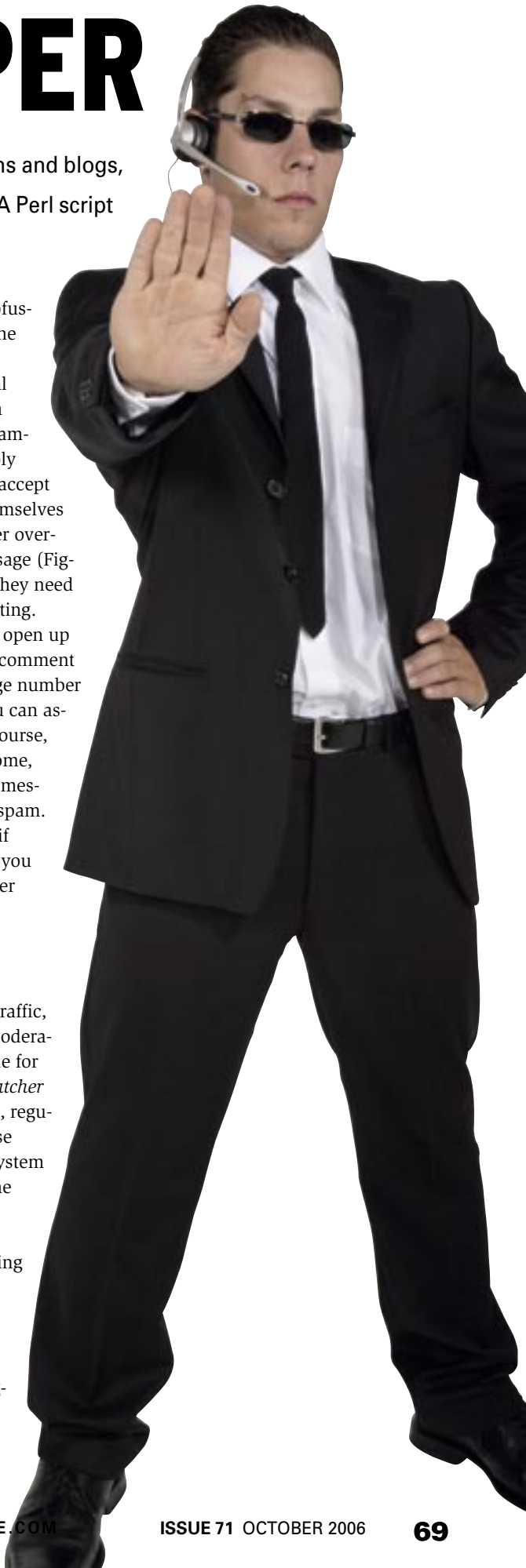
A Perl script protects forums against spam

SPAM STOPPER



Spammers don't just send email. They exploit discussion forums and blogs, posting pseudo-messages full of links to dupe search engines. A Perl script cleans up the mess. **BY MICHAEL SCHILLI**

My own small discussion forum on *perlmeister.com* has attracted too much attention from link spammers recently. These parasites of the Web target their bots at popular forum software tools such as phpBB or blog applications such as Wordpress, bombarding them with postings that contain very little aside from links to poker and sex pages. Spammers try to lure forum visitors into clicking on their sponsors' websites, and they are trying to dupe the major search engines that rate the importance of a page based on the links pointing to it.

Spanner in the Spambot's Works

What is known as comment spam [2] can be reduced by restricting postings to registered users. But this obstacle is also likely to frighten off legitimate users who have qualms about privacy issues. And moderating every posting before it appears on the website may keep the spammers out, but the effort required for these checks is immense – not to mention the unavoidable delays that hamstring any discussion.

Captchas ("Enter the number displayed by the wiggly font") help to ensure that there really is a human being at the other end of the connection and not a computer. Captchas don't need to be as complicated as on major registration sites, as Jeremy Zawodny's blog demonstrates. Users are simply asked to enter the word "Jeremy" in a separate field. Most bots would fail to do this, as they concentrate on mass infestation and can't adapt to customizations that will only work for a tiny percentage of the market. And some bots do not really understand the interaction between JavaScript and the browser's DOM. This means that even a trivial local change to the forum software, with an obfuscating JavaScript would keep the bots at bay.

