Authentication for Web Services

Ray Miller <raym@herald.ox.ac.uk>
Systems Development and Support
Computing Services, University of Oxford
Overview

- Password-based authentication
- Cookie-based authentication
- Using cookies for single sign-on
- Kerberos
- Stanford WebAuth
- Integrating uPortal and WebAuth
Basic authentication

- The simplest authentication scheme used for web services is **HTTP Basic Authentication**, defined in RFC 1945

- When a client requests a protected resource, the server responds with **401 (Authorization required)**

- The client resends the request with an **Authorization** header encoding the username and password
Basic authentication

- Often used in conjunction with `htaccess` and `htpasswd` files under the control of the information provider

- Username and password information can also be stored in a database (GNU DBM, Berkeley DB, mySQL, PostgreSQL, LDAP)
Basic authentication

- HTTP is a *stateless* protocol: every time a client requests a protected resource, it must include an *Authorization* header that the server must verify.

- If account information is held in a remote database, the web server will have to verify the username and password against the remote data source for every client request.
Cookies
Cookies

• When responding to an HTTP request, a server may also send a piece of state information that the client will store:

  Set-Cookie: NAME=VALUE; expires=DATE; path=PATH; domain=DOMAIN_NAME; secure

• This state object is known as a cookie

• The cookie will be sent by the client along with any future requests to the specified domain/path
Cookies

- Cookies enable state to be stored between HTTP requests
- A server can issue a session cookie after authenticating a user for the first time
- The server can use the session cookie to identify the user when they make future requests
- No need to verify the username and password for every request
Using cookies for SSO

- By introducing a central log-in server, we can use cookies to provide intra-institutional single sign-on for web services
  - Pubcookie (University of Washington)
  - BrownTicket (Brown University)
  - Cosign (University of Michigan)
  - WebAuth (Stanford)
SSO components

• These cookie-based single sign-on systems all consist of several components:
  • a user agent (web browser)
  • an application server
  • a login server
  • an external authentication service
The login server

- trusted, central authentication service
- interacts directly with users
- verifies usernames and passwords with backend authentication services
- issues cookies to users to provide single sign-on functionality
- provides authentication information to application servers
The application server

- authentication enforcer
- redirects users who haven’t been authenticated to the login server
- verifies authentication information returned from the login server
- issues cookies to maintain authenticated application sessions
- provides user authentication information to applications
The external authentication service

- verifies user authentication information sent from the login server
  - Kerberos
  - LDAP
  - NIS
How it works
This really is SSO

• If the user requests a protected resource from a new application server, they will be redirected to the login server

• They already have a cookie for the login server, so they don’t need to re-authenticate

• The login server simply issues the appropriate cookie and directs them back to the application server
Benefits

• Passwords, if used, are only sent to the central login server over SSL
• Users need only authenticate once per session to access any protected resource
• Can leverage an existing authentication system
• Works with almost all modern browsers
We’ve seen this before...
We’ve seen this before...

- This looks a lot like the Kerberos single sign-on protocol:
  - cookies issued by the login server grant access to a particular application server; they are like Kerberos service tickets
  - the cookie shared between the browser and the login server is like a Kerberos ticket-granting ticket
Stanford’s WebAuth

• Takes the Kerberos analogue a step further
  • The cookie shared with the login server *really is* a Kerberos ticket-granting ticket
  • The cookie issued for access to an application server *really is* a Kerberos ticket
Why does this matter?

- The designers of the Kerberos protocol have already solved many of the security problems associated with ticketing systems
There’s another advantage

• What happens when the application server needs authenticated access to another service?

• For example, Herald’s WING servers need to authenticate the user when opening a connection to an IMAP server on their behalf
Secondary tickets

- Stanford’s WebAuth system includes support for *secondary tickets*
- When the application server redirects the user to the login server, it can request tickets for additional services
- The user’s ticket-granting ticket shared with the login server takes care of this
- Multiple tickets are encoded in cookies sent to the application server
The WebAuth login server

- The login server component of WebAuth consists of two CGI programs, `login.fcgi` and `logout.fcgi`, and an Apache 2.0 module, `mod_webkdc`.

