RunSignUp OAuth Integration

Audience

This document is intended for a web application software developer, independent of the language or development environment they may be using. Some familiarity with the basics of the HTTP protocol (http://tools.ietf.org/html/rfc2616) is also assumed.

Introduction

The Process of authenticating users with their RunSignUp credentials allows you to defer your user authentication to RunSignup.com. This process is similar to what you might go through to call any of the RunSignUp API functions except here the goal is to redirect the users from your web site to RunSignUp.com long enough for the user to log in and then to use the authorization obtained when they log in to request information about who they are which you can then use to identify them so they have authorized access from your site/application.

The minimum amount of information you would expect to obtain would be their user name or id. You can then use this information to track the authenticated user on your site as they go about their activities. This authentication will last for a set period of time and then the user may be asked to renew the approval they have obtained.

This processed is achieved though OAuth 1.0a protocol. This protocol uses a series of trusted steps to guarantee that not only the user is who they say

they are, but that your website also has authorization to authenticate users at RunSignUp.com. This is done through the exchange of secrets between your website and RugnSignup.com.

This document will discuss the process of registering your website and then using the information you obtain to authenticate and identify your users after they have been validated by providing their credentials to RungSignup.com.

This process is different across the many platforms and languages found on the Internet. To better familiarize you with how deferring authentication is performed, the topic will be covered in the general case and then examples will be discussed in several programming environments to provide practical examples.

If you need more information about this process than is discussed in this document consult RFC-5849, The OAuth 1.0a Protocol at https://tools.ietf.org/html/rfc5849 or the OAuth 1.0a specification at http://oauth.net/core/1.0a .

The General Procedure for OAuth 1.0a authentication.

Step 1. Obtain the authentication endpoints and your shared secrets.

Any registered user of RunSignUp.com can register their web site or application. Once you have registered as a RunSignup user, go to the following URL to register your application.

https://runsignup.com/API

Now go to the API keys section on this page. Make sure you have already signed in. If you have not signed in you will not be able to proceed.

Your Endpoint URLs

On this page your will see a section called OAuth Details. Copy down all three URLs listed there. These are:

```
Request Token Endpoint:
https://runsignup.com/oauth/requestToken.php
OAuth Login URL:
https://runsignup.com/OAuth/Verify
Access Token Endpoint:
https://runsignup.com/oauth/accessToken.php
```

These URLs are the same for all users and represent the endpoints you will be interacting with during each step of the OAuth authentication process.

Your Application Name

Next, use the Register New Application button. This will display a dialog asking you to provide an Application Name and an Application URL. Your Application Name can be anything and is simply used to identify your application in the list of registered applications. Your application URL should be a URL on your website that contains a script that knows how to extract authentication token information from it when it is called by RunSignup.com.

Your Application/Callback URL

Since you may be in the process of developing this functionality, you may not yet have a script like this available and that is not a problem. This URL can be changed later but it is important that you use this exact URL as part of the authentication process. This URL is referred to as the callback URL because this is the URL that RunSignUp will use to direct your users back to your web site once they have been authenticated.

If you do not already know your callback URL, you can use a URL that will work during the development process. For example:

`http://localhost:8080/mysite/auth_callback`

Would work just fine during the development process. This is because, while testing your RunSignUp authentication, your machine knows how to resolve localhost and since you will initially be the only user of your site during the testing process, and your site is running on your local machine, this works just fine.

Keep in mind that once you are ready to go live with RunSignUp.com authentication, this callback URL must be changed to your registered domain name so that when RunSignUp redirects your customers back to you, they can actually find you. This URL should eventually be changed to something like:

`https://mydomain.com/mysite/auth_callback`

But for now, localhost and the port you are testing your application on will work fine.

Record Your Secrets

You actually have three secrets; the first is your Callback URL, which you already have. The other two, your OAuth Key and your OAuth Secret,

combined with the first, are how your website will prove you are who you claim to be. Make sure to keep these pieces of information protected on your website. For now record this information for use in the next step. Note the terms in parenthesis next each title. These are the OAuth terms for each piece of information below and this is how they will be referred to in this document going forward.

Summary

You now know the following facts:

Request Token Endpoint (Temporary Credential Request URI):
`https://runsignup.com/oauth/requestToken.php`
OAuth Login URL (Resource Owner Authorization URI):
`https://runsignup.com/OAuth/Verify`
Access Token Endpoint (Token Request URI):
`https://runsignup.com/oauth/accessToken.php`
Callback URL (oauth_callback):
`http://localhost:8080/mysite/auth_callback`
OAuth Key (oauth_consumer_key):
`cd54e9c9941f147ff37baf1e7260044054c905b1 (SAMPLE)`
OAuth Secret (oauth_consumer_secret):
`2b0357599155dcc1f79cff0147c9bd49d26f92b7  (SAMPLE)`

The last three values will, of course, be different but you have now registered and collected all the information you need to proceed on to the next step.

