PHP: Protecting forms using a CAPTCHA

The CAPTCHA [http://www.google.com/search?q=define:CAPTCHA] approach to securing forms is not new - it first appeared in the late 90's for domain name submissions to search engines and the like - but with the exponential growth of scripted exploits it's coming to the fore once again. The main targets are Guestbook and Contact forms, but any online form can be a target for abuse.

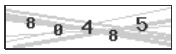
The code presented here shows you how to create a simple CAPTCHA graphic with random lines and digits and how to incorporate it into an HTML form to prevent automated submission by malicious scripts.

1. Creating a CAPTCHA graphic using PHP

The following code needs to be saved as a stand-alone PHP file (we call it `captcha.php`). This file creates a PNG image containing a series of five digits. It also stores these digits in a session variable so that other scripts can know what the correct code is and validate that it's been entered correctly.

```
<?PHP
  // Adapted for The Art of Web: www.the-art-of-web.com
  // Please acknowledge use of this code by including this header.

  // initialise image with dimensions of 120 x 30 pixels
  $image = @imagecreatetruecolor [http://www.php.net/imagecreatetruecolor] (120, 30) or die("Cannot Initialize new GD

  // set background to white and allocate drawing colours
  $background = imagecolorallocate [http://www.php.net/imagecolorallocate] ($image, 0xFF, 0xFF, 0xFF);
  imagefill [http://www.php.net/imagefill] ($image, 0, 0, $background);
  $linecolor = imagecolorallocate($image, 0xCC, 0xCC, 0xCC);
  $textcolor = imagecolorallocate($image, 0x33, 0x33, 0x33);

  // draw random lines on canvas
  for($i=0; $i < 6; $i++) {
    imagesetthickness [http://www.php.net/imagesetthickness] ($image, rand(1,3));
    imageline [http://www.php.net/imageline] ($image, 0, rand(0,30), 120, rand(0,30), $linecolor);
  }

  session_start [http://www.php.net/session_start] ();

  // add random digits to canvas
  $digit = '';
  for($x = 15; $x <= 95; $x += 20) {
    $digit .= ($num = rand(0, 9));
    imagechar [http://www.php.net/imagechar] ($image, rand(3, 5), $x, rand(2, 14), $num, $textcolor);
  }

  // record digits in session variable
  $_SESSION['digit'] = $digit;

  // display image and clean up
  header('Content-type: image/png');
  imagepng [http://www.php.net/imagepng] ($image);
  imagedestroy [http://www.php.net/imagedestroy] ($image);
?>
```

The output of this script appears as follows (reload to see it change):



This image is meant to be difficult for 'robots' to read, but simple for humans (the Turing test). You can make it more difficult for them by addition of colours or textures, or by using different fonts and a bit of rotation.

We've simplified the script presented above as much as possible so that you can easily customise it for your site and add more complexity as necessary. Further down the page you can find examples that use colours, rotation and different fonts, but the basic concept is the same.

2. Adding a CAPTCHA to your forms

In your HTML form you need to make sure that the CAPTCHA image is displayed and that there's an input field for people to enter the CAPTCHA code for validation. Here's a 'skeleton' of how the HTML code for your form might appear:

```
<form method="POST" action="form-handler" onsubmit="return checkForm(this);">
```

```
...

<p><img src="/captcha.php" width="120" height="30" border="1" alt="CAPTCHA"></p>
<p><input type="text" size="6" maxlength="5" name="captcha" value=""><br>
<small>copy the digits from the image into this box</small></p>

...

</form>
```

If you're using JavaScript form validation then you can test that a code has been entered in the CAPTCHA input box before the form is submitted. This will confirm that exactly five digits have been entered, but not say anything about whether they're the right digits as that information is only available on the server-side ($_SESSION) data.

So again, here's a skeleton of how your JavaScript form validation script might appear:

```
<script type="text/javascript">

  function checkForm(form)
  {
    ...

    if(!form.captcha.value.match(/^\d{5}$/)) {
      alert('Please enter the CAPTCHA digits in the box provided');
      form.captcha.focus();
      return false;
    }

    ...

    return true;
  }

</script>
```

Finally, in the server-side script that is the target of the form, you need to check that the code entered in the form by the user matches the session variable set by the `captcha.php` script:

```
<?PHP
  if($_POST && all required variables are present) {
    ...

    session_start();
    if($_POST['captcha'] != $_SESSION['digit']) die("Sorry, the CAPTCHA code entered was incorrect!");
    session_destroy [http://www.php.net/session_destroy] ();

    ...
  }
?>
```

