How to Trust Another CAS Server

By J. Field
November 2011

This article describes the configuration and deployment steps needed to make one CAS server trust another CAS server. This might be needed, for example, when your institution is using two different application suites, each of which bundles a different CAS server. Or, perhaps you are already running CAS, but then you acquire an application that bundles its own CAS Server. The best solution would be to have all the applications and services trust one CAS server deployment, and decommission the others. But sometimes you can’t simplify in this way. Perhaps the second CAS server is maintained by a different department. Having two CAS servers would mean that the SSO experience is spoiled, as the users would have to log in another time, when access services that are protected by the “other” CAS server. You get the idea. To solve this we need to create a “trust bridge” such that logging into one CAS server is sufficient to obtain a Service Ticket from the other CAS server. The solution we describe can be configured for one-way trust, or it can be made symmetrical, where each CAS server trusts the other. The one-way case might be appropriate if you have one CAS server that protects “high value” applications and services, and another CAS server for everything else. Rather than penalizing the users of the “high value” application by making them log in again whenever they access applications or services protected by the “regular” CAS server, one could have that second CAS server delegate the authentication to the “high value” CAS.

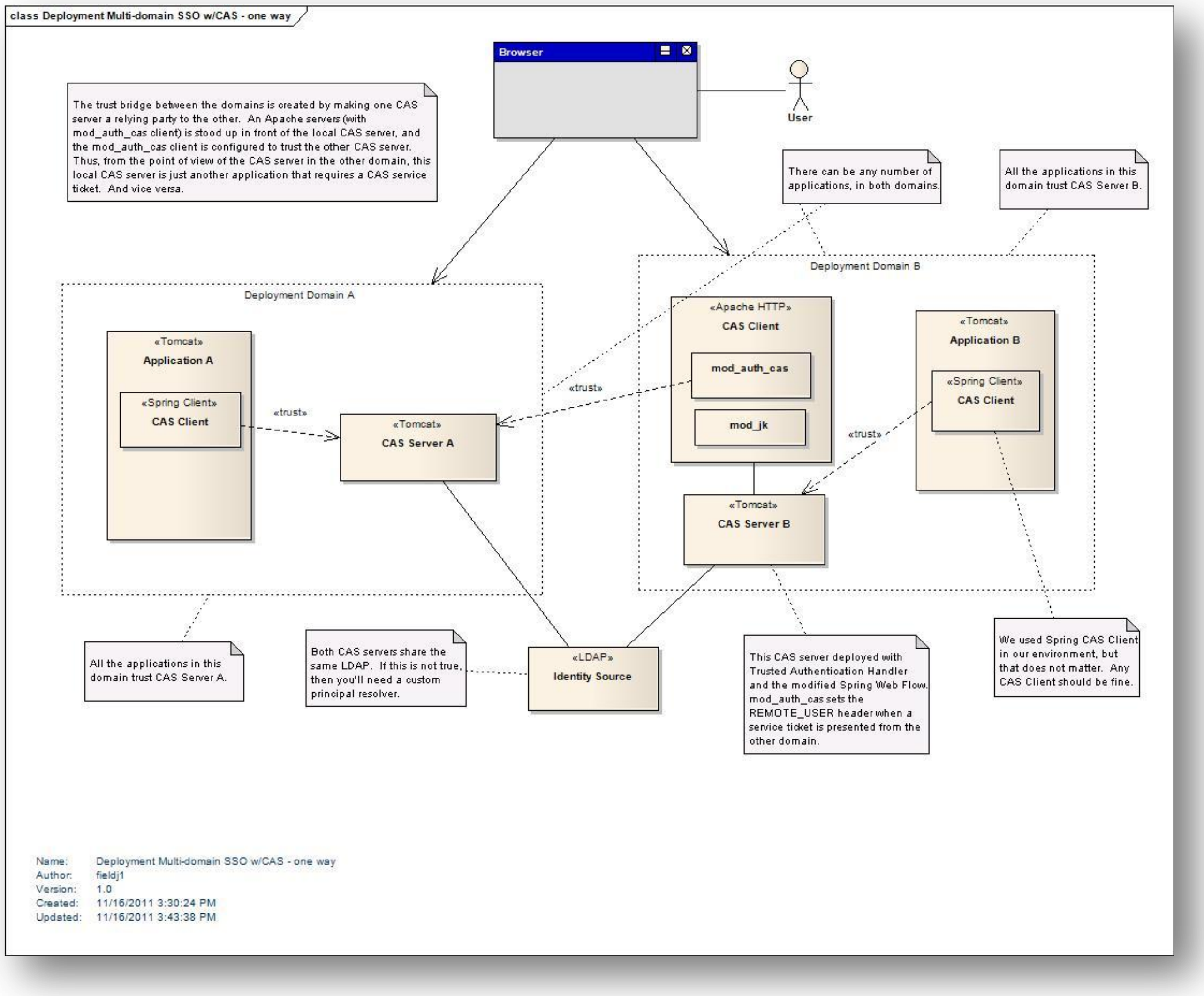
Solution Approach

The solution approach is to have the "local" CAS server treat the other ("remote") CAS server as an "application", and vice versa.

