Enterprise Logon Server for Domain Wide Web-Based Applications

SYLVIA ENCHEV
Stord/Haugesund University College
Department Haugesund
Bjørnsonsg. 45, 5528 Haugesund
NORWAY

SHARIL TUMIN
University of Bergen
IT-Dept.
P. O. Box 7800, 5020 Bergen
NORWAY

Abstract: A growing body of Web-based applications within an educational organization demands a simple and robust Web logon service. Most universities own a collection of Web-based applications for administration, learning and collaborative systems, front-end to databases and directories, plus hosts of other utility applications. Web-based applications provide users with services no matter at what time of the day and where from the users are located. There are no time and space constrains on on-line services. In the age of Internet we are used to this symbiosis between us and the organization we belong to. The system presented here implements the basic logon and SSO functionalities and is written in Python.

Key–Words: Collaborative setup among independent organizations

1 Introduction

In this paper we describe a simple implementation of an enterprise Logon server which support SSO. There exist several implementations of Web authentication system with SSO. The system presented here implements the basic logon and SSO functionalities and is written in Python.

Public Web applications such as those provided by universities and public libraries can easily incorporate logon procedure with SSO based on HTTP redirections. Such implementations should not be difficult once the mechanism of HTTP redirection and web cookies are understood.

Single Web logon server introduces a single point of failure and sensitive to DoS (Denial of Service) attack. DoS attack can be controlled by introducing TCP/IP (Transmission Control Protocol/Internet Protocol) filter rules at the border firewall.

Most Web-based applications impose access control, in which only authenticated users with correct authorization can make use of the services provided. To be authenticated a user must provide a correct credential in the form of (usually) a pair of user-identification and password during a logon process. An organization equipped with a centrally controlled IDM can provide a user with one person - one credential convenience for logon to all systems within the organization. The users are not the only ones that benefit from an IDM, applications developers can also make use of the centrally controlled users’ data, thus saving time and resources for not having to manage local users data, which in turn leads to an increased security.

It is not uncommon that each Web application implements its own way of doing users’ logon process, sometimes with its own local users. However, most open-source applications implement their authentication processes using users’ data stored in a domain wide directory servers based on LDAP (Lightweight Directory Access Protocol), AD (Active Directory) or NIS (Network Information System). One of the most important tasks an IDM has is to synchronize users’ data from the IDM owned central database to those directories mentioned above.

Even if users’ data is obtained from a domain wide directory server, multiple logon services within an organization will not be cost effective. Each of these logon services needs to be implemented and maintained separately. If these logon services run on separate sites (servers), then these servers need separate server certificate in order to support HTTPS (Hypertext Transfer Protocol over Secure Socket Layer). Users are presented with different logon procedures on different servers. Furthermore users need to repeat the logon procedure for accessing different application. In such conditions a naive user can be easily tricked to disclose her credentials to a site that belongs to a group of cyber-thieves.

Authentication is just one part of access control. The other and often abandoned (left out, not implemented) part is authorization. The question here is whether an authenticated user is authorized to continue through the specific application. An application can ask a user for authorization at different stages...
within the user-application interaction. A single logon server can implement SSO and real time authorization mechanism builds on top of IDM users’ data. The logon service provides the users with a consistent and well known URL (Uniform Resource Locator) for authentication procedures. That same server is used by other Web applications for authorization procedures. The enterprise users are asked not to use other logon pages other then the one provided by the enterprise logon server.

The use of a single logon server also helps system administrators to implement better system security. Security administrators have only one logon server to implement, maintain and monitor. Multiple logon attempts can be logged and eventually block offending users in real time.

2 Authentication Procedure

In this paper the term single-sign-on is meant as a single authentication procedure, in which an authenticated user needs not to repeat the authentication procedure again when challenged by a protected application within the umbrella of the SSO framework. Another meaning of single-sign-on is a mapping one credential to another such that a user needs only to remember one user-identification and password pair which can be used to be authenticated to another system using different pair or different mechanism completely.

The authentication process in this SSO implementation gives a particular authenticated user a token as a proof that she is already being authenticated earlier and is a valid user. The token is only valid within a particular session. A session has duration (within a time period) and a range (within a group of applications). A token is not valid outside those particular constrains. Therefore a user is only valid when she holds a valid token within a valid session. With this model a token is associated with authentication and a session with authorization. In Web-based applications, information pertinences to access control to a particular URI (Uniform Resource Identifier) are stored in the user’s Web-client cookies variables.

Within a resource the application checks if the user has a valid session in order to be allowed to proceed. If the session is out of date or out of range, the application will use the user’s token to consult the logon service about that particular user validity and authorization. If the token is valid the logon server will provide the application with the needed information about the user. The application then decides whether to give the user a valid session or not. Using token for authentication and session for authorization solves the logout problem. With a session marks ‘logout’ a user is logout locally from the application’s site. A user is globally logout when her token is marked ‘logout’.

