

To The Graduate Council:

I am submitting a thesis written by Sumit Khanna entitled “Breaking the Multi-Colored Box: A Study of CAPTCHA.” I have examined the final copy of this thesis and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science with a major of Computer Science.

Billy Harris
Billy Harris, Chairperson

We have read this thesis and
recommend its acceptance:

Joseph Kizza
Joseph Kizza

Jack Thompson
Jack Thompson

Accepted for the Graduate Council:

Stephanie Bellar
Stephanie Bellar
Interim Dean of the Graduate School

Breaking the Multi Colored Box:
A Study of CAPTCHA

A Thesis
Presented for the
Master of Science Degree
The University of Tennessee, Chattanooga

Sumit Khanna
April 2009

This work is licensed under the Creative Commons Attribution-Noncommercial-Share Alike 3.0 Unported License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-nc-sa/3.0/> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California, 94105, USA.

2009 Sumit Khanna
Some rights reserved.

Acknowledgements

The research presented here could not be completed without the assistance and support of many different people. I would like to thank Susan Reid, Erin Hazard, Chris Coffey and Erin McArthur for their help in copy-editing my work. I would also like to thank Alex Shiels for helping me find resources on Askimet. Thanks go to my committee chairman Billy Harris, as well as my committee members Joseph Kizza and Jack Thompson, for all their assistance and suggestions. Finally, I'd like to thank everyone who has encouraged and supported me during the research and writing of this thesis.

Abstract

Communication is faster than ever. Innovations in low cost network computing have brought an era in which people can effortlessly and instantaneously view and post opinions collaboratively with others across the world. With such an infrastructure of public message boards, chat rooms and instant messaging systems, there is also a large potential for abuse by people wishing to capitalize on such open services by posting unsolicited advertisements.

An entire industry has been constructed around the prevention of unsolicited electronic advertisements (SPAM). This thesis examines various techniques for preventing SPAM, focusing on Completely Automated Public Turing Tests to Tell Computers and Humans Apart (CAPTCHA), a challenge/response technique where an image is displayed with text that is heavily distorted. It also examines the feasibility of breaking CAPTCHA programmatically, alternatives to CAPTCHA based on filtering, improvements to CAPTCHA using photo recognition and avoiding the need for CAPTCHA using naïve approaches.

Table of Contents

Chapter I – Introduction.....	1
1.1 Problem Statement.....	1
1.2 Explanation of the Problems.....	2
1.3 Understanding CAPTCHA.....	3
1.4 Types of CAPTCHA.....	6
1.5 Alternatives to CAPTCHA.....	10
1.6 Ethical Concerns in Using and Breaking CAPTCHA.....	13
Chapter 2 – Literature Review.....	17
2.1 Chellapill and Simard.....	18
2.2 Mori and Malik.....	19
2.3 Hocevar.....	20
2.4 Atwood.....	22
2.5 Conclusions.....	24
Chapter 3 – Research and Experiments.....	25
3.1 BMCB Engine.....	25
3.2 FlickMeCaptcha.....	49
3.3 Non-CAPTCHA based SPAM Defense.....	51
Chapter 4 – Analysis.....	54
4.1 Degree of Success for the BMCB Engine.....	54
4.2 Pros and Cons of FlickMeCaptcha.....	56
4.3 Filtering versus CAPTCHA.....	57

Chapter 5 – Contributions to the Field.....	58
5.1 BMCB Engine.....	58
5.2 FlickMeCaptcha.....	59
5.3 Conclusion	60
5.4 Future Work.....	61
References and Citations	64
Formatting Conventions	69
Appendix A – Installing the Engine	71
Requirements and Dependencies	71
Downloading the Source Files.....	71
Unpacking the Source Files	72
Compiling the Engine	72
Setting up the Database.....	72
The Configuration Files	73
Running the Program	74
Appendix B – High Level Overview of the Engine.....	76
Appendix C – Developing with the Engine	80
Utility Classes.....	80
Generators.....	82
Image Filters	85
Segmentators.....	86
Analyzers	87
Workflows	88

