Single Sign-On in Service-Oriented Computing

Kurt Geihs[1], Robert Kalcklösch[1], and Andreas Grode[2]

[1] Intelligent Networks and Management of Distributed Systems
Berlin University of Technology, D-10587 Berlin
{geihs, rkalckloesch}@ivs.tu-berlin.de
[2] DIN IT Service GmbH
Burggrafenstr. 6
D-10787 Berlin
andreas.grode@dinits.de

Abstract. Support for Single Sign-On (SSO) is a frequently voiced requirement for Service-Oriented Computing. We discuss SSO strategies and approaches, their requirements and constraints. The two most prominent approaches in this field are presented, i.e. Microsoft Passport and Liberty Alliance. Because implementations of Liberty were not widely available and in order to understand the conceptual implications and practical requirements of SSO we have built our own SSO solution. Its modular and flexible design is compatible with the Liberty specifications. The prototype reveals valuable insights into SSO design and operations.

Keywords: Service oriented computing, security, service authentication, single sign on

1 Introduction

Service-oriented Computing and Web Services are critical ingredients in making the Internet a universal platform for electronic commerce [1]. The new technology supports a loosely coupled collaboration style for business to business and business to consumer scenarios. New technical challenges and requirements arise primarily from the inherent autonomy of actors as well as the heterogeneity of components. Clearly, security concerns are uttermost important in such an open environment. On the one hand we need effective security mechanisms to protect the individual business assets. On the other hand we clearly want to lower the inconvenience hurdles for using the new application potential.

The authentication of a service user to a service provider is one area where this dilemma is evident: While one has to acknowledge the autonomy of services to choose their own authentication mechanism, one should also avoid inefficient and insecure repetitions of service sign-on procedures. Password proliferation is a frequent consequence of multi-service environments. It often leads to violations of security policies and thus weakens the security. Ideally, service users want to sign on to "the distributed system" only once, just like the user of an operating system logs on to a computer once and may then use the different operating

M.E. Orlowska et al. (Eds.): ICSOC 2003, LNCS 2910, pp. 384–394, 2003.

system services. An SSO mechanism should not only provide the necessary level of security but also should be easy to use and should work for arbitrary service types.

This paper addresses Single Sign-On (SSO) strategies for service-oriented computing environments. Solving the SSO problem is much harder in such environments due to the inherent distribution and heterogeneity as well as the autonomy and independence of the involved actors.

The paper is structured as follows. In Section 2 we analyse the SSO requirements and constraints, and we provice a classification of SSO approaches. Section 3 discusses the two most prominent approaches for SSO in Service-Oriented Computing, i.e. Microsoft Passport and Liberty Alliance. Because implementations of Liberty were not widely available and in order to understand the conceptual implications and practical requirements of SSO we have built our own SSO solution. Section 4 describes our prototype system and reports on the lessons learnt. Section 5 concludes the paper and points to further work.

2 Single Sign On

A SSO system enables users to access multiple services or computer platforms after being authenticated just once [12]. This does not mean that a SSO system unifies the account information (e.g., username and password) for all services, even if this is a popular interpretation for many people. Instead, it hides the account information for the participating services behind only one account. Thus, the user logs on to the SSO system once and the system manages the logins for the specific services the user chooses to work with. Especially, the system does not automatically perform a login for the user at all services managed by the SSO system. A login takes place only at those services that the user chooses to work with. The SSO system has to maintain a list of username-password pairs for the services. Obviously, this list is a remunerative target for an attacker. If an attacker gains access to the SSO system he therefore has access to all participating applications. This is also true, if an attacker gets the account information from a specific user. With this information he is able to access all services the genuine user is allowed to access.

Consequently, security is a major aspect of a SSO system. The accounting information has to be stored in a secure way, ensuring that only the owner has access to it. Also, the login to participating services must take place in a secure way.

