
</VirtualHost>
```

/etc/apache/ports.conf

```
# If you just change the port or add more ports here, you will likely also
# have to change the VirtualHost statement in
# /etc/apache2/sites-enabled/000-default.conf

Listen 80

<IfModule ssl_module>
        Listen 443
        Listen 8443
</IfModule>

<IfModule mod_gnutls.c>
        Listen 443
        Listen 8443
</IfModule>

# vim: syntax=apache ts=4 sw=4 sts=4 sr noet
```

10.2 APP server configuration files

/etc/hosts

```
127.0.0.1       localhost
192.168.2.101   app.mistic.lan

192.168.1.200   www.mistic.lan
192.168.2.111   cas.mistic.lan

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

/etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
        address 192.168.2.101/24
        gateway 192.168.2.200
```

10.3 CAS server configuration files

/etc/hosts

```
127.0.0.1       localhost
192.168.2.101   app.mistic.lan          app
192.168.2.111   cas.mistic.lan          cas
192.168.2.200   proxy.mistic.lan        proxy

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

/etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
        address 192.168.2.111/24
        gateway 192.168.2.200
```

/etc/cas/config/cas.properties

```
#========================================
# CAS Server
#========================================

cas.server.name=https://www.mistic.lan
cas.server.prefix=${cas.server.name}/cas
cas.host.name=cas.mistic.lan

#========================================
# Logging
#========================================

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

#========================================
# Service Registry
#========================================

#cas.serviceRegistry.initFromJson=true
cas.serviceRegistry.json.location=file:///etc/cas/services
cas.serviceRegistry.watcherEnabled=true
```

```
#========================================
# Authentication
#========================================

#----------------------------------------
# Static authentication
#----------------------------------------

# Disable static authentication
cas.authn.accept.users=

#----------------------------------------
# X.509 Authentication
#----------------------------------------

cas.authn.x509.principalType=SUBJECT
cas.authn.x509.principalDescriptor=$CN

#----------------------------------------
# LDAP Authentication
#----------------------------------------

cas.authn.ldap[0].ldapUrl=ldaps://ldap.mistic.lan

cas.authn.ldap[0].connectionStrategy=ACTIVE_PASSIVE
cas.authn.ldap[0].useSsl=true
cas.authn.ldap[0].useStartTls=false
cas.authn.ldap[0].connectTimeout=5000

cas.authn.ldap[0].type=DIRECT
cas.authn.ldap[0].bindDn=uid=admin,ou=ca,dc=mistic,dc=lan
cas.authn.ldap[0].bindCredential=mistic

cas.authn.ldap[0].baseDn=ou=users,dc=mistic,dc=lan
cas.authn.ldap[0].subtreeSearch=true
cas.authn.ldap[0].searchFilter=uid={user}

cas.authn.ldap[0].enhanceWithEntryResolver=true

cas.authn.ldap[0].dnFormat=uid=%s,ou=users,dc=mistic,dc=lan

cas.authn.ldap[0].principalAttributeId=uid
cas.authn.ldap[0].principalAttributePassword=userPassword

#========================================
# Attribute retrieval
#========================================

cas.personDirectory.principalAttribute=cn
cas.personDirectory.returnNull=false

cas.authn.attributeRepository.merger=MERGE

#----------------------------------------
# LDAP attributes to retrieve
#----------------------------------------

cas.authn.attributeRepository.ldap[0].order=0
cas.authn.attributeRepository.ldap[0].ldapUrl=ldaps://ldap.mistic.lan
cas.authn.attributeRepository.ldap[0].useSsl=true
cas.authn.attributeRepository.ldap[0].useStartTls=false

cas.authn.attributeRepository.ldap[0].bindDn=uid=admin,ou=ca,dc=mistic,dc=lan
cas.authn.attributeRepository.ldap[0].bindCredential=mistic

cas.authn.attributeRepository.ldap[0].baseDn=ou=users,dc=mistic,dc=lan
cas.authn.attributeRepository.ldap[0].searchFilter=uid={0}

cas.authn.ldap[0].principalAttributeList=cn,uid,employeeType
```

/etc/cas/config/services/App1-1111.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "https://www\\.mistic\\.lan:8443/app1.*",
  "name" : "App1",
  "id" : 1111,
  "evaluationOrder" : 1,
  "logoutType" : "FRONT_CHANNEL",
  "logoutUrl" : "https://www.mistic.lan/app1/logout_cas"
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
   },
  "accessStrategy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
    "requiredAttributes" : {
      "@class" : "java.util.HashMap",
      "employeeType" : [ "java.util.HashSet", [ "10" "20" "30" "40" ] ]
    }
  }
}
```