Entry Point	90
Visual Debugging Tools	90
Appendix D – FlickMeCaptcha	91
Installation	91
Integration.....	91

List of Figures

Figure 1 - Example of a CAPTCHA.....	4
Figure 2 - reCAPTCHA Example.....	8
Figure 3 - KittenAuth Photo-Based CAPTCHA.....	9
Figure 4 - Seemingly Legitimate Text.....	11
Figure 5 - Hidden SPAM.....	11
Figure 6 - Chart outlining several types of easily broken CAPTCHA.....	21
Figure 7 - Chart outlining several types of hard CAPTCHA.....	22
Figure 8 - Types of CAPTCHA.....	28
Figure 9 - Original CAPTCHA.....	29
Figure 10 - CAPTCHA with Modulate Filter 110,100 Applied.....	29
Figure 11 - Original CAPTCHA.....	29
Figure 12 - Noise Reduction Filter Applied.....	30
Figure 13 - OCRAD Analyzing a Trivial Image.....	31
Figure 14 - Graph of Trivial Experiment Results from Guessing.....	36
Figure 15 - Graph of Raw GOCR Test.....	37
Figure 16 - Graph of Raw OCRAD Test.....	38
Figure 17 - Graph of GOCR Analysis of 110 Brightness Adjusted Image.....	39
Figure 18 - Graph of OCRAD Analysis of 110 Brightness Adjusted Image.....	40
Figure 19 - Graph of GOCR Analysis of 180 Brightness Adjusted Image.....	41
Figure 20 - Graph of OCRAD Analysis of 180 Brightness Adjusted Image.....	42
Figure 21 - Graph of GOCR Analysis of 140 Brightness, 5 Contrast Adjusted Image....	43
Figure 22 - Graph of OCRAD Analysis of 140 Brightness, 5 Contrast Adjusted Image.	44

Figure 23 - Graph of GOCR Analysis of 140 Brightness Adjusted Image	45
Figure 24 - Graph of OCRAD Analysis of 160 Brightness Adjusted Image	46
Figure 25 - Graph of GOCR Analysis of Noise Reduced Image.....	47
Figure 26 - Graph of OCRAD Analysis of Noise Reduced Image.....	48
Figure 27 - FlickMeCaptcha	50
Figure 28 - High Level Overview of BMCB Engine.....	77
Figure 29 - Diagram of Utility Classes	81
Figure 30 - Modification to Gotcha to Take Challenge Input	83
Figure 31 - Modification of Freecap to Output Challenge to a File	84
Figure 32 - Modification of Gotcha for Font Path.....	84
Figure 33 - Class Diagram for Filters	85
Figure 34 - Copying One Buffered Image to Another	86
Figure 35 - Class Diagram for Segmentators.....	87
Figure 36 - Class Diagram for Analyzers	88
Figure 37 - Class Diagram for Workflows	89
Figure 38 - Visual Debugger	90
Figure 39 - Adding FlickMeCaptcha to a Form	92
Figure 40 - Verifying FlickMeCaptcha Challenge	92

List of Tables

Table 1 - Trivial Experiment Results from Guessing	36
Table 2 - Results from Raw GOCR Test	37
Table 3 - Results from Raw OCRAD Test	38
Table 4 - Results from GOCR Analysis of 110 Brightness Adjusted Image	39
Table 5 - Results from OCRAD Analysis of 110 Brightness Adjusted Image.....	40
Table 6 - Results from GOCR Analysis of 180 Brightness Adjusted Image	41
Table 7 - Results From OCRAD Analysis of 180 Brightness Adjusted Image.....	42
Table 8 - Results From GOCR Analysis of 140 Brightness, 5 Contrast Adjusted Image	43
Table 9 - Results from OCRAD Analysis of 140 Brightness, 5 Contrast Adjusted Image	44
Table 10 - Results from GOCR Analysis of 140 Brightness Adjusted Image	45
Table 11 - Results from OCRAD Analysis of 160 Brightness Adjusted Image.....	46
Table 12 - Results from GOCR Analysis of Noise Reduced Image.....	47
Table 13 - Results from OCRAD Analysis of Noise Reduced Image.....	48