So, in other words, from the point of view of CAS Server A, the CAS Server B is just yet another "application". The CAS Server A can issue service tickets for the application called "CAS Server B". Of course, we might want this to be symmetrical, and so from the point of view of CAS Server B, it can issue service tickets for the application called "CAS Server A". These "foreign domain" service tickets can be accepted by a corresponding CAS client, positioned to protect the local CAS Server, and configured to trust the remote CAS Server for ticket validation.

The CAS client that protects your local CAS Server checks for a service ticket for CAS login (that has been issued by the other domain). If it determines that a user has presented a valid service ticket from a trusted remote or "foreign domain”, then it tells the local CAS server to issue a local TGC. The visiting user is then authenticated locally, based on having a ticket from their home domain, and is able to subsequently get service tickets issued for any other services in the local domain.

The following figure illustrates the high-level view of the deployment for one-way trust.



The trust bridge between the domains is created by making one CAS server a relying party to the other. An Apache servers (with mod_auth_cas client) is stood up in front of the local CAS server, and the mod_auth_cas client is configured to trust the other CAS server. Thus, from the point of view of the CAS server in the other domain, this local CAS server is just another application that requires a CAS service ticket. And vice versa.

There can be any number of applications, in both domains.

All the applications in this domain trust CAS Server B.

All the applications in this domain trust CAS Server A.