Step 2. Exchange your web site's secrets for permission to authenticate a user.

Your goal now is to gain permission to access RunSignUp's authentication page to allow it to authenticate your user. To access this page you must be able form the correct request to get the authentication page to display for your to web site. Because OAuth 1.0 does not require that your request come over an encrypted connection (HTTPS) you must sign the request to prevent anyone else from using your authentication secrets if they are intercepted.

This permission request must be sent to the Temporary Credential Request URI listed above. The response will be either a new token (oauth_token) and a new secret (oauth_token_secret) if your site's credentials and signature are valid or a failure. A properly formed http request looks like this:

```
POST /oauth/requestToken.php HTTP/1.1
Authorization: OAuth
oauth_callback="http%3A%2F%2F127.0.0.1%3A3000%2Fauth%2Fex
ample%2Fcallback",oauth_consumer_key="cd54e9c8741f147ff37
3af1e7260044054c905b1",oauth_nonce="QgGjlnchj1AlJiR28Ax70
Ylwy5voIi2c",oauth_signature_method="HMAC-
SHA1",oauth_timestamp="1421728747",oauth_version="1.0",oa
uth_signature="sirJ6LqEXeD9UXczetitsm%2B4Qw4%3D"
Host: www.runsignup.com
Accept: */*
Connection: close
User-Agent: Node authentication
Content-length: 0
Content-Type: application/x-www-form-urlencoded
```

Note that authorization occurs in the http header and not as request parameters or posted values in the request. Also note that the **oauth_consumer_secret** does not actually occur in the authorization header. Instead, there is a new parameter called **oauth_signature**, which is generated by encrypting all of the other parameters with the **oauth_consumer_secret** using the method specified in the oauth_signature_method. You will be using a client authentication library to mange the creation of your authorization header but it is important to remember that you will be exchanging these parameters for this response:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded
oauth_token=hh5s93j4hdidpola&oauth_token_secret=hdhd0244k
9j7ao03&oauth_callback_confirmed=true
```

Or if something goes wrong, it's important to know that RunSignUp.com will return an explanation for your rejection in the response body. Here is an example of a request failure:

```
oauth_problem=parameter_absent&oauth_parameters_absent=oa
uth_callback
```

Here you are being told that you did not present or provide the correct **oauth_callback**. This feedback can be invaluable in diagnosing where things have gone wrong.

After all of this, it's important to keep in mind that in step two we have exchanged our **oauth_callback**, **oauth_consumer_key** and **oauth_consumer_secret** for an **oauth_token** and an **oauth_token_secret**.

These parameters will be needed to form the URL you must redirect to at RunSignUp.com to have the user log in (The resource owner authorization URI above). The OAuth client usually performs this exchange without any user interaction (synchronously) on the server and it is easy to miss that there is an exchange going on at all.

Step 3. Redirect your user to the Resource Owner Authorization URI

Once step two has been completed, you are ready to redirect the users to RunSignUp's authorization web page. Since step two can be completed without involving the user, step three is usually part of the same script as step two. Using your client software, the Resource owner authorization URI (see step two) is modified to add your newly obtained **oauth_token** and an **oauth_token_secret.** It is now your site's responsibility to redirect the user to this newly formed URL. Once this is done, your site will wait until they return using the **auth_callback** URL. After authentication is completed at RunSignUp.com, they will be redirected to the **auth_callback** URL and you can extract the information to proceed with step 4.

Step 4. Respond when your auth_callback URL is called

RunSignUp.com will call your auth_callback URL. When it does it will have an **oauth_verifier** in its URL parameter. This **oauth_verifier** can now be exchanged for a new **oauth_token** and **oauth_token_secret.** This new token-secret pair carries with it, the permission of the authenticated user. This has been the goal all along. When your **auth_callback** is called, that URL will look something like this.

