Rodney Beede
CSCI5722 Computer Vision
University of Colorado at Boulder
December 2010

# Analysis of reCAPTCHA effectiveness

## INTRODUCTION

This report provides an analysis of the effectiveness of the reCAPTCHA system and the methods used to defeat it. CAPTCHA stands for "Completely Automated Public Turing test to tell Computers and Humans Apart" and is used in real-world applications as a mechanism for distinguishing whether input is coming from a human or a computer. Its primary purpose is to help prevent, or at least reduce, the amount of SPAM or unsolicited messages in e-mail, online message forums, and online polls (Strickland) by verifying that an actual human is making the submission versus an automated computer. Many spammers (unsolicited message senders) use bots (automated computer software) to send their messages to as many people as possible. By using a CAPTCHA a web site owner hopes to reduce these messages by making the use of bots impossible and requiring the spammer to do many hours of manual entry to get a message through. The system, however, must still allow for actual humans to be able to access the system in an easy way.

A CAPTCHA works by requiring the user to answer a CAPTCHA challenge by typing in some combination of letters, numbers, or words that are displayed in an image. This image, however, is distorted by computer vision algorithms that add random noise to it. Some examples of the types of noise generated are bending the characters, inserting lines through the characters, generating random colored dots in the background, and adding spurious characters such as shadows behind other characters. The end goal is to prevent a computer using OCR algorithms from being able to read the challenge while a human could.

## METHODOLOGY

The methodology used was to collect a large sample of CAPTCHA images and run them against computer software that implemented various character and word recognition algorithms. The CAPTCHA implementation chosen for the sample data was reCAPTCHA (Luis von Ahn, 2008) which is described later. For image recognition a free OCR open source program known as Tesseract OCR (Google) was used as well as an open source PHP program known as antirecaptcha (Wegener, security risks for online services by relying on reCAPTCHA, 2010) both described later. 1,000 samples were collected from Google's reCAPTCHA server (Google) at random and saved as jpeg images. For each image the correct solution was then recorded by hand for later comparison. Every image was processed with Tesseract OCR in an attempt to translate the images into their textual counterparts. The results for each image were recorded and

compared to the expected result to get a percentage of completely correct, partially correct, and incorrect answers. Every image was again processed this time with preprocessing by antirecaptcha and then analyzed with Tesseract OCR. The results were compared against the manually recorded correct answers to get the percentage of response types.

## ANALYSIS OF RECAPTCHA

reCAPTCHA is a technique where the challenge words come from old transcripts, books, and other printed historical materials. The choice of using these older texts arises from the difficulty of OCR algorithms in translating the images into normal text due to the type of paper or font style used. Some types of paper have yellowed texture or faded characters that cannot be easily recognized (Luis von Ahn, 2008). Even without introducing noise into the image regular OCR algorithms have had limited success in translating 100% of these texts. This makes them good candidates for CAPTCHA challenges since it is easier for humans to recognize them than current algorithms. In addition one of the goals of reCAPTCHA is to give additional benefit to a human transcribing the words by helping in the work of converting these old texts into digital format. Thus not only is the CAPTCHA attempting to reduce SPAM it is also providing a useful service in transcribing texts.

reCAPTCHA's methodology works as follows: an image in a distorted form with two words is presented to the user as the challenge. One of the word's solutions is known (the control word) and the other is not known. The unknown word is the one from the historical text that could not be parsed by regular OCR algorithms. The answer given can only be verified for accuracy by the control word therefore the response given to the challenge is only verified for correctness by the control word. If the control word was answered correctly it is assumed a human answered the challenge instead of a computer, and it is assumed that the unknown word was also answered correctly and is thus now transcribed. To handle the case of human entry error for the unknown word multiple humans will be given the same unknown word in a challenge, which may have a different control word, and the results will be compared. After a sufficient number of the same answers the word can then be given high confidence of correct translation.

