LINUX JOURNAL™
Since 1994: The Original Magazine of the Linux Community

Develop Feature-Rich Apps with Command-Line Tools

OCTOBER 2017 | ISSUE 282
http://www.linuxjournal.com

PROGRAMMING

Scripting Project with Bash and PHP

+

How to Use Threading in Python

Introducing Subutai, a New Kind of Cloud

Ubuntu and Bash as a Windows Program

WATCH: ISSUE OVERVIEW

Cool Project: Create a CAPTCHA for Your Website

Jim's CAPTCHA uses photos of his cats, so only friends and family are allowed to view his personal website. You could modify the CAPTCHA to use other photos, like pictures of doors or trees, but I think everyone agrees that the internet needs more photos of cats. Note, this method works for dogs as well.

JIM HALL

PREVIOUS
New Products

NEXT
Feature: Developing Console Applications with Bash

I run a personal web server that I use for various projects and experiments. Over time, it's also become a convenient place for me to post other information to share with friends and family. For example, after much prompting by my family, I finally put up a "wish list" page for Christmas and birthday gift ideas.

Most of what I share on this personal website is intended for friends and family only, not for the general public. It's not sensitive information or private data, but I'd rather not put it out there for just anyone to see.

For a while, I considered protecting my web pages using a proper login system, but I didn't want to manage user names and passwords for all my friends and family who wanted to access my personal website. I might have implemented OpenID or some other decentralized authentication protocol to allow visitors to log in using another service. And although that would be the technically correct thing to do, I thought it was overkill to require a user name and password just for my friends and family to view my wish-list page.

So instead, I wrote a simple web system that asks visitors to demonstrate that they know me. This isn't a formal login system; rather, it's a form of CAPTCHA.

The concept of any CAPTCHA is to present some simple test for an *intended* audience to answer, but that's difficult for anyone else. Most CAPTCHAs try to present a trivial puzzle, such as clicking on any photos that contain water (like fountains or lakes). These CAPTCHAs often are used in online comment systems to verify that the person leaving the comment is a real person and not a bot.

In my case, I wanted to create a CAPTCHA that was simple for friends and family to answer, but difficult for others to figure out. And I decided to do it in a totally adorable way—by using photos of my cats.

My CAPTCHA

My CAPTCHA asks visitors to click on the photo of one our cats against a "forest" of other cats that don't belong to me. If you know me pretty well, this is easy. Obviously, my friends and family know what my cats look like. Selecting the photo of my cat is a simple demonstration that you know me; strangers would be unlikely to select the correct image.

Using photos of my cat isn't a perfect solution. It's entirely possible that attackers could use a brute-force method and keep clicking on photos of cats until they recognize which cats are mine. But this isn't a proper login system; it's just a CAPTCHA. I'm not protecting sensitive information like social security numbers. I merely want to prevent some random Joe User from seeing what books and t-shirts I'm asking for at Christmas.

In the simplest case, creating a cat CAPTCHA involves creating a random "forest" of cat photos, where only one of the cats belongs to you. You can create this "forest" in any number of ways. One easy way to do it in PHP is to create two lists: all images of my cats and all images of other people's cats. Pick one photo of my cat at random from the "my cat" list and several photos at random from the "not my cat" list, then shuffle the result.

So, you need to get started with photos of cats. This is the easiest step. Go through your photo collection and find pictures of your cats:

Figure 1. Jim's Cat

Figure 2.
Jim's Other Cat

contented cats lying in the sun, sleepy cats napping on cushions, playful cats prowling the backyard, confused cats wondering why you're always pointing a camera at them.

Then, crop and resize your cat photos to the appropriate dimensions. So you can use the photos more easily in the CAPTCHA, crop the photos to be square, with the cat taking up most of the photo. In GIMP, this is easily done if you set the Crop tool to use a fixed aspect ratio of "1:1". At this stage, don't worry about the size of the cat photos; I'll describe how to set that in a later step.

Next, you need photos of other cats. You can find those from a variety of online photos, but I went on Facebook and looked for any cat photos posted by my friends. I found about 50 other cat photos this way. Just make sure the other cats don't look too much like your cats, and same as the previous step, crop the other cat photos so they are square.