Figures 2 and 3 show a trivial extension of phpBB that adds a switch to classify posters as spammers first. Spam bots will simply ignore the switch, or they will accept the default value, exposing themselves as spammers. If a human poster overlooks the switch, an error message (Figure 3) lets the user know that they need to flip the switch to enable posting.

Posted messages themselves open up another approach to detecting comment spam. If the message has a large number of links and not much text, you can assume that it will be spam. Of course, false positives can be troublesome, and you want to avoid tagging messages from legitimate users as spam. Users are likely to be annoyed if their postings are trashed, and you may end up losing users to other forums.

Mail for the Decision Maker

For forums without too much traffic, the email-based approach to moderation provides a useful technique for reducing spam. The *posting-watcher* script, which runs as a cronjob, regularly queries the Mysql database used by phpBB on the forum system to discover unread entries in the *phpbb_posts* table, and caches their IDs, along with a random key, on the local disk. After doing so, the script sends a message with the most important data from the posting (see Figure 4) to the moderator.

The moderator can simply ignore messages with legitimate postings. If the posting turns

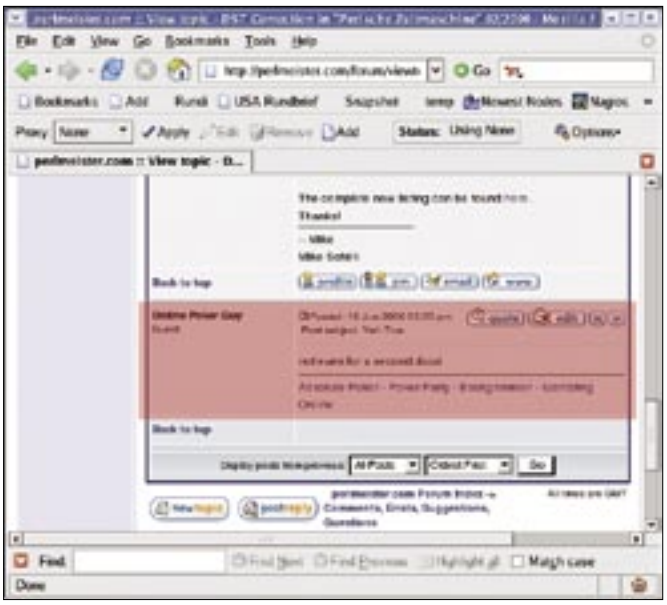


Figure 1: A spammer has set up camp in a forum.

out to be spam, the moderator can simply hit reply on their mail reader, which sends the mail back to the original script. The script then reads the random key stored in the mail, checks the file cache to locate the corresponding phpBB posting ID, and sends a scraper off to the website to remove the posting using the web administration interface provided by the phpBB forum software. I could have modified phpBB's database tables directly, but I didn't want to mess with its inner workings to avoid unwanted side effects.

After deleting the message, the script then sends a confirmation message back to the moderator. Figure 6 shows the whole procedure. Based on this approach, all postings (including spam) are first displayed by phpBB, but spam is quickly removed without too much effort. As communications with the moderator use email, a method that most people use extensively every day, the additional effort is negligible for forums with low traffic levels and, as a side effect, the moderator automatically keeps track of what goes on in the forum.

Preventing Misuse

Both the database monitor run regularly by the cronjob and the scraper serial mail killer are implemented by the same script, *posting-watcher*. When the script is called with the *-c* (check) parameter set, it searches the database for new postings and sends an email to the moderator for each new posting it finds.

When launched with the *-k* (kill) command line parameter set, the script expects an email with a kill key via STDIN.

You may be wondering why the email does not simply contain the posting ID. To make sure that *posting-watcher* understands that the spam killer message originated with the moderator (rather than with an abuser), it generates a random key, which cannot be trivially guessed, when sending the message and then stores the key in a local file (Figure 5), along with the ID of the posting to be investigated. If the moderator sends the message back, the script extracts the random key and uses the table to check if the key is assigned to a posting that is currently being moderated; it will only attempt to delete the posting if these conditions are fulfilled.

