

# security risks for online services by relying on reCAPTCHA

*Benjamin Wegener - <info[at]wegeneredv[dot]de>*

October 22, 2010

revised version October 27, 2010

## Abstract

A description of major flaws in Google's reCAPTCHA service and a proof of concept regarding successful exploitation using freely available tools in order to achieve a recognition rate of approximately 17%.

## 1 Introduction

Many popular websites use the CAPTCHAs (a contrived acronym for “completely automated public turing test to tell computers and humans apart”<sup>1</sup>) provided by reCAPTCHA to prevent the automated subscription and registration or unauthorized and extensive use of their offers and services.

reCAPTCHA was originally developed by Luis von Ahn, Benjamin Maurer, Colin McMillen, David Abraham and Manuel Blum of the Computer Science Department of the Carnegie Mellon University in Pittsburgh PA to help the digitization of books by using CAPTCHAs of scanned words the optical-character-recognition-software used couldn't recognize<sup>2</sup>. In September 2009 Google acquired the system to use it on a wider range of projects like Google Books<sup>3</sup>.

A challenge served by reCAPTCHA consists of two words - one already known by the system and one to be identified by the human solver. The unknown word is given to many humans in a short period of time and the most answered solution is taken. But reCAPTCHA only checks if the known word is typed in correct and also allows variety of accidental errors to be made.

## 2 Major Flaws

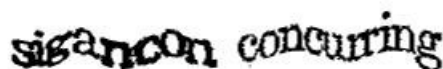


Figure 1

---

1 <http://en.wikipedia.org/wiki/CAPTCHA>

2 Luis von Ahn, Ben Maurer, Colin McMillen, David Abraham and Manuel Blum (2008). "[reCAPTCHA: Human-Based Character Recognition via Web Security Measures](#)" (PDF). *Science* **321** (5895): 1465–1468.

3 "[Teaching computers to read: Google acquires reCAPTCHA](#)". Google. Retrieved 2009-09-16.

As mentioned before, only the already known word is taken for the check if the user is human or not. This allows the use of a dictionary attack for solving the challenge, because it is likely that this word is more common than the other one. This could be accomplished by using the by the used ORC-software recognized phrase and comparing it to the word list. If this word doesn't occur in the list, the most likely replacement would be used. Since reCAPTCHA takes mostly English books and sources to be digitized, a comprehensive and huge word list could be made out of the decompressed archive of the English Wikipedia<sup>4</sup> using a simple algorithm to extract all the words separated by punctuation marks and order them alphabetically.

A user is allowed to enter 32 wrong answers before security measures are taken and a CAPTCHA consisting of two known words is generated which must be answered perfectly. After that the IP-address is blocked for a brief period of time.

The puzzle is monochromatic and besides a simple wave distortion no other effects are applied to the phrase. The font used mostly is of serif type because of the sources - so a OCR-Software doesn't need to be trained for other fonts than this.

### 3 Preprocessing

To attack the mentioned lacks of security, the image is being transformed by some simple algorithms:

#### *3.1 adjustment of contrast*

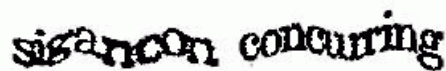


Figure 2

By using different intensities of contrast for each word, two problems are dealt with at once: the distinction between normal fonts and bold ones and the removal of “dirty spots” or artifacts left over by OCR. Hereby the density of black and white pixels in the areas of both words is separately adjusted to a ratio of 1.5 white to black pixel.

#### *3.2 resizing using hq2x*

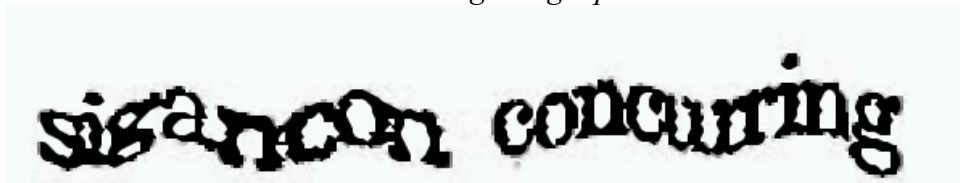


Figure 3

The open source program *hq2x*<sup>5</sup> uses a pixel art algorithm to enlarge pixel based imagery which fits perfectly in the purpose here after the contrast is applied to get a better overall quality.