Chapter I – Introduction

People who frequently use the Internet for communication, social networking and purchases often come across web pages that request that they type the value of an image with distorted text into an input box. The purpose of such requests is to prove that the requester is in fact human and not an automated computer program. This challenge response test is what is known as a Completely Automated Public Turing test to tell Computers and Humans Apart (CAPTCHA). The term was originally coined by several developers at Carnegie Mellon University, and the school's Computer Science Department currently holds the trademark on the acronym [1].

There is a directed effort by commercial interests to break CAPTCHAs, that is to create computer programs to solve the puzzles in order to post unsolicited messages and advertisements. Many such attempts are similar to the nature of attempting to get messages past filters for unsolicited e-mail (also known as SPAM). There are also noncommercial interests in breaking CAPTCHA, either to improve the CAPTCHA itself, force developers to find alternatives by showing CAPTCHA is broken, or general curiosity in bettering the field of Computer Science.

1.1 Problem Statement

My research focuses on three major problems:

1. Is it possible to break common distorted text based CAPTCHA, i.e. recognize and answer challenges programmatically, using currently

- available open source tools for filtering and character recognition?
2. How can CAPTCHA be improved? Are there other challenges that can be created that are easier for humans to answer and more challenging for computers to respond to programmatically?
 3. How effective is CAPTCHA against combating SPAM compared to other prevention techniques?

1.2 Explanation of the Problems

Unsolicited bulk e-mail, advertisements posted to public forms, blogs and bulletin boards and other forms of SPAM are a major problem on the Internet for several reasons. The cost of sending unsolicited advertisements is relatively cheap but consumes large amounts of bandwidth causing cost to be shifted to regular consumers. A majority of SPAM is also fraud and can cause uninformed end-users to lose a considerable amount of money [2]. Because of these facts, multimillion dollar industries arose simply to identify and combat SPAM, which increases costs to service providers, the cost of entry for legitimate businesses and overall costs for end users.

Public forums, bulletin boards and blogs are particularly susceptible to SPAM posting because they are designed to facilitate a high degree of open interaction and discussion with either no verification or simple registration. CAPTCHA becomes important for these services to prevent automated scripts from flooding public discussion areas. Such postings can make legitimate websites completely unusable. However, CAPTCHA does make it more difficult for visually impaired users to participate in such

discussions. My research into attempting to break CAPTCHA is intended to answer the question: is CAPTCHA still a viable form of protection against SPAM?

Regardless of the solution to the first problem, my research also examines alternatives to the traditional distorted text based CAPTCHA and attempts to improve on such techniques and provide innovative approaches. To these ends, I have developed an application that can be integrated into existing websites and that provides a new form of photograph based CAPTCHA utilizing a vast library of user contributed photographs and metadata on those photographs to generate challenges.

CAPTCHA challenges can potentially prevent SPAM but have a considerable number of drawbacks, primarily the inability for the visually impaired to answer most challenges. Many websites and content management systems have attempted to use traditional SPAM filtering techniques such as those used to prevent unsolicited e-mail. The final problem my research addresses is the effectiveness of non-CAPTCHA techniques for preventing SPAM versus CAPTCHA challenges.

1.3 Understanding CAPTCHA

The goals of CAPTCHA are to eliminate automated robots and scripts from using a website as a means of spreading unsolicited advertisements, inflating rankings in search engines and distributing viruses. Although an individual could still accomplish these tasks, using an automated program makes the distribution of SPAM much faster, causes the damage to be more widespread and makes the results considerably more difficult to clean up.