To use these cat photos in a CAPTCHA, you want to ensure that the

visitor cannot guess which cat is yours based on the photo's filenames. Photos named something like mycat_1.jpg can provide clear clues that the cat might belong to you. Instead, name all your photos in the same generic way. I prefer to rename my cat CAPTCHA photos according to the MD5 signature of the photo. This provides suitable non-predictability; you can't guess which photos are my cats and which are not simply based on the photo filename.

To make your pictures more alike, strip your cat photos of any EXIF data. While humans won't see the EXIF data, a bot might. EXIF data could provide a telltale hint that a photo is or isn't your cat. So make them all the same and omit EXIF data.

Note: if your CAPTCHA photos don't change over time, attackers could use a brute-force method to try all of the photos until they guess the MD5 signatures of the correct images. To protect against this, I add some random noise to the photos. I'm sure you can do this in GIMP, but I found it was easiest to use ImageMagick's Convert tool to add noise while I resized the cat photos:

```
convert large_photo.jpg -strip -resize 250x250 +noise
 ➥Laplacian small_photo.jpg
```

ImageMagick has several random noise generators, but I find the Laplacian method doesn't disturb the image too much.

You easily can automate ImageMagick with a script, and that's what I do. A simple Bash script processes all of my image files every day to resize them and add noise while renaming them to something random. This also merges the photos into a single directory and creates the lists of "my cats" and "not my cats". Put the list files in a separate directory, preferably outside the document root. On my personal web server, my document root is under /var/www/html, but I keep my list files in /var/www/etc/cats.

This script runs on my personal web server every day:

```
#!/bin/sh
etcdir=/var/www/etc/cats/
cachedir=/var/www/html/captcha/cache/
```

```
catsdir=$HOME/cats
tempimg=/tmp/cat.jpg


for d in mycats notmycats ; do
 ( cd $catsdir/$d
 >$etcdir/$d.list
 for img in *.jpg ; do
  convert $img -strip -resize 250x250 +noise Laplacian $tempimg
  hashval=$( md5sum $tempimg | awk '{print $1}' )
  mv $tempimg $cachedir/$hashval.jpg
  echo $hashval.jpg >> $etcdir/$d.list
 done )
done
```

This script adds all cat photos saved in $HOME/cats/mycats to the "my cats" list and all cat photos saved in $HOME/cats/notmycats to the "not my cats" list. ImageMagick removes all `EXIF` data (`-strip`), resizes all photos to the same size (`-resize 250x250`) and adds some random noise (`+noise Laplacian`). At the same time, all cat photos are mixed together with random filenames in the /var/www/html/captcha/cache directory.

Writing the PHP Code

Once you have the cat images, you need to write some PHP code that displays the cat images as a CAPTCHA. For my CAPTCHA, I prefer to display nine cat images in a 3x3 grid. One photo is my cat, and the other eight cats are not.

The first step is for the PHP script to know which cats are mine and which cats are not. That's where the mycats.list and notmycats.list files come in.

The `file()` function in PHP reads an entire file into an array—for example:

```
<?php
    $myCats = file('/var/www/etc/cats/mycats.list');
```

```
    $notMyCats = file('/var/www/etc/cats/notmycats.list');
?>
```

The `file()` function will include the newline characters at the end of each line, plus any empty lines that might be in the file. So you need to provide a few options to read the list files without the extra stuff:

```
<?php
    $myCats = file('/var/www/etc/cats/mycats.list',
     ➥FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
    $notMyCats = file('/var/www/etc/cats/notmycats.list',
     ➥FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
?>
```

PHP has a handy function `array_rand()` that picks random entries out of an array. This function returns the indices of random entries, which are easily used as array references. In the simplest case, you can select one cat randomly from the $myCats array like this:

```
<?php
    $myCats[ array_rand($myCats) ]
?>
```