#### *3.3 distortion removal*

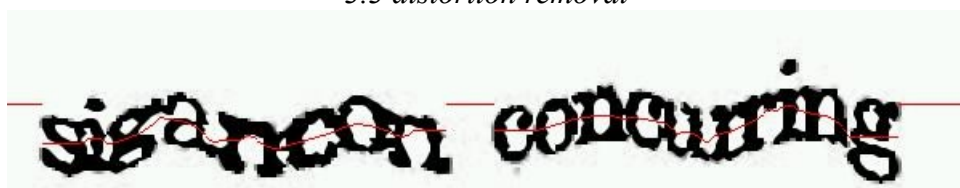


Figure 4

<sup>4</sup> <http://download.wikimedia.org/enwiki/latest/enwiki-latest-pages-articles.xml.bz2>

<sup>5</sup> <http://en.wikipedia.org/wiki/Hqx>

An algorithm to determine the average center line of the phrase served by reCAPTCHA is applied. This is accomplished by subtracting the bottom least black pixel's y-position from the top least ones' for every column of the image. The origin is in the top left corner as usual. To get a nice line without too huge "bumps" in it, an exponential smoothing with a factor of 0.2 is applied to the line in both directions. So 1. every pixel is taken and the y-value is multiplied with 0.8 and then added to the following pixel's y-value multiplied by 0.2 from left to right and afterward from right to left. Then every column is moved by  $(\text{height\_of\_image}/2 - \text{center\_value}(x))$ .

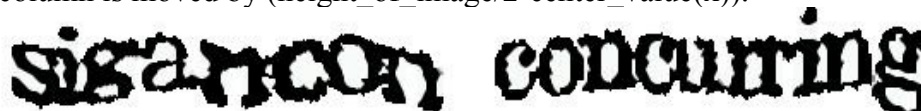


Figure 5

## 4 Training the OCR-software

The OCR-Software used in this proof of concept is *tesseract-ocr*<sup>6</sup>, an open source OCR-engine developed by the Hewlett-Packard Labs<sup>7</sup> between 1985 and 1995 and is now revised by Google itself. The *tesseract* is the 4-dimensional analog to the cube.

The original trained language of *tesseract-ocr* obviously is English, so it doesn't need to be retrained for this special purpose. But the engine in current version 3.0 is only supporting word lists to a limited total count, when it comes to disambiguation. But there is a possibility to just copy a bigger word list to the *tessdata* folder naming it '*eng.user-words*' in this case. I'm working on a word list based on Wikipedia word list as mentioned before and I also prepare training files with a list of most common words in the English language<sup>8</sup>. It is also useful to reject some characters *reCAPTCHA* doesn't use. To achieve this, simply add a config file to *tesseract-ocr* including '*tessedit\_char\_whitelist abcdefghijklmnopqrstuvwxyz1234567890*'<sup>9</sup>. After a test using over 5000 images served by *reCAPTCHA* a successful rate for the control word (the one, *reCAPTCHA* already knows) of 17,23% could be reached<sup>10</sup>. And this is by far more than one needs to reach for a successful exploitation.

## 5 Conclusion

Major internet services such as facebook, twitter, single-click hoster and most public boards and forums rely on the security of *reCAPTCHA*. Using proxies and/or dynamic IPs one could automate for example the registration on twitter to gain the maximum number of (fake) followers.

One year ago Jonathan Wilkins made suggestions<sup>11</sup> to strengthen *reCAPTCHA* (and others) and Chad Houck explained<sup>12</sup>, how to bypass the annoying ellipse and the black bar which both do not appear in *reCAPTCHA* images as of today. This might allow some humans to decipher these puzzles more easily but it also lowers the security level regarding to automated approaches.

I strongly advise Google to bring back things like the ellipse mentioned to increase security of *reCAPTCHA* again. Otherwise the great idea of "stop(ping) spam. read(ing) books."<sup>13</sup> will become useless very soon.

<sup>6</sup> <http://code.google.com/p/tesseract-ocr/downloads/list>

<sup>7</sup> [http://en.wikipedia.org/wiki/Tesseract\\_%28software%29#History](http://en.wikipedia.org/wiki/Tesseract_%28software%29#History)

<sup>8</sup> <http://www.wordfrequency.info/files/entriesWithoutCollocates.txt>

<sup>9</sup> [http://code.google.com/p/tesseract-ocr/wiki/FAQ#How\\_do\\_I\\_recognize\\_only\\_digits?](http://code.google.com/p/tesseract-ocr/wiki/FAQ#How_do_I_recognize_only_digits?)