At one time a simple image with slightly distorted text may have been enough to confuse most web robots and allow web designers to validate human users; however, the presence of such text has led programmers to create even more sophisticated robots capable of using Optical Character Recognition (OCR) algorithms —the same types of algorithms used by scanner software to convert scanned documents into text— to recognize the text within the images.

Such innovations have led to more complicated CAPTCHA, which involves multi-colored distorted text on altered backgrounds that contain added lines, noise and possible faded and rotated characters that are not part of the CAPTCHA itself [Figure 1] [3].



Figure 1 - Example of a CAPTCHA

One of the foundations of security in the field of Computer Science is the ability to create a process or algorithm that is very easy to do computationally but is very difficult to undo. This foundation is used in asymmetric or public/private key encryption, which utilizes keys based on the products of two large prime numbers: something that is very easy to do but difficult to undo due to the complexity of factoring products of prime numbers. This same concept applies to the idea of CAPTCHA, although in a slightly

different context.

With CAPTCHA, a computer program must test the users to see if they are human. By doing so, the program needs to generate a test, to which it knows the answer, but which cannot be solved programmatically. There is an odd paradox here where the program generates a test and grades it for correctness; a test that the program itself cannot pass [4].

CAPTCHA is considered an example of a Reverse Turing Test. In a traditional Turing Test, software developers attempt to generate a program that can simulate written human communication. Typically a user will attempt to communicate with the Turing Machine over a text message system to determine if he or she is talking to a real person or a computer on the remote end. Although there is much debate about whether it is theoretically possible to create a true Turing Test or Reverse Turing Test [5], the concept itself does lend itself to many limitations, technical problems and ethical boundaries when dealing with real people.

Simply put, CAPTCHA works because, even with the advancement and innovations in computing technology over the past several decades, there are still tasks that can be accomplished faster and more easily by humans than they can by computers; specifically simple puzzles that involve images, natural language processing or a combination of the two.

Current advancements in image recognition, shape recognition, artificial intelligence and machine learning may tip that scale back in favor of computer algorithms in solving CAPTCHA challenges. Therefore, programmers and security experts must be

diligent in finding new techniques to correctly identify humans, prevent SPAM and maintain security in website models.

1.4 Types of CAPTCHA

The most prevalent form of CAPTCHA is an image with distorted text, although there are many others. A CAPTCHA needs to be able to automatically determine if the end user is human or a program. Therefore, any test that is easy for a human to solve yet difficult to write an automated program for can be considered a CAPTCHA. Recent advancements have led to CAPTCHAs based on pictures, word puzzles, spoken audio and other challenges, each with their own strengths and weaknesses.

1.4.1 Word Puzzles

One such technique is implemented as a plugin for the commercial bulletin board software, vBulletin. "NoSpam! - an alternative to CAPTCHA images" is a plugin designed by a programmer who goes by the handle Antialiasis [6]. The plugin allows a board administrator to define a set of questions and answers. The questions can be simple (e.g. "What is $2 + 2$?") or technical questions related to the forum. The author also suggests embedding an image and asking the question about the image itself.

The advantages to such an approach include accessibility to the visually impaired as well as providing a less cumbersome mechanism of distorted text which sometimes takes users several tries to decipher correctly. For this technique to be effective, there needs to be a considerably large number of questions so that a programmer simply does not farm the website for all the possible challenge responses. The questions also need to

be simple enough to be quickly and easily answered. A similar tool for Wordpress, WP-Gatekeeper, offers challenge questions such as "How do you spell the color blue?" [7] However, questions such as these could eventually be circumvented by a sophisticated natural language processor.

1.4.2 Sound Based

Some CAPTCHAs provide a sound file alternative for uses that are visually impaired. This allows the user to listen to an audio clip, typically one that is heavily distorted, as a means to identify the text in a visual CAPTCHA. Although a sound only alternative is a possibility, such an implementation would be inaccessible to those who have hearing impairments, users who are at computers without sound cards such as those in libraries or users who are in noisy environments such as coffee shops or public wireless locations.