Then build a new array of eight cats selected at random from the `$notMyCats` array and one random cat from the `$myCats` array by stacking the elements in the `array()` function. This builds a new array `$randCats` from the elements provided. As you can see, the new `$randCats` array is in a predictable order, with the correct cat always in the last position. So after building the array, randomize the order of the array elements with the `shuffle()` function:

```
<?php
    $notMyCatsIdx = array_rand($notMyCats, 8);
    $randCats = array(
            $notMyCats[ $notMyCatsIdx[0] ],
            $notMyCats[ $notMyCatsIdx[1] ],
```

```
            $notMyCats[ $notMyCatsIdx[2] ],
            $notMyCats[ $notMyCatsIdx[3] ],
            $notMyCats[ $notMyCatsIdx[4] ],
            $notMyCats[ $notMyCatsIdx[5] ],
            $notMyCats[ $notMyCatsIdx[6] ],
            $notMyCats[ $notMyCatsIdx[7] ],
            $myCats[ array_rand($myCats) ]
    );
    shuffle($randCats);
?>
```

Once you have randomly selected eight incorrect cats and one correct cat in the $randCats array, you can display the nine cat photos on a CAPTCHA web page. You'll use HTML styles later to ensure the photos are displayed as a 3x3 grid. In my CAPTCHA implementation, each cat photo is a separate Submit button in a web form. On the captcha/index.php CAPTCHA page, iterate through the `$randCats` array using the `foreach()` method:

```
<form method="POST">
<?php
    foreach ( $randCats as $cat ) {
            echo<<<EOF
<button type="submit" name="cat" value="$cat">
<img src="$cat" alt="image of a cat" />
</button>
EOF;
    }
?>
<input type="hidden" name="captcha" value="catCAPTCHA" />
</form>
```

The hidden input "captcha" with the value "catCAPTCHA" will be used later to detect that the user submitted the web form and not simply visited the web page.

Validating the CAPTCHA is fairly straightforward: read the POST data

to the web form and search for the cat CAPTCHA value in the `$myCats` array. PHP's `in_array()` function checks whether a value is in an array, so you can use this function to simplify the lookup. At the top of the captcha/index.php page, add code to detect the posting of the cat CAPTCHA, then search for the cat CAPTCHA value:

```
<?php
 /* test if the visitor guessed the correct cat */


 if (strcmp( $_POST['captcha'], 'catCAPTCHA' ) == 0) {
  $cat = $_POST['cat'];


  if (in_array($cat, $myCats) === TRUE) {
   /* success */
  }
  else {
   /* failure */
  }
 }
?>
```

The Completed Captcha

With these pieces, it's pretty simple to complete the CAPTCHA page.

I've parameterized the cookie information via the cookie-conf.php file. The parameter file retrieves the cookie value from a separate /var/www/etc/cats/catCAPTCHA.cookie file. Also, cookie-conf.php defines a standard cookie name, path and domain, and sets the cookie expiry to one day.

This CAPTCHA page is just an example. You can improve the system by checking the value of the `redir=` URL parameter before redirecting the user there to prevent cross-site attacks. And, you might add further checks to detect multiple CAPTCHA attempts from the same user within too short a time frame.

Listing 1 shows the completed CAPTCHA page.