## ANALYSIS OF TESSERACT OCR

Tesseract OCR is an open source optical character recognition engine that was originally developed at Hewlett Packard (Vincent, 2006) in 1985. It was released as open source in 2005 and is now maintained by Google. The basic technique (Smith) used begins with adaptive thresholding (R. Fisher, 2003) which separates the image pixels into a binary image of foreground and background. Next the connected components of text lines and words are determined. Outlines of the characters are generated with polygonal approximation and outline fragments are used as the features. A dictionary of known words for the specified language is used for training data in a machine learning method of templates and relations. Two passes are

made to process the words: the first with a static classifier, and the second with an adaptive classifier based on words that passed a threshold during the first pass (Language Technologies Unit (Canolfan Bedwyr), Bangor University, 2008). Tesseract does lack page layout analysis, but for recognizing simple images such as a CAPTCHA challenge it is not needed.

## ANALYSIS OF ANTIRECAPTCHA

"antirecaptcha" (Wegener, antirecaptcha version 0.2e, 2010) is a software program by Benjamin Wegener that is designed to break reCAPTCHA challenges by applying transformations to the distorted image so that OCR software can recognize it. The OCR engine that is used to process the modified image is Tesseract, the same version which was used before in the regular OCR pass. The author doesn't expect it to be able to solve 100% of the challenges, but it does show promise in being able to solve many more. Since the reCAPTCHA challenge only requires that the control word be given correctly to pass the challenge it increases the chance of success. "antirecaptch" is composed of the following components:

- index.php – The actual code for antirecaptcha. Minor modifications to accept command line source (reCAPTCHA challenge image) files and output to a results folder were made. In addition the modified images were saved for analysis.
- tesseract.exe – Called by index.php after a modified image is generated to run OCR against the modified image. "antirecatpcha" also specifies that only alphanumeric characters are to be recognized, and the default English dictionary is to be used
- hq2x – A magnification filter for enlarging an image without much quality loss (Stepin) which is used to improve the quality of the image and remove noise from the actual characters themselves

The algorithm works as follows:

1. The source image is read into memory
2. The start of the first word is found by looking for the first black pixel going left to right and top to bottom (column wise search)
3. The end of the first word is found by starting at the next column (on x axis) and scanning the entire column (all of y axis) for black pixels.
    a. If no black pixels are found then the current column is a "empty white space column" and the next column is checked
    b. Once a certain threshold of white columns is reached it is assumed that the end of the word has been found
    c. If a black pixel is found during the search the count of "empty white space columns" is reset to zero
    d. If the end of the first word could not be found (image never had threshold amount of all empty white columns) then the algorithm will fail

4. The bounds of the second word are also found in a similar manner with the start and end corresponding to x axis values
5. For each point on the x axis the top and bottom most pixels of the words in the column are mapped based on a color threshold
6. Take a copy of the source image and a loop is now begun for processing the image
7. Across the columns of the first word (from word start to word end on x axis) we look at all the y axis points for each column (x axis) that have a red component that exceeds a certain threshold for that word.
    a. These would be the pixels inside the top and bottom of the word
    b. If the threshold is met we set that pixel to all white
    c. If not we set it to all black
    d. This creates a contrast of white and black
8. Repeat the same for the second word as above
9. Count the number of black and white pixels for each word
10. For each word if the ratio of white to black pixels meets a certain threshold (1.5) then we go on.
    a. If not adjust the threshold we used in 7 & 8 and go back to step 7 again
    b. "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" (Wegener)
11. Take the newly generated contrast image and magnify it to improve the quality.
    a. h2qx was used by antirecaptcha which uses a pixel art algorithm (Stepin)
    b. Making the image larger allows us to sharpen it later without risking removing too much detail to the inside of characters and between them
12. Sharpen the resized contrast image
13. Calculate the top and bottom of each column (x axis) again for the sharpened image
14. Calculate the middle of the black pixels for each column (x axis)
    a. A smoothing factor of 0.2 is used to reduce bumps in the calculated middle line for the image
15. Every column is moved by (height of image / 2) - center value(x)
    a. This mostly removes the curvy distortion from the image
16. The top and bottom of the image (whitespace) are cropped
17. An OCR engine is then applied to the modified image to digitize the words

