

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Wednesday, September 28, 2011

ZeroMQ + libevent in PHP

While waiting for a connection in Frankfurt I had a quick look at what it would take to make ZeroMQ and libevent co-exist in PHP and it was actually quite easy. Well, easy after Mikko Koppanen added a way to get the underlying socket fd from the ZeroMQ PHP extension.

To get this working, install the PHP ZeroMQ extension and the PHP libevent extension.

First, a little event-driven server that listens on loopback port 5555 and waits for 10 messages and then exits.

Server.php

Posted by Rasmus in PHP at 23:10

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Saturday, May 22. 2010

Writing an OAuth Provider Service

Last year I showed how to use pecl/oauth to write a Twitter OAuth Consumer. But what about writing the other end of that? What if you need to provide OAuth access to an API for your site? How do you do it?

Luckily John Jawed and Tjerk have put quite a bit of work into pecl/oauth lately and we now have full provider support in the extension. It's not documented yet at php.net/oauth, but there are some examples in svn. My particular project was to hook an OAuth provider service into a large existing Kohana-based codebase. After a couple of iterations this should now be trivial for others to do with the current pecl/oauth extension.

Step 1 - Create a Consumer Key registration page

In order for an application to communicate with your service you assign it what is essentially a user id and password. Except in OAuth terms these are known as the Consumer Key (CK) and shared secret. Your main job here is to make sure the CK and secret are unique and unguessable which means you need a decent entropy source. On Linux you have `/dev/random`, and the non-blocking potentially slightly weaker `/dev/urandom` mechanism. You can check how much entropy is available with:

```
cat /proc/sys/kernel/random/entropy_avail
```

In general something like the following should give you a decent random string of characters that you can use for your CK and secret:

```
function new_consumer_key() {
    $fp = fopen('/dev/urandom','rb');
    $entropy = fread($fp, 32);
    fclose($fp);
    // in case /dev/urandom is reusing entropy from its pool, let's add a bit more entropy
    $entropy .= uniqid(mt_rand(), true);
    $hash = sha1($entropy); // sha1 gives us a 40-byte hash
    // The first 30 bytes should be plenty for the consumer_key
    // We use the last 10 for the shared secret
    return array(substr($hash,0,30),substr($hash,30,10));
}
```

You can of course read more entropy and use a longer hash, like sha256 or whirlpool if you want longer keys.

Step 2 - The OAuth endpoints

It would probably be a good idea to skim the OAuth Spec at this point. We need two main end points for the OAuth sequence of requests. The first is called the "request token" end point. In my case I named it `/v1/oauth/request_token`. And since I am sliding this into a Kohana-based system, I wrote a controller (`application/controllers/v1.php`) which contained this in the constructor:

```
try {
    $this->provider = new OAuthProvider();
    $this->provider->consumerHandler(array($this,'lookupConsumer'));
    $this->provider->timestampNonceHandler(array($this,'timestampNonceChecker'));
    $this->provider->tokenHandler(array($this,'tokenHandler'));
    $this->provider->setParam('kohana_uri', NULL); // Ignore the kohana_uri parameter
    $this->provider->setRequestTokenPath('/v1/oauth/request_token'); // No token needed for this end point
    $this->provider->checkOAuthRequest();
}
```

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

```
} catch (OAuthException $E) {
echo OAuthProvider::reportProblem($E);
$this->oauth_error = true;
}
```

This makes the pecl/oauth extension do all the heavy lifting for us. We register a couple of callback functions. `lookupConsumer` will look up the CK and check if it is valid. `timestampNonceChecker` will check whether the timestamp of the request is sane and falls within the window of our Nonce checks. And this function will, of course, also check whether the provided Nonce has been used already to prevent replay attacks. And finally the `tokenHandler` callback will check whether a request or access token is valid. The `setParam('kohana_uri',NULL)` call tells the extension to ignore the `kohana_uri` parameter in the URLs because this was injected by an nginx rewrite rule and wasn't part of the original request.

The `lookupConsumer` callback happens early on. My version looks like this:

```
public function lookupConsumer($provider) {
    $consumer = ORM::Factory("consumer", $provider->consumer_key);
    if($provider->consumer_key != $consumer->consumer_key) {
        return OAUTH_CONSUMER_KEY_UNKNOWN;
    } else if($consumer->key_status != 0) { // 0 is active, 1 is throttled, 2 is blacklisted
        return OAUTH_CONSUMER_KEY_REFUSED;
    }
    $provider->consumer_secret = $consumer->secret;
    return OAUTH_OK;
}
```

Just a simple lookup in the data model and if found, the shared secret is attached so it can be used to check the signature of the request.

So, now we need to write code for the different stages of the OAuth request. I have an `oauth` method with a switch:

```
public function oauth($action=NULL) {
    if($this->oauth_error) return;

    switch($action) {
    case 'request_token':
        $token = Token_Model::create($this->provider->consumer_key);
        $token->save();
        // Build response with the authorization URL users should be sent to
        echo 'login_url=https://'.Kohana::config('config.site_domain').
            '/session/authorize&oauth_token='.$token->tok.
            '&oauth_token_secret='.$token->secret.
            '&oauth_callback_confirmed=true';
        break;
    }
```

Remember, the CK and the signature of the request has been checked already in the constructor, so by the time we get here, as long as `$this->oauth_error` is false, we can generate a request token and respond with a urlencoded set of parameters that include the unauthorized request token and secret we just generated along with a login url that the 3rd-party application will redirect the user to in order for them to authorize the request token to be used on their behalf.

Step 3 - Authorizing the request token

Back to your regular web UI, you now need to add a landing page for authorizing request tokens. Make sure the language on the page makes it clear to the user that they are authorizing a 3rd-party application to act on their behalf. It is a good idea to make them re-enter their password and explicitly click a button to do this authorization.