Say It with Flowers

The phpBB forum software distributes the forum data over 30 Mysql tables. Postings are stored in *phpbb_posts*, but not the body text of the message. Instead, phpBB uses the *post_id* column in the *phpbb_posts* table to establish a link between *phpbb_posts* and another table, *phpbb_posts_text*, and thus associates the posting with the full message text (Figure 8).

Figure 2: A new switch keeps bots at bay.

Figure 3: If a users forgets to flip the switch from "Spammer" to "User", an error message draws their attention to the mistake.

Queries that link two tables in this way are easy to handle with SQL. This said, a mixture of Perl and SQL is not exactly elegant, and this is why developers now tend to drop object-oriented Perl wrappers over relational databases.

This approach involves using object methods to query the database and manipulate the data stored in it. SQL is not required. You may remember me talking about *Class::DBI* in [5], but just recently, a new, and far more exciting framework was released: *Rose* is not only more flexible, but has a better turn of speed.

The framework, which is available from CPAN, uses the *Rose::DB* abstraction to talk to databases and *Rose::DB::Object* to access rows in database tables.

Relationships

As Listing 1 *PhpbbDB.pm* shows, there is no need to manually enter table column names, just to define the object-relational wrapper. *auto_initialize()* lets Rose autonomously search popular database systems (MySQL, Postgres, …), and automatically creates the methods required to build up an abstraction layer.

Normally, links between tables in relational databases are implemented by means of a foreign key in an extra column, but phpBB does its own thing here, using two primary keys, both of which are called *post_id* (Figure 8). This is a pity, because Rose has a convention manager that guesses table relations based on conventions (in a similar approach to Ruby on Rails). If everything is set up in a standard way, you can simply call *Rose::DB::Object::Loader* and let Rose automagically take care of everything else.

If Rose finds a table titled *phpbb_posts* (in the plural), it creates a class titled *PhpbbPost* (in the singular). Table names with underscores become *CamelCase* class names. Thus there is no need for *PhpbbDB.pm* to call *__PACKAGE_->meta->table('phpbb_topics');*, as Rose will automatically guess the table name of *phpbb_*

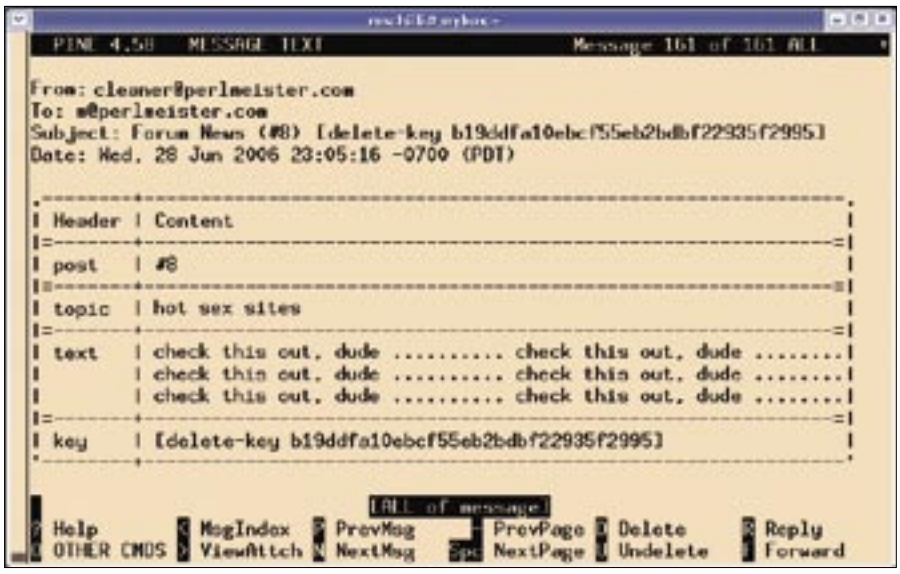


Figure 4: The forum moderator can check the message and decide whether to keep or discard the posting.

topics, based on the class name of *PhpbbTopic*.

If Rose detects a foreign key named *post_text*, it looks for another table titled *post_texts*. Rose then links the two classes *PhpbbPost* and *PostText* using a "many to one"-(N:1) relation. Note that table names are always plural (*post_texts*), whereas column names for foreign keys in 1:1 and N:1 relations are singular.

