Strong CAPTCHA Guidelines
v1.2

Jonathan Wilkins - <jwilkins[at]bitland[dot]net>

http://www.bitland.net/

December 21, 2009

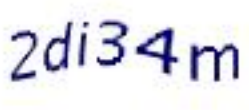
Abstract

An introduction to developing secure CAPTCHA (Completely Automated Public Turing test to tell Computers and Humans Apart)[1] systems. In addition to describing common weaknesses in CAPTCHA puzzles, focus is placed on the system as a whole, including replay detection and attack detection.

1 Introduction

When abuse is detected on a site, CAPTCHA seems to be the knee jerk response to limiting it. Developers have seen the typical warped character type of challenge currently in common usage and jump to implement one of their own quickly.

Here's a sample used by a WiFi Hotspot.



Simply running it through ocropus[2] yields the correct text:

```
jwilkins@silence:~$ hocr=0 ocrocmd hotspot-captcha.jpg
2di34m
```

These easily OCR'ed puzzles usually work for a period of time, depending on what asset is being protected. Most small sites using a commonly attacked message board software could effectively protect themselves with a single hard coded question like 'What color is an orange?'. As long as attackers have no real cause to examine that particular site, the scripts they use to post spam in comments will break and the forum will remain spam free[3].

[1] http://en.wikipedia.org/wiki/Captcha
[2] See a description in appendix A
[3] Though this wouldn't properly be called a CAPTCHA.

It is for everyone else that the following content is written.

There are three major components involved in building a strong CAPTCHA solution. First, the basis for the puzzle or challenge must be something that is truly difficult for computers to solve. Second, the way puzzles and responses are processed must not introduce any flaws. Lastly, the system should have adequate logging so that it is easy to determine when an attack is happening and what the nature of the attack is.

It is also worth realizing that CAPTCHAs are not the solution for every problem. At best, they increase the cost of a given task to that of paying people[4] to solve the puzzles and the overhead to manage this process. However, for completely unauthenticated transactions, they're an important tool and can be extremely effective.

2 The Puzzle

The puzzle must be very difficult for computers to solve and relatively easy for humans. Simple character recognition isn't one of those problems, despite the fact that developers and users are so used to seeing it on the sites they frequent. On those sites that are successfully using warped text, the real problem preventing scripting is segmentation[5].

Software is still not as good as humans at determining where one character ends and the next one begins. The basis for a strong text based CAPTCHA is ensuring that segmentation is hard. In fact, once segmentation is solved, computers are much better at recognizing individual characters than people are[6]. This means that characters should have some overlap and any decoy lines should be the same thickness and texture as the lines used in the letters. They should also run in the same direction as the strokes that compose the letters. Many people use techniques other than overlap and decoy lines to obfuscate challenges. The general approach seems to be to generate random text and apply a grab bag of simple filters or image processing operations. Many of these alterations are quite simple to reverse and only serve to confuse legitimate users.

2.1 Common Weaknesses

2.1.1 Noise that doesn't resemble the text

Many of these operations are easily reversible or have other issues. One more amusing one created a very noisy colorful image where the widely spaced and unwarped letters to be recognized were outlined in black. It also happened that the outlined characters were the only things that were black. Given this it is a trivial task to eliminate all non-black pixels and run it through off the shelf software to have a very high solution rate.

[4] whether directly or through access to resources
[5] http://research.microsoft.com/~kumarc/pubs/chellapilla_hip05.pdf
[6] http://research.microsoft.com/~kumarc/pubs/chellapilla_ceas05.pdf

Ocropus has no trouble with this third version:

```
jwilkins@silence:~$ hocr=0 ocrocmd technicolor.jpg
CFETLV
```

Others employ noise lines that were much thinner than the challenge text. By applying two basic image processing techniques called erode (which thins the edges of all objects) and dilate (which thickens them) software can automatically eliminate these lines[7]. What happens is that the erode eliminates all thin lines and then dilate mostly restores the original thickness of the characters.

Thresholding is another trivial operation which is very effective at removing common types of noise. By setting a given value as a dividing line between black and white (all values below become black, all above become white) most color noise is eliminated.