Sound CAPTCHAs also run into the same limitations as picture based CAPTCHAs as they require large numbers of voice recordings to be effective. One solution is automatically generated sounds using voice-synthesizing software; however, such sound based challenges could be circumvented using voice recognition software. Many audio challenges also add in background noise and various other voices chattering. Although this addition makes it more difficult for voice recognition programs to extract the correct response, it can also make it difficult for humans to understand what the correct response should be.

The current official version of CAPTCHA created by the term's trademark holder, Carnegie Mellon University, named reCAPTCHA, has support for an auditory

CAPTCHA for users who are visually impaired. In addition, reCAPTCHA is more useful than just a SPAM prevention mechanism. It actually helps facilitate digitizing books into an electronic form [Figure 2].



Figure 2 - reCAPTCHA Example

It works by providing two words, one which is known and the other taken from a book digitalization project that an optical character recognition (OCR) program could not correctly identify with confidence. If the CAPTCHA is validated correctly with the known word, the user submitted value of the unknown word is stored. The same unknown word is presented to multiple people to gain a total confidence score on what the word actually is [8].

1.4.3 Photograph Identification

One means of determining if an individual is human is by using a matrix of photographs. A challenge is presented where the user is asked to select a set of photos which have something in common with one another. An example is the KittenAuth

project by Oli Warner. Using KittenAuth, a user is presented with a series of nine images [Figure 3]. The user must pick out the three which are kittens in order to prove he or she is human [9].

The advantage to such a challenge is that for people who are visually impaired, it may be easier to recognize photographs than it is to read distorted text challenges. The disadvantage is that a massive repository of both kitten and non-kitten related photos would be necessary for such a system to be practical against SPAM prevention. If the program contained only a few hundred photos, given enough time, an attacker could manually identify many of the kittens and then proceeded to using image comparison techniques to break the challenge and send automated requests.



Figure 3 - KittenAuth Photo-Based CAPTCHA

Part of my research deals with this specific type of CAPTCHA challenge and improves it to be more robust and less vulnerable to attack by utilizing a larger public repository of images. The result is an application called FlickMeCaptcha, which interacts

with the popular photo sharing website Flickr and is covered in more detail in Chapters 3 and 4.

1.5 Alternatives to CAPTCHA

1.5.1 Bayesian Networks

There are solutions that simply test the contents of the message body itself rather than add an additional CAPTCHA test, similar to how e-mail SPAM filters work. Early spam filters for e-mail used developer defined rules such as searching e-mail for specific types of websites or words and phrases. As spammers became adept at circumventing sets of known rules and the rules themselves grew to enormous sizes, more dynamic approaches based on machine learning came into play.

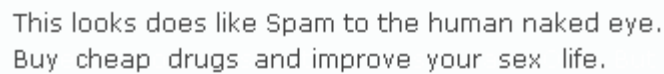
Modern SPAM filters are based on the concept of a naïve Bayes classifier, also known as a Bayesian Network. In 1998, Sahami and others trained the first of such filters with promising results [10]. Today, the same Bayesian filtering has been incorporated into many end user applications and SPAM filters such as Mozilla Thunderbird and the free project SpamAssassin [11].

Bayesian Networks work off a probabilistic graph model. Under the surface they are standard acyclic directed graphs that can be trained for classification using machine learning techniques. Given a certain threshold they can be trained to return, within the tolerance of a given percentage, items that are likely to be SPAM.

Bayesian Networks are not the perfect solution to preventing SPAM since they still have several problems when used with websites. E-mail based filters have the

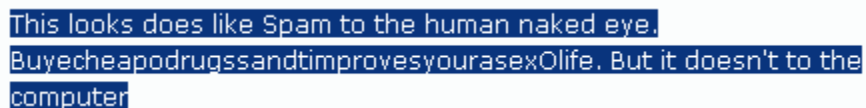
advantage of also being able to perform checks based on e-mail headers, reverse domain name service (DNS) lookups and other techniques in addition to the Bayesian analysis to derive a total confidence score. On a website where user input is available, many of these options are not available and the filter must determine the legitimacy of the posted item based solely on the text entered.