Getting Help

If an application's database schema does not follow this convention, you can either modify the convention manager, or step in to help Rose. Thus, Lines 50ff. of Listing 1 *PhpbbDB.pm* specifies the relationships between the *phpbb_posts* table, and the *phpbb_posts_text* and *phpbb_topics* tables, using *add_relationships* to do so. The *text* relationship (in Line 51) specifies a "one to one"-(1:1) relationship between the *phpbb_posts* table and the *phpbb_posts_text* table. Note that the table name is not in the plural here, since phpBB uses singular. Rose plays along just fine, because the definition of PhpbbPostsText explicitly defined the table name to be *phpbb_posts_text*.

"One to one" means that every row in *phpbb_posts*, is associated with a row in *phpbb_posts_text*, and vice-versa. There are also "*one to many*" (1:N), "*many to one*" (N:1), and "*many to many*" (N:N) relation types. The important thing is to call *add_relationships before* calling *auto_initialize()*; otherwise Rose won't create the necessary relationship methods in *auto_initialize()*.

This mainly automatically applied abstraction wrapper lets developers query the value of, say, the *post_id* column in a

Listing 1: PhpbbDB.pm

```
01 ##################################
02 package Phpbb::DB;
03 ##################################
04 use base qw(Rose::DB);
05 __PACKAGE__
06   ->use_private_registry();
07 __PACKAGE__->register_db(
08   driver   => 'mysql',
09   database => 'forum_db',
10   host     =>
11     'forum.db.host.com',
12   username => 'db_user',
13   password => 'XXXXXX',
14 );
15
16 ##################################
17 package Phpbb::DB::Object;
18 ##################################
19 use base
20   qw(Rose::DB::Object);
21
22 sub init_db {
23   Phpbb::DB->new();
24 }
25
26 ##################################
27 package PhpbbTopic;
28 ##################################
29 use base "Phpbb::DB::Object";
30 __PACKAGE__->meta
31   ->auto_initialize();
32
33 ##################################
34 package PhpbbPostsText;
35 ##################################
36 use base "Phpbb::DB::Object";
37 __PACKAGE__->meta->table(
38   'phpbb_posts_text');
39 __PACKAGE__->meta
40   ->auto_initialize();
41
42 ##################################
43 package PhpbbPost;
44 ##################################
45 use base "Phpbb::DB::Object";
46
47 __PACKAGE__->meta->table(
48   'phpbb_posts');
49 __PACKAGE__->meta
50   ->add_relationships(
51   text => {
52     type  => "one to one",
53     class =>
54       "PhpbbPostsText",
55     column_map => {
56       post_id => 'post_id'
57     },
58   },
59   topic => {
60     type  => "one to one",
61     class => "PhpbbTopic",
62     column_map => {
63       topic_id => 'topic_id'
64     },
65   }
66   );
67
68 __PACKAGE__->meta
69   ->auto_initialize();
70 __PACKAGE__->meta
71   ->make_manager_class(
72   'phpbb_posts');
73
74 1;
```

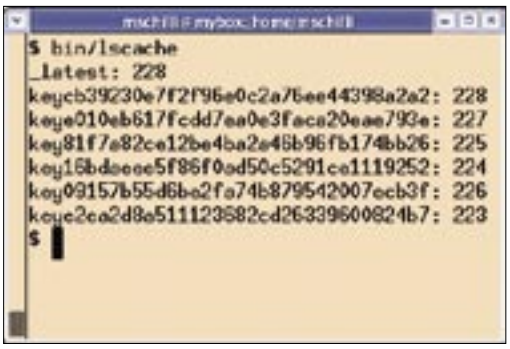


Figure 5: The content of the file cache which stores the random key and the corresponding post IDs.

PhpbbPost class object using the *post_id()* method, and, thanks to the relationship created previously, to call the *text()->post_text()* chain of methods to access the text string of the posting, which is located in the *phpbb_posts_text* table but linked to by *phpbb_posts*!

A comprehensive description of the object-oriented *Rose* database wrapper, and practical examples of its use, are available in the CPAN distribution of the module, and you'll find an excellent Rose tutorial at [4].

Key Generator

To support SELECT-style collective querying of an abstracted table, the *PhpbbPost* class is told in Line 71 of *PhpbbDB.pm* to create the *PhpbbPost::Manager* class, whose *query()* method performs the query.