This CAPTCHA page defines a few styles, both to make the page content

Listing 1. Completed CAPTCHA Page

```
<?php
 require ('/var/www/etc/cats/cookie-conf.php');


 /* initialize values */


 $myCats = file('/var/www/etc/cats/mycats.list',
  ➥FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);
 $notMyCats = file('/var/www/etc/cats/notmycats.list',
  ➥FILE_IGNORE_NEW_LINES | FILE_SKIP_EMPTY_LINES);


 $CAPTCHAsuccess = FALSE;
 $CAPTCHAmessage = '<p>Hi there!</p>';


 /* test if the visitor guessed the correct cat */


 if (strcmp( $_POST['captcha'], 'catCAPTCHA' ) == 0) {
  $cat = $_POST['cat'];


  if (in_array($cat, $myCats) === TRUE) {
   $CAPTCHAsuccess = TRUE;
   $CAPTCHAmessage = '<p class="success">Success!</p>';
   setcookie($cookieName, $cookieValue, $cookieExpire,
    ➥$cookiePath, $cookieDomain);


   $redir = $_GET['redir'];
  }
  else {
   $CAPTCHAsuccess = FALSE;
   $CAPTCHAmessage = '<p class="error">No, that is not
    ➥our cat. Try again?</p>';
  }
 }
?>
<!DOCTYPE html>
<html>
<head>
 <title>Which one is our cat?</title>
```

```
 <link rel="shortcut icon" href="/favicon.png" />
 <meta charset="UTF-8">
 <meta name="theme-color" content="firebrick">
 <meta name="viewport" content="width=device-width" />
<?php
 /* if CAPTCHA was correct, and there is a redirect URL,
    then redirect */


 if ( ($CAPTCHAsuccess) && (strlen($redir) > 0) ) {
  echo "<meta http-equiv='refresh' content='0;URL=$redir' />";
 }
?>
 <style>
body {
 background-color: white;
 color: black;
 font-family: sans-serif;
 margin: 0;
}
header {
 background-color: firebrick;
 border-bottom: .5em solid darkred;
 color: white;
}
 header h1 {
 font-size: 1em;
 margin: 0;
 padding: 2em 0;
 text-align: center;
 text-transform: uppercase;
 }
main {
 border-bottom: 1em solid gray;
}
main section {
 border-left: 1px dotted lightgray;
 border-right: 1px dotted lightgray;
 margin: 0 auto;
 padding: 2em 1em;
 max-width: 900px;
}
 main h2 {
 border-bottom: 1px solid darkred;
 color: darkred;
 font-family: serif;
```

```
 font-size: 1.1em;
 margin: 2em 0;
 }
 p.success {
 color: green;
 }
 p.error {
 color: red;
 }
 form {
 margin: 1em auto;
 max-width: 800px;
 }
 form button {
 border: 1px solid #333;
 margin: 2px;
 padding: 0;
 -moz-appearance: none;
 -webkit-appearance: none;
 }
 </style>
</head>
<body>
<header>
 <h1>Login</h1>
</header>
<main>
 <section>
<?php
 echo $CAPTCHAmessage;


 if ( ! $CAPTCHAsuccess ) {
  echo<<<EOF
<p>The stuff on this website isn't private information, but
I'd rather not put it out there for just anyone to see. I
prefer
to keep this just for friends and family.</p>


<p>So to prove that you know me, I'm giving you
this little quiz:</p>


<h2>Which one is our cat?</h2>
```

```
<p>We have several cats, but only one of them is shown here.
Click on the picture of our cat.</p>


<form method="POST">
EOF;


  $notMyCatsIdx = array_rand($notMyCats, 8);
  $randCats = array(
   $notMyCats[ $notMyCatsIdx[0] ],
   $notMyCats[ $notMyCatsIdx[1] ],
   $notMyCats[ $notMyCatsIdx[2] ],
   $notMyCats[ $notMyCatsIdx[3] ],
   $notMyCats[ $notMyCatsIdx[4] ],
   $notMyCats[ $notMyCatsIdx[5] ],
   $notMyCats[ $notMyCatsIdx[6] ],
   $notMyCats[ $notMyCatsIdx[7] ],
   $myCats[ array_rand($myCats) ]
  );
  shuffle($randCats);


  foreach ( $randCats as $cat ) {
   echo<<<EOF
<button type="submit" name="cat" value="$cat">
<img src="cache/$cat" alt="image of a cat" />
</button>
EOF;
  }


  echo<<<EOF
<input type="hidden" name="captcha" value="catCAPTCHA" />
</form>
EOF;
 }
?>
 </section>
</main>
</body>
</html>
```

look nice and to ensure that the CAPTCHA images are displayed in a 3x3 grid. Since each cat photo is 200x200 pixels, setting the page width to 700 pixels forces the CAPTCHA images to wrap to the next line after three