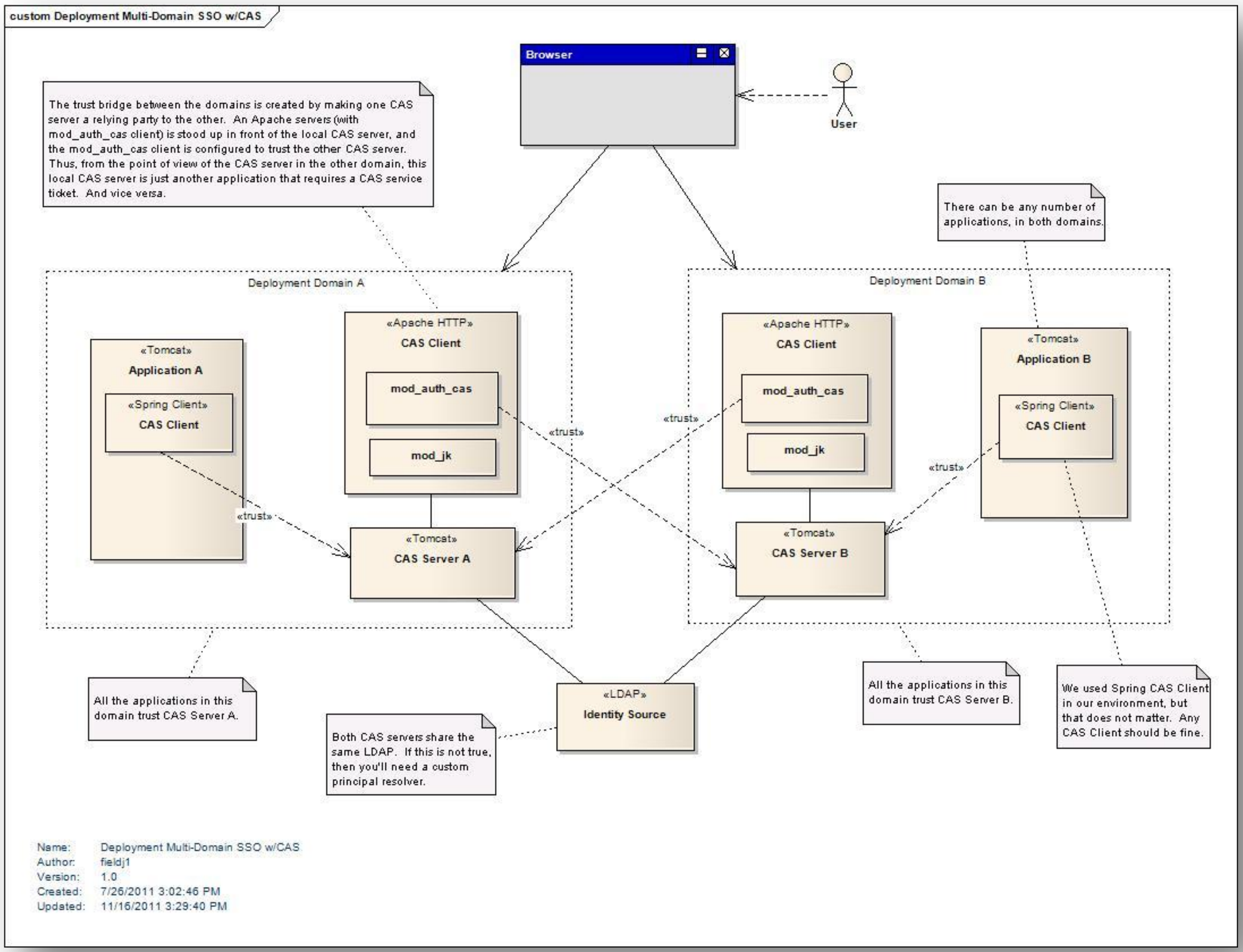
Both CAS servers share the same LDAP. If this is not true, then you'll need a custom principal resolver.

This CAS server deployed with Trusted Authentication Handler and the modified Spring Web Flow. mod_auth_cas sets the REMOTE_USER header when a service ticket is presented from the other domain.

We used Spring CAS Client in our environment, but that does not matter. Any CAS Client should be fine.

Name:	Deployment Multi-domain SSO w/CAS - one way
Author:	fieldj1
Version:	1.0
Created:	11/16/2011 3:30:24 PM
Updated:	11/16/2011 3:43:38 PM

The following is a deployment diagram for the two-way, or symmetrical case:



The trust bridge between the domains is created by making one CAS server a relying party to the other. An Apache servers (with mod_auth_cas client) is stood up in front of the local CAS server, and the mod_auth_cas client is configured to trust the other CAS server. Thus, from the point of view of the CAS server in the other domain, this local CAS server is just another application that requires a CAS service ticket. And vice versa.

There can be any number of applications, in both domains.

All the applications in this domain trust CAS Server A.

Both CAS servers share the same LDAP. If this is not true, then you'll need a custom principal resolver.

All the applications in this domain trust CAS Server B.

We used Spring CAS Client in our environment, but that does not matter. Any CAS Client should be fine.

Name:	Deployment Multi-Domain SSO w/CAS
Author:	fieldj1
Version:	1.0
Created:	7/26/2011 3:02:46 PM
Updated:	11/16/2011 3:29:40 PM

In this figure we show the Apache mod_auth_cas component as the CAS Client that protects the CAS servers. Notice the trust relationships. The mod_auth_cas client sits in front of the CAS server and redirects the request for the /cas/login page to the other CAS server. As illustrated, this can be symmetrical and work both ways. The applications in this figure use the Spring CAS client, but that does not matter. There is no specific limitation on what CAS clients the other applications use.