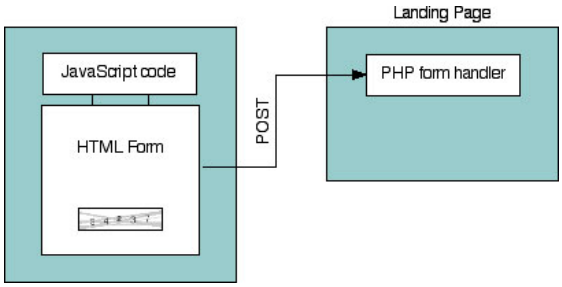
It's important to call `session_start()` both in the `captcha.php` script (when seting the session variable) **and** in the server-side validation script (in order to retrieve the value) as those files are processed independently and can't otherwise share information. We call `session_destroy()` only after the form submission has been verified.

You can see this code working in the Feedback form below.

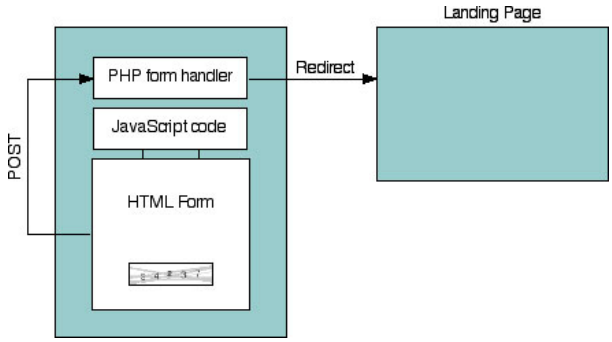
3. Putting it all together

There has been feedback sent by a number of people confused about which code to put where to get this working on their own website. To make it clearer I've put together a couple of diagrams which illustrate the two most common solutions.

Here you can see illustrated the simplest and most common setup, but by no means the best solution. The form is checked using JavaScript and then POSTed to another page/script where the data is processed:



A more 'professional' solution involves a practice called Post/Redirect/Get (PRG) which means that the data is first processed and then the user is redirected to a landing page:



This avoids a number of issues including problems caused when someone reloads the landing page which in the first configuration would cause all the POST data to be re-submitted.

This can also be implemented using three scripts where the form handler has it's own file and decides whether to redirect back to the FORM or forward to the landing page depending on whether the data validates.

In any case the PHP form handler code needs to appear as the first item before any HTML code is generated.

4. Upgrading the CAPTCHA to block new bots

The CAPTCHA image presented above was 'cracked' after a matter of months by one or two bots. Fortunately a few small changes to the code can send them packing at least for a while.

Here's some code to 'jazz up' our CAPTCHA to give it a better chance of being bot-proof. The sections of code that have been changed are highlighted:

```
<?PHP
  // Adapted for The Art of Web: www.the-art-of-web.com
  // Please acknowledge use of this code by including this header.

  // initialise image with dimensions of 120 x 30 pixels
  $image = @imagecreatetruecolor(120, 30) or die("Cannot Initialize new GD image stream");

  // set background and allocate drawing colours
  $background = imagecolorallocate($image, 0x66, 0x99, 0x66);
  imagefill($image, 0, 0, $background);
  $linecolor = imagecolorallocate($image, 0x99, 0xCC, 0x99);
  $textcolor1 = imagecolorallocate($image, 0x00, 0x00, 0x00);
  $textcolor2 = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);

  // draw random lines on canvas
  for($i=0; $i < 6; $i++) {
    imagesetthickness($image, rand(1,3));
    imageline($image, 0, rand(0,30), 120, rand(0,30) , $linecolor);
  }

  session_start();

  // add random digits to canvas using random black/white colour
  $digit = '';
  for($x = 15; $x <= 95; $x += 20) {
    $textcolor = (rand() % 2) ? $textcolor1 : $textcolor2;
    $digit .= ($num = rand(0, 9));
    imagechar($image, rand(3, 5), $x, rand(2, 14), $num, $textcolor);
  }

  // record digits in session variable
  $_SESSION['digit'] = $digit;

  // display image and clean up
  header('Content-type: image/png');
  imagepng($image);
  imagedestroy($image);
?>
```

And here is the modified CAPTCHA graphic produced by the new code:

All we've done here is changed the background colour from white to green, the lines from grey to light green and the font colour from black to a mixture of white and black.

This method has now also been cracked by a small number of bots. In recent days we've seen 10-20 succesful exploits a day, but we're not going to give up. Read on for details of a more advanced CAPTCHA.

For information on how the CAPTCHA images can be cracked, read this article [http://www.mperfect.net/aiCaptcha/] .