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Step 4 - The Access token

Once the user has authorized the request token, the 3rd-party app needs to exchange the authorized request token for an access token. My switch case for this looks something like this:

```
case 'access_token':
    $access_token = Token_Model::create($this->provider->consumer_key, 1);
    $access_token->save();
    $this->token->state = 2; // The request token is marked as 'used'
    $this->token->save();
    // Now we need to find the user who authorized this request token
    $utoken = ORM::factory('utoken', $this->token->tok);
    if(!$utoken->loaded) {
        echo "oauth error - token rejected";
        break;
    }
    // And swap out the authorized request token for the access token
    $new_utoken = Utoken_Model::create(
        array('token' => $access_token->tok,
            'user_id' => $utoken->user_id,
            'application_id' => $utoken->application_id,
            'access_type' => $utoken->access_type));
    $new_utoken->save();
    $utoken->delete();
    echo "oauth_token={$access_token->tok}&oauth_token_secret={$access_token->secret}";
    break;
```

As you can see I have a couple of data models that deal with tokens here. Token_Model is the token itself while UToken_Model links users to tokens. A token is either a request token (type 0) or an access token, (type 1) and they can be in various states. The tokenHandler callback keeps track of whether a token is used correctly. It looks like this:

```
public function tokenHandler($provider) {
    $this->token = ORM::Factory("token", $provider->token);
    if(!$this->token->loaded) {
        return OAUTH_TOKEN_REJECTED;
    } else if($this->token->type==1 && $this->token->state==1) {
        return OAUTH_TOKEN_REVOKED;
    } else if($this->token->type==0 && $this->token->state==2) {
        return OAUTH_TOKEN_USED;
    } else if($this->token->type==0 && $this->token->verifier != $provider->verifier) {
        return OAUTH_VERIFIER_INVALID;
    }
    $provider->token_secret = $this->token->secret;
    return OAUTH_OK;
}
```

Once we have handed out an access token, the 3rd-party app can store this and sign API requests with the associated shared secret and send that signature along with the consumer key. As you can see in the above token handler, there is a revocation check. At any point the user can go in through the Web UI and revoke an access token for a specific 3rd-party application.

Step 5 - An actual API call

Now that we have completed the OAuth dance, we can write API end points. Because all the oauth magic happens in our controller constructor, we are left to just implement our own api logic in our api methods. Mine look something like this:

```
public function user($cmd, $arg1=NULL, $arg2=NULL) {
    if(!$this->checkAccess()) return;
    $res = array();
    switch($cmd) {
    case 'settings':
        foreach($this->user->settings as $key=>$val ) {
            $res[$key] = $val;
        }
        $this->jsonResult($res);
        break;
    }
```

The checkAccess function uses the access token to look up the user who authorized it and sets \$this->user the same way your login flow will load the user once authenticated.

For the most part, the new OAuth Provider support in pecl/oauth takes the pain out of writing an OAuth Provider. It is smart about detecting the URL of the end points. You only have to indicate what the request_token (or 2-legged) endpoint is called, and it takes it from there. You can also pass the endpoint explicitly to checkOAuthRequest at each stage of the OAuth dance, but then you have to separate the steps more than I have here. I find this approach nice and compact and it lets me focus on the API part. I also like that it supports the OAuth ProblemReporting extension. This makes it a bit easier to track down problems. And by problems I mean signature mismatches. Everyone who has ever done anything with OAuth has struggled with them, and you will too. With the ProblemReporting extension you will get the raw string that the provider used to generate the signature. You just have to match that against what your consumer signed and you should be able to track down signature problems quickly.

Let me know if you do something interesting with the pecl/oauth extension. We may still be missing some use cases and right now there is decent momentum to fix things, so get in there and try it out.

Posted by Rasmus in PHP at 21:50

Sunday, January 10, 2010

SQLi Detection - Duh Moment

Not sure why it took me so long to figure out what I am sure is obvious to most other people who have thought about this, but it never clicked for me how to get anywhere near useful SQL Injection detection. The injection itself is trivial, of course, but determining whether it actually worked and weeding out false positives in an automated manner was something that seemed too hard.

During my run on Friday I had a Duh! moment on it. Annoyingly simple. Do it in 3 requests. Request #1 is a normal request. For example, "?id=1" in the URL. If the id is being passed to an SQL request it will return a single record or perhaps no record, it doesn't really matter. Now on request #2 do "?id=1 or 3=4", that is, inject a false 'OR' condition. If the output changes, we are done. Nothing to see here. However, if the output does not change we send request #3 with "?id=1 or 3=3" and if that output differs from request #2 then we have a potential SQLi situation. There are of course still chances of false positives (and negatives) with page stamps and such, but filtering out the response headers and html comments cuts down on that a bit. Add different combinations of single and double-quotes, like "?id=1'or'3='3" (without the double-quotes, of course) and it might be able to catch something.

The best thing about it is that it can slide into an existing scanner framework quite easily. If you have a base reference request, then it just adds a single request to the common case where the false 'OR' condition output does not match the base reference. You only need to do the true 'OR' condition request in case it does match.

Anybody have any other approaches?

Posted by Rasmus in Software at 18:44

Monday, April 27, 2009

Using pecl/oauth to post to Twitter

I have seen a lot of questions about OAuth and specifically how to do OAuth from PHP. We have a new pecl oauth extension written by John Jawed which does a really good job simplifying OAuth.

I added Twitter support to Slowgeek.com the other day and it was extremely painless. The goal was to let users have a way to have Slowgeek send a tweet on their behalf when they have completed a Nike+ run. Here is a simplified description of what I did.

First, I needed to get the user to authorize Slowgeek to tweet on their behalf. This is done by asking Twitter for an access token and secret which will be stored on Slowgeek. This access token and secret will allow us to act on behalf of the user. This is made a bit easier by the fact that Twitter does not expire access tokens at this point, so I didn't need to worry about an access token refresh workflow.

First, a bit of DB groundwork that has nothing to do with OAuth itself. I needed a place to store the access tokens and secrets and relate them to the existing user ids. My Slowgeek user table has a numeric u_id field for each user, so that is what I am using as my primary key here:

```
CREATE TABLE twitter (  
  u_id int(10) not null,  
  name char(32) default NULL,  
  state smallint default 0,  
  token varchar(64) default NULL,  
  secret varchar(64) default NULL,  
  description varchar(255) default NULL,  
  status varchar(140) default NULL,  
  location varchar(80) default NULL,  
  followers smallint default 0,  
  mtime timestamp default CURRENT_TIMESTAMP on update CURRENT_TIMESTAMP,  
  PRIMARY KEY (u_id)  
) TYPE=MyISAM;
```

And the related DB code using PDO.

So, I now have a mechanism for storing Twitter-related data. Now for the real oauth work. First I registered my application with Twitter to get a consumer key and consumer secret. You do that at http://twitter.com/oauth_clients/new.

Because I am updating the status I needed read/write access for this app. Now the PHP code. It will probably be easier to read if you copy and paste it out of the iframe into your favourite editor.

The last thing this script did after it had gotten the access token and secret was to call `verify_credentials.json` to get the user's credentials.