```
http://localhost:8080/mysite/auth_callback?
oauth_token=hh5s93j4hdidpola&oauth_verifier=hfdp7dh39dks9
884
```

You must now respond by taking this **oauth_verifier** and sending it back to RunSignUp.com at the Token Request URI (that you obtained in Step 1) so that you can obtain the token-signature pair that represents your authenticated user. That request should look like this:

```
POST /token HTTP/1.1
Host: runsignup.com
Authorization: OAuth
oauth_consumer_key="
cd54e9c9941f147ff37baf1e7260044054c905b1",
oauth_token="hh5s93j4hdidpola",
oauth_signature_method="HMAC-SHA1",
oauth_timestamp="137131201",
oauth_nonce="walatlh",
```

```
oauth_verifier="hfdp7dh39dks9884",
oauth_signature="gKgrFCywp7rO0OXSjdot%2FIHF7IU%3D"
```

It will be the responsibility of your client software to add these parameters to this URL and then send the request synchronously and wait for the response before continuing. The response will look like this:

```
HTTP/1.1 200 OK
Content-Type: application/x-www-form-urlencoded

oauth_token=nnch734d00sl2jdk&oauth_token_secret=pfkkdhi9sl3r4s00
```

You have now received the users OAuth token and secret. This should be stored, either in their session or for longer term use, in a database if further API calls are to be attempted in their name. For the needs of completing this authentication, it is assumed that you have at least preserved it in the session.

If you allow the user to remain logged in beyond the lifetime of your session then you will most likely have stored some piece of identifying information in a cookie. Since this information will be unique to your site, you should use this information to look up the last recorded **oauth_token** and **oauth_secret**. When revalidating this user, you will be expected to perform step 4 again to make sure these tokens are still valid.

Step 4. Requesting User Profile Information

The act of using your user's **oauth_token** and **oauth_secret** to access the user's profile information is the only means you have of making sure that they are valid and have not expired. Since you will have to generate an **oauth_signature** to accompany this request, you will have to use your OAuth client to prepare the request. Simply creating the header without the correct **oauth_signature** in the header will cause the request to fail.

Below is an example of how you would request the user profile from RunSignUp once you have obtained the user's tokens.

```
GET /rest/user?format=json HTTP/1.1
   Host: photos.example.net
   Authorization: OAuth
      oauth_consumer_key="dpf43f3p2l4k3l03",
      oauth_token="nnch734d00sl2jdk",
      oauth_signature_method="HMAC-SHA1",
      oauth_timestamp="137131202",
```

```
            oauth_nonce="chapoH",

   oauth_signature="MdpQcU8iPSUjWoN%2FUDMsK2sui9I%3D"
```

Which will return this user profile in the document body.


```
{
   "user": {
      "user_id": 3114896,
      "first_name": "Bill",
      "last_name": "Reichardt",
      "email": "bill.reichardt@gmail.com",
      "address": {
         "street": "107 Jadds Lane",
         "city": "Northville",
         "state": "NJ",
         "zipcode": "08033",
         "country_code": "US"
      },
      "dob": null,
      "gender": "M",
      "phone": "856-333-9373",
      "profile_image_url":
"\/\/d368g9lw5ileu7.cloudfront.net\/users\/user3114896_profile.bs
CM1Z.png"
   }
}
```


Client Libraries and Platform Integration

Since you are reading this document it is assumed that you have an existing web site or service that you intend to integrate RunSignUp.com as a source for user authentication. Now that we have discussed the general steps in OAuth 1.0 authentication let's talk about specific client examples of integrations for your development environment. This process is complicated because there are usually multiple choices of OAuth client software available written in the language you are working with and this document cannot cover all possibilities. We have provided simple examples in PHP and Node that will hopefully be transferable to the client you choose for your integration.

PHP

RunSignUp provides a client example based on OAuth implementation provided as a binary library extension of PHP itself described at http://php.net/manual/en/book.oauth.php. If you do not have this PHP

extension installed on your system you can install it using pear/pecl. If you are not sure, perform the verification starting at step 6, below. If you do not have it installed follow the procedure below for getting the example to run on a Unix based system. These steps are provided as a guideline for you to do your own installation but Apache Http installs vary greatly and these steps are not meant as an exact set of instructions for your system.

1. If you do not already have the pear/pecl commands on your system, follow the installation procedure here. http://pear.php.net/manual/en/installation.php
2. It is always a good idea to update your pear database before doing an install. Use these commands:
   ```
   sudo pear channel-update pear.php.net
   sudo pear upgrade-all
   ```
3. Find your php.ini file by running php –i
   ```
   php -i | grep 'Configuration File'
   ```
4. Use pecl to build and install the oauth extension with the command:
   ```
   sudo pecl install oauth
   ```
   This will download an automatically compile the OAuth PHP extension and add a configuration line for it in your php.ini file. Note the path that the .so or .dll file gets installed to. You may also need to add extension_dir= value to your php.ini file if Apache cannot find oauth.so when it starts up.
5. Restart the Apache web server:
   ```
   sudo apachectl –k restart
   ```
6. Verify that the oauth extension is running. Look in your httpd.conf file to find your DocumentRoot directory. In this path create a PHP file named phpinfo.php which contains the following lines:
   ```
   <?php
       phpinfo();
   ?>
   ```
7. In your web browser, go to http://localhost/phpinfo.php. It should respond with a detailed configuration of PHP. Make sure you have an oauth section present that looks like the one shown below.

OAuth

OAuth support	enabled
PLAINTEXT support	enabled
RSA-SHA1 support	enabled
HMAC-SHA1 support	enabled
Request engine support	php_streams, curl
source version	$Id: oauth.c 325799 2012-05-24 21:07:51Z jawed $
version	1.2.3

8. Clone the PHP example:
   ```
   git clone https://github.com/RunSignUp-Team/OpenSource.git
   ```

9. Copy RunSignUpApiExamples/simpleOauthExample.php to your server's DocumentRoot.
10. Insert Your **oauth_consumer_key and oath_consumer_secret** into simpleOauthExample.php as shown below: (Around line 5)