Ocropus gives us:

```
jwilkins@silence:~$ hocr=0 ocrocmd threshold-after.png
abcdcigh
```

which isn't perfect, but it hasn't been trained on this font. The more important thing is that it had no difficulty with the segmentation.

[7] These are called minimum and maximum in Photoshop

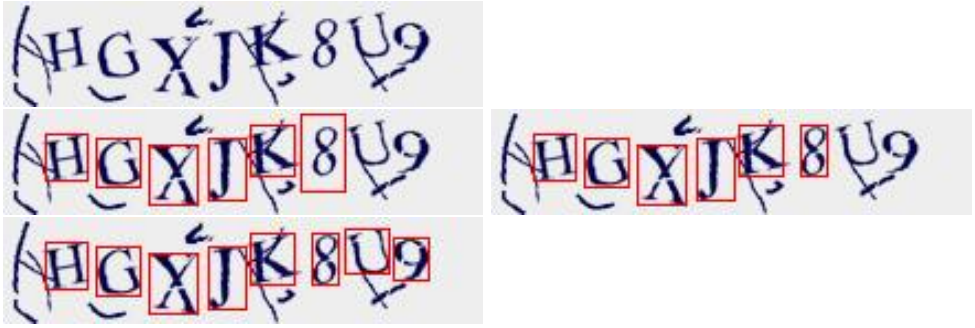
2.1.2 Modifying the Whole Image

Many CAPTCHA puzzles place a random string on a given background and then apply a simple warp to the whole image. Certain types of warp are easy to reverse. For instance, a spiral type of warp can be removed by applying its inverse. If this spiral is often used, an attacker can simply automatically try various values and see which one yields the best result.

```
jwilkins@silence:~$ hocr=0 ocrocmd swirl-reverse.png
ABCD
```

2.1.3 Excess Spacing

Some CAPTCHAs which employ noise lines to make segmentation harder still allow excess spacing between characters. This allows an attacker to perform a rough slicing attack where they look for bounding boxes of an approximate size (the size of the average individual character plus a small fuzz factor for warping) at offsets from the edge of the last bounding box and then reducing where possible. In puzzles that have noise lines that are thinner than the actual characters this can be quite effective, especially with erode and dilate.



Knowing the above, it is possible to eliminate the reversible techniques commonly used to obfuscate a challenge (which pose no burden to automated attacks, only legitimate users) and yield a challenge that is both easier for humans to read and increases difficulty for software.

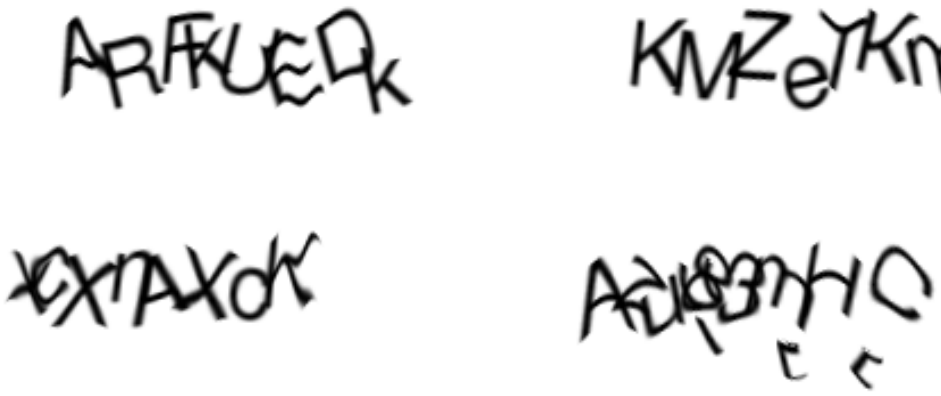
2.2 Designing strong puzzles

2.2.1 Rotation and warping of individual characters

Rotation and warping of individual characters seemed to provide the best resistance to OCR when combined with overlap and this is backed up by prior research[8]. By applying these to each character individually, the attacker is forced to perform segmentation before being able to try inverting the function. Creating each character (and decoy lines) individually on a transparent background, applying warp and then compositing gives the best results.