That gives us json back which we can decode with `json_decode()`. There is some interesting stuff in your Twitter record. Here is mine from a few minutes ago:

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

stdClass Object

```
(
  [favourites_count] => 0
  [profile_text_color] => 666666
  [description] => Breaking the Web
  [screen_name] => rasmus
  [utc_offset] => -28800
  [profile_background_image_url] => http://static.twitter.com/images/themes/theme9/bg.gif
  [profile_link_color] => 2FC2EF
  [following] =>
  [profile_sidebar_fill_color] => 252429
  [url] =>
  [name] => Rasmus Lerdorf
  [time_zone] => Pacific Time (US & Canada)
  [protected] =>
  [status] => stdClass Object
  (
    [truncated] =>
    [in_reply_to_status_id] => 1642930101
    [text] => @DonMacAskill Floating point values are approximations in all computer languages #php #broken
    [in_reply_to_user_id] => 813491
    [favorited] =>
    [in_reply_to_screen_name] => DonMacAskill
    [id] => 1643107564
    [source] => Nambu
    [created_at] => Tue Apr 28 21:57:56 +0000 2009
  )

  [profile_sidebar_border_color] => 181A1E
  [notifications] =>
  [profile_background_tile] =>
  [followers_count] => 2112
  [friends_count] => 71
  [profile_background_color] => 1A1B1F
  [profile_image_url] => http://s3.amazonaws.com/twitter_production/profile_images/52489510/rl_normal.jpg
  [location] => Sunnyvale, California
  [id] => 928961
  [statuses_count] => 577
  [created_at] => Sun Mar 11 15:39:19 +0000 2007
)
```

The `$debug = $oauth->getLastResponseInfo();` after an oauth call will always give you info about the last call. For this verify_credentials call the LastResponseInfo is (with the actual token and secret deleted):

```
[u_id] => 1823955881
[name] => rasmus
[state] => 2
[token] => XXX
[secret] => XXX
[description] => Breaking the Web
[status] => @cdamian I don't like things strapped to my chest. My brain is a perfectly good built-in heart rate monitor.
[location] => Sunnyvale, California
[followers] => 2080
[utype] =>
[mtime] => 2009-04-26 09:07:08
```

And now you can actually send a tweet. This assumes the filename is 'twitter.php'. This shows how to POST a tweet to

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

twitter. Two things to note here. Because Twitter requires a status update to be sent as a POST request, we have to use OAUTH_AUTH_TYPE_FORM when we instantiate the oauth object here. And second, I have a CSRF-preventing crumb in the POST data. The idea here is to tie the POST body to the current user's login cookie so the bad guys can't spoof a form post to our twitter.php script.

I am hoping the code is for the most part self-explaining. You can also have a look at the great pecl/oauth examples.

Posted by Rasmus in PHP at 15:20

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Thursday, March 19, 2009

Select * from World

I have been having a lot of fun with two Yahoo! technologies that have been evolving quickly. YQL and GeoPlanet. The first, YQL, puts an SQL-like interface on top of all the data on the Internet. And the second, GeoPlanet, introduces the concept of a WOEID (Where-On-Earth ID) that you can think of as a foreign key for your geo-related SQL expressions.

First some example YQL queries to get you used to this concept of treating the Internet like a database. Go to the YQL Console and paste these queries into the console to follow along.

```
select * from geo.places where text="SJC"
```

This looks up "SJC" in GeoPlanet and returns an XML result containing this information:

```
12521722
Airport
Norman Y Mineta San Jose International Airport
United States
California
Santa Clara
```

```
Downtown San Jose
```

```
95110
```

```
37.364079
-121.920662
```

```
37.35495
```

Posted by Rasmus in PHP at 15:06

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Thursday, May 8, 2008

SearchMonkey

One of the things I have been playing with lately is Yahoo!'s SearchMonkey project. It appeals to me on many different levels. The geeky name is a play on GreaseMonkey. But instead of writing plugins that run locally in the browser, SearchMonkey is a way to write plugins for the Yahoo! Search results page that change the appearance of the results themselves. Best explained with an example. Assume I am looking for a Japanese restaurant, and on my search results page I see:

That's ok, I guess. It tells me it is somewhere in Redwood City and that it is a neighborhood restaurant, whatever that means. Compare that to:

This gets me a real address and phone number plus a number of other useful bits of information. That is the first level SearchMonkey appeals to me on. The usefulness is obvious. My usefulness test is to see if I can explain it to my mother. Having her search for recipes and get pictures of dishes, ingredients and preparation times right on the search results page makes this an easy sell.

The second level this appeals to me on is the way it is implemented. Writing these SearchMonkey plugins becomes much simpler if the site you are writing the plugin for uses microformats of some sort. hCard, hCalendar, hReview, hAtom, xfn or generic structured eRDF or RDFa tags. The data can also be collected via a separate XML feed that can then be converted via XSLT in the SearchMonkey developer tool. The microformat data is collected and indexed and when you go to write a plugin and specify the url pattern you are writing the plugin for, it will find whatever indexed metadata it has for that url. If it doesn't have what you are looking for, you can still write a custom data scraper to get it, but that gets a bit more involved. I really like that the easy path is to add some sort of semantic markup to the pages. Yes, as Micah points out, this is not the (uppercase) Semantic Web, but it is still a push towards semantic markup. Having such a tangible and visible result of adding semantic tags is going to encourage people other than microformat geeks to do so. The more semantic markup we get, the better off the Web is.

The third part that appeals to me is the way the plugins are written. You write a little snippet of PHP. It is actually a method in a class you can't see, but its job is to return an associative array of data such as the title to display, the summary, extra links to show and whatever other key/value pairs you might want in the output. Because you have a full-featured scripting language available, you can write quite complicated logic in one of these plugins and pull whatever data you want from the site the plugin is written for.

You can also write an add-on to your plugin which is called an Infobar. It is a little bar that is shown below the plugin and from an Infobar you can access arbitrary external services. This example shows it well:

This one shows an OpenTable reservation link and a Yelp review, but almost anything can go there as long as you can squeeze it into the limited space you have.

The SearchMonkey is still in its infancy. It needs developer support. If you are in Silicon Valley, please come to the Developer Launch Party next week on Thursday May 15. See the link for details. If you aren't in the area, or even if you are, sign up for a developer account at <http://developer.yahoo.com/searchmonkey/preview.html> and help encourage the Web to become more semantic.