```
// Fill in your key and secret
 define('OAUTH_CONSUMER_KEY', 'cd54e9c8741f147ff373af1e7260044054c905b1');
 define('OAUTH_CONSUMER_SECRET', '2b0357503155dcc1f79cee0147c9bd49d26f92b7');
```

11. Replace the callback_url found on line 53 as shown below:

```
// Set callback URL to this script (you probably should
have an absolute URL here) $cbUrl =
'http'.(isset($_SERVER['HTTPS'])?'s':'').'://'.$_SERVER[
'SERVER_NAME'].$_SERVER['REQUEST_URI'];
```

With your registered callback_url:

```
$cbUrl = "http://localhost/simpleOauthExample.php";
```

Note*: It is very important that this* ***callback_url*** *match the one you registered at https://runsignup.com/API. If they don't match, change the one at RunSignUp.com to match the current URL of this script on your server.*

12. You are now ready to run the example. In your browser go to http://localhost/simpleOauthExample.php. If everything is configured correctly you will be redirected to RunSignUp.com to log in and then redirected back to your site. You will then be redirected to this same page and the user's profile will be displayed as an XML document as shown below:

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE user SYSTEM
"https://runsignup.com/rest/rsu.dtd">
<user>
<user_id>3004896</user_id>
<first_name>William</first_name>
<last_name>Reichardt</last_name>
<email>william.reichardt@gmail.com</email>

<address>
<street>107 Ladds Lane</street>
<city>Westville</city>
```

```
<state>NJ</state>
<zipcode>08093</zipcode>
<country_code>US</country_code>
</address>

<gender>M</gender>
<phone>856-345-9473</phone>

<profile_image_url>//d368g9lw5ileu7.cloudfront.net/users/
user3004896_profile.bsCM1Z.png</profile_image_url>

</user>
```

From here you can integrate the steps in this example into your PHP site's authentication mechanism. When a customer returns to your site, simply re-issue a request for their profile. If this request fails, you must re-obtain their **oauth_token** and **oauth_signature** again since it may have expired or have been revoked.

Express/Node.js

RunSignUp authentication has been integrated into the Passport middleware package (http://passportjs.org/) for the Express web framework (http://expressjs.com/) for Node.js.

It has been integrated as a library into a forked version of easy-node-authentication (https://github.com/obiwan314/easy-node-authentication) which is a sample Passport application created by Scotch.io (https://scotch.io/collections/easy-node-authentication). This project can be checked out to create a fully functional example of RunSignUp authentication for Node.js. Here is the procedure to check it out and run it.

1. Install Node.js if you don't already have it from http://nodejs.org/ .
2. Check out the source.
   ```
   git clone https://github.com/obiwan314/easy-node-authentication.git
   ```
3. Change to the newly created directory.
   ```
   cd easy-node-authentication
   ```
4. Run npm to install the dependent libraries
   ```
   npm install
   ```
5. Edit the file ./config/auth.js. Enter your **oauth_consumer_key** and **oauth_consumer_secret** in place of the placeholders in the runsignup section of this file.

6. At https://runsignup.com/API change your callback URL to match the one already present in the file ./config/auth.js.
7. This example requires Mongo DB for Database Storage. Install this product at http://www.mongodb.org/. Alternatively, you could stand a database up at https://modulus.io/ as the Scotch.io tutorial suggests. Enter your database URL in your config/database.js file. This database will be used to store your tokens.
8. From the easy-node-authentication directory run the following command to start the application.
   > node server.js
9. In your web browser go to http://localhost:8080. You will see the login page shown below. Choose the RunSignUp button.