[8] http://research.microsoft.com/~patrice/PDF/hip9.pdf

The following examples use warped character segments for noise to make feature detection much less effective. For the more difficult examples, it is believed that OCR has such a low solve rate that requiring 7/8 correct characters is acceptable.



2.2.2 Character Set

In order to ensure that the difficulty of OCR is as high as possible, it is important to make sure that there are many possibilities for each character. For instance, if the character set is $[0, 1]$, the recognition task is much simpler for software. Use of a full alphanumeric character set is best though typically some characters are eliminated for usability. Some sites eliminate vowels to avoid offensive terms randomly appearing. Others eliminate characters that are very frequently confused such as the number 0 vs the letter O.

To increase usability it is helpful to provide the user with some guidelines for the puzzles such as saying that the characters are not case sensitive and there are no numbers. This doesn't decrease the difficulty for automated attackers as they will be able to determine these rules through observation.

A recommended character set is included in the appendices.

2.2.3 Font Selection

Font selection can also introduce flaws. Fonts can be broadly divided into serif and sans-serif fonts. Serifs are the small features at the ends of individual characters. Some serif fonts have unique features for certain characters that can make recognition much easier. In the below image, the Q, J and also C are particularly susceptible to this sort of analysis.

For this reason, Sans-serif fonts are recommended.

2.3 Other considerations

2.3.1 Dictionary Words

Use of dictionary words make analysis much easier. When OCR can extract the majority of the characters it is a simple matter to run the result through a spell checking library and choose the most likely suggestion. This is especially true as the words get longer. The word 'they' has many words within a one character edit distance[9] such as 'them', 'the', 'then' and 'whey'. Xenophobic has fewer. This is a feature of most languages, especially English.

2.3.2 Use of a Known or Public Source of Puzzles

Using a publicly available source of puzzles has its hazards. For instance, if the puzzle revolves around answering common sense questions, such as 'What sounds do dogs make?', a system like OpenCyc[10] may be integrated. If a custom set of questions or images are used, the difficulty can be reduced to that of rebuilding the knowledge base. Since this is less difficult[11] for a spammer than for the developers building the original data set, this is not an ideal solution.

2.3.3 Success Rates

It is also critical to remember that an attacker is often perfectly happy with a very low automated solution rate. Even if they are only able to solve 1 in 100 challenges automatically, they are content to just throw resources at the puzzle since most of the resources they are using don't belong to them in the first place. This is due to the low cost of compromising computers and building or renting bot nets. Given this, the attacker doesn't have to rebuild a complete set of solutions, just enough to get this minimal success rate.

For instance, with a 10,000 machine botnet (which would be considered relatively small these days), given broadband connections and multi-threaded attack code, even with only 10 threads per machine, a 0.01% success rate would yield 10 successes every second, which would provide the attacker with 864,000 new accounts per day if they were attacking a registration interface.

2.3.4 Accessible CAPTCHA

Audio CAPTCHAs are often used to make puzzles accessible to vision impaired users. This generally doesn't satisfy relevant accessibility requirements and legislation as users may have multiple impairments. Audio CAPTCHAs are also generally more susceptible to automation as speech recognition is a simpler problem than image segmentation.

Depending on the site's relationship to users and how much information is available, other methods may be much more effective while avoiding the weaknesses of an audio challenge. For instance, automatically phoning the user or sending a text message[12] may be reasonable for protecting certain assets if the user is

[9] Also known as the Levenshtein distance.

[10] http://www.opencyc.org

[11] Since spammers are generally using low cost labor overseas, sometimes employing rooms full of people who are simply typing in CAPTCHA solutions.

[12] Here the system would be relying on the expense and difficulty of replacing a phone number and would, naturally, require a mechanism for disabling accounts and blacklisting the associated phone number if they are later caught abusing the service.

unable to complete a visual CAPTCHA. Here, authentication essentially replaces CAPTCHA.

The goal is to make the effort required by attackers higher than the effort needed for solving the visual CAPTCHA while avoiding exposing the site to a DOS if the spammer were to shift their attention to the alternative system.

