Interstage Support

Fujitsu America Inc.
1250 East Arques Avenue
Sunnyvale, CA 94085

Java Servlet Support for Lightweight Open Authentication

Updated: 14 August 2015
Editor: Keith Swenson

This is a design document for a protocol for the SSOFI Provider to allow for three-party authentiation.

Table of Contents

1 Executive Summary

Please see the associated document on the Lightweight Open Authentication Protocol to understand how the exchange works. This document describes two java classes that are useful for implementing the SERVER side of the protocol in the a java server.

2 Goals

The server part of the interaction must do two things:

2.1 Generate a Challenge

The client will make a call asking for a randomly generated challenge. The challenge is a string, and the only requirement is that this string be unique. No two calls should return the same challenge. The challenge is the unique key for the login attempt, and no two attempts should have the same key. The challenge should nto be easily guessable either, because a hacker that could guess the value of a challenge might use that to gain access. So the challenge should be a long value, and randomly generated.

The challenge, value is returned to the client, and it is also remembered as part of a session with the client.

2.2 Verify a Token

The client will give the challenge to the identity provider, and will receive a token. Using the challenge and the token the server can then ask the identity provider who the user is that was logged in at the time that the token was gotten. If the challenge and token do not match, or if the provider has never seen one or both of the values, then verification will fail. A failure is simply that: the user is not logged in. If the challenge and token match with its own records, then verification is returned as true. Along with the positive verification, the identify server delivers the email address and user name, and the server considers them logged in.

2.3 Assumptions

HttpSession: The server is stateless EXCEPT that it remembers a session for the user. This is accomplished with cookies. When a browser makes multiple requests to the server, the server knows that it is the same browser because of the cookie. Java server environments, like TomCat, manage the cookies for you, and represent the session using an HttpSession object. The approach this code takes is to store user information in this HttpSession object. (Other session approached could be made to work, but for discussion we will focus on this one.)

Into the session we will put a single object instance: AuthStatus. This will have methods to clear indicate the state of the exchange: authenticated or not. It will also have members for the bookkeeping while the procol is working through all the steps.

The http requests will be received by a Servlet object, which is passed HttpServletRequest and HttpServletResponse objects. This is the LightweightAuthServlet class

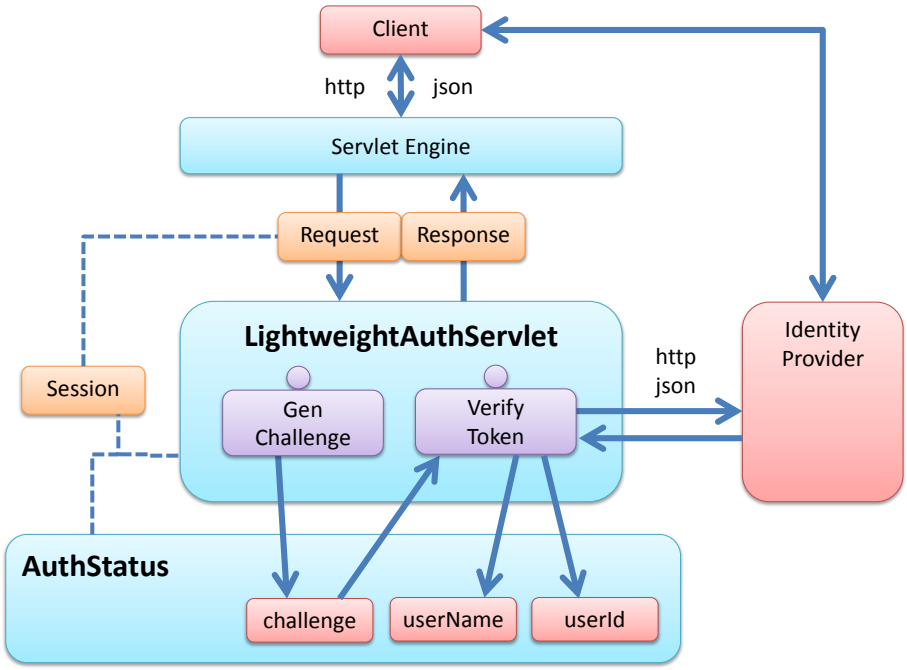
We assume this servlet is mapped to the path “/auth/” in the application. Clearly it could be mapped elsewhere, but for discussion will be assume it is there.