5. Yet another CAPTCHA

Here's the next version that we've been using until recently. The main change from those presented above is that we now use a larger range of fonts to confuse the spambots. ~~You can find a good resource for~~

~~GDF fonts under References below.~~ Unfortunately GDF fonts are now hard to come by, but there are alternatives using TrueType (TTF) fonts. The positioning of the lines has also changed to make them more random.

```
<?PHP
  // Adapted for The Art of Web: www.the-art-of-web.com
  // Please acknowledge use of this code by including this header.

  // initialise image with dimensions of 120 x 30 pixels
  $image = @imagecreatetruecolor(120, 30) or die("Cannot Initialize new GD image stream");

  // set background and allocate drawing colours
  $background = imagecolorallocate($image, 0x66, 0xCC, 0xFF);
  imagefill($image, 0, 0, $background);
  $linecolor = imagecolorallocate($image, 0x33, 0x99, 0xCC);
  $textcolor1 = imagecolorallocate($image, 0x00, 0x00, 0x00);
  $textcolor2 = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);

  // draw random lines on canvas
  for($i=0; $i < 8; $i++) {
    imagesetthickness($image, rand(1,3));
    imageline($image, rand(0,120), 0, rand(0,120), 30 , $linecolor);
  }

  // using a mixture of system and GDF fonts
  $fonts = array(3,4,5);
  $fonts[] = imageloadfont('$fontdir/bmcorrode.gdf');
  $fonts[] = imageloadfont('$fontdir/bmreceipt.gdf');
  $fonts[] = imageloadfont('$fontdir/checkbook.gdf');
  shuffle($fonts);

  session_start();

  // add random digits to canvas using random black/white colour
  $digit = '';
  for($x = 15; $x <= 95; $x += 20) {
    $textcolor = (rand() % 2) ? $textcolor1 : $textcolor2;
    $digit .= ($num = rand(0, 9));
    imagechar($image, array_pop($fonts), $x, rand(2, 14), $num, $textcolor);
  }

  // record digits in session variable
  $_SESSION['digit'] = $digit;

  // display image and clean up
  header('Content-type: image/png');
  imagepng($image);
  imagedestroy($image);
?>
```

And here's the result - a little less readable perhaps for humans, but a lot less readable for robots who were starting to get around the previous version:

If you have trouble locating GDF fonts you can also use `imagettftext` in place of `imageloadfont/imagechar` which lets you use TTF fonts instead of architecture dependent GD fonts. This is demonstrated in the following section. There are also tools for converting TTF fonts into GD format.

6. A more readable CAPTCHA using TTF fonts

As you can see we've upgraded our CAPTCHA image once again. Mainly to make it more human-friendly with larger characters. The new version is similar to the above, but uses a (free) TTF font and some rotation. The background hasn't changed.

```
<?PHP
  // Adapted for The Art of Web: www.the-art-of-web.com
  // Please acknowledge use of this code by including this header.

  // initialise image with dimensions of 160 x 45 pixels
  $image = @imagecreatetruecolor(160, 45) or die("Cannot Initialize new GD image stream");

  // set background and allocate drawing colours
  $background = imagecolorallocate($image, 0x66, 0xCC, 0xFF);
  imagefill($image, 0, 0, $background);
  $linecolor = imagecolorallocate($image, 0x33, 0x99, 0xCC);
  $textcolor1 = imagecolorallocate($image, 0x00, 0x00, 0x00);
  $textcolor2 = imagecolorallocate($image, 0xFF, 0xFF, 0xFF);

  // draw random lines on canvas
  for($i=0; $i < 8; $i++) {
    imagesetthickness($image, rand(1,3));
    imageline($image, rand(0,160), 0, rand(0,160), 45, $linecolor);
  }

  session_start();

  // using a mixture of TTF fonts
  $fonts = array();
  $fonts[] = "ttf-dejavu/DejaVuSerif-Bold.ttf";
  $fonts[] = "ttf-dejavu/DejaVuSans-Bold.ttf";
  $fonts[] = "ttf-dejavu/DejaVuSansMono-Bold.ttf";

  // add random digits to canvas using random black/white colour
  $digit = '';
```

```
  for($x = 10; $x <= 130; $x += 30) {
    $textcolor = (rand() % 2) ? $textcolor1 : $textcolor2;
    $digit .= ($num = rand(0, 9));
    imagettftext [http://php.net/imagettftext] ($image, 20, rand(-30,30), $x, rand(20, 42), $textcolor, $fonts[arr
  }

  // record digits in session variable
  $_SESSION['digit'] = $digit;

  // display image and clean up
  header('Content-type: image/png');
  imagepng($image);
  imagedestroy($image);
?>
```

You can reference TTF font files either by using the complete filesystem path starting with '/', or by placing them in a directory included in the `default font path` in PHP which will depend on your system.