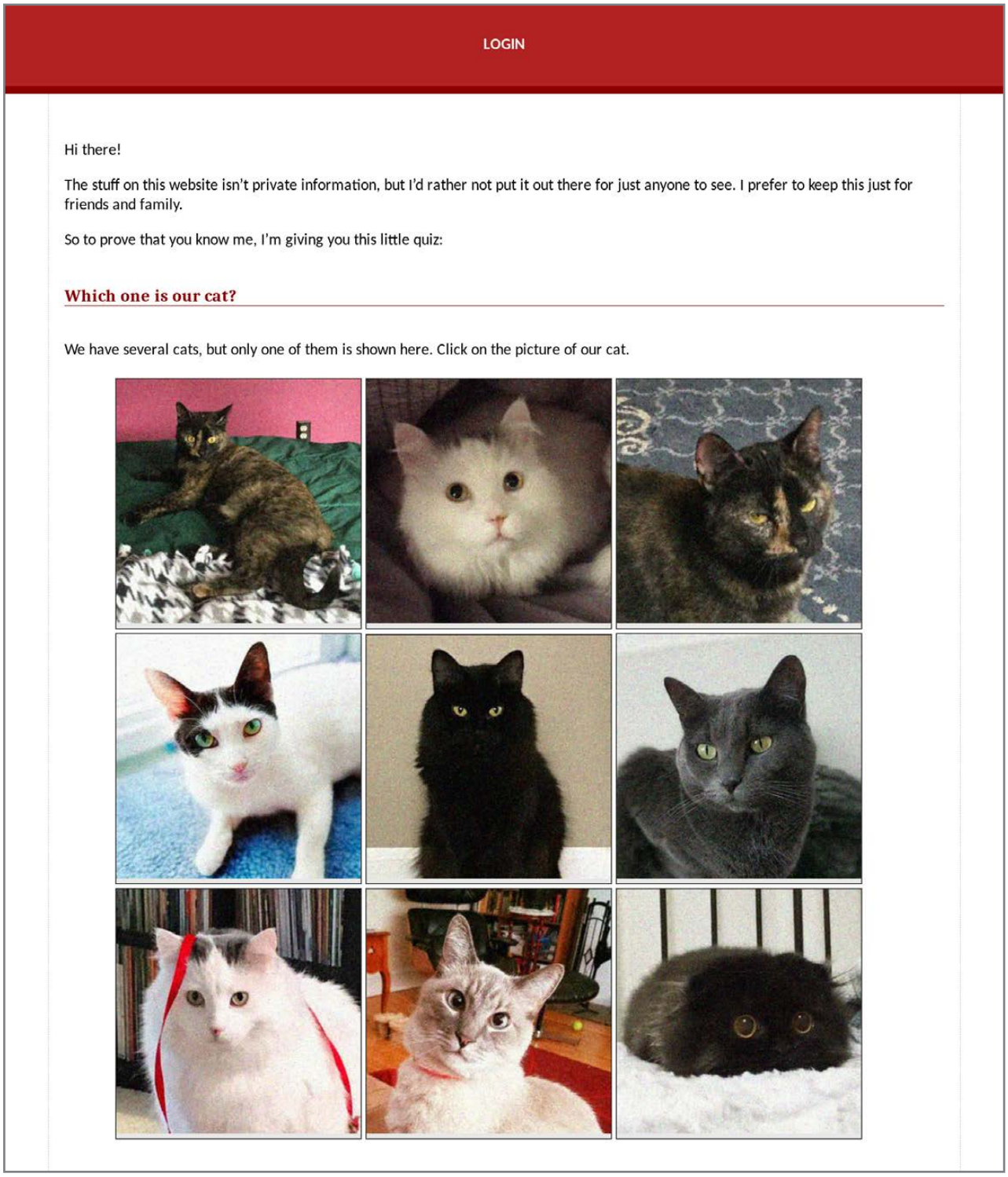

Figure 3. Cats CAPTCHA

photos. This effectively presents them as a 3x3 grid.

The `-moz-appearance: none;` and `-webkit-appearance: none;` styles for the form buttons erase any extra decoration from Mozilla and Safari web browsers. Without those styles, an iPad will add rounded edges to the buttons, which would ruin the effect of the CAPTCHA.

The other styles provide a pleasant viewing experience. For example, the styles apply a red banner at the top of the page as well as different styling for text and headings.

Rotating the CAPTCHA Cookie

One final improvement remains. Remember, when the visitor correctly selects the right cat, the CAPTCHA page sets a cookie with a value read from a file. But, why read the cookie value from a file rather than set a static value?

The answer is security. If you keep the cookie value in a separate file, you can change the contents of cookie file whenever you want to "invalidate" the old cookie value. On my personal web server, I set up a daily job that writes a random value to the cookie value. Even I couldn't tell you what the new cookie value will be.