Example of the algorithm as it is run:

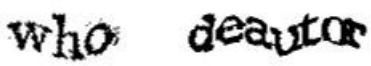

Figure 1: Original reCAPTCHA

Figure 2: During processing (curvature line imposed in red for debug purposes)

Figure 3: Final image cropped and handed to OCR

The algorithm relies on removing the curvature and sharpening the characters with contrast as a means of transforming the image into something that OCR software can correctly parse. Again since the unknown word was difficult for regular OCR algorithms to parse before it was ever distorted the primary goal is to solve the control word in order to successfully pass the CAPTCHA challenge.

## RESULTS

| | Regular OCR (Tesseract) | antirecatpcha |
|---|---|---|
| **Complete Match** | 0.2% | 2% |
| **Partial Match** | 10.7% | 21.6% |
| **No Match** | 89.1% | 77.4% |

Table 1: Out of 1,000 samples. Complete means both words matched, Partial means only one word, No means neither word.

From these results we can see that antirecaptcha's algorithm does increase the amount of images that could be successfully recognized into digital text. A recognition rate of 2% is still useful at slowing a spammer down, but with automated means a large amount of SPAM could still get through. If you assume that all the partial matches got the control word as the correct word then 23.6% of the attempts would succeed which would still result in a large amount of SPAM. It would take more effort for the spammer to get through, but since many spammers use automated bots the only burden would be the initial setup to work through the CAPTCHA challenge.

It is interesting to note that even regular OCR was able to get a complete match or partial match. This would indicate that the amount of noise or obfuscation generated by reCAPTCHA is simply not sufficient to prevent spammers from getting past the challenge via automated means. Jonathan Wilkins in his paper Strong CAPTCHA Guidelines v1.2 (Wilkins, 2009) makes the argument that spammers don't care if they only have a 0.01% success rate since many of the resources they use are not even their own but instead botnets they control. The number of computers available to them makes the attack still very feasible.

Another interesting result was seen in the complete match solutions that regular OCR performed. The two complete matches done without any pre-processing, with just Tesseract, were actually not solved by antirecaptcha. One example is the "adonees from" challenge:


**Figure 4: Original Image**


**Figure 5: antirecaptcha debug image (resized)**

Regular OCR gave the correct solution of "adonees from" while antirecaptcha gave "acmrys from". The antirecaptcha algorithm's attempt at removing the distortion caused some of the characters in the first word to bend further or touch each other (such as the 'a' to 'd') which caused an incorrect result. One possible improvement would be for the algorithm to run the image through OCR first without any pre-processing to see if the words match any trained dictionary words.

The primary failure of reCAPTCHA is that segmentation of the individual characters and words is still easy to achieve. OCR algorithms work best when they can successfully detect where one character ends and the next begins (Wilkins, 2009) or solve the problem of segmentation. Other weaknesses in CAPTCHA challenges as noted by Wilkins are:

- Random noise that doesn't resemble text such as colored backgrounds. These are easily removed by taking out all the color pixels.
- Random lines that are thinner than the text. These can be removed by eroding the image and then dilating it back to restore the characters.

Wilkins recommends that to improve the resistance to automated attacks a CAPTCHA should rotate and warp individual characters and combine them to overlap. This makes the segmentation and curve adjusting more difficult because the characters touch in more random ways which makes whitespace detection harder.

## CONCLUSION

reCAPTCHA does make a spammers job more difficult, but it isn't full proof. One interesting note is that after Wegener released his first version of antirecaptcha the reCAPTCHA

system was modified to block it. reCAPTCHA modified their algorithm for generating the distorted challenge images by changing how the bends were produced. One day later Wegener released an improved version that was able to work around tweaks to the distortion amounts. A few days later reCAPTCHA added in some Vietnamese words to the challenges which antirecaptcha's OCR engine could not easily solve since it lacked trained recognition data for that language. However it would be trivial to train the OCR engine with an appropriate language dictionary. Thus you have a game of cat and mouse where initially reCAPTCHA was far ahead but since OCR and recognition computer vision algorithms have become more advanced it has lost its lead.