There exist three major mechanisms for user authentication. First, user and service share a not commonly known secret (e.g., a password) which is bound to a special identity. The second mechanism involves some material token, including personal characteristics (e.g., a driving license or a magnetic card). The third one is a variation of the token, whereby the token is a biometric attribute (e.g., the fingerprint or the signature). The main point is that no mechanism is per se better than the other. A carefully chosen password which is changed regularly may be better than a token which can be easily faked. Thus, not only

the mechanism itself, but also the behavior of the user contributes to the level of security of the authentication.

Although an authentication maps a network identity to a real life person, it is not necessary that all services behind a Single Sign-On know exactly with whom they are dealing. Thus, a SSO system should support pseudonymity in a way, that the user can decide, based on the specific needs of an application, to which extend he is willing to reveal his identity. However, it must be ensured that at least at one point the mapping between the pseudonym and the real identity can take place. Pseudonymity is an issue especially in the European Community with its strong privacy policies.

2.1 Cooperative vs. Non-cooperative SSO

Basically, there exist two different approaches for SSO: cooperative and non-cooperative SSO. In a cooperative SSO system the participating services know about the SSO system. SSO is not transparent for the services in this case. Normally, the services have to be modified, at least regarding the login procedure, to cope with the requirements of the cooperative SSO system. Through cooperation among themselves the services are able to build a network, where a user, once successfully logged in, could move from one service to another, without further login procedures. Also, it is possible to link the different identities together building a group of services and enriching each others services.

The alternative is non-cooperative SSO. In this approach the participating services do not know about the SSO system. This is no contradiction, if one looks at the desired behavior of a Single Sign-On mechanism. It should group together services and offer the user a single authentication point to login to all services. Such a system could be a local application at the user-side or a server-side proxy, managed and operated by an administrator who manages the login on behalf of the user. This can be fully transparent to the involved services. Obviously, not all services may be suited to be unified in such a way. There are several requirements they have to match. The usage of standardized techniques for the authentication is the major requirement. If the service provider is using proprietary software for the authentication, the provider of the Single Sign-On may not be able to map his account information to the desired procedure. Also, the service entry point and the authentication procedure should be stable and not change to often, as this would lead to an increasing effort in keeping the SSO up-to-date.

A rather straightforward approach to build such a non-cooperative SSO is conceivable for the World Wide Web. User login can be handled by an appropriately enhanced proxy such that the SSO is transparent for the services. Assuming that the login procedure for a service is known, it can be automated through an intercepting proxy that provides the authentication information automatically to the service. The proxy retrieves the authentication data from e.g. a configuration file set up by the system administrator. Thus, after the login to the proxy, the login to the services is hidden from the user.

3 Related Work

Today, for Web-based service environments two major approaches to SSO exist: Microsoft Passport and Liberty Alliance. Both systems are based on the cooperative SSO model. Their differences lie in the architectural structure. Passport is using a centralized approach, where only one entity (e.g. Microsoft) is able to authenticate a user. In contrast to that Liberty is designed to operate in a decentralized way. There may be many (cooperating) entities, that are allowed to authenticate a user. Both approaches are described briefly in the next sections.

A good taxonomy for SSO systems is given in [10], where the authors try to categorize different SSO approaches.

3.1 Microsoft Passport

The main objective of Passport [8] is the centralized storage of account information in order to simplify the login procedures and thus to ease the eCommerce activities of registered partner applications. Therefore, the Passport Server manages the authentication of users and only transmits a unique user identifier to the services.

Both the user and the service provider have to register themselves on the Microsoft Passport Server in order to use this SSO service. As part of the registration a service provider has to provide Microsoft with information about his safety guidelines and the offered service, and he has to comply with the technical requirements of Passport. Microsoft then assigns a unique ID to this service provider and transmits it to the provider together with a symmetric key, which is used for the encrypted communication between Passport and the provider. The encryption is done by a software provided by Microsoft, which has to be installed on the server side.

The users register with a valid email address and a password. The email address is used to transmit a confirmation mail which therefore checks the validity of the address. After that the user gets assigned a Passport Unique Identifier (PUID). Subsequently, this PUID is used to identify the user in all participating applications. It is a perfect pseudonym which does not reveal any links to the original user. As it is not told to the user, the user account is the valid email address.

The sequence chart of a successful login is depicted in figure 1. After initiating the login process on the webpage of a service provider the user will be redirected to a Passport server. There, the actual login form is created and the user performs the login. For the transmission of the user data a ssl connection is used. If the provided pair of username and password is valid, the server sends a redirect to the service provider and places some cookies on the users computer. These cookies are encrypted using 3DES and contain among other information the PUID of the user. With this PUID the service provider is able to identify the user and can thus be sure that the user has successfully logged in to the Passport server.

If the user now wants to use another Passport-enabled service he just needs to push the "login with Passport"-Button on the according web-site. With the

following redirect the already existing Passport-cookie will be sent to a Passport server. Instead of creating the login form, the server now checks the cookie and, if it is valid, acts like after a successful login depicted in figure 1.

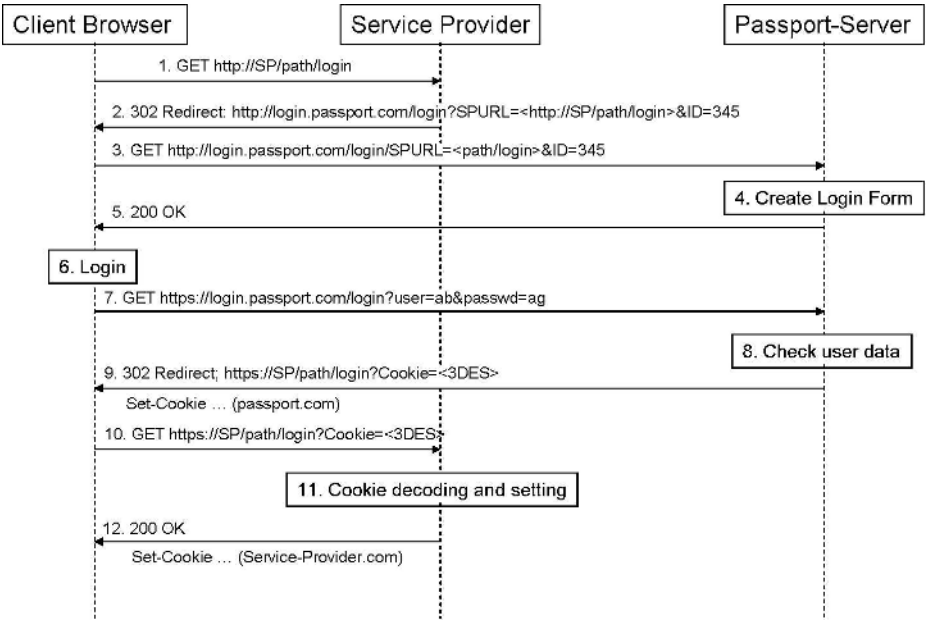

Fig. 1. Passport Single Sign-On Protocol

3.2 Project Liberty

The Liberty Alliance was founded to set up a SSO standard which should lead to different interoperable products from different vendors. The main objective is the coupling of multiple user identities distributed over cooperative service providers. The standard aims to support all popular operating systems, programming languages, and network structures [7] and is designed to ensure the compability and security between Liberty-aware applications [11]. As this was a new standard, no Liberty-aware applications existed in a productive environment when we started our work. Only a reference implementation from SUN [13] had been released. Today, there are several Liberty-enabled products[1].

The benefits for the users are given mainly through three points. First of all, Liberty allows the coupling of identities a user has for different service providers without announcing these to other providers. Within this group the user is able

[1] http://www.projectliberty.org/resources/enabled.html

to stretch a single login for one provider to all coupled providers leading to a single sign-on. Also, a single log-out is possible where a user logs out of all participating providers simultaneously.

A special service provider in the sense of Liberty is the Identity Provider (IP). It offers the management of user accounts for other service providers and users. Additionally, it offers authentication mechanisms. The Liberty standard does not prescribe any particular authentication mechanism. The choice of mechanisms is up to the Identity Providers. Naturally, a service provider could demand a specific method for the authentication as a minimum requirement for a successful login.

The SSO protocol used by Liberty is depicted in figure 2. After navigating to a service provider, the user chooses one of the proposed Identity Providers, that a service provider is willing to work with. The user is then redirected to that IP. Contained within the redirect is a defined XML structure (*AuthRequest*) which controls the behavior of the IP for that specific service provider and user. If the user is not yet logged in to the IP, he has to login first. After the successful authentication the user is redirected to the service provider. This redirect contains a SAML artifact with a random number. SAML is an XML-based framework for exchanging authentication and authorization information [9]. The random number in the SAML artifact serves as a handle for the service provider to access information about the user. After decoding the artifact, typically the service provider issues a SOAP-Request to the IP to fetch this information. In the SOAP-Response an authentication assertion is included, which holds the information. Among other things it includes the user ID. With this the service provider is now able to identify the user.

A group of service providers can build a so-called "Circle of Trust" in which a user, once authenticated can move easily from one service to another. Thus, the user can choose from a list provided by the service provider to which other provider within this circle he wants to go. By clicking of the according link, his profile is sent to the new provider. If the authentication level is acceptable for the new provider, that means, that the authentication mechanism chosen by the original provider meet his own security requirements, the user is automatically logged in without the need for any other interaction.

4 SSO Prototype

In order to study the practical requirements and limitations of SSO in a WWW environment, we have built our own prototype SSO system. In particular, we were looking for a general and comprehensive, but nevertheless easy to use and easy to administer solution. We aimed at a non-cooperative SSO system that does not require modification of the participating target services. Thus, it should transparently maintain individual passwords for each service.

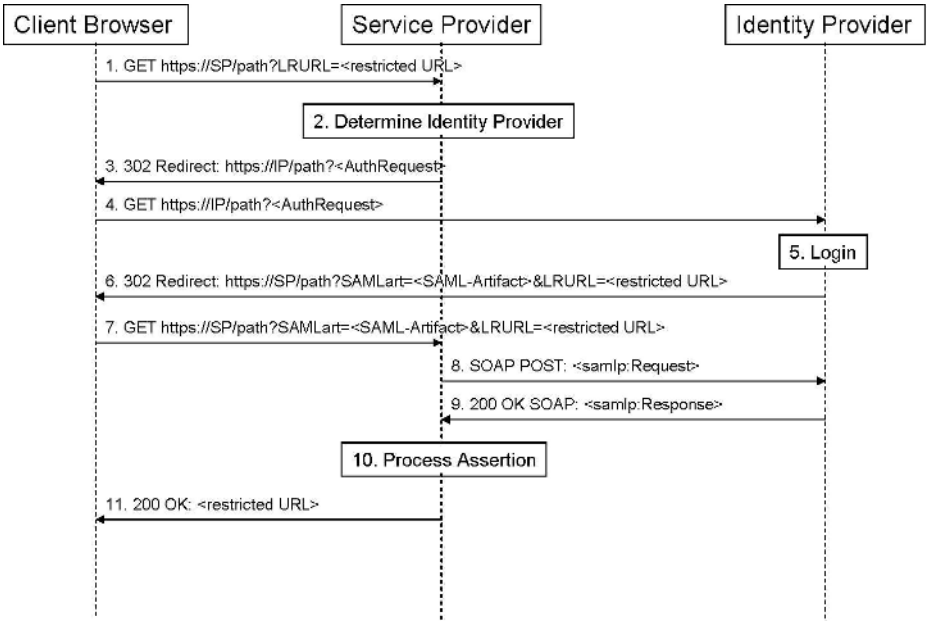

Fig. 2. Liberty Single Sign-On Protocol

4.1 Overview

The implemented SSO system provides an non-cooperative SSO service, where remote, protected web-applications are encapsulated through a "normal" protected intermediate SSO service. The different services do not need to know each other and do not get any information about the users outside of their scope. The service-internal implementation and authentication mechanisms continue to function as before. Therefore, the web applications are still usable even without the SSO server. The SSO server himself stores information about each user and his credentials for each of the participating web application. A proxy server in between the user's browser and the service is the key component. The desired content is proxied to the user's http-client (browser). Hence, no direct connection between remote service and user browser will be established. To prevent the browser from connecting to the remote server directly by following an absolutely addressed link, every tag with embedded hyperlink attributes has to be changed (transcoded) to "point" to the SSO-Server, preserving the information which remote document should be fetched. The SSO-Server therefore acts as a *"transcoding reverse proxy"* or *"transcoding surrogate"*. Different authentication mechanisms, access restrictions and session-tracking techniques may be employed and are hidden from the user, who only needs to authenticate to the SSO server using one authentication scheme.

This approach of creating a SSO service is rather simple. By the modular design of the chosen components, which are created with open source software only, integration of additional remote services with different protection schemes (such as HTTP authentication or cookies) is relatively straightforward. If needed, new transcoding rules have to implemented and added to the transcoding surrogate. In addition, configuration data needs to be included for accessing the desired URLs of the remote services.

The main objective of this first implementation was to provide a lightweight and extendable prototype of a SSO server which can be installed and configured easily. Extensions for a comfortable user and configuration management have to be created separately. As an administrator for a specific SSO server knows best, which applications are participating and what their configuration possibilities are, the development of a sophisticated configuration interface would be up to him. Naturally, a simple and generic interface could be provided as a starting point.

Because of the danger that the SSO service might become a single point of failure for many encapsulated services, the design payed careful attention to availability concerns for the SSO server. For example, configuration and user management is decoupled from the internal transcoding engine. This rather static information is stored in a separate database with internal redundancy in a central or distributed architecture. These "helper applications" are accessed via a defined API and are independent from the implementation of the SSO server itself, which can be installed on a couple of servers to provide redundancy and a higher availability.

4.2 Implementation

As mentioned above the implementation of the SSO server is built with open source software only. The main component is the well known and popular Apache HTTP Server [2]. One of its benefits besides stability and usability is its modular design which supports the integration of modules from third parties in a rather straightforward way. In our implementation we use mainly the Perl module to extend the Apache API with Perl applications inside the HTTP server. The development of the Apache module which drives the SSO server is done in the Perl programming language with all well-known benefits for writing web applications and the big number of available Perl modules from CPAN [3]. In the scope of the SSO server the Apache HTTP server is used to provide the framework for HTTP(S) transactions. It acts as a container for the reverse proxy (the so called *"surrogate"*) with the internal rewriting engine. All communication activities (proxy functions) and the transcoding of HTML documents is done with the Perl modules `LWP` [2] and `HTML::Parser` [3] and their derivates within the developed module `Apache::Transcode`. The Apache HTTP server can be used in conjunction with `mod-ssl`, which lets the HTTP server act as a HTTPS server

[2] http://www.cpan.org/modules/by-module/LWP
[3] http://www.cpan.org/modules/by-module/HTML

by the use of OPENSSL, the open source implementation of the SSLv3/TLS 1.0 [5,4] specification. HTTPS is needed for confidential communication between the SSO server and the user's browser. It also allows the use of client-side SSL certificates for user authentication. All data which has to be kept across multiple HTTP transactions is stored outside the scope of the server application with use of DBI [4], the database abstraction layer for Perl, available from CPAN [3].

4.3 Discussion

In our prototype implementation we have shown that SSO is possible in a way that it is transparent for service providers. So far, this is working only for sites built out of plain HTML, whereat it does not matter if the content is generated dynamically or served from static files. Hence, more work has to be done to integrate more sophisticated web techniques. Especially DHTML pages require careful examination to guarantee that the user is not leaving the SSO server during his session by accident.

Another subtle problem is the copyright issue. Although the service provider does not know about the SSO system it is necessary to inform him and to get his permission for the participation of his site, because the contents of the web pages are under the copyright of the providers. A cooperation between the service providers and the SSO service seems to be advisable here. In order to alleviate the copyright problem, our SSO server would be suited best for an Intranet, where many services with different authentication schemes are provided.