/etc/cas/config/services/App2-2222.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "https://www\\.mistic\\.lan:8443/app2.*",
  "name" : "App2",
  "id" : 2222,
  "evaluationOrder" : 2,
  "logoutType" : "FRONT_CHANNEL",
  "logoutUrl" : "https://www.mistic.lan/app2/logout_cas"
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
   },
  "accessStrategy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
    "requiredAttributes" : {
      "@class" : "java.util.HashMap",
      "employeeType" : [ "java.util.HashSet", [ "20" "30" "40" ] ]
    }
  }
}
```

/etc/cas/config/services/App3-3333.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "https://www\\.mistic\\.lan:8443/app3.*",
  "name" : "App3",
  "id" : 3333,
  "evaluationOrder" : 3,
  "logoutType" : "FRONT_CHANNEL",
  "logoutUrl" : "https://www.mistic.lan/app3/logout_cas"
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
   },
  "accessStrategy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
    "requiredAttributes" : {
      "@class" : "java.util.HashMap",
      "employeeType" : [ "java.util.HashSet", [ "30" "40" ] ]
    }
  }
}
```

/etc/cas/config/services/App4-4444.json

```
{
  "@class" : "org.apereo.cas.services.RegexRegisteredService",
  "serviceId" : "https://www\\.mistic\\.lan:8443/app4.*",
  "name" : "App4",
  "id" : 4444,
  "evaluationOrder" : 4,
  "logoutType" : "FRONT_CHANNEL",
  "logoutUrl" : "https://www.mistic.lan/app4/logout_cas"
  "attributeReleasePolicy" : {
    "@class" : "org.apereo.cas.services.ReturnAllAttributeReleasePolicy"
   },
  "accessStrategy" : {
    "@class" : "org.apereo.cas.services.DefaultRegisteredServiceAccessStrategy",
    "requiredAttributes" : {
      "@class" : "java.util.HashMap",
      "employeeType" : [ "java.util.HashSet", [ "40" ] ]
    }
  }
}
```

10.4 LDAP server configuration files

/etc/hosts

```
127.0.0.1       localhost
192.168.2.110   cas.mistic.lan          cas
192.168.2.111   ldap.mistic.lan         ldap
192.168.2.200   proxy.mistic.lan        proxy

# The following lines are desirable for IPv6 capable hosts
::1     localhost ip6-localhost ip6-loopback
ff02::1 ip6-allnodes
ff02::2 ip6-allrouters
```

/etc/network/interfaces

```
# This file describes the network interfaces available on your system
# and how to activate them. For more information, see interfaces(5).

source /etc/network/interfaces.d/*

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
allow-hotplug enp0s3
iface enp0s3 inet static
        address 192.168.2.101/24
        gateway 192.168.2.200
```

/etc/ldap/slap.d/slapdssl.ldif

```
dn: cn=config
changetype: modify
replace: olcTLSCertificateKeyFile
olcTLSCertificateKeyFile: /etc/ldap/certs/ldap.mistic.lan.key
-
replace: olcTLSCertificateFile
olcTLSCertificateFile: /etc/ldap/certs/ldap.mistic.lan.crt
-
replace: olcTLSCACertificateFile
olcTLSCACertificateFile: /etc/ldap/certs/ca.mistic.lan.crt
```