Is it worth it to use CAPTCHA to try to protect an online site from SPAM? Initially it was because spammers had no good way to automate attacks against it. However, with better algorithms more spammers are breaking through, and the effectiveness of the system has been reduced. Also the additional complexity for a human end-user to access a site may outweigh the little protection that CAPTCHA provides. Some implementations of CAPTCHA lack ways for those who are disabled, say someone who is blind, to even attempt the challenge. reCAPTCHA does have handicapped assistance through the alternate choice of an audio CAPTCHA (Google), but this also adds another attack vector for spammers which has been shown to be successful (Tam). The cost of a more difficult user experience because of a CAPTCHA challenge and hoping a spammer will skip over your site because of the CAPTCHA will have to be made by the site owner. If a spammer feels your site is worth spamming the CAPTCHA won't stop them from using automated means.

# Works Cited


Google. (n.d.). *Guidelines for reCAPTCHA*. Retrieved Nov 10, 2010, from http://www.google.com/recaptcha/captcha

Google. (n.d.). *reCAPTCHA*. Retrieved Nov 3, 2010, from http://www.google.com/recaptcha

Google. (n.d.). *tesseract-ocr Version 3.00 for Win32*. Retrieved Nov 1, 2010, from http://code.google.com: http://code.google.com/p/tesseract-ocr/

Language Technologies Unit (Canolfan Bedwyr), Bangor University. (2008, Apr). *An overview of the Tesseract OCR (optical character recognition) engine, and its possible enhancement for use in Wales in a pre-competitive research stage.* Retrieved Nov 9, 2010, from saltcymru.org: http://saltcymru.org/english/saltcymru_document5.pdf

Luis von Ahn, B. M. (2008, Sep 12). reCAPTCHA: Human-Based Character Recognition via Web Security Measures. *SCIENCE vol. 321* .

R. Fisher, S. P. (2003). *Adaptive Thresholding*. Retrieved Nov 9, 2010, from http://homepages.inf.ed.ac.uk/rbf/HIPR2/adpthrsh.htm

Smith, R. (n.d.). *Tesseract OCR Engine, What it is, where it came from, where it is going*. Retrieved Nov 9, 2010, from googlecode.com: http://tesseract-ocr.googlecode.com/files/TesseractOSCON.pdf

Stepin, M. (n.d.). *hq2x.* Retrieved Nov 11, 2010, from http://web.archive.org/web/20070624082212/www.hiend3d.com/hq2x.html

Strickland, J. (n.d.). *How CAPTCHA Works*. Retrieved Nov 1, 2010, from howstuffworks.com: http://computer.howstuffworks.com/captcha2.htm

Tam, J. (n.d.). *Breaking Audio CAPTCHAs.* Retrieved Nov 12, 2010, from captcha.net: http://www.captcha.net/Breaking_Audio_CAPTCHAs.pdf

Vincent, L. (2006, Aug). *Announcing Tesseract OCR*. Retrieved Nov 2010, from blogspot.com: http://googlecode.blogspot.com/2006/08/announcing-tesseract-ocr.html

Wegener, B. (2010, Oct 27). *antirecaptcha version 0.2e.* Retrieved Nov 3, 2010, from http://wegeneredv.de/arc/

Wegener, B. (2010, Oct 27). *security risks for online services by relying on reCAPTCHA.* Retrieved Nov 4, 2010, from wegeneredv.de: http://wegeneredv.de/antirecaptcha.pdf

Wilkins, J. (2009, Dec 21). *Strong CAPTCHA Guidelines v1.2.* Retrieved Nov 10, 2010, from bitland.net: http://bitland.net/captcha.pdf