<sup>10</sup> <http://wegeneratedv.de/arc>

<sup>11</sup> <http://bitland.net/captcha.pdf>

<sup>12</sup> <http://n3on.org/projects/reCAPTCHA/>

<sup>13</sup> <http://www.google.com/recaptcha>

## 6 Appendix

### Source Code:

---

```
<?php
# anti-recaptcha v0.2c (c)opyleft 2010 http://wegeneredv.de/arc *****
# *****
# future fixes and changes:
# - gaining greater recognition rates by using wikipedia as wordlist
# - see included paper 'antirecaptcha.pdf' for further information or
#   download it from http://wegeneredv.de/antirecaptcha.pdf
# - clean up source / replace repeating code by functions
# - convert to c++ using http://github.com/facebook/hiphop-php
#
# *****
# Installation:
# copy directly to your jDownloader installation directory (e.g.
# c:\program files\jDownloader and extract.
# -runs only on Windows systems-
#
# *****
# Changelog:
#
# v0.2c
# - applied selective contrast to both words
# - improved success rate to 17% by adding a bigger wordlist to tesseract
# - some hoster aren't supported any longer due to plugin errors caused by jD
#
# v0.2b
# - removed 'best match' for speed reasons
# - removed intersection cleaning due to ineffectivity
# - added experimental sharpening
# - changed from VietOCR to tesseract
#
# v0.2a
# - pushed recognition rate to ca. 10%
# - multiple iterations to ocr ('best match')
# - partially cleaned intersections between letters
# - cleaned up sourcecode
# - added support for the following onc-click-hosters:
# combozip.com,crमित.in,crazyupload.com,drop.io,enterupload.com,extabit.com,
# filebling.com,filechip.com,filescloud.com,fileserv.com,filesmonster.com,
# filesonic.com,filestab.com,filestrack.com,filevo.com,freakshare.net,
# free-share.ru,hatlimit.com,hidemyass.com,hostingcup.com,maknyos.com,
# mediafire.com,putshare.com,quickupload.com,slingfile.com,tgf-services.com
# uploadfloor.com
#
# v0.1b
# - changed the directory structure
#
# v0.1
# - first version tested
#
# known bugs:
# - hotfile.com folder gets deleted by jDownloader:
#   -> set NTFS-permissions to 'read only' for the user group 'everyone'
# - sometimes the captcha input window pops up - just *ignore* it
#
# *****
# variables *****
$debug_mode=true;
$enable_sharpen=true;
$shit="!$%&@";
$white_columns_needed_for_separation=7;
$h2qx_commandline='h2qx.exe contrast.bmp resized.bmp';
$tesseract_commandline='tesseract.exe cropped.jpg output -l eng nobatch +arc';
$input_image=".input.jpg";
$mid_exponential_smoothing_factor=0.15;
$vector_threshold=0.33;
$surrounding_pixel_threshold_min=7;
$surrounding_pixel_threshold_max=8;
$selective_contrast_threshold=1.5;
$contrast=127;
# function imagesetpixel_rgb *****
# *****
function imagesetpixel_rgb($im,$x_pos,$y_pos,$r,$g,$b){
    imagesetpixel($im,$x_pos,$y_pos,65536*$r+256*$g+$b);
}
# function readfile_chunked *****
# *****
function readfile_chunked ($filename) {
    $chunksize = 1*(1024*1024); # how many bytes per chunk
    $buffer = "";
    $handle = fopen($filename, 'rb');
    if ($handle === false) {
```

```

        return false;
    }
    while (!feof($handle)) {
        $buffer = fread($handle, $chunksize);
    }
    fclose($handle);
    return $buffer;
}
# function imagecreatefrombmp *****
# *****
# save Bitmap-File with GD library
# written by mgutt of http://www.programmierer-forum.de/function-imagecreatefrombmp-laeuft-mit-allen-bitraten-t143137.htm
# based on the function by DHKold of http://www.php.net/manual/de/function.imagecreate.php#53879
if (!function_exists('imagecreatefrombmp')) { function imagecreatefrombmp($filename) {
    # version 1.00
    if (!($fh = fopen($filename, 'rb'))) {
        trigger_error('imagecreatefrombmp: Can not open ' . $filename, E_USER_WARNING);
        return false;
    }
    # read file header
    $meta = unpack('vtype/Vfilesize/Vreserved/Voffset', fread($fh, 14));
    # check for bitmap
    if ($meta['type'] != 19778) {
        trigger_error('imagecreatefrombmp: ' . $filename . ' is not a bitmap!', E_USER_WARNING);
        return false;
    }
    # read image header
    $meta += unpack('Vheadersize/Vwidth/Vheight/vplanes/vbits/Vcompression/Vimagesize/Vxres/Vyres/Vcolors/Vimportant', fread($fh,
40));
    # read additional 16bit header
    if ($meta['bits'] == 16) {
        $meta += unpack('VrMask/VgMask/VbMask', fread($fh, 12));
    }
    # set bytes and padding
    $meta['bytes'] = $meta['bits'] / 8;
    $meta['decal'] = 4 - (4 * (($meta['width'] * $meta['bytes'] / 4) - floor($meta['width'] * $meta['bytes'] / 4)));
    if ($meta['decal'] == 4) {
        $meta['decal'] = 0;
    }
    # obtain imagesize
    if ($meta['imagesize'] < 1) {
        $meta['imagesize'] = $meta['filesize'] - $meta['offset'];
        # in rare cases filesize is equal to offset so we need to read physical size
        if ($meta['imagesize'] < 1) {
            $meta['imagesize'] = @filesize($filename) - $meta['offset'];
            if ($meta['imagesize'] < 1) {
                trigger_error('imagecreatefrombmp: Can not obtain filesize of ' . $filename . '!', E_USER_WARNING);
                return false;
            }
        }
    }
    # calculate colors
    $meta['colors'] = !$meta['colors'] ? pow(2, $meta['bits']) : $meta['colors'];
    # read color palette
    $palette = array();
    if ($meta['bits'] < 16) {
        $palette = unpack('l' . $meta['colors'], fread($fh, $meta['colors'] * 4));
        # in rare cases the color value is signed
        if ($palette[1] < 0) {
            foreach ($palette as $i => $color) {
                $palette[$i] = $color + 16777216;
            }
        }
    }
    # create gd image
    $im = imagecreatetruecolor($meta['width'], $meta['height']);
    $data = fread($fh, $meta['imagesize']);
    $p = 0;
    $vide = chr(0);
    $y = $meta['height'] - 1;
    $error = 'imagecreatefrombmp: ' . $filename . ' has not enough data!';
    # loop through the image data beginning with the lower left corner
    while ($y >= 0) {
        $x = 0;
        while ($x < $meta['width']) {
            switch ($meta['bits']) {
                case 32:
                case 24:
                    if (!($part = substr($data, $p, 3))) {
                        trigger_error($error, E_USER_WARNING);
                        return $im;
                    }
                    $color = unpack('V', $part . $vide);
                    break;
                case 16:
                    if (!($part = substr($data, $p, 2))) {
                        trigger_error($error, E_USER_WARNING);
                        return $im;
                    }
                    $color = unpack('v', $part);