Posted by Rasmus in PHP at 14:50

Thursday, February 8, 2007

Pipes

<http://pipes.yahoo.com> is a cool toy, and by toy I mean it in the useful and cant-stop-playing-with-it sense. My first impression when I saw an early version a couple of months ago was, "How the heck did they do that?" I was reading the Javascript source code for quite a while. Once you get beyond the fact that this is a browser-based app doing this without Flash, or Java or any similar cheats, you get down to what the app actually does.

Years ago I wrote this silly little Mashup example:

<http://buzz.progphp.com/?q=4>

It grabs an RSS feed, in this case the top daily search term % movers from <http://buzz.yahoo.com/feeds/buzzoverm.xml> which gives you an indication of what is on the minds of web searchers right now. I took these searches and did a Yahoo Image search and a News search and combined them in that oval interface you see. I had to do a bit of RSS and XML parsing to take these different data sources and combine them. This is what Pipes is all about. It provides a visual environment for manipulating data sources and then provides a number of different ways to get the results and integrate them into other things. Directly in your RSS reader is probably the simplest, but you could also feed it to PHP and do further data manipulation.

A simplified Pipes version of the above takes the same Buzz.yahoo.com RSS feed and does a Flickr search on each search term. The result looks like this:

http://pipes.yahoo.com/pipes/DnudMIO32xGDclu7pRr_og

The point here is not the visual output. It is meant to be fed to something else. Hover over the "Subscribe" link on the right there. Then click on the "How this pipe was made" image on the left to see how it works.

This is a particularly lame and simple pipe. Some much cooler ones include:

Blog Buzz for Pipes combines a couple of different blog watching feeds, filters out duplicates and gives you a combined feed in reverse chronological order. When you look at how it was made it becomes immediately obvious what it does. You can save a copy and make your own version that watches for whatever terms you want.

Another interesting one takes the New York Times front page, runs a content analysis on it to get a set of representative keywords and then does a Flickr search on each of those. <http://pipes.yahoo.com/pipes/vvW1cD212xGMiR9aqu5IkA/>.

Here is a much more complex pipe that takes some user input. It finds apartments near things. In this case it looks for apartments within 2 miles of a Park in Palo Alto, California by searching Craigslist, then doing a location extraction and then doing a Yahoo! Local Search for that location.

Even if you have no use for processing data sources this way, open up one of these Pipes and drag the boxes around and watch the pipes react. Web apps don't get any cooler than this right now.

Posted by Rasmus in Software at 02:03

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Tuesday, January 30, 2007

Want a PHP job?

Want to work on some of the busiest and coolest web apps in the world?

Do you like Flickr, and want to work downtown San Francisco?

Or perhaps you are into music, movies or TV and want to work out of Santa Monica? Jumpcut? Or have you seen answers.yahoo.com? Address Book, Personals, Search, Premium Services, Hot Jobs? Want to do interesting things combining PHP and Flash?

Yes, I get a referral bonus, but I need more toys. You get a cool job though, so I think we are even.

Send me your resume and let me know what sort of stuff you are interested in or poke around on <http://careers.yahoo.com/> and let me know which job interests you and I will forward your resume to the appropriate hiring manager.

[edited to remove RSS ad test I had forgotten about]

Posted by Rasmus in PHP at 23:36

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Sunday, March 5, 2006

Get your geo plugin here

PHP 5, JSON from pecl cvs, Map Tile API and the Yui Connection Manager.

Toss them in a bag and shake and you get something like this. Enter a city name, or even a pseudo-name like Philly or SFO and hit return. Each time you enter a new name you will get a map tile for that city.

It's not all that fancy, but it sure is easy to do:

33 lines total which includes about 6 lines of PHP which consists mostly of building the URL to the map tile service. Then the Javascript which has a callback function to read the form values and make the backend GET request and a simple function to take the JSON response and add the map tile to the document. And finally the actual HTML form.

With a slight tweak you can change it to make a nice geocode lookup field.

So, fire up your text editors and start writing some plugins for blog and forum packages and perhaps image gallery applications as well.

Posted by Rasmus in PHP at 15:50

Monday, February 27, 2006

The no-framework PHP MVC framework

March 1, 2006 - Disclaimer: Since a lot of people seem to me misunderstanding this article. It isn't about OOP vs. Procedural programming styles. I happen to lean more towards procedural, but could easily have gone more OOP. I simplified the code a bit for brevity, but have added a light OO layer back in the model now. Not that it makes a difference. What I was hoping to get across here is a simple example of how you can use PHP as-is, without additional complex external layers, to apply an MVC approach with clean and simple views and still have all the goodness of fancy Web 2.0 features. If you think I am out to personally offend you and your favourite framework, then you have the wrong idea. I just happen find most of them too complex for my needs and this is a proposed alternative. If you have found a framework that works for you, great.

So you want to build the next fancy Web 2.0 site? You'll need some gear. Most likely in the form of a big complex MVC framework with plenty of layers that abstracts away your database, your HTML, your Javascript and in the end your application itself. If it is a really good framework it will provide a dozen things you'll never need.

I am obviously not a fan of such frameworks. I like stuff I can understand in an instant. Both because it lets me be productive right away and because 6 months from now when I come back to fix something, again I will only need an instant to figure out what is going on. So, here is my current approach to building rich web applications. The main pieces are:

PHP 5
Yahoo! User Interface Library
JSON

MVC?

I don't have much of a problem with MVC itself. It's the framework baggage that usually comes along with it that I avoid. Parts of frameworks can be useful as long as you can separate the parts out that you need. As for MVC, if you use it carefully, it can be useful in a web application. Just make sure you avoid the temptation of creating a single monolithic controller. A web application by its very nature is a series of small discrete requests. If you send all of your requests through a single controller on a single machine you have just defeated this very important architecture. Discreteness gives you scalability and modularity. You can break large problems up into a series of very small and modular solutions and you can deploy these across as many servers as you like. You need to tie them together to some extent most likely through some backend datastore, but keep them as separate as possible. This means you want your views and controllers very close to each other and you want to keep your controllers as small as possible.

Goals for this approach

Clean and simple design

HTML should look like HTML

Keep the PHP code in the views extremely simple: function calls, simple loops and variable substitutions should be all you need

Secure

Input validation using `pecl/filter` as a data firewall

When possible, avoid layers and other complexities to make code easier to audit

Fast

Avoid `include_once` and `require_once`

Use APC and `apc_store/apc_fetch` for caching data that rarely changes

Stay with procedural style unless something is truly an object

Avoid locks at all costs

Example Application

Here is the example application I will be describing.

It is a form entry page with a bit of Javascript magic along with an sqlite backend. Click around a bit. Try to add an entry, then modify it. You will see the server->client JSON traffic displayed at the bottom for debug purposes.