Another common problem is Bayesian Filtering Poisoning. Using this technique, a spammer incorporates several legitimate words together with the SPAM so the filter either lets the message through or incorrectly marks legitimate e-mails as SPAM leading to false positives within the SPAM filter. One particularly tricky example involves using text that is the same color as the background [Figure 4].



This looks does like Spam to the human naked eye.
Buy cheap drugs and improve your sex life.

Figure 4 - Seemingly Legitimate Text



This looks does like Spam to the human naked eye.
BuycheapdrugssandtimprovesyourasexOlife. But it doesn't to the
computer

Figure 5 - Hidden SPAM

Using changes in text and background color, an e-mail that seems like an unsolicited advertisement to the user may pass through a SPAM filter. Highlighting the text reveals the characters that are used to run the words together [Figure 5] [12].

Utilizing a Bayesian Network based filter on user submitted input could be used as an alternative to CAPTCHA. The advantages include not bothering the user with the CAPTCHA puzzle and not taking into consideration users who may be visually impaired and unable to read the challenge. However, there is the possibility that the filter could identify false positives and incorrectly label legitimate content as SPAM.

In my research I examined several non-CAPTCHA based alternatives that examine the contents of messages to determine if they are SPAM. Such alternatives include Akismet [13], Mollom [14] and Defensio [15]. These alternatives are implemented as web based services that can be accessed through a publicly available interface. All of them are free for non-commercial use with optional licenses for businesses and large websites. Each also has plugins for popular content management systems such as Wordpress, Drupal and Movable Type. They most likely implement a series of techniques such as Bayesian Networks and blacklists although their exact techniques are kept secret to prevent attackers from being able to circumvent the systems.

1.5.2 The Naïve Approach

There is a less sophisticated approach to preventing SPAM based on the assumption that automated scripts written to post SPAM are not designed to be very intelligent. In other words, the programmers that create SPAM posting scripts design them to post as many websites as possible without paying much attention to any other content on the page. An example of a naïve approach is placing an input field on a website and then changing its display property in the style sheet to make it invisible. A simple automated program would most likely not be designed to download and parse the

style sheet and would fill out the form field that could not be seen in a real web browser, clearly indicating the post is SPAM.

In my research I examine naïve approaches used on websites. Naïve techniques do not provide a high degree of SPAM prevention and can easily be circumvented, however the cost and effort for identifying and circumventing these implementations is often not economical for posting SPAM to small websites.

1.6 Ethical Concerns in Using and Breaking CAPTCHA

Although added visual distortions may make it more difficult for a program to efficiently identify the characters, these distortions pose a second challenge of also making it more difficult for humans to identify the characters as well, especially since one in twelve people in the United States have some form of color blindness [16]. Being color deficient in one or more major color group can make it difficult if not impossible to identify characters in CAPTCHA.

There are many ethical dilemmas found both in using CAPTCHA and in trying to circumvent or break CAPTCHA. Issues of legality, security and access for those with visual impairments are just a few of the many issues surrounding both CAPTCHA and various other SPAM prevention technologies. Some companies have issued cease and desist orders against developers and companies who create software to circumvent CAPTCHA while others have taken software developers to court. The issues that arise begin to congeal over the very idea of creating illegal software. The simple act of developing software brings up questions of liability when it comes to damages, whether

real or virtual, when faced with the concept types of software that are defined as illegal to develop.