3 Implementation

The idea of Web-based SSO is not new, many SSO for Web exist (JA-SIG CAS, Pubcookie, WebAuth, Cosign A-Select). A more recent open source middleware software called Shibboleth (Shibboleth), which provides SSO by implementing OASIS SAML is available for anyone interested for free. Shibboleth was designed with inter-institutional collaboration in mind, where a number of organizations join together under a single Shibboleth federation. The Shibboleth framework can also be used as a stand-alone system for SSO within organizational boundaries. Shibboleth employs Apache Tomcat Java based Web server. The 14.1 MB compressed downloadable package contains a collection of jar, java, xml and others files for building a complete system using ant.

HTTP (Hypertext Transfer Protocol) is inherently a stateless request-response message passing protocol. The communication is closed after each request-response is completed. We have to remember that HTTP was originally designed for serving Web clients (browsers) with static HTML (Hypertext Markup Language) documents. This simple design principle is so successfully reflected by the success the Web had gained since its introduction in 1990 (Tim Berners-Lee, 1998). In order to keep states between invocations web cookies were introduced in 1995 (Montulli, Giannandrea, 1995). Cookies are packets of text information send by a server to a client and then send back unchanged to the server each time it connects to that server. Using cookie the servers can then store states of its clients.

The basic technique in implementing SSO is the use of web cookies and HTTP redirect. Let’s denote application server as WAS and the enterprise logon server as LOS. A (server, tokens) signifies web cookies for a particular server and tokens are its values. Also let denote φ(url, parameters) as a HTTP redirect directive to a server and η(url, parameters) as an HTTP request from a client to a server with query string given in parameters for either PUT or POST. An url is written in the form server:app-name. A nil signifies empty or no value.

Below we describe stages of logon procedure:-

1. client: η(WAS:start, nil); γ(WAS, nil) → φ(LOS:logon, WAS:whoami, WAS:start)
2. client: η(LOS:logon, id, pwd, WAS:whoami, WAS:start) → logon: authen(id,pwd)
3. if not authen(id, pwd) repeat 2.
   else create logon token tid associated with that particular id and token tida associated with that particular id and application server WAS.

4. logon: \( \gamma(\text{LOS, tid}); \phi(\text{WAS:whoami, tid, WAS:start}) \)

5. (web-service) whoami: \( \eta(\text{LOS:whoami, tid}) \rightarrow \text{user} \)

6. if not user repeat 1.
   else create session sess associated with user

7. whoami: \( \gamma(\text{WAS, tida, sess}); \eta(\text{WAS:start, user}) \)

At stage 1 a user is accessing a Web application named start at the application server. Without the proper session cookies, the application redirects the user’s browser to Logon server with two URL, whoami and the original application start located under Web application site. In stages 2 and 3 the user provides her credentials and proceeds to stage 4 if she is successfully authenticated. In stage 4, the Logon server sets her browser’s cookies with logon token tid and redirects the browser to application server’s whoami. The two whoami applications communicate with each other in stages 5 and 6. If the token tida is valid and produces a valid user, the logon procedure proceeds to the 7 stage. Under stage 7 the user browser cookies are set with session information tida and sess and finally the user is redirected to the original application start.

The cookie path for tida is set to be applicable site-side, that is to say path = ‘/’. While the cookie path for sess is set to be applicable for that particular application. The SSO is implemented using the site-wide tida cookie. Any Web applications under WAS will receive the tida cookie if is valid. Such Web application will proceed to the redirection part of stage 4.

To provide authorization mechanism, the Logon server also provides a simple HTTP-based web-service \( \eta(\text{LOS:mygroups, tid}) \rightarrow \text{(user, \{grp1, grp2 ...\})} \). The service mygroups receives tida token and replies with a list of groups that belong to a particular user. The application can then use the list to set session cookies for authorization purposes.

Users need not use their system identities (Unix account names or Active Directory user names) to logon to Web-based applications. The users can use their mail aliases (for example Jason.Man@mki.uib.no) or nick name (for example ‘Jason the Man’) to be used as identities during logon procedure. This practice is useful for the protection of system account names.

Web cookies should be signed by the issuing application. Cookies containing sensitive information should be encrypted. Whenever possible cookies should contain data such that it can be uniquely related to the user’s browser, for example IP address and browser information (for example type and version). All these will help in preventing cookies theft and poisoning.

4 Conclusion

The system described in this paper is a light-weight implementation of SSO and access control based on tokens and tokens and is implemented using Apache Web server, Python scripting language and SQLite database. Using an interpretive dynamically type language as Python contributes in making the system described to be simple, transparent and extensible. New functionalities can be added step-wise when needed. In stead of using Java classes we employed Python scripts callable from inside Apache Web server via mod_python.

References:


