Single Sign-On Using Cookies for Web Applications

Vipin Samar
vsamar@us.oracle.com
Oracle Corporation

Abstract

The proliferation of web applications forces users to manage multiple user names and passwords daily. Various single sign-on solutions that depend upon PKI, Kerberos, or password-store have been proposed, but they require client side infrastructure and new administrative steps. Single sign-on in a web based intranet environment can be achieved using http cookies. This paper analyses cookie security issues and proposes a cookie based single sign-on architecture.

1. Introduction

Single Sign-On (SSO) has been a difficult problem for quite some time. This was not a big issue until recently, because people typically accessed only a few applications. Now with the integration of web-based desktops and applications, the number of services a typical user accesses has grown multi-fold. It is no longer acceptable to enter one's username and password 25 times daily. Besides being insecure, this is inconvenient and expensive to administer. The lack of single-sign-on at the surface may seem like a relatively simple problem that causes some user inconvenience, but the impact across an enterprise is actually much deeper. Besides burdening the help-desk system, it increases administrative costs of each application, compromises security with the yellow-sticky-notes, and impedes productivity.

Various SSO solutions have been proposed that depend upon PKI, Kerberos, or password-store, but they require client side infrastructure and new administrative steps.

Since HTTP is a "stateless" protocol, it is difficult to differentiate between visits to a web site, unless the server can somehow "mark" a visitor. This “marking” is done by sending a piece of state information (called a *cookie*) to the client. Any future HTTP requests made by the client to this server includes this state.

Many web applications have started to use cookies for session management and single-sign-on. However, very little actual design and analysis information is available.

This paper analyses the security issues with both session and SSO cookies and proposes a scalable cookie based single sign-on architecture. This paper specifically does not address the single-sign-on problem across multiple tiers and how to propagate the identity across layers. It also does not address the single-sign-on problem for fat clients, IIOP clients and non-web servers. These issues are important, but not the focus of this paper.

2. General Requirements

Here are SSO requirements for web applications:

- Support multiple authentication systems such as name/passwords, certificates, one-time-passwords, etc.
- Support web access via middle tiers to legacy applications and provide name/password mapping
  - Scale to hundreds of thousands of users
- Simple to administer and configure
- Support a whole range of deployment scenarios:
  - Only one, to a few, physical web servers
  - Only one logical web server (multiple physical) for high availability and load balancing
  - Many web servers (> 5) supporting many applications but within the same domain
  - Multiple web servers spanning different domains
- Support policy considerations including authentication, authorization, and auditing

3. HTTP Cookies: An Overview

3.1 Cookie Specification

A cookie is sent to the client by including a Set-Cookie header as part of an HTTP response [1,2]. A cookie has 6 parameters: cookie name, value, expiration date, URL path for which the cookie is valid, the domain for which the cookie is valid, and whether the cookie should only be sent using secure connection with SSL.

3.2 Security Considerations for Cookies

3.2.1 Clear Text Cookies

Like all other *http* traffic, the information in the *Cookie* headers is unprotected unless SSL is used. Encryption of the cookie by itself provides limited extra security, because this same encrypted cookie can be played back to impersonate the user. Cookies are sometimes encrypted with the same session key, so if the attacker can find the session key for even one cookie, every user's cookie is now vulnerable. If the user authentication is based on username and password, the password should be sent through SSL.

We should only include the SSO session ID inside the cookie to avoid exposing any session information. The ID can be mapped to the session context on the web server.

3.2.2 Cookie Storage Security

We should assume that the cookies on the browser machine can be stolen through plug-ins or other means. The cookies should have limited lifetime (for example, eight hours) as determined by the risk tolerance by the business to limit the replay vulnerability. Furthermore, the server should request the browser to not store the cookie on the client's file system by marking it as an in-memory/session cookie.

3.2.3 Cookie Destination Control

The set of servers to whom this cookie can be sent is based upon the DNS naming structure. Cookies can be sent either only to the originating server or all the servers within the DNS domain of the originating server. There is no way to specify that a cookie should be sent only to individual servers from different domains such as *A.foo.com*, *B.foo.com*, and *C.bar.net*. Any of the DNS name server spoofing attacks can subvert these controls.

We neither want to send a cookie accidentally to anyone who is not in the SSO domain nor send a spoofed cookie to our servers. For example, if the domain name included is "*.us.oracle.com*", a visit to any of the web servers within this domain could set a cookie that is sent to any of the web servers under "*.us.oracle.com*". Given the proliferation of web servers in any intranet, this is possible and can be dangerous. We could create a sub domain which includes all such application servers.