/etc/ldap/slap.d/lan.mistic.ldif

This file contains the actual list organizational units and users for the MISTIC organization:

```
### MISTIC LDIF file

## MISTIC.LAN
dn: dc=mistic,dc=lan
objectClass: top
objectClass: dcObject
objectClass: organization
dc: mistic
o: MISTIC

## CA
dn: ou=ca,dc=mistic,dc=lan
objectClass: organizationalUnit
ou: MISTIC CA
```

```
dn: uid=admin,ou=ca,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: admin
userPassword: mistic
cn: admin
sn: Admin

## USERS
dn: ou=users,dc=mistic,dc=lan
objectClass: organizationalUnit
ou: Users

dn: uid=user10,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user10
userPassword: user10
cn: user10
sn: User 10
employeeType: 10

dn: uid=user20,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user20
userPassword: user20
cn: user20
sn: User 20
employeeType: 20

dn: uid=user30,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user30
userPassword: user30
cn: user30
sn: User 30
employeeType: 30

dn: uid=user31,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user31
userPassword: e1NIQX1zTmF4OGR2MVhXSmhoWU9mK05lQnpMN1BsVms=
cn: user31
sn: User 31
employeeType: 30

dn: uid=user40,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user40
userPassword: user40
cn: user40
sn: User 40
employeeType: 40

dn: uid=user41,ou=users,dc=mistic,dc=lan
objectClass: inetOrgPerson
uid: user41
userPassword: e1NIQX1zTmF4OGR2MVhXSmhoWU9mK05lQnpMN1BsVms=
cn: user41
sn: User 41
employeeType: 40
```

lan.mistic.sh

This scheme can be loaded as many times as needed using an script like this at the LDAP server:

```
#!/bin/bash

# Clean directory
echo "Deleting previous organization..."
ldapdelete -v -D cn=admin,dc=mistic,dc=lan -x -w mistic -r dc=mistic,dc=lan
echo

# Create LDAP scheme
echo "Creating MISTIC organization..."
ldapadd -v -D cn=admin,dc=mistic,dc=lan -x -w mistic -c -f lan.mistic.ldif
echo
```

10.5 Applications source code

10.5.1 Home page

The home page is a simple HTML file that links to the applications installed in the server.

apps/WebContent/index.html

```
<!DOCTYPE HTML>
<html>
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta charset="utf-8">
    <title>MISTIC CORPORATE SITE</title>
</head>
<body>
  <h1>Welcome to the MISTIC Corporate Site</h1>
  <h4>running at app.mistic.lan</h4>

  <h3>App1</h3>
  <a target="_blank" href="https://www.mistic.lan/app1">Enter App1</a>

  <h3>App2</h3>
  <a target="_blank" href="https://www.mistic.lan/app2">Enter App2</a>

  <h3>App3</h3>
  <a target="_blank" href="https://www.mistic.lan/app3">Enter App3</a>

  <h3>App4</h3>
  <a target="_blank" href="https://www.mistic.lan/app4">Enter App4</a>

</html>
```

10.5.2 Client applications

The Java applications developed are all very similar, because we included no specific business logic besides configuring them as CAS client Java applications. The exception is App4, which has an additional requirement that needs specific code, as shown later.

As an example of a Java CAS client application, these are the files needed to create the App2 client application (App1 and App3 are similar):

app2/src/main/java/lan/mistic/app2/App2Application.java

```
package lan.mistic.app1;

import org.jasig.cas.client.boot.configuration.EnableCasClient;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;

@SpringBootApplication
@EnableCasClient
public class App1Application {

        public static void main(String[] args) {
                SpringApplication.run(App1Application.class, args);
        }

}
```

app2/src/main/java/lan/mistic/app2/ServletInitializer.java

```
package lan.mistic.app1;

import org.springframework.boot.builder.SpringApplicationBuilder;
import org.springframework.boot.web.servlet.support.SpringBootServletInitializer;

public class ServletInitializer extends SpringBootServletInitializer {
        @Override
        protected SpringApplicationBuilder configure(SpringApplicationBuilder application) {
                return application.sources(App1Application.class);
        }

}
```