The Code

This is the code layout. It uses AJAX (with JSON instead of XML over the wire) for data validation. It also uses a couple of components from the Yahoo! user interface library and PHP's PDO mechanism in the model.

The presentation layer is above the line and the business logic below. In this simple example I have just one view, represented by the add.html file. It is actually called add.php on the live server, but I was too lazy to update the diagram and it really doesn't matter. The controller for that view is called add_c.inc. I tend to name files that the user loads directly as something.html or something.php and included files as something.inc. The rest of the files in the presentation layer are common files that all views in my application would share.

ui.inc has the common user interface components, common.js contains Javascript helper functions that mostly call into the presentation platform libraries, and styles.css provides the stylesheet.

A common db.inc file implements the model. I tend to use separate include files for each table in my database. In this case there is a just single table called "items", so I have a single items.inc file.

Input Filtering

You will notice a distinct lack of input filtering yet if you try to inject any sort of XSS it won't work. This is because I am using the pecl/filter extension to automatically sanitize all user data for me.

View - add.html

Let's start with the View in add.html:

The main thing to note here is that the majority of this file is very basic HTML. No styles, or javascript and no complicated PHP. It contains only simple presentation-level PHP logic. A modulus operation toggles the colours for the rows of items, and a loop around a heredoc (

Posted by Rasmus in PHP at 14:39

Sunday, February 26. 2006

Apache 1.3.34 Debian Package w/ mod_deflate and no pthreads

Debian's Apache1 package doesn't quite do what I need. I have been building my own and overwriting the files from the Debian package, but that can get annoying. So I hacked my changes into the Debian source package and built real .debs. I figure they might be useful to others.

The main changes are to get rid of -lpthread (needed by mod_perl) and -lexpat and to add mod_deflate. To enable mod_deflate make sure your /etc/apache/modules.conf file has:

```
AddModule mod_deflate.c
```

And in your httpd.conf add:

```
DeflateEnable      on
DeflateMinLength   1024
DeflateCompLevel   8
DeflateProxied     on
DeflateDisableRange "MSIE 4."
DeflateVary        on
DeflateTypes       text/css
DeflateTypes       text/plain
DeflateTypes       text/rtf
DeflateTypes       text/xml
DeflateTypes       text/javascript
DeflateTypes       image/vnd.dwg
DeflateTypes       image/vnd.dxf
DeflateTypes       application/msword
DeflateTypes       application/vnd.hp-HPGL
DeflateTypes       application/vnd.ms-access
DeflateTypes       application/vnd.ms-excel
DeflateTypes       application/vnd.ms-powerpoint
DeflateTypes       application/vnd.ms-project
DeflateTypes       application/vnd.visio
DeflateTypes       application/x-javascript
```

Posted by Rasmus in Software at 23:06

Sunday, November 6, 2005

More fun with the Yahoo! Maps API

I spent a few hours with the Javascript-Flash Yahoo! Maps API today. The result is here:

<http://lerdorf.com/map>

Have a look at the source. There is a source link at the top-right on the page. It is about 90 lines of HTML and Javascript and virtually no server-side scripting.

I usually throw PHP code at all sorts of problems, but in this case I could do pretty much everything I wanted to directly from Javascript via the excellent API. The initial zooming from way out is a bit distracting and doesn't work too well, but I wanted to play with that a bit. Once you are zoomed in and searching for things, the search is updated as you move around the map and everything is event-based so the page doesn't need to be redrawn from scratch.

Both input fields are free-form. You can put a city name, a zip code or a full address in the Location field and in the What field you put what you are looking for. There is a special hack that checks for something like 4* and the filters the results to only show you those entries that were rated 4* or higher on Yahoo! Local. You can of course also just put something like "pizza" or "mexican" in that field.

Most of the magic here is done by 2 things. The LocalSearchOverlay and the event handling. Note how Map.EVENT_MOVE and Map.EVENT_ZOOM_END are registered events in the onInitialize() function. When you scroll the map or zoom it the onOverlayInit function will get called and the LocalSearchOverlay will be recalculated for the new map coordinates. Same thing happens when you change something in the input fields. The updateMap() function is called which will center the map at the new location and update the LocalSearchOverlay appropriately. There are a few more Javascript tricks here and there in it, like updating the link at the top so you always have a way to grab a link to your current search and send it to someone, but other than that there really isn't all that much to figure out here. Once you understand which events happen when and which methods are available where, you can do some really powerful things with this. It is all documented here:

<http://developer.yahoo.net/maps/flash/V2/flashReference.html>

and people are discussing it here:

<http://groups.yahoo.com/group/yws-maps/messages>

In my last entry I showed how to parse a geocoded XML file and put markers for each entry on the map. Someone asked me how I would do the using the JS-DHTML API instead of the JS-Flash API. It's a whole lot harder in DHTML, but it works. You can see it here:

<http://lerdorf.com/php/ymap/dquakes.php>

And I talk a bit about it here:

<http://groups.yahoo.com/group/yws-maps/message/612>

Posted by Rasmus in Software at 17:09

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Thursday, November 3, 2005

GeoCool!

Web 2.0 and the programmable web that I and others have been talking about for a while has mostly been vapourware so far. There are a few generic components that are useful, but it is somewhat limited what you can do with them. And yes, you may consider this a somewhat biased view, but I think Yahoo!'s new geocoding platform is a huge step in the right direction.

There is of course the fancy new maps.yahoo.com/beta site which is fun, but as far as I am concerned the killer app here is the geocoding platform that drives this. And it is completely accessible for anyone to use. It's also a sane API that anybody can figure out in minutes. Here are a few tips for using this API from PHP 5.

Step 0 - The raw geocoding API

Whenever I do anything with web services, I always add a request caching layer. So here are the base building blocks implemented in 2 functions. One for doing request caching and the second to do the actual REST query to the geocoding service.

```
Result['precision']; foreach($xml->Result->children() as $key=>$val) { if(strlen($val)) $ret[(string)$key] = (string)$val; } return $ret;?>
```

The above code is the contents of `geo.inc` which you will see included in the following examples.

Easy enough? No real tricks here. We simply send a regular GET request to <http://api.local.yahoo.com/MapsService/V1/geocode> with the location parameter set to an address. You can try it yourself directly from your browser by clicking here:

<http://api.local.yahoo.com/MapsService/V1/geocode?appid=rlerdorf&location=701%20First%20Ave,%2094089>

You can read more about the geocoding service here:
<http://developer.yahoo.net/maps/rest/V1/geocode.html>

Step 1 - Writing your first application

We can just toss a form around this and dump the results to make sure things are working.