Here you can see the output of the above script. The TTF fonts used in this example come from the (free) `ttf-dejavu-core` package on Debian, but there are thousand of Truetype fonts to choose from, many of them free. Typefaces where the characters have gaps or a bit of flair will work better against spambots, but remember it also has to be human-readable.

The `imagettftext` function is quite simple to use in this situation:

```
imagettftext($image, $fontsize, $angle, $xpos, $ypos, $color, $fontfile, $text);
```

We're using a font size of 20 pixels and a rotation (`$angle`) of between -30 and +30 degrees for each digit. The characters are 30 pixels apart (`$xpos`) and have a random vertical offset - enough just to touch the top or bottom of the canvas as that makes it tougher for a robot to decipher.

So how long will we need to keep using CAPTCHAs to protect forms? Unfortunately, until Microsoft invents a secure operating system and puts an end to botnets [http://en.wikipedia.org/wiki/Botnet] we need to keep evolving our security to stay ahead of the spammers. For small websites something like the above example will work fine. For more popular sites there are any kind of protective measures, but that's another story.

7. Usability improvements

It's the little things that make your visitors more relaxed about filling in forms. The code below has been modified to limit the input to only numbers using the `onkeyup` event, and adding an option to reload/refresh the CAPTCHA image in the case that it's not readable.

```
....

<p><img id="captcha" src="/captcha.php" width="160" height="45" border="1" alt="CAPTCHA">
<small><a href="#" onclick="
  document.getElementById('captcha').src = '/captcha.php?' + Math.random();
  document.getElementById('captcha_code').value = '';
  return false;
">refresh</a></small></p>
<p><input id="captcha_code" type="text" name="captcha" size="6" maxlength="5" onkeyup="this.value = this.val

...
```

In your form, the CAPTCHA section will then appear something like the following:

Security Check

 refresh

CAPTCHA*:

<- copy the digits from the image into this box

You can see this code in action in the Feedback form at the bottom of the page, using the latest CAPTCHA.

8. Related Articles - Form Validation

- HTML5 Form Validation Examples [HTML]
- Validating a checkbox with HTML5 [HTML]
- Password Validation using regular expressions and HTML5 [JAVASCRIPT]
- Date and Time [JAVASCRIPT]
- Preventing Double Form Submission [JAVASCRIPT]
- Form Validation [JAVASCRIPT]
- Credit Card numbers [JAVASCRIPT]
- A simple modal feedback form with no plugins [JAVASCRIPT]

- **Protecting forms using a CAPTCHA** [PHP]
- Basic Form Handling in PHP [PHP]

9. References

- User-defined Bitmap Fonts [http://www.devtrolls.com/qdf_fonts/]

10. User Comments

Tom B [http://www.ashcomllc.com/] 19 January, 2015

Followed the examples of creating captcha.php, adding the cpatcha ot the form, adding tghe javascript to the head of the contact form page and adding hte session start script to the beginning of the process page. Captcha show up on form but the form will nto longer process or display any error message. Could you possibly clarify what might be the problem?
Thank you

saleem raza 21 December, 2014

hii..! sir let me know what is" php form handler"
i am confused on it coz i,ve got captcha digits but form handler not appear...kindly if u take few mint to it it would b great ,,,thans

Tom 10 December, 2014

I can't get the PHP-code to check if the digits entered are correct. Could you help me out as I'm new to these things.

RB 17 April, 2014

Very good! Could you include a complete set of files, including the contact form, as a downloadable zip? Thanks!

> No, but you can find code for a standalone feedback form in this article. 😎

Richard [http://golfcarportal.com/] 4 April, 2014

Tried number #1 with javascript.
Processes fine but I can enter any 6 digits and it processes and sends the email.

> You need to have the PHP code checking the CAPTCHA, as well as the CAPTCHA itself in the HTML form. The correct digits are not (can not be) known by JavaScript - only when the form is submitted and checked on the server. 😎

Bloke 4 December, 2013

Example #6 is not working for me. Says it cannot be displayed because of errors. I am using TTF fonts. My fonts are in /fonts/ folder which is inside /captcha/ folder. The other examples worked fine.

Clare [http://radicalworlddesign.com] 28 August, 2013

Great tutorial!
However, I too am having some confusion about the last part.
Is the captcha.php page supposed to be the landing page? If so, exactly where do we drop the last bit of code?
Do we create another page and drop the code there??

> No the PHP script *is* the CAPTCHA graphic. You reference it as you would any image using the IMG tag. 😎

Rama 16 July, 2013

Hi !
nice work nice tutorial!
Could you just give the code that change the "!" to "v" when the captcha is correct?
Thx a lot !