3.2.4 Client IP Address Restriction

The regular http protocol does not secure the association between the HTTP data and the cookie. Including the client's IP address in the cookie can help thwart replay attacks, but this does not work very well in scenarios where network address translations (NAT) are involved with remote access servers (RAS), proxy servers (for firewalls) or otherwise. If there are multiple proxy servers for load balancing, the IP address may change between multiple HTTP connections.

3.2.5 Cookies as an URL parameter

Given some of the differences in which the browsers deal with cookies including allowing users to turn off cookies, some applications include cookies as part of the URL itself. This does not address the SSO issue, because the URL cookie cannot be shared across different web servers, but it does solve the problem of keeping session state across multiple *http* calls to the same base URL.

Cookies passed as URL parameters are arguably more secure than the http cookies, because there is no cookie storage and no domain security issues. Thus, where session management is the only requirement across multiple http calls to the same server, URL cookies should be preferred.

4. Cookies for a SSO Solution?

Cookies are quite useful for session management across multiple http calls, but given these security issues, why should we consider a SSO solution based upon cookies?

4.1 Appeal of the Cookie Based Approach

- Installation: No new extra software has to be installed on the browser side.
- User Experience: No change or minimum change. Even the authentication mechanism can remain the same as before. The cookies are not seen by the user.
- Independence with the authentication mechanism: The cookie mechanism can be supported with application username/password through http POST, basic http authentication, http digest authentication, secure-ID authentication, PKI or any other forms of authentication.
- Maintainability: Since cookies are server controlled, the format and content of the cookie can be changed easily without requiring any upgrades on clients. This makes it possible to quickly fix any vulnerabilities.

4.2 Drawbacks of the Cookie Based Approach

In addition to the security issues with session cookies, there are other vulnerabilities if cookies are used for SSO:

- Authentication and data relationship: Since there is no secure binding between the cookie and the actual data sent to the server, it is possible to steal a cookie and include it as part of some other transaction. This is in contrast with, say, Kerberos where it is not possible to transplant the ticket because there is a secure association between the server, client and the ticket.
- Who initially authenticates the user and how secure is that server? Participating servers trust other web servers that authenticate the user to issue an initial cookie. In Kerberos, the servers have to trust only the KDC.

Even though there are security vulnerabilities in the cookie approach, it is, however, a significant improvement over sending user names and passwords each time.

5. Possible SSO Solutions

In the following sections, we describe three approaches to provide SSO for web applications: Centralized Cookie Server, Decentralized Cookie Server, and Centralized Login Server. We have described and analyzed only the first approach in detail from various perspectives including performance, policy, and security. Many of the ideas and concepts from the first approach can be easily applied to the other two. Depending upon the specific requirements, one of these three, or a combination of the three, may be used for solving the SSO problem.

6. Centralized Cookie Server Approach

6.1 Overall Architecture

This section presents the overall high-level architecture for SSO using centralized cookie issuance and verification. Refer to Figure 1: Control Flow for SSO, below.

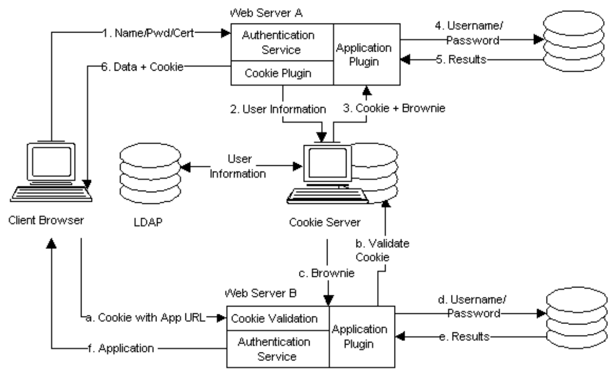

Figure 1: Control Flow for SSO

1. The user connects to Web Server A over *http* using any of the web server supported authentication services.
2. Web Server A sends over SSL relevant information to a centralized Cookie Server including the name of the user, requested expiry time, and any session specific information such as language, etc.
3. The Cookie Server creates a local entry in its database about this specific user session and returns the associated cookie and the cookie attributes (called *brownie*) back to Web Server A. One of the attributes of the brownie is the global user name.
4. Web Server A passes the user, session context, and credential information to the application plug-in. Based upon the target application's authentication requirements, the plug-in may fetch other user information from LDAP or any other user repository.
5. The database authenticates the user and if successful returns the data associated with this particular query.
6. Web Server A sends the cookie back in the *http* header along with the application data to the Client Browser.