Use Case Flow

When the user from “Domain A” goes to the application in the remote domain (say, Domain “B”) they get redirected to the local CAS server trusted by that application. The login page of that CAS Server B is being protected by the Apache server running mod_auth_cas. That CAS client, in turn, redirects again to the CAS Server A in the user’s home domain, which it trusts. If the user has a session back there then they are issued an ST for the "application" which is the CAS Server B (the one at which they are considered a guest). That CAS server B then issues an ST for any target application in Domain B. This works nicely in practice because the query string parameters such as

“?service=” and “ST=…” are nicely pre-pended and “nested” as part of the HTTP redirects that embody the CAS login.

Architecture and Implementation Details

The solution approach described above relies upon the use of three specific CAS features/options:

- The "Trusted" authentication handler
- A modified Spring Web Flow for the CAS login sequence
- The use of CAS "gateway" mode

We will describe these CAS capabilities and explain how together they helped us solve the problem.

The Trusted Authentication handler is an optional feature that must be included in the CAS server build for the target domain. You need to modify the Maven POM to include this support, similar to including the REST support. This handler enables us to delegate the authentication of the user to the Apache server that is deployed in front of the CAS Tomcat server.

The Spring Web Flow for the CAS login sequence must also be amended, to invoke the Trusted Authentication handler, prior to rendering the JSP page that presents the HTML login form to the user. Again, this is part of the build of the CAS war file for the server to be deployed in the target domain.   The Trusted Authentication handler sets the HTTP REMOTE_USER header. More details below.

The "gateway" mode is a native feature of the CAS 2.0 protocol specification, and it is supported by the standard CAS webapp build. Use of gateway mode ensures that users of the “local” CAS server (those who are not cross-domain users) will never see the CAS login page from the other domain. Gateway mode must be turned on in the configuration of the Apache mod_auth_cas client.   Note that there is some inefficiency in this particular use of gateway mode. Users who are strictly “local” (in that they never intend to log into applications in the other domain) will still be redirected over there to check for a session (and then immediately redirected back). The gateway redirect is for the benefit of the users who are users of applications in both domains, and have already logged in, back in their home domain. This happens quickly, and only once, but it is admittedly an unfortunate side effect for the users who normally only use applications in one domain.

Trusted Authentication Handler

The deployment relies upon use of the CAS Trusted Authentication Handler. The trusted authentication handler is used when CAS wants to delegate the actual authentication to another authority. This is included in the CAS build for the target or "trusting" domain, i.e. the CAS server that is willing to accept inbound users, coming from the other domain. The Trusted Authentication handler enables CAS to delegate the authentication to another authority, and issue TGT and ST, if some other actor that CAS trusts asserts that it has authenticated the user. In our case the CAS server delegates the authentication to the Apache server that has been deployed in front of CAS, with Apache mod_auth_cas installed.

The CAS Trusted Authentication handler has support for two options (i.e. choose which Spring bean to configure in CAS). The first option is to rely on the CGI environment variable REMOTE_USER. To use this option, CAS must be deployed in conjunction with a front end proxy such as Apache, configured with mod_auth_cas. Using this configuration, the CAS Server will trust the contents of the HTTP header REMOTE_USER, as set by the Apache server. Apache and Tomcat are connected via installation of mod_jk in Apache. As noted, the Apache server is also configured with mod_auth_cas, and that plug-in will check for service tickets issued by the "other" CAS server. If a valid ST is presented (it can be validated back at the issuing CAS server), then the REMOTE_USER header is set by mod_auth_cas, and the HTTP request is passed to the CAS server in Tomcat. If it is not valid, then the REMOTE_USER is not known, and the user needs to log in, just like any other request. The Spring Web Flow in the CAS build needs to be modified to look for the REMOTE_USER header. If it is set, trust it, and issue a TGT. If it is not set, return the login page.

The second option is very similar, but does not rely on the REMOTE_USER HTTP header. Instead, it works directly with the Java security context, and looks for the authenticated Principal that has been set in the context of the Java JVM. This might be set by, say, a JAAS compliant security provider. Or, perhaps another filter such as Spring Security, or the app container (i.e. JBOSS does the authentication). This approach seems to be useful for non-HTTP situations, and/or for situations where for some reason you want to rely on another container authentication feature, rather than deploy the Apache HTTP server with mod_auth_cas.