The server is configured with a single trusted identity provider. It will check only with that one provider, so any challenge/token pair made at a different provider will obviously not validate. The client must use the same identity provider.

To keep things simple, only two classes are needed.

3 Architecture

This diagram depicts the roles and relationships of the two classes:



The host program need communicate only with the AuthStatus instance, which it gets from the session object. From this object it sees whether the user is authenticated or not. Any other servlets servicing reuqests should first get this object, find out if the user is authenticated, find out who the user is, and then limit access accordingly.

The LightweightAuthServlet handles the requests from the client. GenChallenge is easy, t just generates a large random challenge value and returns it to the client. The other operation, VerifyToken, is a little more involved. It has to make a call to the identity provider, to verify the challenge and token association that it has been given, and to pick up the verified user id and user name for that user. It stores these in the AuthStatus object.

4 AuthStatus object

Each user session gets an AuthStatus object. You get the auth status with the following call:

```
AuthStatus aStat = AuthStatus.getAuthStatus(req.getSession());
```

Where req is a HttpServletRequest object. It looks into the session for an instance, and if it does not find one, it creates one, and stores it in the session.

These methods are available:

```
boolean isAuthenticated()
```

says whether the user is authenticated or not.

```
String getId()
```

Returns the Id (email address actually) of the user (if logged in).

```
String getName()
```

Returns the display name of the user (if logged in)

```
Void logout()
```

If for some reason the host program wants to invalidate the user login status, a call to logout will clear the settings and make it one again as if the user is not logged in. It is not clear that you ever need to do this, since session timeout will dispose of all this. If the user wants to log in as someone else, the login sequence will replace the settings with those of the new user.

Those are the only methods designed to be used by the host system. There are a few more apis that are used by the LightweightAuthServlet class to remember the challenge, and to set things.

5 LightweightAuthServlet class

This is installed like any servlet class, and associated with the address "/auth/"

There are two calls that the servlet will support

```
http://<server>/<app>/auth/getChallenge
http://<server>/<app>/auth/verifyToken
```

Both of these are POST method operations. They both take JSON formatted input, and return JSON output.

For debugging purposes, the GET method is support, and it simply returns an indicator of whether the session has a logged in user or not.

6 Source

6.1 AuthStatus Class

```
package org.socialbiz.cog.api;

import javax.servlet.http.HttpSession;

public class AuthStatus {

    //these are official verified value from trusted authentication server
    private String authID = null;
    private String authName = null;
    private String challenge = "";

    public static AuthStatus getAuthStatus(HttpSession session) {

        AuthStatus aStat = (AuthStatus) session.getAttribute("AuthStatus");
        if (aStat == null) {
            aStat = new AuthStatus();
            session.setAttribute("AuthStatus", aStat);
        }
        return aStat;
    }

    public boolean isAuthenticated() {
        return (authID!=null);
    }

    public void logout() {
        authID = null;
        authName = null;
    }

    public String getId() {
        return authID;
    }
    public void setId(String newID) {
        authID = newID;
    }

    public String getName() {
        return authName;
    }
    public void setName(String newName) {
        authName = newName;
    }

    public String generateChallenge() {
        challenge = generateKey();
        return challenge;
    }
    public void clearChallenge() {
        challenge = "";
    }
    public String getChallenge() {
        return challenge;
    }
```

```
    private static long lastKey = System.currentTimeMillis();
    private static char[] thirtySix = new char[] {'0','1','2','3','4','5','6','7','8','9',
        'a','b','c','d','e','f','g','h','i','j', 'k','l','m','n','o','p','q','r','s','t',
        'u','v','w','x','y','z'};
    /**
    * Generates a value based on the current time, and the time of
    * the previous id generation, but checking also
    * that it has not given out this value before.  If a key has
    * already been given out for the current time, it increments
    * by one.  This method works as long as on the average you
    * get less than one ID per millisecond.
    */
    private synchronized static String generateKey() {
        long ctime = System.currentTimeMillis();
        if (ctime <= lastKey) {
            ctime = lastKey+1;
        }
        long lastctime = lastKey;
        lastKey = ctime;

        //now convert timestamp into cryptic alpha string
        //start with the server defined prefix based on mac address
        StringBuffer res = new StringBuffer(8);
        while (ctime>0) {
            res.append(thirtySix[(int)(ctime % 36)]);
            res.append(thirtySix[(int)(lastctime % 10)]);  //always a numeral
            ctime = ctime / 36;
            lastctime = lastctime / 10;
        }
        return res.toString();
    }
}
```