app2/src/main/java/lan/mistic/app2/App2Controller.java

```
package lan.mistic.app2;

import java.util.Iterator;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.jasig.cas.client.authentication.AttributePrincipal;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class App2Controller
{

    @Autowired
    @HttpSession session;

    @Autowired
    HttpServletRequest request;

    @RequestMapping("/")
    public String index(Model model)
    {
        if (request.getUserPrincipal() != null)
        {
            AttributePrincipal principal = (AttributePrincipal) request.getUserPrincipal();

            final Map attributes = principal.getAttributes();
            if (attributes != null)
            {
                Iterator<?> attributeNames = attributes.keySet().iterator();
                if (attributeNames.hasNext())
                {
                    for (; attributeNames.hasNext(); )
                    {
                        String attributeName = (String) attributeNames.next();
                        final Object attributeValue = attributes.get(attributeName);
                        model.addAttribute(attributeName, attributeValue);
                    }
                }
            }
        }

        return "index";
    }

    @RequestMapping("/logout")
    public String logout_GET()
    {
        return "logout";
    }

    @RequestMapping("/logout_cas")
    public String logout_CAS()
    {
        session.invalidate();

        return "logout";
    }

}
```

app2/src/main/java/resources/application.properties

```
cas.server-url-prefix=https://www.mistic.lan/cas
cas.server-login-url=https://www.mistic.lan/cas/login
cas.client-host-url=https://www.mistic.lan
```

app2/src/main/java/resources/templates/index.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta charset="utf-8">
    <title>MISTIC APP2</title>
</head>
<body>
        <h1>Welcome to App2 at the MISTIC Corporate Site</h1>
        <h4>running at app.mistic.lan</h4>

        <p>Welcome user: <b><span th:text="${uid}">UNKNOWN</span></p></b>

        <p>Your access level is: <b><span th:text="${employeeType}">UNKNOWN</span></p></b>

        <p>You logged in using the credential type: <b><span th:text="$
{credentialType}">UNKNOWN</span></p></b>
        <p>at: <b><span th:text="${authenticationDate}">UNKNOWN</span></p></b>

        Click <a href="/app2/logout">here</a> to log out.
</html>
```

app2/src/main/java/resources/templates/logout.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Refresh" content="0; url=https://www.mistic.lan/cas/logout" />
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta charset="utf-8">
    <title>MISTIC APP1</title>
</head>
<body>
        <h1>App1 Logout</h1>
</body>
</html>
```

app2/pom.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 https://maven.apache.org/xsd/maven-
4.0.0.xsd">
        <modelVersion>4.0.0</modelVersion>
        <parent>
                <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-parent</artifactId>
                <version>2.2.1.RELEASE</version>
                <relativePath/> <!-- lookup parent from repository -->
        </parent>
        <groupId>lan.mistic</groupId>
        <artifactId>app2</artifactId>
        <version>0.0.1</version>
        <packaging>war</packaging>
        <name>app2</name>
        <description>App2@MISTIC.lan</description>

        <properties>
                <java.version>11</java.version>
        </properties>

        <dependencies>
                <dependency>
                <groupId>org.springframework.boot</groupId>
```

```
                        <artifactId>spring-boot-starter-web</artifactId>
                </dependency>
                <dependency>
                         <groupId>org.springframework.boot</groupId>
                         <artifactId>spring-boot-starter-thymeleaf</artifactId>
                </dependency>
                <dependency>
                        <groupId>org.jasig.cas.client</groupId>
                        <artifactId>cas-client-support-springboot</artifactId>
                        <version>3.6.1</version>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-tomcat</artifactId>
                        <scope>provided</scope>
                </dependency>
                <dependency>
                        <groupId>org.springframework.boot</groupId>
                        <artifactId>spring-boot-starter-test</artifactId>
                        <scope>test</scope>
                        <exclusions>
                                <exclusion>
                                        <groupId>org.junit.vintage</groupId>
                                        <artifactId>junit-vintage-engine</artifactId>
                                </exclusion>
                        </exclusions>
                </dependency>
        </dependencies>

        <build>
                <finalName>${artifactId}</finalName>
                <plugins>
                        <plugin>
                                <groupId>org.springframework.boot</groupId>
                                <artifactId>spring-boot-maven-plugin</artifactId>
                        </plugin>
                </plugins>
        </build>