Any web cookie is really just a long string of random letters and numbers. So on my web server, I want to set a different long string every day. I do this in two steps:

1. Create a random file using `dd` and /dev/urand.

2. Generate a hash of that random file using `sha256sum`.

The value of the SHA256 checksum on the random file becomes the new CAPTCHA cookie value. That's 64 letters and numbers to "describe" a random file, which makes a pretty good CAPTCHA cookie value. I run this simple Bash script via cron to overwrite the cookie file with a random value every day:

```
#!/bin/sh
cookiefile=/var/www/etc/cats/catCAPTCHA.cookie
dd if=/dev/urandom bs=1M count=4 of=/tmp/urandom.tmp 2> /dev/null
sha256sum /tmp/urandom.tmp | awk '{print $1}' > $cookiefile
```

Testing for the CAPTCHA

Once you've put the CAPTCHA page in place, your other web pages need to recognize when a visitor has "passed" the CAPTCHA test. You easily can test for the CAPTCHA by examining cookies.

Referring back to the completed CAPTCHA page, when the visitor correctly selects the right cat, the CAPTCHA page sets a cookie with a specific value. So if the cookie exists and contains that value, any PHP page can determine if the visitor "passed" the CAPTCHA. I define a function to do this for me in the cookie-conf.php page:

```
<?php
 $cookieName = 'catCAPTCHA';
 $cookieValue = file_get_contents('/var/www/etc/cats/catCAPTCHA.cookie');
 $cookieExpire = time() + 86400; /* 86400 = (3600 * 24) = 1 day */
 $cookiePath = '/';
 $cookieDomain = 'freedos.org';


 function is_catCAPTCHA_ok()
 {
 /* shortcut to determine if a login is successful ..
    uses the cookie value */


  global $cookieName, $cookieValue;


  if ( (isset($_COOKIE)) &&
       (isset($_COOKIE[$cookieName])) &&
       (strcmp($_COOKIE[$cookieName], $cookieValue) == 0) ) {
   return TRUE;
  }
  else {
   return FALSE;
  }
 }
?>
```

In any PHP page that I want to protect with the cat CAPTCHA, I simply include the cookie-conf.php file and call `is_catCAPTCHA_ok()` to test the CAPTCHA. In a more complete implementation, you might instead redirect non-validated visitors to the CAPTCHA page, setting the `redir=` parameter so that users automatically return to the correct page. Here's a simple example to test the CAPTCHA:

```
<?php
 require ('/var/www/etc/cats/cookie-conf.php');
?>
<!DOCTYPE html>
<html>
<head>
 <title>Test page</title>
 <meta charset="UTF-8">
 <meta name="viewport" content="width=device-width" />
</head>
<body>


<?php
 if ( is_catCAPTCHA_ok() ) {
  echo 'cat CAPTCHA is set';
 }
 else {
  echo 'cat CAPTCHA is <b>not</b> set';
 }
?>


</body>
</html>
```

CAPTCHAs Are Easy

If you aren't protecting very sensitive information, consider a simple CAPTCHA. In my case, I wanted to keep random strangers from seeing

my Christmas wish list. That's not exactly "Fort Knox" material. So a straightforward CAPTCHA does the job.

A good CAPTCHA is easy for certain people to solve and difficult for others to solve. For my web pages, I wanted to let in friends and family, and it's easy enough to do that with a CAPTCHA that uses photos of my cats.

Every time visitors load the CAPTCHA page, they see a different set of nine cats. Eight of these cats do not belong to me; only one of them is the correct cat. The correct cat is different each time and appears in a random location. The first time you load the page, my cat might be in the upper-left position. The next time, you might find my cat in the lower-right or somewhere else. This is a "good enough" test to demonstrate that you know me. Only my friends and family should know what my cats look like.

In practice, it takes me about a second to find my cat among the "forest" of other cats. And my friends and family say they like seeing the cats whenever they access my personal website. They say it's just adorable, and I agree.■

Jim Hall is an advocate for free and open-source software, best known for his work on the FreeDOS Project. Jim earned his MS in Scientific and Technical Communication from the University of Minnesota, focusing on the usability of open-source software. At work, Jim is the Chief Information Officer for Ramsey County, Minnesota.

Send comments or feedback via http://www.linuxjournal.com/contact or to ljeditor@linuxjournal.com.

RETURN TO CONTENTS