```

```

(($color[1] & 0x001f) << 3);

$color[1] = (($color[1] & 0xf800) >> 8) * 65536 + (($color[1] & 0x07e0) >> 3) * 256 +
break;
case 8:
$color = unpack('n', $vide . substr($data, $p, 1));
$color[1] = $palette[ $color[1] + 1 ];
break;
case 4:
$color = unpack('n', $vide . substr($data, floor($p), 1));
$color[1] = ($p * 2) % 2 == 0 ? $color[1] >> 4 : $color[1] & 0x0f;
$color[1] = $palette[ $color[1] + 1 ];
break;
case 1:
$color = unpack('n', $vide . substr($data, floor($p), 1));
switch (($p * 8) % 8) {
case 0:
$color[1] = $color[1] >> 7;
break;
case 1:
$color[1] = ($color[1] & 0x40) >> 6;
break;
case 2:
$color[1] = ($color[1] & 0x20) >> 5;
break;
case 3:
$color[1] = ($color[1] & 0x10) >> 4;
break;
case 4:
$color[1] = ($color[1] & 0x8) >> 3;
break;
case 5:
$color[1] = ($color[1] & 0x4) >> 2;
break;
case 6:
$color[1] = ($color[1] & 0x2) >> 1;
break;
case 7:
$color[1] = ($color[1] & 0x1);
break;
}
$color[1] = $palette[ $color[1] + 1 ];
break;
default:
trigger_error('imagecreatefrombmp: ' . $filename . ' has ' . $meta['bits'] . ' bits and this is not
supported!', E_USER_WARNING);
return false;
}
imagesetpixel($im, $x, $y, $color[1]);
$x++;
$p += $meta['bytes'];
}
$y--;
$p += $meta['decals'];
}
fclose($fh);
return $im;
}}
# function imagebmp *****
# *****
# create Bitmap-File with GD library
# written by mgutt of http://www.programmiererforum.de/imagebmp-gute-funktion-gefunden-t143716.htm
# based on the function by legend(legendsky@hotmail.com) of http://www.ugia.cn/?p=96
function imagebmp($im, $filename="", $bit=24, $compression=0) {
if (!in_array($bit, array(1, 4, 8, 16, 24, 32))) {
$bit = 24;
}
else if ($bit == 32) {
$bit = 24;
}
}
$bits = pow(2, $bit);
imagecolortopalette($im, true, $bits);
$width = imagesx($im);
$height = imagesy($im);
$colors_num = imagecolorstotal($im);
$rgb_quad = "";
if ($bit <= 8) {
for ($i = 0; $i < $colors_num; $i++) {
$colors = imagecolorsforindex($im, $i);
$rgb_quad .= chr($colors['blue']) . chr($colors['green']) . chr($colors['red']) . "\0";
}
}
$bmp_data = "";
if ($compression == 0 || $bit < 8) {
$compression = 0;
$extra = "";
$padding = 4 - ceil($width / (8 / $bit)) % 4;
if ($padding % 4 != 0) {
$extra = str_repeat("\0", $padding);
}
for ($j = $height - 1; $j >= 0; $j--) {
for ($i = 0;