Assuming that the user has successfully authenticated to Web Server A, and was able to get the cookie, the user can now login to any other Web Server 'B' within this SSO domain without doing any explicit login action. Here is the control flow:

a) If Web Server B is part of the SSO domain, the cookie is sent with the *http* request.

b) Web Server B passes the cookie to the cookie validation plug-in module. This module determines which cookie server should be contacted for validation and sends the cookie for validation.

c) The cookie server checks its own local database for validation and returns the brownie associated with the given cookie. The web server performs any other checks, if required, such as IP address checking or any other policies as defined by the SSO domain.

Steps d, e, and f in Figure 1 are identical to steps 4, 5, and 6 above except the cookie is *not* sent back to the browser.

Assuming that the web server has cached both the cookie and the brownie, any further interactions are much faster. It still has to perform any other required checks and check whether the cookie has not expired. It skips steps b, and c, and proceeds directly to the steps d, e, and f.

6.1.1 The Cookie Request Structure

The request sent from the web server to the cookie server contains the following information:

• Name of the user as known to the application
• Type of initial authentication used
• Client's IP address
• Requested expiry time (optional)
• Web server identity (implicit through the SSL identity)

6.1.2 The Cookie Result Structure

Since the cookie is sent back to the browser, we do not want to include anything more than what is absolutely necessary to identify the state information with the cookie. The cookie itself is not being encrypted because it does not contain any user-specific information that can be stolen. Besides its name, the cookie contains:

Bakery ID: This is the index of the cookie server where the cookie was first *baked*. This information is used by the web server to find the right cookie server for validation and to get the associated brownie.

Cookie identifier: The identifier should be long enough and random to make brute force attack impossible.

Cookie integrity check: The cookie could include a keyed MD5/SHA-1 digest to protect against modifications.

6.1.3 The Brownie Structure

The term "brownie" refers to the session structure pointed to by the cookie. Brownies are created by the cookie server on request of the authenticating web server. Note that brownies are *never* returned back to the browser, but are for the consumption of authorized web servers only. Brownies contain the following information:

- *Primary user as known to the authenticating web server* (e.g., user name, SSL identity). The global user name can be used by the validating web server to locate any other static information about the user from LDAP.
- *Primary authentication mechanism used:* The validating web server uses it to implement its own policies about acceptable authentication mechanisms.
- *Name of the login web server:* Used by the validating web server to determine whether they trust the login server to authenticate the user. The login server name is obtained implicitly through SSL connection.
- *User credential information:* There might be some specific user credential information available with LDAP. For example, the database username/password for accessing the database could be stored in LDAP.
- *IP address of the browser:* Used by the validating web server to determine whether it is coming from the same IP address. Specially useful if SSL is not used.
- *Time when the cookie was first baked:* Used by the validating server to implement session life-time policies.
- *Expiry time for cookie:* Keeping a much shorter expiry time (such as 15 minutes) for the cached cookie with the web server as compared to the expiry time for the http cookie allows for a very coarse-grain cache consistency between different web servers, especially if the user has logged out or if some brownie attributes have changed.
- *Session information included by the web server:* Used by the validating web server to implement its own session management: it may include role, responsibility, or language preference information. Attaching session information with the cookie obviates the need for the web server to create and manage a separate mechanism. The authorized web server can request all session attributes.

6.2 Policy Management

The cookie server can play an important role in implementing the cookie policy for the domain. For example:

- Accept requests for new cookies only from selected authenticating web servers?
- Acceptable types of primary authentication?
- Static cookie expiry policy: 8/4/2 hours?
- Dynamic expiry time based upon the type of authentication and the port of entry?
- Whether to accept cookie if coming from different IP address (as with mobile users)?
- Should secure flag be used? Should it depend upon the location of the client?
- Should the IP address of the client be checked?

Such policy information would typically be stored in LDAP and made available to all web and cookie servers.

6.3 Security Analysis

6.3.1 Cookie Security Analysis

We will look at the cookie life-cycle to examine the conditions under which the cookie can be attacked.