One of the more recent and predominant cases involving software that circumvents CAPTCHA is in the case of TicketMaster vs. RMG. TicketMaster found that certain ticket brokers were purchasing large numbers of tickets in very small amounts of time. Some of these brokers used a service from RMG called ticketbrokertools.com which was available only to RMG clients. Through cooperation with brokers, TicketMaster found that RMG's PurchaseMaster software actually made a slew of automated requests to TicketMaster's website. RMG had developed software to break the CAPTCHA used by TicketMaster and funneled the request through their client's PCs to make the request look like they were coming from several sources [17].

Another questionable ethical practice involves employing people to solve massive amounts of CAPTCHA problems in a farming type situation. Technically a real human would be solving the problems; however, it would be for the purposes of posting SPAM or launching some type of attack. There are widespread, although unsubstantiated, reports of such farming [18], however the feasibility of such a concept comes into serious question. Jeff Atwood of the website Coding Horror puts the feasibility of such an operation into perspective in the follow blog article:

“Let's say spammers set up a sweatshop to employ people to look at computer screens and answer CAPTCHA challenges. They get to send one message for each challenge passed. Assuming 10 seconds per challenge,

and paying roughly \$5 per hour, that represents \$14 per thousand messages [sic]. A typical spam run of 1 million messages per day would cost \$14,000 per day and require 116 people working 24/7.

This would break the economic model used by most current spammers. A recent Wired article showed one spammer earning \$10 for each successful sale. At that rate, the cost of \$14,000 for 1,000,000 spam emails requires a 1 in 1000 success rate just to break even, whereas current spammers are managing a 1 in 100,000 or even 1 in 1,000,000 success rate [19].”

A more viable model would be to get real people to solve CAPTCHA challenges either for free or in exchange for something with insignificant cost. According to Computer World Magazine, such a technique was implemented by one group using a virtual stripper [20]. The animated image of the stripper would gradually remove clothing as users enter in solutions to CAPTCHAs. Each CAPTCHA was actually taken from Yahoo’s e-mail service and the solutions were used to generate a collection of accounts to use for spamming purposes.

There are many ethical considerations surrounding CAPTCHA. As far as their use, the primary concern is accessibility for those who are impaired. Improvement in the use of audio and word puzzle substitutes has greatly reduced this concern in newer implementations.

As far as solving CAPTCHA programmatically, issues have been raised as to the legality of breaking CAPTCHA for the purpose of sales. The Ticketmaster vs RMG

shows that the courts hold that development of software specifically for the purpose of violating TicketMaster's terms of service is illegal, however, this raises further concerns about the ethics of creating software to break CAPTCHA for research purposes, in an effort to find either better CAPTCHA or to enhance the field of artificial intelligence as a whole.

Chapter 2 – Literature Review

Applications designed to programmatically break CAPTCHA do not need to have a high degree of accuracy. Even if a program can only get ten percent of the image challenges correct, it could access and submit to a page several hundred times a minute negating the inaccuracy. Typically the more SPAM prone websites will also add monitors that can detect several connection attempts by a single client in a short interval and will ban such connections. In this sense software that prevents Denial of Service attacks has the side-effect of also helping deflect CAPTCHA breaking attacks. This forces spammers to find ways of getting other clients to make such attempts, either by means of spreading viruses, using unsecured open proxy servers or coaxing individuals with offers of free services or money.

There has been considerable research put forth into creating algorithms that can identify and successfully answer CAPTCHA challenges. There are several challenges facing image analysis. Each of the papers I examined dealt with issues of segmentation, that is separating individual letters in a CAPTCHA challenge, and shape recognition, that is identifying individual characters or glyphs.

Before beginning my own experiments and writing my own applications, I studied existing academic research as well as individual blogs and user experience. I read the works of Chellapilla and Simard who authored the paper *Using Machine Learning to Break Visual Human Interaction Proofs* [21], Mori and Malik who wrote *Breaking a Visual CAPTCHA* [22], Hocevar who created a program called PWNtcha [24] that attempted to circumvent several common forms of CAPTCHA, and Jeff Atwood of the

blog Coding Horror [19] who comments on the state of CAPTCHA breaking as well as using a naïve approach to prevent SPAM.