GeoCoding API Example

You can see this one in action here:

<http://lerdorf.com/php/ymap/geo1.php>

Note how it is able to fill in missing details for a partial address. eg.

<http://lerdorf.com/php/ymap/geo1.php?location=701+First+Avenue+94089>
results in:

```
[precision] => address  
[Latitude] => 37.416384  
[Longitude] => -122.024853  
[Address] => 701 FIRST AVE  
[City] => SUNNYVALE  
[State] => CA  
[Zip] => 94089-1019
```

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

[Country] => US

This means that you can use it for a bunch of different things. Address to lat/long, of course, but also address to city, or city to zip code conversions. Or 5-digit zip to 5+4. This is of course rather US-centric right now, but that will improve over time.

Step 2 - Adding a map

The geocoding is cool, but an actual map is cooler. Easy enough:

```
#mapContainer { height: 600px; width: 800px; } GeoCoding API Example mymap.addMarkerByLatLon( new CustomP  
OIMarker("  
", "", '0x0012f0', '0xFFFFF'), new LatLon());
```

You can also let the API figure out your markers for you which makes this even simpler. If the RSS feed is using georss correctly you can use the GeoRSSOverlay mechanism. Here it is using the earthquake RSS feed directly:

<http://lerdorf.com/php/ymap/rssquakes.php>

And here is the code. I am still loading the RSS feed myself from PHP because I want to get the pubDate and title from it, but everything else is handled automatically.

```
#mapContainer { height: 600px; width: 800px; }
```

Posted by Rasmus in Software at 15:32

Thursday, September 1, 2005

Flickr API Fun

I like stuff I can pick up and do something useful with in an hour or two. Perhaps my attention span is too short, but if I have to read a 300 page spec before I get to Hello World, then it's not for me. Or you would at least have to pay me a lot of money to suffer through it. I think people refer to this as "immediacy". For me I think it is mostly lazyness. If I can't figure it out in an hour, it's broken as far as I am concerned.

Flickr's REST API is not broken. You can read all about it at <http://flickr.com/services/api>.

There are links there to various wrappers for the API, but I ended up writing my own. I have a bad habit of doing that. This entry will focus on my PHP wrapper for the Flickr API. It is based on Cal's version and is compatible with it, but it expands on it and puts some PHP 5.1 features to good use. You can see it here:

http://lerdorf.com/php/flickr_api.phps

Before you get started, in case you want to follow along, go get yourself an API key at

<http://flickr.com/services/api/key.gne>

You will need two pieces of information to fully use the API. An API key and an API secret. And if you are going to do anything that requires authentication, you need to set a callback url as well. More on that later. To get your secret after applying for and getting your API key, go to

http://www.flickr.com/services/api/registered_keys.gne and click on "Edit Configuration".

Many functions in the API do not require authentication. Getting a list of someone's public photos, for example, is something anybody can do by just browsing Flickr, or by just going to this URL:

http://flickr.com/services/rest/?method=flickr.people.getPublicPhotos&user_id=56053642@N00&api_key=3aba8184848f9263b80795c95529bcd1

Guess what, you just sent a REST Web Services query.

Or, slightly cooler. A list of tags related to the tag you provide based on Flickr's clustering code.

http://flickr.com/services/rest/?method=flickr.tags.getRelated&tag=monkey&api_key=3aba8184848f9263b80795c95529bcd1

The whole point of web services is to provide data in a machine-readable way so you can do something more interesting with it. That's where the API wrapper comes in. You can of course also use Flickr's feed mechanism to do this.

But back to the API and the PHP wrapper. Getting a list of someone's public photos is done like this:

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

```
$secrets = array('api_key'=>'your_key_here','api_secret'=>'your_secret'); $flickr = new Flickr($secrets); $photos = $flickr->peopleGetPublicPhotos('56053642@N00');
```

This will give you an array of photos. Or to be precise, an array of information about the photos. Note the mapping of the method name. `$flickr->peopleGetPublicPhotos` maps to `flickr.people.getPublicPhotos` in the documentation. And the returned XML is converted to a more useful (and more memory-cacheable - I'll write something up soon on that) PHP array. The example result XML for 2 photos looks like this:

Which gets mapped to this PHP array (in `print_r` format):

```
Array (
  [page] => 1
  [pages] => 1
  [perpage] => 100
  [total] => 2
  [photos] => Array (
    [39006009] => Array (
      [id] => 39006009
      [owner] => 56053642@N00
      [secret] => f2086066d5
      [server] => 33
      [title] => IMG_7564.JPG
      [ispublic] => 1
      [isfriend] => 0
      [isfamily] => 0
    )

    [39006000] => Array (
      [id] => 39006000
      [owner] => 56053642@N00
      [secret] => 4ec57bd51f
      [server] => 28
      [title] => IMG_7551.JPG
      [ispublic] => 1
      [isfriend] => 0
      [isfamily] => 0
    )
  )
)
```

To turn a photo into a URL you can use in an IMG tag you would call the `$flickr->getPhotoURL()` method. It isn't very complex. Here is what it does:

```
function getPhotoURL($p, $size='s', $ext='jpg') { return "http://photos{$p['server']}.flickr.com/{$p['id']}{$p['secret']}{$size}.$ext"; }
```

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

The default size is the small 75x75 square thumbnail. See the URL Documentation for further info. So here is the full code to put up the first 50 thumbnails from someone's photostream:

The contents of secrets.inc is just that \$secrets array I referred to above. You can see the output of this script at flickr_demo1.php.

Flickr users are uniquely identified by a very cryptic-looking nsid. You don't see this id anywhere when you are clicking around on Flickr. But you can look up a user's nsid if you know their photo url, their user name or their email address. flickr_demo2.php shows you how to do that. Change the u= parameter in the URL to look up other users.

Playing with the non-authenticated functions of the API can get you far, but Flickr also lets you authenticate, and it will let the users using your application authenticate themselves. That lets you do a whole class of cool things that something like the RSS feed mechanism doesn't provide. For example, I wrote a Gallery to Flickr migration tool that can take my Gallery photo albums and copy the pictures to Flickr and put them in a Flickr set with the same name as the Gallery album they came from. You could also write alternative frontends for it and integrate your Flickr photos with your own web site. Or perhaps write a Gallery plugin that uses Flickr as the backend. All sorts of possibilities here.