6.2 LightweightAuthServlet

```
package org.socialbiz.cog.api;

import java.io.InputStream;
import java.io.OutputStream;
import java.io.OutputStreamWriter;
import java.io.Writer;
import java.net.HttpURLConnection;
import java.net.URL;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

import org.workcast.json.JSONArray;
import org.workcast.json.JSONObject;
import org.workcast.json.JSONTokener;

/**
 * This servlet implements the Lightweight Authentication Protocol
 * which consists of two operations, both POST method, and both
 * receiving and returning JSON structures.  A JavaScript client
 * can use these method to authenticate the user to this server.
 *
 * <server>/<app>/auth/getChallenge
 *
 * This receives a request for a challenge value, generates a challenge
```

```
 * value, associates the challenge with the current session, and returns
 * the challenge in a JSON file.
 *
 * <server>/<app>/auth/verifyToken
 *
 * This receives a JSON with the challenge, the token from the auth
 * provider.  This will make a call to the auth provider, and if
 * it gets an acceptable response (verified) then it records the
 * user information in the session, and returns a value saying
 * that the user is logged in.
 *
 *
 */
public class LightweightAuthServlet extends javax.servlet.http.HttpServlet {

    private static String trusterProviderUrl = "https://interstagebpm.com/eid/";

    public static void init(String _trusterProviderUrl) {
        trusterProviderUrl = _trusterProviderUrl;
    }

    private void setRequirements(HttpServletRequest req, HttpServletResponse resp) {
        //this is an API to be read by others, for these really strange rules
        //set up by the browsers, you have to set the CORS to
        //allow scripts to read this data from a browser.
        //Longer story, the browser will accept setting allow origin to '*'
        //but if you do, it will not send the cookies.  If you tell it to send
        //the cookies with "withCredentials" then it will not allows the '*'
        //setting on allow origin any more.  The only alternative is that you
        //MUST copy the origin from the request into the response.
        //This is truely strange, but required.

        String origin = req.getHeader("Origin");
        if (origin==null || origin.length()==0) {
            System.out.println("COG-LAuth: got a null origin header???");
            //this does not always work, but what else can we do?
            origin="*";
        }
        resp.setHeader("Access-Control-Allow-Origin",      origin);
        resp.setHeader("Access-Control-Allow-Credentials", "true");
        resp.setHeader("Access-Control-Allow-Methods",     "GET, POST, OPTIONS");
        resp.setHeader("Access-Control-Allow-Headers",     "Authorization");
        resp.setHeader("Access-Control-Max-Age",           "1");
        resp.setHeader("Vary",                             "*");
    }

    @Override
    public void doGet(HttpServletRequest req, HttpServletResponse resp) {

        setRequirements(req, resp);
        AuthStatus aStat = AuthStatus.getAuthStatus(req.getSession());
        JSONObject jo = new JSONObject();
        try {
            if (aStat.isAuthenticated()) {
                jo.put("msg",  "You appear to be logged in as "+aStat.getName());
            }
            else {
                jo.put("msg",  "You are not logged in");
            }
            Writer w = resp.getWriter();
            jo.write(w);
```

```
            w.flush();
        } catch (Exception e) {
            System.out.println("COG-LAuth FAILURE handling GET request "+e);
            e.printStackTrace(System.out);
        }
    }

    @Override
    public void doPost(HttpServletRequest req, HttpServletResponse resp) {

        setRequirements(req, resp);
        Writer w = null;
        String pathInfo = req.getPathInfo();

        resp.setContentType("application/json");
        try {
            w = resp.getWriter();
            System.out.println("COG-LAuth POST: "+pathInfo);
            AuthStatus aStat = AuthStatus.getAuthStatus(req.getSession());

            //receive the JSONObject
            InputStream is = req.getInputStream();
            JSONTokener jt = new JSONTokener(is);
            JSONObject objIn = new JSONObject(jt);
            is.close();

            if (pathInfo.startsWith("/getChallenge")) {