```

```

        while ($i < $width) {
            $bin = 0;
            $limit = $width - $i < 8 / $bit ? (8 / $bit - $width + $i) * $bit : 0;
            for ($k = 8 - $bit; $k >= $limit; $k -= $bit) {
                $index = imagecolorat($img, $i, $j);
                $bin |= $index << $k;
                $i++;
            }
            $bmp_data .= chr($bin);
        }
        $bmp_data .= $extra;
    }
}

# RLE8
else if ($compression == 1 && $bit == 8) {
    for ($j = $height - 1; $j >= 0; $j--) {
        $last_index = "\0";
        $same_num = 0;
        for ($i = 0; $i <= $width; $i++) {
            $index = imagecolorat($img, $i, $j);
            if ($index != $last_index || $same_num > 255) {
                if ($same_num != 0) {
                    $bmp_data .= chr($same_num) . chr($last_index);
                }
                $last_index = $index;
                $same_num = 1;
            }
            else {
                $same_num++;
            }
        }
        $bmp_data .= "\0\0";
    }
    $bmp_data .= "\01";
}

$size_quad = strlen($rgb_quad);
$size_data = strlen($bmp_data);
}
else {
    $extra = "";
    $padding = 4 - ($width * ($bit / 8)) % 4;
    if ($padding % 4 != 0) {
        $extra = str_repeat("\0", $padding);
    }
    $bmp_data = "";
    for ($j = $height - 1; $j >= 0; $j--) {
        for ($i = 0; $i < $width; $i++) {
            $index = imagecolorat($img, $i, $j);
            $colors = imagecolorsforindex($img, $index);
            if ($bit == 16) {
                $bin = 0 << $bit;
                $bin |= ($colors["red"] >> 3) << 10;
                $bin |= ($colors["green"] >> 3) << 5;
                $bin |= $colors["blue"] >> 3;
                $bmp_data .= pack("v", $bin);
            }
            else {
                $bmp_data .= pack("c*", $colors["blue"], $colors["green"], $colors["red"]);
            }
        }
        $bmp_data .= $extra;
    }
    $size_quad = 0;
    $size_data = strlen($bmp_data);
    $colors_num = 0;
}

$file_header = 'BM' . pack("V3", 54 + $size_quad + $size_data, 0, 54 + $size_quad);
$info_header = pack("V3v2V*", 0x28, $width, $height, 1, $bit, $compression, $size_data, 0, 0, $colors_num, 0);
if ($filename != "") {
    $fp = fopen($filename, 'wb');
    fwrite($fp, $file_header . $info_header . $rgb_quad . $bmp_data);
    fclose($fp);
    return true;
}
echo $file_header . $info_header . $rgb_quad . $bmp_data;
return true;
}

$source_img=imagecreatefromjpeg($input_image);
# contrast *****
# *****
$source_img=imagecreatefromjpeg($input_image);
$height=imagesy($source_img);
$width=imagesx($source_img);
$first_word_start=0;
for($x=0;$x<$width;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($source_img,$x,$y);
        if($c==0){
            $first_word_start=$x;
        }
    }
}