Notice that the syntax for this query is an amazingly clean construct in pure Perl (see Line 99 and Line 100 in *posting-watcher*): *[post_id => { gt => $latest }]*.

The query corresponds to *SELECT ... WHERE post_id > $latest*. If the search is successful, the *get_phpbb_posts()* method returns a reference to an array that contains matching objects of the class *PhpbbPost*. Thanks to the *post_id()*, *text()->post_text()*, and *topic()->topic_title()* methods, and the relations defined for the tables, the objects return the posting ID, its title, and the text as strings. The *posting-watcher* script uses this data to fill an email to the moderator with content.

The persistent cache, which maps the post IDs to the random keys mentioned previously, is implemented by the CPAN *Cache::FileCache* module. As specified in Line 25, entries become obsolete after 14 days and are then treated as if accepted. The *purge()* method (Line 37) clears up obsolete entries.

Listing 2: posting-watcher

```
001 #!/usr/bin/perl -w
002 use strict;
003 use PhpbbDB;
004 use Cache::FileCache;
005 use Digest::MD5;
006 use Mail::Mailer;
007 use Mail::Internet;
008 use Text::ASCIITable;
009 use Text::Wrap qw(wrap);
010 use Getopt::Std;
011 use
012   WWW::Mechanize::Pluggable;
014 getopts( "kc", \my %opts );
016 my $FORUM_URL =
017   "http://foo.com/forum";
018 my $FORUM_USER =
019   "forum_user_id";
020 my $FORUM_PASS = "XXXXXXXX";
022 my $TO = 'moderator@foo.com';
023 my $REPL =
024   'forumcleaner@foo.com';
025 my $EXPIRE = 14 * 24 * 3600;
027 my $cache =
028   Cache::FileCache->new(
029   {
030     cache_root =>
031 "$ENV{HOME}/phpbb-cache",
032     namespace =>
033       "phpbb-watcher",
034   }
035   );
037 $cache->purge();
038 039 if ( $opts{k} ) {
040   my @data = <>;
041   my $body = join '', @data;
042   if ( $body =~
043     /\[delete-key (.*?)\]/ )
044   {
045     my $id = kill_by_key($1);
046     my $mail =
047       Mail::Internet->new(
048       \@data );
049     if ($mail) {
050       my $reply =
051         $mail->reply();
052       $reply->body(
053         [
054 "Deleted posting $id.\n\n",
055           @data
056         ]
057       );
058       $reply->send()
059         or die
060         "Reply mail failed";
061     }
062   }
063 }
064 elsif ( $opts{c} ) {
065   check();
066 }
067 else {
068   die "Use -c or -k";
069 }
071 ################################
072 sub kill_by_key {
073 ################################
074   my ($key) = @_;
075   my $id =
076     $cache->get("key$key");
077   if ( defined $id ) {
078     msg_remove($id);
079   }
080   else {
081     die "Invalid key $key";
082   }
084   return $id;
085 }
087 ################################
088 sub check {
089 ################################
090   my $latest =
091     $cache->get("_latest");
092   $latest = -1
093     unless defined $latest;
095   my $new_posts =
096     PhpbbPost::Manager
097     ->get_phpbb_posts(
098     query => [
099       post_id =>
100         { gt => $latest }
101     ]
102     );
104   foreach my $p (@$new_posts)
105   {
106     my $id = $p->post_id();
108     my $key = genkey();
110     mail(
111       $id,
112       format_post(
```

The ID of the last message to be moderated is stored by Line 129 under the *_latest* key in the cache, which maps the random keys to the post IDs for emails mentioned previously. The *check()* function in Listing 2 *posting-watcher* extracts the ID of the last posting currently under investigation from the cache and issues an SQL query that returns all postings with more recent IDs.

The 32 bytes hex format key string is created by the *genkey()* function in Line 138. It uses a method copied from the Apache::Session CPAN module, which runs the current time, a memory address, a random number, and the ID of the current process through an MD5 hash twice to produce a random parameter.