                String challenge = aStat.generateChallenge();
                objIn.put("challenge",  challenge);
                objIn.write(w, 2, 0);
                w.flush();
            }
            else if (pathInfo.startsWith("/verifyToken")) {

                String chg1 = aStat.getChallenge();
                String chg2 = objIn.getString("challenge");

                if (objIn.getString("token").length()==0) {
                    throw new Exception("Need to have a 'token' member of the "
                                +"passed JSON in order to verify it.");
                }

                if (!chg1.equals(chg2)) {
                    System.out.println("COG-LAuth - challenge ("+chg1+") does "
                                +"not match passed ("+chg2+")");
                    aStat.clearChallenge();
                    throw new Exception("Got a request to verify a token and "
                            +"challenge that is not that which was given out.  "
                            +"Authentication transaction aborted.");
                }

                objIn.put("challenge", chg2);

                if (trusterProviderUrl==null || trusterProviderUrl.length()==0) {
                    throw new Exception("the LightweightAuthServlet has not been "
                           +"initialized with the address of the provider");
                }

                //Now, actually call the provider and see if this is true
                String destUrl = trusterProviderUrl + "?openid.mode=apiVerify";
```

```
                JSONObject response = postToRemote(new URL(destUrl), objIn);
                Boolean valid = response.getBoolean("Verfied");

                if (valid) {
                    aStat.setId(response.getString("userId"));
                    aStat.setName(response.getString("userName"));
                }
                else {
                    //after a failed login, don't leave any previous login around....
                    aStat.logout();
                }
                response.write(w, 2, 0);
                w.flush();
            }
            else {
                throw new Exception("Lightweight Auth Servlet can not handle address:”
                                    +pathInfo);
            }
        }
        catch (Exception e) {
            System.out.println("COG-LAuth FAILURE handling "+pathInfo);
            e.printStackTrace(System.out);
            try {
                JSONObject err = new JSONObject();
                JSONArray msgs = new JSONArray();
                Throwable t = e;
                while (t!=null) {
                    msgs.put(t.toString());
                    t = t.getCause();
                }
                err.put("exception", msgs);
                if (w!=null) {
                    err.write(w,2,0);
                    w.flush();
                }
            }
            catch (Exception eeee) {
                //can't seem to do anything to let the client know.
                System.out.println("COG-LAuth FAILURE sending exception to client: "+ eeee);
            }
        }
    }

    /**
     * Send a JSONObject to this server as a POST and
     * get a JSONObject back with the response.
     */
    private static JSONObject postToRemote(URL url, JSONObject msg) throws Exception {
        try {
            HttpURLConnection httpCon = (HttpURLConnection) url.openConnection();
            httpCon.setDoOutput(true);
            httpCon.setDoInput(true);
            httpCon.setUseCaches(false);
            httpCon.setRequestProperty( "Content-Type", "text/plain" );
            httpCon.setRequestProperty("Origin", "http://bogus.example.com/");

            httpCon.setRequestMethod("POST");
            httpCon.connect();
            OutputStream os = httpCon.getOutputStream();
            OutputStreamWriter osw = new OutputStreamWriter(os, "UTF-8");
            msg.write(osw, 2, 0);
```

```
            osw.flush();
            osw.close();
            os.close();


            InputStream is = httpCon.getInputStream();
            JSONTokener jt = new JSONTokener(is);
            JSONObject resp = new JSONObject(jt);

            return resp;
        }
        catch (Exception e) {
            throw new Exception("Unable to call the server site located at "+url, e);
        }
    }


}
```