```

```

    }
    if($first_word_start>0){
        break;
    }
}
$first_word_end=0;
for($x=$first_word_start;$x<$width;$x++){
    $black_pixel_found=false;
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($source_img,$x,$y);
        if($c==0){
            $black_pixel_found=true;
        }
    }
    if(!$black_pixel_found){
        $white_column_count++;
        if($white_column_count>$white_columns_needed_for_separation){
            $first_word_end=$x-$white_column_count;
        }
    }
    }else{
        $white_column_count=0;
    }
    if($first_word_end>0){
        break;
    }
}
$second_word_start=0;
for($x=$first_word_end+$white_columns_needed_for_separation;$x<$width;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($source_img,$x,$y);
        if($c==0){
            $second_word_start=$x;
        }
    }
    if($second_word_start>0){
        break;
    }
}
$second_word_end=0;
for($x=$width-1;$x>$second_word_start;$x--){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($source_img,$x,$y);
        if($c==0){
            $second_word_end=$x;
        }
    }
    if($second_word_end>0){
        break;
    }
}
}
# selective contrast *****
$source_top=null;
$source_bottom=null;
for($x=1;$x<$width-1;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($source_img,$x,$y);
        if($c<8355840){
            $source_top[$x]=$y;
            break;
        }
    }
    for($y=$height-1;$y>0;$y--){
        $c=imagecolorat($source_img,$x,$y);
        if($c<8355840){
            $source_bottom[$x]=$y;
            break;
        }
    }
}
$word1_contrast=1;
$word2_contrast=1;
a:
$contrast_img=imagecreatefromjpeg('input.jpg');
for($x=$first_word_start;$x<$first_word_end;$x++){
    for($y=0;$y<$height;$y++){
        if(($y>=$source_top[$x])&&($y<=$source_bottom[$x])){
            $rgb=imagecolorat($source_img,$x,$y);
            $r=($rgb>>16)&0xFF;
            $g=($rgb>>8)&0xFF;
            $b=$rgb&0xFF;
            if(($r>$word1_contrast)&&($r>$word1_contrast)&&($r>$word1_contrast)){
                imagesetpixel($contrast_img,$x,$y,16777215);
            }else{
                imagesetpixel($contrast_img,$x,$y,0);
            }
        }
    }
}
}
for($x=$second_word_start;$x<$second_word_end;$x++){
    for($y=0;$y<$height;$y++){