</project>
```

App4, on the other hand, requires some dedicated code in the controller to deal with the extra validations about the type of credentials used in the login operation.

app4/src/main/java/lan/mistic/app4/App4Controller.java

```
package lan.mistic.app4;

import java.util.Iterator;
import java.util.Map;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpSession;

import org.jasig.cas.client.authentication.AttributePrincipal;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.ui.Model;
import org.springframework.web.bind.annotation.RequestMapping;

@Controller
public class App4Controller {

    @Autowired
    HttpSession session;

    @Autowired
    HttpServletRequest request;

    @RequestMapping("/")
    public String index(Model model)
    {

        if (request.getUserPrincipal() != null)
        {
            AttributePrincipal principal = (AttributePrincipal) request.getUserPrincipal();
            final Map attributes = principal.getAttributes();
            if (attributes != null)
            {
                Iterator<?> attributeNames = attributes.keySet().iterator();
                System.out.println("<b>Attributes:</b>");
                if (attributeNames.hasNext())
                {
                    for (; attributeNames.hasNext(); )
                    {
                        String attributeName = (String) attributeNames.next();
                        final Object attributeValue = attributes.get(attributeName);
                        model.addAttribute(attributeName, attributeValue);

                        // Check for a correct credential type
                        if (attributeName.equals("credentialType") &&
                              ! attributeValue.equals("X509CertificateCredential"))
                        {
                           return "unauthorized";
                        }
                    }
                }
            }
        }

        return "index";
    }

    @RequestMapping("/logout")
    public String logout_GET()
    {
        return "logout";
    }

    @RequestMapping("/logout_cas")
    public String logout_CAS()
    {
        session.invalidate();

        return "logout";
    }
}
```

In case of denying the access, at the application level, an intermediate web page template is returned to redirect the user to the logout page after showing a message:

app4/src/main/java/resources/templates/unauthorized.html

```
<!DOCTYPE HTML>
<html xmlns:th="http://www.thymeleaf.org">
<head>
    <meta http-equiv="Refresh" content="10; url=https://www.mistic.lan/cas/logout" />
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
    <meta charset="utf-8">
    <title>MISTIC APP4</title>
</head>
<body>
        <h1>App4 Logout</h1>