2.3.5 Randomness

The usual warnings about randomness apply. Use a strong source of random numbers instead of weaker sources such as the standard C library rand() function. Also important to note is that using modulo on random numbers is usually unsafe. Imagine using a six sided dice to generate numbers between 1 and 3 (1D6 mod 3). This is safe, as there is still a random distribution of numbers, however using that same dice to generate numbers between 1 and 4 (1D6 mod 4) will yield twice as many 1's and 2's as 3's and 4's.

3 Case Study: Breaking reCAPTCHA

3.1 reCAPTCHA Overview

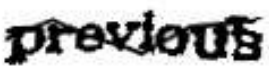
reCAPTCHA takes two words scanned from books that their OCR engine is unable to identify and then adds noise and warping, composites them then displays the resulting image to the user. They use a basic k of n system to figure out whether the submitted answers are correct:

> Each new word that cannot be read correctly by OCR is given to a user in conjunction with another word for which the answer is already known. The user is then asked to read both words. If they solve the one for which the answer is known, the system assumes their answer is correct for the new one. The system then gives the new image to a number of other people to determine, with higher confidence, whether the original answer was correct. [13]

3.2 Issues

The reCAPTCHA system breaks many of the above guidelines. Firstly, they're using English text with a few exceptions such as where a word is broken across two lines (Ele- phant for example). This means that we have a pretty easy way to check whether a given OCR operation has been successful.

They've further weakened the system by allowing off by one errors to be accepted as correct. This saves us in the case that a line traverses a letter in such a way as to make it problematic to guess, 'lone' vs 'tone' for example. Skipped characters are also acceptable, for instance, when the challenge is 'base defined', 'base define' is accepted as is 'bass deined'. An edit distance of 1 for each word is accepted. One can even skip a word entirely, if the word is short.



[13] From: http://recaptcha.net/learnmore.html

For the above challenge 'previous' is an accepted solution.

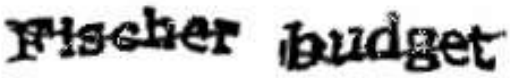
The noise lines are generally close to the same weight as the font but they are generally only horizontal. Certain erode/dilate matrices are very effective at removing only horizontal lines.

3.3 The Attack

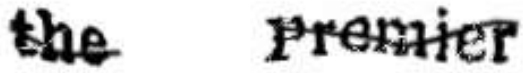
I downloaded and hand solved 200 reCAPTCHA challenges in early 2008. Since the words are widely spaced, it is easy to split the challenge into it's component parts. Word separation was performed using the blobs.rb code in Appendix E. Then code was developed that attempted various erode/dilate matrices on each individual word before OCR'ing[14] the result. The Levenshtein distance between each guess and the correct answer was recorded. I then took the best matrices and used them to apply to new challenges.

The solver code iterates through the above described list of effective matrices and stores each answer. The answer list is then checked against a word list and if the word is known, it is stored with a count of how many times that result was given by the OCR engine. If it is not a known word, then it is run through aspell and the most likely guess is chosen and added. After each matrix has been attempted, the resulting words and their counts are evaluated. The longest word with a non-trivial count is usually correct. If there is no likely candidate, this challenge can be discarded and the next one attempted. Since the reCAPTCHA system is using dictionary words as well as the names of people and places (aside from word fragments like the previously discussed 'phant'), if it doesn't exist in our word lists, it is probably incorrect and it's worth not submitting so as to not draw attention to our submissions.

Here is some sample output from the solver:



```
0087 Win                  fischer (  188)                  budget ( 192)
  Highest count:          fischer (  188)                  budget ( 192)
  Longest word:           pitcher (    5)                  widget ( 2)
  Longest w/ count:       fischer (  188)                  budget ( 192)
```



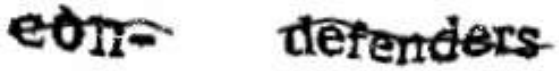
```
0097 Win                    the (  483)                  premier ( 42)
  Highest count:            the (  483)                  premier ( 42)
  Longest word:          mamma (    4)                  premier ( 42)
  Longest w/ count:         the (  483)                  premier ( 42)
```

Two typical successes.