```



```

        if(($y>=$source_top[$x])&&($y<=$source_bottom[$x])){
            $rgb=imagecolorat($source_img,$x,$y);
            $r=($rgb>>16)&0xFF;
            $g=($rgb>>8)&0xFF;
            $b=$rgb&0xFF;
            if(($r>$word2_contrast)&&($r>$word2_contrast)&&($r>$word2_contrast)){
                imagesetpixel($contrast_img,$x,$y,16777215);
            }else{
                imagesetpixel($contrast_img,$x,$y,0);
            }
        }
    }
}
$word1_black_pixel=0;
$word1_white_pixel=0;
for($x=$first_word_start;$x<$first_word_end;$x++){
    for($y=0;$y<$height;$y++){
        if(($y>=$source_top[$x]-1)&&($y<=$source_bottom[$x]+1)){
            $c=imagecolorat($contrast_img,$x,$y);
            if($c==0){
                $word1_black_pixel++;
            }else{
                $word1_white_pixel++;
            }
        }
    }
}
$word1_contrast_ratio=($word1_black_pixel/$word1_white_pixel);
$word2_black_pixel=0;
$word2_white_pixel=0;
for($x=$second_word_start;$x<$second_word_end;$x++){
    for($y=0;$y<$height;$y++){
        if(($y>=$source_top[$x]-1)&&($y<=$source_bottom[$x]+1)){
            $c=imagecolorat($contrast_img,$x,$y);
            if($c==0){
                $word2_black_pixel++;
            }else{
                $word2_white_pixel++;
            }
        }
    }
}
$word2_contrast_ratio=($word2_black_pixel/$word2_white_pixel);
if($word1_contrast_ratio<$selective_contrast_threshold){
    $word1_contrast++;
}
if($word2_contrast_ratio<$selective_contrast_threshold){
    $word2_contrast++;
}
if(($word1_contrast_ratio<$selective_contrast_threshold)||($word2_contrast_ratio<$selective_contrast_threshold)){
    imagedestroy($contrast_img);
    goto a;
}
imagedestroy($source_img);
imagebmp($contrast_img,'contrast.bmp');
# resize image *****
# *****
//imagebmp($sharp_img,"contrast.bmp");
$h2qx_output=shell_exec($h2qx_commandline);
$img=imagecreatefrombmp('resized.bmp');
$height=imagesy($img);
$width=imagesx($img);
# sharpen *****
# *****
$sharp_img=imagecreatetruecolor($width,$height);
imagecopy($sharp_img,$img,0,0,0,$width,$height);
imagedestroy($img);
if($enable_sharpen){
    for($surrounding_pixel_threshold=$surrounding_pixel_threshold_max;
$surrounding_pixel_threshold>$surrounding_pixel_threshold_min;$surrounding_pixel_threshold--){
        for($x=1;$x<$width-1;$x++){
            for($y=1;$y<$height-1;$y++){
                $surrounding_pixel_counter_black=0;
                $surrounding_pixel_counter_white=0;
                $c1=imagecolorat($sharp_img,$x-1,$y-1);
                $c2=imagecolorat($sharp_img,$x-1,$y);
                $c3=imagecolorat($sharp_img,$x-1,$y+1);
                $c4=imagecolorat($sharp_img,$x,$y-1);
                $c5=imagecolorat($sharp_img,$x,$y+1);
                $c6=imagecolorat($sharp_img,$x+1,$y-1);
                $c7=imagecolorat($sharp_img,$x+1,$y);
                $c8=imagecolorat($sharp_img,$x+1,$y+1);
                if($c1==0){
                    $surrounding_pixel_counter_black++;
                }else{
                    $surrounding_pixel_counter_white++;
                }
                if($c2==0){
                    $surrounding_pixel_counter_black++;
                }else{

```

```

        $surrounding_pixel_counter_white++;
    }
    if($c3==0){
        $surrounding_pixel_counter_black++;
    }else{
        $surrounding_pixel_counter_white++;
    }
    if($c4==0){
        $surrounding_pixel_counter_black++;
    }else{
        $surrounding_pixel_counter_white++;
    }
    if($c5==0){
        $surrounding_pixel_counter_black++;
    }else{
        $surrounding_pixel_counter_white++;
    }
    if($c6==0){
        $surrounding_pixel_counter_black++;
    }else{
        $surrounding_pixel_counter_white++;
    }
    if($c7==0){
        $surrounding_pixel_counter_black++;
    }else{
        $surrounding_pixel_counter_white++;
    }
    if($c8==0){
        $surrounding_pixel_counter_black++;
    }
    if($surrounding_pixel_counter_black>=$surrounding_pixel_threshold){
        imagesetpixel($sharp_img,$x,$y,0);
    }
    if($surrounding_pixel_counter_white>=$surrounding_pixel_threshold){
        imagesetpixel($sharp_img,$x,$y,16777215);
    }
}
}
}

# reaglnment *****
# *****
$temp=@imagecreatetruecolor($width,$height) or die('Cannot Initialize new GD image stream');
for($x=0;$x<$width;$x++){
    for($y=0;$y<$height;$y++){
        imagesetpixel($temp,$x,$y,16777215);
    }
}

# calculate top and bottom *****
$top=null;
$bottom=null;
for($x=1;$x<$width-1;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $top[$x]=$y;
            break;
        }
    }
    for($y=$height-1;$y>0;$y--){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $bottom[$x]=$y;
            break;
        }
    }
}

# calculate mid *****
$mid=null;
$mid_1stpass=null;
$mid_2ndpass=null;
for($x=0;$x<$width;$x++){
    if($bottom[$x]-$top[$x]==0){
        $mid_1stpass[$x]=abs($height/2);
    }else{
        $mid_1stpass[$x]=abs((1-$mid_exponential_smoothing_factor)*$mid_1stpass[$x-1]+
$mid_exponential_smoothing_factor*(($top[$x]+$bottom[$x])/2));
    }
}
for($x=$width-1;$x>=0;$x--){
    if($bottom[$x]-$top[$x]==0){
        $mid_2ndpass[$x]=abs($height/2);
    }else{
        $mid_2ndpass[$x]=abs((1-$mid_exponential_smoothing_factor)*$mid_2ndpass[$x+1]+
$mid_exponential_smoothing_factor*(($top[$x]+$bottom[$x])/2));
    }
}
for($x=0;$x<$width;$x++){
    $mid[$x]=(($mid_1stpass[$x]+$mid_2ndpass[$x])/2);
}
}