But in order to do any sort of reading of non-public information or writing to your Flickr account via the API, you have to authenticate. Flickr uses a token-based authentication system where you make a roundtrip to flickr.com for the user to log into his flickr account and choose whether or not to grant your application the requested level of access. That means that your application never sees the user's credentials, but instead gets a token with the appropriate rights associated with it that it can then use. Each API call then includes this token, the application's key and all the arguments for whatever method you are calling and a signature using your application-specific secret across all the arguments. That means that even if someone sniffs your traffic, all they can do is replay the exact API call. They can't use it to execute arbitrary things against your account. Users can also remove an application's access later by going to <http://www.flickr.com/services/auth/list.gne>. The system is described at <http://www.flickr.com/services/api/auth.spec.html> and is worth a read if you are interested, but you don't really need to understand it. Just use the wrapper. Here is how:

This probably looks a bit cryptic. This says that If you already have a token, we simply create a \$flickr object and we are ready to go. If there is no token on the request and there is no 'frob', then redirect the user to the authentication URL which is generated by the call to \$flickr->getAuthUrl with the desired permission level as an argument. The user will then get sent back to your callback url, which you would set to this same script most likely, and on that callback we still don't have a token, but you will be called with a frob parameter. A call to \$flickr->getFrobToken turns the frob into a token. You actually get back an auth array containing not just the token but also the user's nsid, permission level for the token, username and fullname. The idea is then that you include the above

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

blurb on your pages, as flickr_auth.inc, for example and pass the token along from page to page in your web application.

Now we can write our first full little authenticated example. Not much to it. We just call `$flickr->authCheckToken` on our token to see what Flickr thinks of the token we are using.

You can see the output by clicking on flickr_demos.php and selecting flickr_demo3.

So, by having those 3 includes at the top, by the time we get control we have a fully authenticated `$flickr` object that we can start using.

So, an all-out demo. In flickr_demo4 we upload a photo:

```
$photo_id = $flickr->upload($fname,$title,$desc,$tags,$perms,0);
```

Check to see if you already have a set named "Sample Set". If you don't, create it (adding the uploaded photo at the same time):

```
$set = $flickr->photosetsCreate("Sample Set", $photo_id);
```

If you do already have that set, add the uploaded photo to it:

```
$flickr->photosetsAddPhoto($set_id, $photo_id);
```

Then we can add a note:

```
$note_id = $flickr->photosNotesAdd($photo_id,342,70,50,50,"This is Carl");
```

Get info on the photo:

```
$photo = $flickr->photosGetInfo($photo_id);
```

And get a direct URL to it:

```
$url = $flickr->getPhotoURL($photo,'m');
```

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Have a look at the full source code and try it by going to http://lerdorf.com/php/flickr_demos.php and clicking on demo4.

Even if you don't use Flickr, I think there are a lot of interesting things here. An interesting web service authentication mechanism, a nice and clean REST API that allows for complex operations, and some PHP 5.1 XML and stream handling if you look closely at the flickr_api code.

Posted by Rasmus in Software at 09:00

Wednesday, March 2, 2005

Buzzing the Yahoo! Search Web Services

March 22. Update: And here is the Flickr version: flickr.progphp.com

PHP5 has a revamped XML architecture that makes dealing with SOAP and REST Web Services extremely simple. I wrote a little demo application against Yahoo!'s new search web services. It uses the various search buzz RSS feeds to seed it or you can provide your own search terms. It then uses those terms to pull image, web and news search results which it arranges somewhat haphazardly. You can play with it at <http://buzz.progphp.com>. The orange box shows results from a news search, and when a term doesn't have enough news hits I supplement them with web search results which is shown in green to distinguish them.

Apart from a bunch of messy CSS, the application is actually quite simple. Pulling from the RSS and REST servers is trivial. Here is a one-liner to pull an RSS feed from a url:

```
$url = 'http://buzz.yahoo.com/feeds/buzzoverl.xml';  
$xml = simplexml_load_file($url);
```

The actual implementation wraps this and returns an associative array with just the title and the link, like this:

```
foreach($xml->channel->item as $item) {  
    $ret[(string)$item->title] = (string)$item->link;  
}  
return $ret;
```

For the search REST queries it isn't much harder. You build your query string:

```
$url = 'http://api.search.yahoo.com/';  
$url .= 'ImageSearchService/V1/imageSearch';  
$url .= '?query='.rawurlencode($q);  
$url .= "&appid=$appid";  
$url .= "&results=$results";  
$url .= "&type=$type";
```

I then throw a cacheing layer in front of all these so I don't hit the feeds on every request. The core of the cache layer looks like this:

```
$stream = fopen($url,'r');  
$tmpf = tempnam('/tmp','YWS');  
file_put_contents($tmpf, $stream);  
fclose($stream);  
rename($tmpf, $dest_file);
```

A straight fopen() can be used since this is a simple REST query and the result is streamed directly to a temp file which is then renamed when complete to make sure other processes never see a half-written file. Check the mtime on \$dest_file and read it until it gets too old, then refresh it.

Although I am not using any SOAP in this particular example, it isn't much harder to pull from a SOAP service. Here is a simple example that pulls from Amazon's SOAP service (they have a REST interface as well). It caches a serialized version of the generated object based on the service index and keywords requested.

```
$amazon_index = array(
```

```
'DVD', 'Photo', 'Electronics', 'OfficeProducts', 'HealthPersonalCare',
'Toys', 'Baby', 'VideoGames', 'MusicTracks', 'OutdoorLiving',
'Blended', 'MusicalInstruments', 'Magazines', 'DigitalMusic',
'Jewelry', 'Video', 'Tools', 'PCHardware', 'SportingGoods',
'Classical', 'Software', 'Books', 'VHS', 'Wireless', 'Restaurants',
'Music', 'GourmetFood', 'Miscellaneous', 'Kitchen', 'WirelessAccessories',
'Merchants', 'Beauty', 'Apparel'
);

function amazon($index, $keywords, $timeout=7200) {
    $dest_file = "/tmp/aws_{$index}_".md5($keywords);
    if(file_exists($dest_file) && filemtime($dest_file) > (time()-$timeout)) {
        $result = unserialize(file_get_contents($dest_file));
    } else {
        $aws = new SoapClient('http://webservices.amazon.com/'.
            'AWSECommerceService/US/AWSECommerceService.wsdl',
            array("trace" => 1));
        $result = $aws->ItemSearch(array(
            'SubscriptionId'=>'XXXXXXXXXXXXXXXXX',
            'AssociateTag'=>'lerdorf-20',
            'Request'=>array(array('SearchIndex'=>$index,
                'Keywords'=>$keywords))
        )
        );
        $tmpf = tempnam('/tmp','YWS');
        file_put_contents($tmpf, serialize($result));
        rename($tmpf, $dest_file);
    }
    return $result;
}
```

I still much prefer the REST services out there. SOAP always reminds me of being stuck behind the guy in a hat driving a Lincoln Towncar. You eventually get to where you want to go, but the journey is painful. With REST you can just toss your query into your browser and have a look at the returned XML. SOAP starts to make more sense when the queries you are sending get more complex than just tossing a couple of keywords to a search service and setting a couple of flags. But don't even try to read the SOAP spec. If you managed to fight your way through that spec already, try the new WSDL 2.0 Draft Spec. This is the sort of stuff that makes my brain hurt.