        <p>You accessed this application using a credential type <b>not allowed</b>.</p>
        <p style="color:red">You will be logged out automatically in 10 seconds...</p>
</body>
</html>
```

11 Glossary

authentication: the action of verifying that a user/application is who claims to be.

authorization: the action of granting permissions to access or use a resource.

BYOD: *Bring your own device.* A policy that allows, and fosters, the users to use their own personal devices (e.g. smartphones) in the organization network.

certification authority: an organization that is trusted to sign other certificates so the client can trust the final certificates by validating the chain of certification from the final certificate up to the trusted authority (abbr. **CA**).

commercial off the shelf: a software component, already built by a third party, usually selected to be used to compose, or integrate with, a more complex product along with other parts (abbr. **COTS**).

CDDL: *Common Development and Distribution License.* A free and open source license from Sun Microsystems**.**

demilitarized zone: a kind of perimeter network (abbr. DMZ).

fork: forking an open source project is the process of copying the codebase under a new name and start from there a new, different project from the original, usually due to different criteria of the future development of the product by a set of developers or due to changes in the licensing of the original product.

LDIF: *LDAP Data Interchange Format.* A file format to interchange data and provide configuration information to LDAP servers.

perimeter network: an intermediate network set between a public network, like the Internet, and the internal network to isolate and protect the internal servers from direct access. To achieve this, all the connections from the external users pass through servers located in this intermediate layer.

predictive project management: an approach to managing a project where all the tasks are defined and planned before the actual work on these tasks starts. This approach is also known as *waterfall* project management.

proxy web server: an intermediate server used to cache and serve web pages from an external web server from the local network, to avoid consume Internet bandwidth, control the pages accessed, etc.

reverse proxy: an intermediate server, usually in a DMZ, dedicated to protecting internal server from external users.

role base access control: a model to account for the user's roles in an organization to assign specific permissions to some set of secured resources (abbr. **RBAC**)

servlet: a small Java program running in a servlet container or application server, that interacts via HTTP/HTTPS requests with its clients.

single point of failure: a component or part of a system that, when unavailable, renders the whole system inaccessible.

software as a service: a paradigm for software distribution where, instead of providing copies of the software to the user on some media, like CD-ROM, DVD, etc., the solution is provided on the Internet for the users to use it without any local software installation.

user repository: a database with information about the users in the system with their identifiers, names, position, and other corporate attributes, etc. plus meta-data about the user like the account status, statistics, etc.

vendor lock-in: an undesired situation where a company or organization has developed its solutions tightly based on some proprietary product or technology, thus making the costs of changing to another provider or technology very expensive.

12 Bibliography

Bell. E., LaPadula, L. (1973) *Secure Computer Systems: Mathematical Foundations.* MITRE Technical Report 2547, Volume I. March 1973.

Coulouris, G., Dollimore, J., Kindberg, T., Blair, G. ***(2012)*** *Distributed Systems: Concepts and Design.* International 5th Edition. Pearson Education Limited.

Deeming, D. (1976) *A lattice model of security information flow*. Comm. of the ACM. Vol 19. Num.5. May 1976. https://doi.org/10.1145/360051.360056

Duncan, A., Creese, S., Goldsmith, M. (2015) *An overview of insider attacks in cloud computing*. Concurrency and Computation: Practice & Experience. Vol 27. Num. 12. https://doi.org/10.1002/cpe.3243

Fett, D., Küsters, R., Schmitz, G. (2016) A *Comprehensive Formal Security Analysis of OAuth 2.0* Proc. of the 2016 ACM SIGSAC Conference on Computer and Communications Security. October 2016. p 1204–1215 https://doi.org/10.1145/2976749.2978385

Gollmann, D. (2002) *Computer Security*. John Wiley and Sons.

Grimes, R. **(2019)** *The many ways to hack 2FA.* Network Security, September 2019, p. 8-13.

Kotrappa, S., Kulkarni, P.J. (2010) *Multilevel Security Using Aspect Oriented Programming AspectJ.* 2010 International Conference on Advances in Recent Technologies in Communication and Computing. https://doi.org/10.1109/ARTCom.2010.87

Mayer, A., Niemietz, M., Mladenov, V., Schwenk, J. (2014) *Guardians of the Clouds: When Identity Providers Fail*. Proc. of the ACM Conference on Computer and Communications Security, 7 November 2014, p. 105-116 https://doi.org/10.1145/2664168.2664171

Maachaoui M., Abou El Kalam A., Fraboul C., Ait Ouahman A. (2012) *Multi-level Authentication Based Single Sign-On for IMS Services*. In: De Decker B., Chadwick D.W. (eds) Communications and Multimedia Security. CMS 2012. Lecture Notes in Computer Science, vol 7394. Springer, Berlin, Heidelberg https://doi.org/10.1007/978-3-642-32805-3_14

Naik T., Koul. S. (2013) *Multi-Dimensional and Multi-Level Authentication Techniques*. International Journal of Computer Applications 75, p. 17-22, August 2013 https://doi.org/10.5120/13163-0845

Sandhu, R.S. (1993) *Lattice-Based Access Control Models*. IEEE Computer. Vol. 26. Num. 11. November 1993 https://doi.org/10.1109/2.241422

San-Tsai S. and Beznosov, K. (2012) *The devil is in the (implementation) details: an empirical analysis of OAuth SSO systems.* CCS '12: Proc. of the 2012 ACM conference on Computer and Communications Security. October 2012. p. 378–390 https://doi.org/10.1145/2382196.2382238

Usha Rani, K. and Rajeswari, P. **(2018)** *An Efficient Approach for Multilevel Authentication System for Banking Services*. Int. Journal for Research in Applied Science & Engineering Technology (IJRASET) Volume 6 Issue II, March 2018.