[14] With Ocropus

```
0101 Fail                      25 (    9)              telephoned ()
  Highest count:               as (  660)                      85 ( 33)
  Longest word:            demand (    8)                   momma ( 1)
  Longest w/ count:        demand (    8)                      85 ( 33)
```

Here we have a complete failure. '25' is short and while it did have some hits, it wasn't a solid guess. 'telephoned' had no hits.



```
0102 Half                    con- (     )               defenders ( 1)
  Highest count:               em (  184)                  norman ( 25)
  Longest word:              eqyy (    1)               defenders ( 1)
  Longest w/ count:            em (  184)                  norman ( 25)
```

"con-" is only a word fragment so we don't have any likely solution, though we did get defenders.

3.4 Observations

Interestingly, it wasn't the matrices that removed horizontal lines entirely that were most effective. Training a new OCR engine against these segmented and damaged characters will likely result in an even higher success rate than simply using the Ocropus engine.

Short words were the most problematic. If the solver saw 'it' as 'if' due to noise lines, it was very likely to consistently get it wrong. For this reason, short words were rejected and a new attempt was made instead.

3.5 Success Rates

Running against 200 challenges, this method solved 10 correctly - a success rate of 5 percent. It further got one word correct in 25 other cases. If we presume that in half the cases the failed word would be the unknown word for reCAPTCHA, this gives us a total success rate of 17.5 percent.

Also worth noting, ocropus alone solved 0 of the 200 challenges. When ocropus was provided with the challenge split into single word portions it was able to get 5 single words, a success rate of 1.25 percent.

Some changes were made to the reCAPTCHA system since this analysis was originally performed. However, it appears that the changes that have been made weaken rather than strengthen their system. The major change has been to eliminate the line that ran through the challenge. When ocropus was run against 100 challenges fetched on December 16th, 2009, after they were split into their two halves, it was able to solve one word out of two in 23 cases (23/100). This is much higher than the 5/200 base line solve rate seen

before.

Some analysis was also done against the live service. 40 puzzles were requested and passed through tesseract without doing any of the erode or dilate that was needed against the original data set. The output was passed in as a solution without even the benefit of aspell. Of 40 attempts, two submissions were accepted as correct. This equates to a 5 per cent success rate for simple OCR, which confirms that the new puzzle is in fact weaker. While it remains possible that reCAPTCHA is doing something on the back end to prevent large scale automated solutions, the strength of the puzzle does not appear to be part of the equation.

4 Generation and Processing

The second part of a secure CAPTCHA system revolves around how challenges are generated and processed. Processing should disallow replay of previously submitted puzzles, prevent multiple guesses of the same puzzle and limit the lifetime of all puzzles.

There's a common desire to reduce the cost of a CAPTCHA system by avoiding any database access. CAPTCHA puzzles which place the solution in a cookie after a simple (reversible) encoding are all too common. Since this is sent to the user, it can be unencoded far more easily than attempting OCR.

Further, if a hash is used instead of a simple encoding, it is susceptible to a dictionary attack. For example, if the challenge is only 4 bytes long and consists of lowercase a-z, then an attacker can trivially build a list of hashes and their values, enabling the attacker to simply recognize the hash rather than having to OCR the image. [15]

4.1 Replay

Even if the solution is properly protected (encrypted in a cookie using AES in CBC mode, with a random IV, HMAC and using a key known only by the site and only used for CAPTCHAs), as long as there is nothing keeping track of which puzzles have already been submitted reuse of previously solved puzzles is an easy way to automate the system. In order to avoid replay attacks (where a solved puzzle is used repeatedly) a list of solved challenges must be maintained[16].

When answers are submitted they must be checked against this list and rejected if they have been attempted already. If there is no list, then an attacker only has to solve a single challenge to continually bypass any protection the CAPTCHA is supposed to provide[17].

By not maintaining a centralized list of attempted CAPTCHAs, an attacker gets an immediate multiplier equal to the number of machines used to process responses. This is because any solution will be accepted

[15] While salting the hash or using multiple rounds of a slower algorithm can help, it is better not to put yourself in the position of having to worry about this attack.

[16] Alternatively, a list of outstanding valid challenges may be used.