```

```

$first_word_start=0;
for($x=0;$x<$width;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $first_word_start=$x;
        }
    }
    if($first_word_start>0){
        break;
    }
}
$first_word_end=0;
for($x=$first_word_start;$x<$width;$x++){
    $black_pixel_found=false;
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $black_pixel_found=true;
        }
    }
    if(!$black_pixel_found){
        $white_column_count++;
        if($white_column_count>$white_columns_needed_for_separation){
            $first_word_end=$x-$white_column_count;
        }
    }
    $white_column_count=0;
}
if($first_word_end>0){
    break;
}
}
$second_word_start=0;
for($x=$first_word_end+$white_columns_needed_for_separation;$x<$width;$x++){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $second_word_start=$x;
        }
    }
    if($second_word_start>0){
        break;
    }
}
$second_word_end=0;
for($x=$width-1;$x>$second_word_start;$x--){
    for($y=0;$y<$height;$y++){
        $c=imagecolorat($sharp_img,$x,$y);
        if($c==0){
            $second_word_end=$x;
        }
    }
    if($second_word_end>0){
        break;
    }
}
for($count=0;$count<20;$count++){
    $mid[$first_word_start+$count]=$mid[$first_word_start+20];
    $mid[$first_word_end-$count]=$mid[$first_word_end-20];
    $mid[$second_word_start+$count]=$mid[$second_word_start+20];
    $mid[$second_word_end-$count]=$mid[$second_word_end-20];
}
for($x=0;$x<$width;$x++){
    $vector[$x]=((($height/2)-((1-$vector_threshold)*$mid[$x])));
}
# debug *****
if($debug_mode){
    $debug=imagecreatefrombmp('resized.bmp');
    for($x=0;$x<$width;$x++){
        imagesetpixel_rgb($debug,$x,$mid[$x],255,0,0);
    }
    imagejpeg($debug,"debug.jpg");
}
# remove distortion *****
for($x=0;$x<$width;$x++){
    if($bottom[$x]-$top[$x]>0){
        for($y=floor($mid[$x]);$y<$height;$y++){
            if($y<=$bottom[$x]){
                $c=imagecolorat($sharp_img,$x,$y);
                imagesetpixel($temp,$x,$y+$vector[$x],$c);
            }
        }
        for($y=floor($mid[$x])-1;$y>0;$y--){
            if($y>=$top[$x]){
                $c=imagecolorat($sharp_img,$x,$y);
                imagesetpixel($temp,$x,$y+$vector[$x],$c);
            }
        }
    }
}
}

```

```

}
# crop captcha top and bottom *****
$crop_top=0;
for($y=0;$y<$height;$y++){
    for($x=0;$x<$width;$x++){
        $c=imagecolorat($temp,$x,$y);
        if($c==0){
            $crop_top=$y;
        }
    }
    if($crop_top>0){
        break;
    }
}
$crop_bottom=0;
for($y=$height;$y>0;$y--){
    for($x=0;$x<$width;$x++){
        $c=imagecolorat($temp,$x,$y);
        if($c==0){
            $crop_bottom=$y;
        }
    }
    if($crop_bottom<$height){
        break;
    }
}
$new_height=$crop_bottom-$crop_top;
$new_width=$width;
$cropped_img=imagecreatetruecolor($new_width,$new_height);
imagecopyresampled($cropped_img,$temp,0,0,0,$crop_top,$new_width,$new_height,$width,$new_height);
imagejpeg($cropped_img,'cropped.jpg');
# output to ocr *****
# *****
$tesseract_output=shell_exec($tesseract_commandline);
if($debug_mode){
    echo "<img src=input.jpg><br><img src=contrast.bmp><br><img src=debug.jpg><br>";
    imagedestroy($debug);
}else{
    @imagedestroy($temp);
    @imagedestroy($img);
    @unlink('resized.bmp');
    @unlink('cropped.jpg');
    @unlink('contrast.bmp');
    @unlink('debug.jpg');
}
?>

```