And yes, I know the thumbnails don't jump to the front in IE. IE's z-index handling on position: absolute elements is braindead. So use Firefox or Safari or some other browser with decent CSS support. Also, you'll need to let the cookie through. It's just a javascript cookie with your window dimensions so I'll know how big to make the oval. And no, it isn't really meant to be useful. Just a bit of fun visual candy.

Posted by Rasmus in Software at 22:38

Wednesday, November 24, 2004

Skype - Talk to your Laptop

Skype has been out for a couple of months now, but I only recently had a look. I have never been very impressed with the audio quality of these various voice-chat systems. However, Skype is a whole different story. With similar-looking clients for Windows, Linux, OSX and PocketPC, a mechanism called SkypeOut for making calls to regular phones and some nifty P2P principles applied to the problem of getting in and out from behind firewalls and NAT gateways all combining to create something that works extremely well for most people. The biggest problem people generally have with it is figuring out how to get a microphone installed and configured for their systems.

I installed it on a machine at home before heading to Paris and have been using it to talk to Christine from Paris. I also got my parents to install it by themselves and have had 3-way conference calls with me in Paris, my parents in Toronto and Christine in California. I have also called landlines in the US from Paris and Christine called my hotel landline in Paris using Skype. Overall the quality of all these calls were amazing. Every now and then there would be a slight drop or when 2 or 3 people all spoke at once it would occasionally garble things, but it is definitely the coolest piece of software I have used in a while. Although I may have to invest in a headset for it. You get some strange looks when you sit there talking to your laptop.

Posted by Rasmus in Software at 02:03

Saturday, September 11, 2004

Gallery and the Coral Distribution Network

The Coral Distribution Network (CDN) is a very nice shiny toy for all of us who sometimes struggle with limited bandwidth. Let the NSF pay for it!

My photo album is what chews up the most bandwidth from my site, so it made sense to rig it up first to optionally be available via CDN. CDN is basically just a big Akamai-like network of servers that cache things and serve them up to you from servers that are close to you network-wise. It has some fancy code behind it, but who cares how it works, it just does. Let's get to the interesting part. As you have probably figured out by now, you can access any site on the Web via Coral by simply appending ".nyud.net:8090" to the domain part of the URL and there are plugins available that add a menu item to automatically do this for you client-side. The problem with this is that you don't want to have to do this for every page on a site. It would be much nicer if the site would adjust its links such that if you enter the site via Coral, all the local links from that site would be Coral links. You may of course want to make some exceptions for pages that are interactive in some way, but for something like a photo album where 99% of people just look at mostly static content it works well.

You can see the patch at <http://gallery.menalto.com>

Note again that this quick hack does not in any way handle links that shouldn't be Coral'ed. Like the login link, comment stuff or all the administrative tools. It turns your Gallery into a read-only site if you come in using Coral. As the administrator of my album I can figure out that I need to go directly to it in order to add photos. But a more complete patch that doesn't Coralize stuff that shouldn't be would be nice.

You can have a look at it in action at <http://www.phpics.com.nyud.net:8090/>

I submitted it to the Gallery Folks so let's see if they take it and run with it to integrate it completely.

Update: A variation of this that also supports people running their Gallery on weird ports has been committed to Gallery's CVS, so you may just want to update from there to get it.

Posted by Rasmus in Software at 01:26

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Monday, January 12. 2004

Spamassassin

...is normally a very good toy. I use it extensively with Razor2 and Bayes hooked into it, it catches nearly everything. Today, however, I was suddenly inundated with Viagra spam. Odd, since any such spam usually sets off all sorts of SA alarms. And sure enough, the alarms were there:

```
X-Spam-Status: No, hits=0.9 required=5.0 tests=BAYES_99,BIZ_TLD,CLICK_BELOW,
HABEAS_SWE,HTML_50_60,HTML_LINK_CLICK_HERE,HTML_MESSAGE,
MIME_HTML_ONLY,MIME_HTML_ONLY_MULTI
```

So what gives? I have my BAYES_99 rule cranked way up to 5.4, so that alone should have put it over the 5.0 spam requirement, never mind all the other ones it triggered. The answer is of course that there is a large negative rule in there. In this case it was this HABEAS_SWE thing. The default SpamAssAss scores file has:

```
score HABEAS_SWE -8.0
score HABEAS_VIOLATOR 16.0
```

And it turns out that Habeas is some sort of "good spam" company that you can pay to get yourself whitelisted if you really need to spam people. I can see how that could be useful if you have a newsletter or something that people subscribe to and then it can't get through because of filters, but then these Habeas people damn well better be on the ball and triple-check the intentions of everyone and also run a tight ship security-wise. Given the fact that I received at least 20 Viagra spams before I killed that -8 rule, they obviously weren't quite on the ball and I don't particularly appreciate that this rule was in the default SA config to begin with. I haven't tracked down exactly who put that rule in and what sort of compensation changed hands. If someone knows, I'd like to know.

I would suggest that you find your local.cf file. Mine is in /etc/spamassassin/local.cf and add:

```
score HABEAS_SWE 0.0
```

Making it neutral.

Posted by Rasmus in Software at 03:47

Blog Export: Rasmus' Toys Page, <http://toys.lerdorf.com/>

Saturday, November 8, 2003

Cool PHP Apps

People are always asking me which PHP applications I prefer. It is obviously a very subjective thing, but here is a list of ones I have worked with and liked:

FUDForum is a very nice forum package writtern by Iliia.

Gallery is the ultimate photo album application. You can see it in action at phpics.com

Serendipity is a cool blog package. I really don't like blogs, but as you can see from this toy page which is using Serendipity it doesn't have to be just for your standard boring and useless blog. It can be used for a useless toy page too.

Posted by Rasmus in Software at 02:01