[17] A timestamp alone may be used to limit the lifetime of a given puzzle, but an attacker can resubmit a solved challenge hundreds of thousands or millions of times in a few minutes.

by a machine and only added to its own internal list while still remaining valid on all other machines. For any large scale site, this multiplier can be considerable.

4.2 Multiple Guesses

Never allow multiple guesses on a given challenge. Many times, character recognition algorithms will be able to reduce the possibilities for a given glyph to two or three possibilities. By allowing multiple guesses, the effectiveness of the system is significantly reduced. Failed attempts should therefore be added to the above list[18].

4.3 Puzzle Lifetime

Puzzles should only be valid for a short period of time since this increases the cost to spammers as they can't solve a large batch of puzzles and use them later. They must have a human online at the time of issuance to solve it. This should be tied to how long it takes a user to fill in the form that the puzzle is on. If it typically takes a user a long time to complete a page, don't place the CAPTCHA on that page. Place it on an interstitial page that is only displayed when the user submits an error free form. Requiring the user to repeatedly solve a CAPTCHA just to find out that their phone number is missing or in the wrong format only increases user frustration.

4.4 Reuse of Puzzles

Puzzles should also never be reused[19]. For instance, if a large set of images of puppies is created and the CAPTCHA revolves around identifying puppies versus cows, all an attacker has to do is identify most of the puppy images (or cow images if there are fewer of those). Then anytime a known image is seen again (even if minor hash breaking noise has been introduced)[20], it can be easily recognized as being more similar to that entry in the solved set than any other.

4.5 Recommended Architecture

Ideally, the system should generate a puzzle, create a unique identifier and store the corresponding solution keyed with that identifier in a resource that is available to all servers processing CAPTCHAs [21] and then send the puzzle to the client with that unique identifier[22] When the user submits a correct solution, it should be marked as solved and then the user's request should be granted. If the answer is incorrect, the puzzle should be marked as invalid and no further attempts to solve that puzzle should be permitted.

[18] For some puzzle types, attempting to solve a new challenge may be less expensive than retrying on a previously failed challenge. If you're sure your puzzle falls into this category, then allowing multiple guesses might be reasonable.

[19] Though randomly generating two puzzles with the same solution is expected, but should happen infrequently if you have a random puzzle with sufficient entropy

[20] If the image is used unmodified, then it is trivial to apply a standard hash algorithm such as MD5. If the image is modified in a trivial way (such as changing the gamma or tweaking random pixels), algorithms like MD5 and SHA-1 will not produce matches and more advanced image comparison algorithms will be needed. These are not yet easily available when compared to MD5, but are not beyond the capabilities of spammers.

[21] Memcached (http://en.wikipedia.org/wiki/Memcached) is one good option. Databases can be used as a last resort.

[22] The answer shouldn't be sent to any client in any form, only a unique, hard to guess identifier such as a randomly generated 128-bit number, which is only used to look up the answer on the back end.

All actions should be logged, allowing for in-depth incident response and attack detection.

All actions should be logged, allowing for in-depth incident response and attack detection.

5 Logging and Incident Response

If there is a sudden surge of new (CAPTCHA gated) activity on your site, it is crucial to be able to tell whether the CAPTCHA system is failing or if the site is simply attracting a large boost of legitimate use. For this reason it is important to log the following:

Correct answer and Submitted answer These are important because it is possible to determine which characters legitimate users are having trouble with as well as identify when attackers are getting close to successfully breaking your CAPTCHA. Humans tend to consistently make small errors, while automated code will frequently make large ones.

Time of puzzle generation and Time of answer submission These intervals shouldn't be either too long or too short. If you're seeing solution times of 10ms, it is a fair bet that your system is broken.

IP address requesting puzzle and IP address submitting answer Consistent discrepancies may indicate sites redisplaying your puzzles to users who unwittingly solve the puzzles on behalf of spammers.[23]

When you suspect an automated attack, making a substantive change to the puzzle such as significantly changing the font or the amount of noise for a short time can tell you whether the attack is code based.