- *Initial user authentication:* If this is username/password based authentication, this is protected through SSL.
- *Initial cookie issuance:* The cookie server can control which web servers can initially authenticate the users.
- *Sniffing on the wire:* The cookie itself does not include any user information that is of any value.
- *Replay attack:* The risk can be reduced with certain implementation guidelines and policies:
  - For intranet users, IP address based security in combination with a shorter cookie lifetime can suffice for sites that do not want the SSL performance overhead.
  - For internet scenarios, use SSL when sensitive data is being accessed.
- *Cookie sent to the wrong server:* In conjunction with a spoofed DNS server, it is possible to persuade the browser to send the cookie to a wrong server. This risk can be reduced if the server is SSL authenticated.
- *Cookie modification:* The cookie is protected against modification by keyed-hash.
- *User log out:* Upon user log out, the web server clears the cookie inside the browser by sending an identical cookie with an expired time. It also informs the cookie server which then deletes this cookie and its associated brownie. If they are cached with other web servers, they are deleted only when their time-to-live value expires.

6.3.2 Brownie Security Analysis

Brownies are sent only to the authorized web servers using SSL. Since they may include information such as user credentials, they have to be protected on the web servers. The cookie server can decide which credentials should be sent to the web server based upon the policy configuration.

6.4 Performance and Scalability

6.4.1 User Authentication

SSL performance degradation (200-500ms) during login can be addressed with hardware accelerators.

6.4.2 Cookie creation

The initial authentication step requires one call to the cookie server to fetch the brownie and the cookie. Since the web server caches the cookie and the brownie, it can verify the cookie without going to the cookie server for all subsequent browser communication. A new cookie imposes the following "encryption" load on the servers:

- MD5/SHA-1 for cookie identifier on the cookie-server
- keyed-MD5 with RC2 for cookie integrity check
- SSL encryption for the cookie/brownie sent back and forth to the cookie server

6.4.3 Cookie Validation

Cookie validation is faster because the cookie and the brownie already exist at the cookie server. For the most common case of going back to the same web server, the web server can validate the cookie internally using the cached cookie and the brownie.

7. Decentralized Cookie Server Approach

One of the deficiencies of the centralized cookie server model is the run-time dependency on yet another server along with the administrative overhead of managing this service. This may be acceptable for larger sites, but this overhead may be a burden for smaller sites.

One alternative is that after authenticating the user as usual, the web server assembles the authentication information such as username, user IP address, web server name, and cookie expiry time, and digitally signs it. Basically, this signed cookie has all the information contained in the brownie except for passwords and any information that should not be exposed to anyone. The validating web server verifies the cookie signature and performs the policy checks.

The drawbacks include:

- Policies can be specified centrally, but there is no centralized policy enforcement. Adding a new policy requires a change within the web application itself.
- Including all identity information increases the size of the cookie and the network traffic for every transaction. Signature generation can take as much as 200 ms. This can impact a server’s Monday 8 AM load capacity.
- The applications would have to devise their own mechanism for maintaining session state.

This model is suitable for those applications that do not want to share session state between applications or already have an existing infrastructure to manage the state information between calls to the same server.

Some of the performance and bandwidth limitations can be alleviated if the login web server encrypts the cookie instead, and the validating server decrypts the cookie and performs integrity checks. One can use fairly simple out-of-band mechanisms to distribute the session key.

8. Centralized Login Server Approach

Even though it is possible for every web server to authenticate the user, it is better from the security perspective to assign the task of initially authenticating the user to a selected login server. This allows consolidation of user authentication information at one server. From the user's perspective, the password is changed only at one place. This makes it possible to implement the policy about user authentication at the login web server only.

One of the security features of http cookies is that cookies cannot be shared across domains. However, this gets in the way of the user experience if a user needs to single sign-on between *foo.com* and *bar.com*. This limitation can be bypassed if two sites cooperate and trust each other as described below and illustrated in Figure 2: Cross-Domain SSO Flow.

.

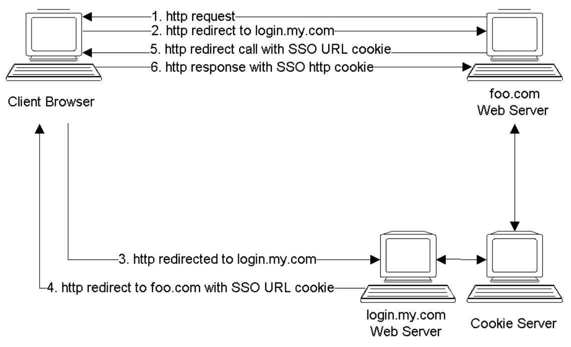
1. http request
2. http redirect to login.my.com
5. http redirect call with SSO URL cookie
6. http response with SSO http cookie
Client Browser
foo.com Web Server
3. http redirected to login.my.com
4. http redirect to foo.com with SSO URL cookie
login.my.com Web Server
Cookie Server