As noted, we implemented with mod_auth_cas and Apache proxy. Configure the Apache server with mod_jk to forward requests for the CAS login page to the CAS Tomcat server. Configure mod_auth_cas to trust the "other" CAS server, not the one it is sitting in front of :-). Also configure mod_auth_cas to turn on CASGateway (set to true).

Finally, once the Apache server with mod_auth_cas has been set up, the applications and services in that domain should be re-configured to redirect users to the CAS login page via the Apache server port. That is, rather than redirecting to the port for CAS login page at the Tomcat server, they CAS clients for the applications should redirect to the Apache port. Without this, change those applications will be oblivious to the cross-domain trust.

More details available here.

Spring Web Flow

The Spring Web Flow of the CAS login sequence has to be modified to have an additional state transition. The idea is that normally when the user does a GET for

/cas/login, the page is delivered. But, when supported Trusted Authentication Handler we first check if the HTTP GET request for the /cas/login page contains a “REMOTE_USER“ Header, and if it does then we trust it, and transition to state “Issue TGT” rather than to state “ViewLoginForm”. In other words, there is no need to authenticate the user again. We skip the webflow steps of presenting the login page and processing the submitted form, and instead just issue the TGT and then naturally flow into issuing the ST. If no “REMOTE_USER” Header is found in the GET request for the /cas/login page, then just present the JSP page as before.

The details on how to modify the Web Flow configuration can be found here.

Use of Gateway Mode

The gateway feature is a way to tailor the behavior of the CAS SSO flow, to improve the user experience when they do not yet have a current SSO session at CAS. The idea is that the target application can take advantage of the SSO session if it already exists, but it can defer the decision to force the authentication if the user is not already logged in.

Normally, if the user is not logged in, and they ask for a page at a service, the service will redirect the user to CAS, where they will see a login form, and be expected to comply. However, this experience can be rather abrupt. That immediate bounce to a login page may also be unnecessary, since some sites have a public home page(s). If the user arrives unauthenticated, there may still be some useful pages they can still see. So, if the user is surfing application A and clicks a link to application B, then application B has the option to use the SSO session if it is available, or to defer authentication until later if it is not.

To do this, the CAS Client on application B can redirect the user to its trusted CAS server with the query string argument "?service=foo&gateway=true". This means: "redirect to CAS and attempt to get the advantage of an existing SSO session by obtaining a service ticket. But if the user is not already logged in, then don’t present the login page”. Instead, CAS will immediately send the user right back to the requested application page at application B with no authentication.

We use gateway mode when we redirect the user back to their home CAS server. If they have a session at their home CAS server, it will issue an ST for the remote CAS server. If they do not have a session at the other CAS server, they will see the login page of the remote CAS server (the public "landing" page). This means that strictly local users (those who are not cross-domain users, and could not possibly have a session over there) will not be surprised by seeing the login page of the other CAS domain. However, it also means that users who **are** cross-domain -- but have not yet logged in back at their home CAS server -- will be presented with the login page from the destination domain. Since in the proposed deployment the OpenLDAP back end is actually shared, they COULD in fact log in if they wanted to. (But, in the "one way trust" case, the session created would not be meaningful back at their home CAS domain. In this case it would be better to modify the login page to provide a link so that the user could manually surf back to their normal CAS server.

For example, to handle this, we could add a link to the login page that says "If you are a guest user from other domain click here". This will encourage the users to surf back to their home domain to log in there. Note also, however, that if we proceed to implement symmetric Multi-domain SSO, then this link would not be needed, since a login to either domain is valid in both places. Rather than limit the approach to the special case of one-way trust, we took advantage of the shared LDAP and adopted the two-way trust approach. This means that the user can benefit from any session that exists back at their home domain, but they can also create a new session in the remote domain. Due to the bidirectional trust, that remote CAS session will be a meaningful back in their home domain, if and when they surf back to a home application/product.