> That's actually done just with HTML5 and CSS - no scripting required.
> You can find instructions here: HTML5 Form Validation Examples
> 😎

Dave 15 December, 2012

What I am looking for is an example of the PHP form handler and how to implement it?

mike 9 November, 2012

thanks, finally got this working but would like to enhance...so two noobie q's:

1. I tried using: $image = @imagecreatefromjpeg('../images/captcha.jpg');

but captcha is too big for box and digits arent all visible. How would you set the size?

2. how do you get the digits below in the circles?

thanx

> 1) If you first get it working using a coloured background (imagecreatetruecolor) then you can switch to using an image background (imagecreatefromjpeg) using an image ***with the same dimensions***.
>
> 2) The circles are actually a part of the font we're using 😎

JC 25 April, 2012

Hi, I am having so much trouble with the last step...the server-side thing. I have a php file through which my contact form processes, but I cannot figure out where to put the script to make the CAPTCHA code actually process. I have tried putting it numerous places based on your answer from like 5 years ago, but I have no idea what you're talking about. Would you mind please explaining it in plain language for those who don't know what you're referring to? Would appreciate it. I at least got the CAPTCHA code to go into my html script. Now I just need to actually make it work.

Thank you.

RD [http://www.adventurelinks.ie/] 20 February, 2012

To confuse the bots that much more you could use an image for the background instead of a solid colour:
$image = @imagecreatefromjpeg('../images/captcha.jpg');
Make sure captcha.jpg is the same size of your captcha and that the colours work (ie the picture is faded and the numbers are darker)
Give those bots a bit more to fight with!!!
Great info and very well explained. Thanks

tom 2 April, 2011

I want to get the same blue the full latest version CAPTCHA like you. It is possible to give URL to me for downloading the PHP File (captcha.php).
Many thanks, Tom.

> The code's there now for copying. Also with the new TTF option it's easy to create your own unique CAPTCHA which is always a better option.

Shaks 17 March, 2011

Hi, I must say this is the clearest and most helpfull instruction to use code that I have come across in a while.

Thanks.

> The code's there now for copying. Also the new TTF option makes it easy to create your own unique CAPTCHA using different fonts and effects which is always better than using someone else's.

Gary 15 October, 2009

HI, I am new to this, maybe you can help.
Where exactly is this suposed to go?

```
if($_POST && all required variables are present) {
...
}
```

> I've tried to illustrate this in the section "Putting it all together". Look for where "PHP form handler" appears in the graphics.

Mewp 3 August, 2009

I have only one question: how is it better than ReCAPTCHA [recaptcha.net]?

also, your CAPTCHA is barely readable sometimes (due to small numbers), and have few weaknesses:
By scanning the first row I can determine color of background and lines. If i do that, i know that any other color is a number, and I have only to detect characters. Assuming three fonts in three sizes, there are 3*3*10 possibilities of generated character bitmap. That's rather easy to break for a specialized bot.

Your idea to rotate numbers wouldn't work either (I've seen bots break that).

> You only have one question? 😃
>
> - Noone said this was 'better' than any other system. It's a **tutorial**;
> - Any CAPTCHA can be broken by a 'specialized' bot. We're only looking at protecting from 'wild' bots, which it does quite well 😎

Brian [http://thinklike.net/] 15 November, 2008

Thanks for a very informative and easy to follow tutorial!

Paul Smith [http://www.5stardisco.com/] 10 March, 2008

1st of all thank you for a straightforward script that works well.

You mention adding colour to the background and making the numbers black & white (as in your own Captcha). Can you provide a sample script with these changes.

> You can find the code for the second CAPTCHA now on the page. We only make public new versions of the code as and when we upgrade our own 'live' version, which has now happened.

Lisa [http://www.2createawebsite.com] 3 February, 2007

First of all, your captcha instructions are the best I've seen yet! I almost got it going but I'm confused about the last part where I need to add the session start variable.

My PHP feedback forms has a .php validation file that validates everything on the form before the form is submitted. Is this the target file you mentioned? I'm not sure where to put that session start info into my script. I pasted it into that file but I get the captcha code is incorrect no matter what I enter. If you could help that would be great thanks!

> By definition PHP can validate only AFTER the form is submitted. In your PHP code you need to call session_start() before any $_SESSION variables become available, and then destroy the session. If you have problems try using the print_r function [http://www.php.net/print_r] to display the $_POST and $_SESSION data to make sure they are populated. If your form uses GET instead of POST as the submission method then replace $_POST with $_GET.