This combination creates an almost one hundred percent unique key, which is also very difficult to guess. Whoever knows this key can delete the posting that's mapped to the key in the cache from the forum. The *posting-watcher* script sends the key to the moderator and, if the moderator sends the key back to the script, the script launches the agent that removes the posting from the

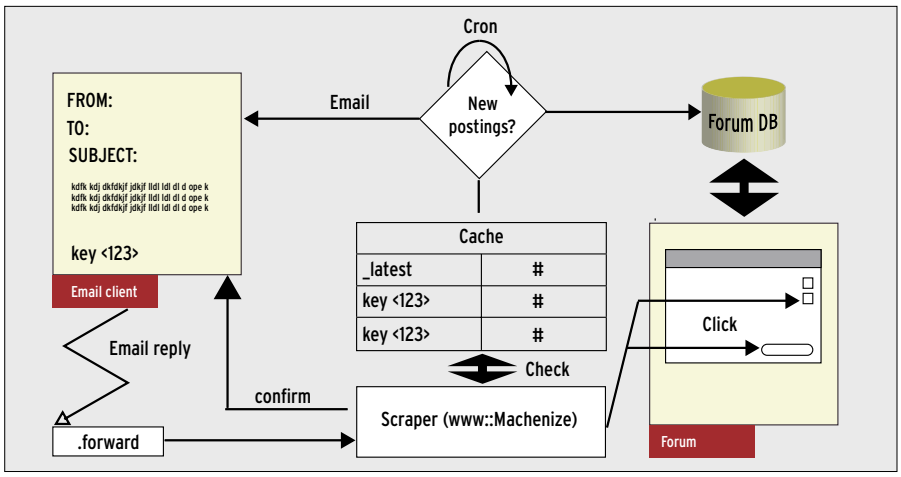


Figure 6: The content spam killer uses email to talk to the moderator.

Listing 2: posting-watcher

```
113         $id,
114         $p->text()
115           ->post_text(),
116         $p->topic()
117           ->topic_title(),
118         $key
119       ),
120       $key
121     );
123     $cache->set( "key$key",
124       $id, $EXPIRE );
126     $latest = $id;
127   }
128
129   $cache->set( "_latest",
130     $latest );
131 }
133 ##################################
134 sub genkey {
135 ##################################
136   return
137     Digest::MD5::md5_hex(
138     Digest::MD5::md5_hex(
139           time()
140         . {}
141         . rand()
142         . $$
143     )
144     );
145 }
147 ##################################
148 sub mail {
149 ##################################
150   my ( $id, $body, $key ) =
151     @_;
153   my $m =
154     Mail::Mailer->new(
155     'sendmail');
157   $m->open(
158     {
159       To      => $TO,
160       Subject =>
161 "Forum News (#$id)
    [delete-key $key]",
162       From => $REPL
163     }
164   );
166   print $m $body;
167 }
169 ##################################
170 sub format_post {
171 ##################################
172   my (
173     $id,    $text,
174     $topic, $key
175     )
176     = @_;
178   my $t =
179     Text::ASCIITable->new(
180     { drawRowLine => 1 } );
182   $t->setCols( 'Header',
183     'Content' );
184   $t->setColWidth( "Header",
185     6 );
187   $Text::Wrap::columns = 60;
189   $text =~
190     s/[^[:print:]]/./g;
192   $t->addRow( 'post',
193     "#$id" );
194   $t->addRow( 'topic',
195     $topic );
196   $t->addRow( 'text',
197     wrap( "", "", $text ) );
198   $t->addRow( 'key',
199     "[delete-key $key]" );
201   return $t->draw();
202 }
204 ##################################
205 sub msg_remove {
206 ##################################
207   my ($post_id) = @_;
209   my $mech =
210     WWW::Mechanize::Pluggable
211     ->new();
212   $mech->get($FORUM_URL);
214   $mech->phpbb_login(
215     $FORUM_USER,
216     $FORUM_PASS
217   );
218   $mech->get(
219 "$FORUM_URL/viewtopic.
    php?p=$post_id"
220   );
221   $mech->phpbb_post_remove(
222     $post_id);
223 }
```

forum by simulating mouse clicks in the forum's administrative GUI.