Another point is, that in heterogeneous environments where different HTTP-based applications, using different approaches for the authentication and administration of users, already exist, it could be difficult to migrate at once to a centralized directory-based approach using for example LDAP as the underlying protocol. Nevertheless, this technique does not implement SSO out of the box. It just provides the simplification to use only one account for every application, but the login itself has still to be done for each application. To address this problem, as mentioned before, it is possible to automate the login process in two ways: Firstly, on the client side, where a central database could be used as many popular commercial SSO solutions do, or secondly on the server side with the described SSO server.

Several points remain open at this stage of our investigations. First of all, the system is only a prototype. Especially, the reliability and the scalability of the system are issues we have to take a closer look at. For a productive system we will also need to study more carefully the efficiency and the resource requirements of our transcoding algorithms.

As the design of the prototype SSO server is quite modular, the current authentication mechanism in the proxy could be easily exchanged. For example, it could be replaced by a call to a Liberty Identity Provider. This would make the participating services Liberty-aware, regardless of their own capabilities and without changing their implementation.

[4] http://www.cpan.org/modules/by-module/DBI

To gain more experiences and to further improve our implementation, it is envisaged to use the SSO server inside the Intranet of the DIN IT Service GmbH, to group multiple document management systems (DMS) of the same vendor. Therefore, the prototype is now used for local testing with one DMS.

5 Conclusion and Outlook

In future Service-Oriented Computing environments service users in B2B and B2C scenarios need to sign on frequently to many different independent services. Each service may have its own authentication procedure. Without support for SSO, service users are forced to handle as many authentication credentials and procedures as there are accessed services. Clearly, the more services a user navigates, the greater the likelihood of user errors and thus compromised security. With a properly configured SSO system, once authenticated the user has immediate and convenient access to a number of services.

In this paper we have analysed the requirements and constraints of SSO for web environments. Furthermore, we have presented a prototype SSO system that works according to the non-cooperative model as a proxy that is transparent to the service applications. The prototype demonstrates that SSO support can be implemented effectively based on transcoding HTTP requests. No changes are required for services in order to participate. Consequently, our prototype solution works with arbitrary web-based services.

The future of SSO for Web Services and Service-Oriented Computing is not clear today. While the need for SSO has been clearly identified and expressed by many people, it is questionable whether and how fast users will overcome their scepticism and will trust another service component that has full control over the user's credentials and decides about very sensitive authentication activities. Microsoft's Passport has faced a lot of headwind already [6]. The Liberty Alliance undertakes great efforts to gain more acceptance. For e-commerce over the web, SSO remains one of the great technical and personal challenges.

References

1. *Communications of the ACM*, 46(6), June 2003.
2. Apache Software Foundation. www.apache.org.
3. CPAN. Comprehensive Perl Archive Network. www.cpan.org.
4. T. Dierks and C. Allen. The TLS Protocol Version 1.0. January 1999.
5. Alan O. Freier, Philip Karlton, and Paul C. Kocher. The SSL Protocol Version 3.0. November 1996.
6. David P. Kormann and Aviel D. Rubin. Risks of the Passport Single Signon Protocol. *Computer Networks, Elsevier Science Press*, 33:51–58, 2000.
7. Liberty Alliance. Liberty Architecture Overview Version 1.1. January 2003.
8. Microsoft. Microsoft .NET Passport Review Guide. March 2003.
9. OASIS. Assertions and Protocol for the OASIS Security Assertion Markup Language (SAML) v1.1. July 2003.

10. Andreas Pashalidis and Chris J. Mitchell. A Taxonomy of Single Sign-On Systems. In *The Eighth Australasian Conference on Information Security and Privacy (ACISP 2003)*, volume 2727 of *LNCS*. Springer, January 2003.
11. Birgit Pfitzmann. Privacy in Enterprise Identity Federation - Policies for Liberty Single Signon. Dresden, December 2003. 3rd Workshop on Privacy Enhancing Technologies (PET 2003), Springer.
12. R. Shirey. RFC: 2828: Internet Security Glossary. May 2000.
13. SUN. Interoperability Prototype for Liberty. 2002.