Depending on how you are using CAPTCHA puzzles, the following may be needed

- Puzzle source, for instance Signup vs Message sending
- Difficulty level

One complicating factor is that certain ISPs and companies use a single externally facing IP for many thousands or even tens of thousands of legitimate users. Care needs to be taken when responding to an apparent attack to ensure that you don't block access to legitimate users.

These logs should be monitored as it is possible to detect an attack before it is even successful. When it appears that an attacker is getting close to success or has broken the puzzle, changing the difficulty by increasing overlap of characters or adding more decoy lines or warping the characters more will force an attacker to retrain their system. This can even be done in an automated fashion, perhaps by linking to a complaint system. As users mark more communications from a given interface as spam, the difficulty for that interface can be automatically increased. Increasing the frequency is also often appropriate.

[23] However this also happens when, for example, a user's IP address is reassigned through DHCP during the transaction. Other issues can occur due to proxies.

6 Other Considerations

6.1 Applicability

There are many problems CAPTCHA challenges are suitable for, other than just spam reduction.

Replacing Account Lockout Instead of providing a static account lockout, which enables an attacker to completely deny service to a targeted user, requiring a CAPTCHA to be solved slows attackers while maintaining access for legitimate users.

Limiting Account Registration This is the canonical example for CAPTCHA. Since the service provider has no information on the user (such as a phone number, email address) they have to rely on CAPTCHA to provide some modicum of required effort on the part of unauthenticated and effectively anonymous users.

Limiting Resource Intensive Tasks Instead of resorting to a hard limit on a resource heavy request, providing a CAPTCHA can slow the pace of requests while not denying access. This is especially useful for unauthenticated services.

Preventing Screen Scraping If automated extraction of site contents is unwanted, CAPTCHA puzzles are an effective tool.

6.2 Avoiding challenging users

An important consideration is reducing the frequency of challenges. Doing this allows the remaining challenged operations to be harder. A legitimate user will complain less about a hard challenge if they see it every other week than if they have to answer one every third click.

To this end, identify the operations that spammers are actually targeting and use CAPTCHAs mainly on those. Once a certain number of messages have been exchanged between two users and none considered spam, the likelihood of future messages being spam is lower. Content scanning is useful for detecting spam and malware. If a user is sending many messages, it can be effective to first determine whether the messages are similar to known spam [24] or is too similar to messages already sent by the same user instead of prompting blindly with a CAPTCHA.

[24] Using a distance based hash like Nilsimsa (http://nilsimsa.rubyforge.org)

A Thanks

I'd like to thank Jesse Burns, Brad Hill, Chris Palmer and Jake Appelbaum for their comments on early drafts of this document. Thanks are also due to Adam Back, Josh Benaloh, Kumar Chellapilla, Cem Paya, Patrice Simard and the whole Passport team for assorted conversations in this arena while I was at Microsoft. Thanks also to Luis von Ahn of the reCaptcha project. Post release thanks go to Fyodor and Steve Weis.

Much of the text related to strong CAPTCHA guidelines was written while I was working at iSEC Partners and I would like to thank everyone there.

Sincere thanks also go to the development teams responsible for the various CAPTCHA systems I've reviewed over the years (who shall remain nameless). Conversations with them have clarified much of the guidance offered here.

B Tools

B.1 gocr

GOCR (available at http://jocr.sourceforge.net) is a simple OCR package that requires no training. It is very fast and useful for sanity checking CAPTCHA puzzles, however it is not as accurate as other engines..

B.2 tesseract/ocropus

Tesseract (http://code.google.com/p/tesseract-ocr/) is another OCR engine that is freely available. It requires a training set to be useful and is more complex to configure than gocr but is more accurate.

Google has released ocropus (http://code.google.com/p/ocropus/), which is based on tesseract, and provides a reasonable training set making it the best choice for simple testing of CAPTCHA puzzles.

B.3 gamera

Gamera (http://ldp.library.jhu.edu/projects/gamera/) is the best freely available software for testing CAPTCHA puzzles. It was written as a framework for building domain specific recognizers for unusual data such as music and archaic documents. For this reason, it is especially good at breaking CAPTCHA puzzles. It does require a lot more effort to be useful however.