Figure 2: Cross-Domain SSO Flow

1. A user goes to any application site, *foo.com*.
2. Since a SSO cookie was not presented, the web server infers that this user has not signed in. The server *http redirects* to the login web server and includes the original URL used by the user.
3. If this user had earlier signed on to *login.my.com*, then the login *http* cookie would be sent with this request. If there were no existing cookie, the login server would do the appropriate authentication and create the login cookie and URL for the appropriate SSO cookie.
4. The login server sends an *https* redirect request to the original server using the URL (from the earlier request) along with the SSO cookie as an URL parameter.
5. The browser *http-redirects* to the original *foo.com* server with the original URL but now with the SSO cookie parameter. Note that we have bypassed the cookie domain constraint with this approach.

6. The web server verifies the cookie and returns its own session cookie to the browser along with the response.

Two cooperating web servers thus bypass the http cookie security constraints and provide single sign-on across servers in different domains.

9. Integrating with a PKI based Solution

If client certificates are available, the user authentication will take place using SSL with client-certificates instead of user-name/password. Cookies and brownies would then be used primarily for session information.

Cookies can improve the performance of authenticating the user to the secondary web servers. Today, not many web sites check certificate revocation status or any other information about the user (a time-consuming check). Assuming that the login web server issues the user a cookie only after a successful client side SSL authentication coupled with the certificate revocation check and any other checks as required by the enterprise, the subsequent login operations to other web servers will become faster.

While a PKI can provide for single sign-on across different applications, the cookie infrastructure is useful for implementing centralized policy and session management, supporting legacy applications, and mapping the primary username to the application-specific username.

10. Recommendations

Selecting any of these proposals requires a good understanding of the application and customer requirements.

For small sites, the encrypted cookie model with no dependence on a cookie server might be the best approach. Since the number of web servers are few, the administrator can share the cookie keys through out-of-band mechanisms. This system would be the fastest to deploy because there are very few outside dependencies. In such environments, policies typically do not change very often, and if they do, it may be acceptable to make these changes individually. This approach may also be very attractive to existing applications that already manage their own session state.

If out of band cookie key management is not acceptable, signed cookies should be considered.

For larger sites, the centralized cookie server approach offers centralized control and performance benefits. This model also accommodates multiple user repositories including legacy directories and applications. The cookie server model also allows for a manageable trust model. As applications evolve over time, there would be lot of information that they may want to share. Since this list would continue to evolve, it is better if this information is not hard coded within the applications themselves.

We recommend that sites adopt the centralized login server model to consolidate passwords to one place and also allow users to modify their passwords only at one place. If the site moves to PKI, the centralized login server can check the certificate revocation status and life-cycle management functions as determined by site policy.

Any cookie based design is vulnerable to stolen cookies and spoofing attacks. Because of the various security risks, we recommend that this cookie based SSO solution be considered initially for intranet applications. We acknowledge that this solution is weaker than a PKI based SSO solution, but this practical solution would be significantly stronger than the current practices.

11. Conclusions

With the growing proliferation of web applications, users are forced to deal with multiple user names and multiple passwords many times a day. Single sign-on can be addressed through client side certificates, but as yet, very few people are using client side certificates. In absence of such an infrastructure, it is important to be able to address the single sign-on issue for a large majority of users now.

Cookie properties such as its opaqueness to regular users and its full control by the web server, make cookies very appealing for a single sign-on solution. While recognizing the security deficiencies with the cookie mechanism, we have proposed solutions for achieving single sign-on and session management. Depending upon the customer requirements of deployability, performance and centralized management, an appropriate single-sign-on solution can be chosen for web applications.

12. Acknowledgments

This paper has benefited from the experience of various groups within Oracle Corporation who are planning to use cookies for session management and single-sign-on. In particular, I would like to thank Gordon Buhle, Gilbert Gonzalez, Naresh Kumar, Douglas McMahon, Tri Phan, Andrew Philips, Edward Screven, Malcolm Speedie, Subash Srivastava, and Craig Stephen for discussing these ideas and reviewing earlier drafts of this paper.

13. References

[1] RFC 2109: HTTP State Management Mechanism

[2] HTTP Cookies:
http://www.netscape.com/newsref/std/cookie_spec.html