ASCII Art by Email

To make sure the messages about new postings received by the moderator are neatly formatted, the *format_post* function defined in Line 170 places the ID, title, and text in the ASCII box shown in Figure 4. The box is created without much ado by the *Text::ASCIITable* CPAN module. As the text contains tabs and other non-printing characters, the regular expression in Line 190 filters them, inserting dots instead. Before the text is inserted into the box, the *Text::Wrap* module sets the line width to 60 characters. Finally, the *format_post* inserts the secret key into the last line of the ASCII table. A *Subject:* line is added to the email, allowing the moderator to press the *Reply* button in the mail client to return the key as part of the subject line (but with "*Re: ...*" added) to the spam killer.

CPAN's Serial Killer

The *msg_remove* in Line 205 launches the *WWW::Mechanize* screen scraper using a plugin mechanism developed by my coworker at Yahoo, Joe McMahon, using the *WWW::Mechanize::Pluggable* CPAN module. The module supports constructing simple plugins, which can then be uploaded and shared on CPAN, to extend *WWW::Mechanize* with

Figure 7: The database tables used by the phpBB forum software.

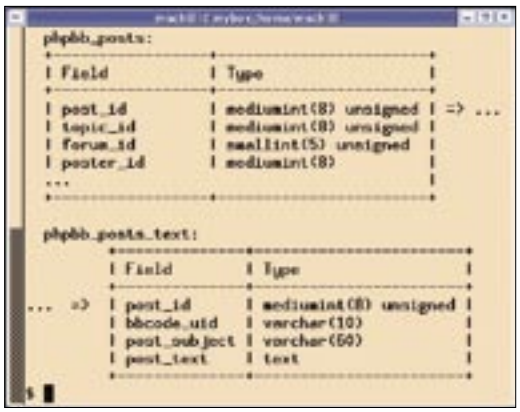


Figure 8: The unusual link between two tables in the phpBB forum software.

practical features. For example, *WWW::Mechanize::Pluggable::Phpbb* adds the *phpbb_login* and *phpbb_post_remove* methods to *WWW::Mechanize* to log the virtual browser user in as a Phpbb forum administrator, and press the right buttons to remove a posting identified by its ID. *msg_remove* then mails the moderator to confirm that the job has been done (Figure 9).

Installation

To make sure that emails sent to the spam killer's Unix account are forwarded to *posting-watcher*, we need a *.forward* file with the following content in the user's home directory: *| /full/path/to/posting-watcher-kill.sh*

The full command line is then available in the executable shell script, *posting-watcher-kill.sh*:

```
#!/bin/sh
/full/path/to/posting-watcher -k
```

Figure 9: Confirming deletion.

The reason for this is that the *.forward* file can't handle command line options, just plain program or script calls.

Make sure you enter your own values in Lines 16 through 25 of *posting-watcher* with valid email addresses, a forum URL, and the username and password of the forum admin.

The cronjob that checks the forum database for new entries every 15 minutes is called by entering *crontab -e*, and then adding: **/15 * * * * /full/path/to/posting-watcher -c*.

The important thing is to ensure that the script finds the *PhpbbDB.pm* module file. If *PhpbbDB.pm* is stored at a different location than the other Perl modules, you can point *posting-watcher* to the module directory by specifying *use lib '/path/to/moduledirectory';*

You can extend the script to automatically delete postings that match specific criteria (that contain more than a certain number of links, or certain keywords, for example) without asking the moderator for permission. Fight spam! ■

INFO

[1] Listings for this article: *http://www.linux-magazine.com/Magazine/Downloads/71/Perl*

[2] Benjamin Trott, "Comment Spam" *http://www.sixapart.com/about/news/2003/10/comment_spam.html*

[3] Forum software phpBB: *http://phpbb.com*

[4] Excellent tutorial on Rose: *http://search.cpan.org/dist/Rose-DB-Object/lib/Rose/DB/Object/Tutorial.pod*

[5] Michael Schilli, "Go Get It!", Destop Search in Perl, Linux Magazine 10/2005, *http://www.linux-magazine.com/issue/59/Perl_Desktop_Searches.pdf*

THE AUTHOR

Michael Schilli works as a Software Developer at Yahoo!, Sunnyvale, California. He wrote "Perl Power" for Addison-Wesley and can be contacted at *mschilli@perlmeister.com*. His homepage is at *http://perlmeister.com*.