C Character Set

Using [A-Z][a-z][0-9] as your base and then eliminating characters is recommended. Depending on the usability requirements and chosen font you may have to eliminate more or be able to eliminate fewer. The

following list is weighted to usability and eliminates all numbers.

'l', '1', 'I' These are too similar to one another

'W', 'w' It is easy to mistake with v or vv depending on warping and overlap

'O', '0', 'Q' Too similar, especially with noise

'g', '9' Too similar, especially after displacement

'3', '8' Too similar to B

'4' Too similar to A

'5' Too similar to S

'a', 'Q' Too distinctive in most fonts

'7', 'J', 'f', 'i', 'j', 'v', 't', 'Y', y', 'v' Confusing w/ overlap (Gets lost in M, V, etc)

'h', 'p' Warping can make h look like n and p look like o, especially with overlap

'L' With rotation, can look like V

'r' Confused with n in lots of cases

'd', 'b' Gets warped and easy to confuse with a, o

This still leaves a 1/656 billion chance for random guesses. Image selection challenges (eg. identify the pictures that have a cow in them) are happy with 1/4096 for a 12 image puzzle.

D Demonstration Code

D.1 reCaptcha word separation - blobs.rb

```
# Run jpg2bmp first, has to be TrueColor bmp
require 'rubygems'
require 'RMagick'
require 'camellia'
require 'jwutil'
require 'cam_util'

include Camellia

def get_halves(filename)
  unless image = cam_load_bmp(filename)
    puts "Couldn't load #{filename}"
    return nil, nil
  end

  yuv=image.to_yuv

  # Just use the V (red) plane
  yuv.set_roi(CamROI.new(1,0,0,yuv.width,yuv.height))

  # dilate until we're left with 2 blobs
  blob_count = 100
  until blob_count == 2
    tries = 1
    while true
      yuv.dilate_square3!
      thr=yuv.encode_threshold(150) # threshold and encode
      blobs=thr.labeling!

      blob_count = blobs.nb_blobs
      case blobs.nb_blobs
      when 0
        puts "No blobs detected, can't continue - ERR"
        save_markup(image, blobs, "images/blobs/#{filename[-8..-5]}-#{tries}.bmp")
        return nil, nil
      when 1
        puts "Only 1 blob, too much dilation - ERR"
        save_markup(image, blobs, "images/blobs/#{filename[-8..-5]}-#{tries}.bmp")
        return nil, nil
      when 2
        puts "2 blobs detected - OK"
        break
      else
        puts "#{blobs.nb_blobs} blobs detected, retrying"
        save_markup(image, blobs, "images/blobs/#{filename[-8..-5]}-#{tries}.bmp")
        tries += 1
```

```
      end
    end
  end

  sorted=blobs.sort {|a,b| b.surface<=>a.surface}
  if sorted[0].left < sorted[1].left
    first = sorted[0]
    second = sorted[1]
  else
    first = sorted[1]
    second = sorted[0]
  end
  return first, second
end

if $0 == __FILE__
  make_image_subdirs

  imgparams = open('imginfo.csv', 'w+')

  unless Dir['images/bmp/*.bmp'].length > 0
    puts "Run jpg2bmp.rb first"
  end

  Dir.open('images/bmp').each { |f|
    if f =~ /.bmp/
      if File.exists?("images/parts/#{f[-8..-5]}a.jpg")
        next
      end

      puts "---------\n#{f}"

      one, two = get_halves("images/bmp/#{f}")
      unless one && two
        next
      end
      #puts "  1: x: #{one.left}, y: #{one.top}" \
      #     "     width: #{one.width}, height: #{one.height}"
      #puts "  2: x: #{two.left}, y: #{two.top}" \
      #     "     width: #{two.width}, height: #{two.height}"
      imgparams << "#{f},#{one.width},#{two.width}\n"

      # save each as a separate file
      img1 = Magick::Image::read("images/#{f[0..-4]}jpg").first
      img2 = img1.dup

      img1.crop!(one.left, one.top, one.width, one.height)
      img1.write("images/parts/#{f[-8..-5]}a.jpg")

      img2.crop!(two.left, two.top, two.width, two.height)
      img2.write("images/parts/#{f[-8..-5]}b.jpg")
```

```
    end
  }
end
```