- You don’t need to worry about this: the login server will be provided centrally.
The WebAuth application server

• The *mod_webauth* module provides an authentication handler for Apache 2.0

LoadModule webauth_module modules/mod_webauth.so

WebAuthKeyring conf/webauth/keyring
WebAuthKeytab conf/webauth/keytab
WebAuthServiceTokenCache conf/webauth/service_token_cache
WebAuthLoginURL https://webkdc/login/
WebAuthWebKdcURL https://webkdc/webkdc-service/
WebAuthWebKdcPrincipal service/webkdc

<Location /private/>
AuthType WebAuth
Require valid-user
</Location>
The WebAuth application server

- The WebAuth module provides information in additional environment variables to the SSI and CGI namespace

<table>
<thead>
<tr>
<th>Variable</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>WEBAUTH_USER</td>
<td>Name of the WebAuth authenticated user</td>
</tr>
<tr>
<td>WEBAUTH_TOKEN_CREATION</td>
<td>When the token was created</td>
</tr>
<tr>
<td>WEBAUTH_TOKEN_EXPIRATION</td>
<td>When the token will expire</td>
</tr>
<tr>
<td>WEBAUTH_TOKEN_LASTUSED</td>
<td>When the token was last used</td>
</tr>
<tr>
<td>REMOTE_USER</td>
<td>Name of the WebAuth authenticated user</td>
</tr>
<tr>
<td>AUTH_TYPE</td>
<td>Set to WebAuth</td>
</tr>
</tbody>
</table>
Integration with Perl and PHP

- Perl CGI scripts can access information about the authenticated user via environment variables

  ```perl
  my $webauth_user = $ENV{'WEBAUTH_USER'}
  ```

- As can PHP

  ```php
  $webauth_user = getenv('WEBAUTH_USER');
  ```
Integration with Java/Tomcat

- When Apache 2.0 is used to proxy requests to Tomcat, mod_webauth can be used in conjunction with mod_jk in order to pass environment variables to the Java servlet
The sticky issue of authorization

• Now that we’ve authenticated the user, how do we decide whether or not they’re authorized to use this service?
The sticky issue of authorization

- The latest version of WebAuth includes an Apache 2.0 authorization module, `mod_webauthldap`
- Can allow access to groups of individuals defined in LDAP
- Can include arbitrary attributes from an LDAP record in environment variables
The sticky issue of authorization

• Alternatively, an application can make its own authorization decision based on the identity of the authenticated user.

• For example, by performing its own LDAP lookup.
Integration with uPortal

• We should be able to use Apache 2.0 with mod_webauth and mod_jk to proxy requests to the Tomcat instance running uPortal

• Will require some changes to uPortal to extract information about the authenticated user from environment variables
Integration with uPortal

- This might not be flexible enough for uPortal channels that require secondary tickets

- Implement the WebAuth protocol in pure Java for uPortal?
Where we are now (May 2003)

• We have a working kdc (Kerberos domain controller), but none of our services are Kerberos-enabled

• We have a working WebAuth login server

• We have a working WebAuth application server
Where we are now (May 2003)

- We have not yet tried the latest WebAuth with LDAP support for authorization
- We have not set up the Apache 2.0 proxy to uPortal
- The kdc and WebAuth login server are some way from being production-ready
References

- http://users.ox.ac.uk/~raym/talks/webauth.pdf
- http://www.pubcookie.org/
- http://www.brown.edu/Facilities/CIS/Network_Services/web-auth/
- http://www.umich.edu/~umweb/software/cosign/
- http://webauthv3.stanford.edu/
- http://web.mit.edu/kerberos/www/