2.1 Chellapill and Simard

In the paper *Using Machine Learning to Break Visual Human Interaction Proofs*, Chellapilla and Simard [21], two software engineers from Microsoft, examine breaking hard CAPTCHA using a combination of recognition, machine learning algorithms and segmentation techniques. During the course of their research, they determine that most simple CAPTCHAs, which they referred to as Human Interaction Proofs or HIPs, were simple recognition problems while the harder ones required significantly more complex segmentation algorithms.

For simple CAPTCHAs such as Milblocks, Chellapilla and Simard were able to achieve an end-to-end segmentation success rate of 88.8% with a 95.9% recognition for those correctly segmented. Similarly, the Register CAPTCHA had a 95.4% segmentation success rate with an 87.1% recognition rate of successful segmentation. Harder CAPTCHAs to segment, such as Ticketmaster's, which uses diagonal intersecting lines, yielded a segmentation success rate of only 16.6%. Of those correctly segmented, the recognition rate was 82.3%. Their conclusions showed that once segmentation can be broken, the remaining recognition problem can be solved easily with a machine learning algorithm.

Chellapilla and Simard pose the question: What makes segmenting characters in CAPTCHA difficult? Their analysis shows that segmentation is very computationally

expensive requiring examination of many different patterns to locate candidates. The segmentation functions are also very complex because they must identify patterns over the set of all possible valid and invalid patterns, which is substantially more difficult than traditional classification problems. Finally, identifying symbols over a set of valid and invalid candidates is a combinatorial problem, which can very quickly explode into a high order problem size. For example correctly identifying 10 characters among 20 candidates has a 1 in 184,756 (20 choose 10) chance in succeeding by random guessing [21].

Unlike Chellapilla and Simard, in my own research I do not attempt to use a machine learning algorithm to analyze CAPTCHA challenges. Instead I use freely available optical character recognition (OCR) software combined with image filtering over a large set of challenges. The software I have developed is modularized to accommodate a variety of different filtering techniques. Using an object orientated approach, the analysis tool can be easily expanded to accommodate different filters and analyzers and then perform experiments using different combinations of filters and analyzers to gather results.

The software I designed attempts to test several different CAPTCHA scripts with a set of analysis techniques. Although it gathers data on the amount of both correct letters and correct words, it does not have a means of gathering data on correct segmentation.

2.2 Mori and Malik

One of the more famous examples of defeating CAPTCHA is documented in the

paper *Breaking a Visual CAPTCHA* by Greg Mori and Jitendra Malik from the University of California Berkeley and Simon Fraser University, respectively [22]. They took on the challenge of breaking Gimpy and EZ-Gimpy, the Yahoo CAPTCHA systems. Using shape recognition techniques, the one word EZ-Gimpy CAPTCHA could be broken 92% of the time, while the more difficult two overlaid word Gimpy CAPTCHA could still be broken 33% of the time.

The technique employed by Mori and Malik involves three very basic steps at its highest level. First, each individual shape is identified and a list of possible letters assigned to it. Second, a set is composed by linking every possible combination of the letters. Third, the set is compared to a dictionary to find the actual word the image is displaying [23]. This technique has obvious limits as the EZ-Gimpy system uses actual dictionary words and not random letters. Although this makes the system easier for a human to use, it also makes it significantly easier to automatically decipher.

My own research uses CAPTCHA scripts which generate random characters instead of dictionary words. Because of this my engine does not attempt to try to match potential choices with a dictionary; but attempts to analyze each image individually and gathers data on the percentage of correct letters and correct challenges.

2.3 Hocevar

Sam Hocevar, a developer, has worked diligently on his project "PWNtcha" which stands for "Pretend We're Not a Turing Computer but a Human Antagonist." Hocevar has discovered poorly designed generation techniques in many of the common

forms of CAPTCHA used in a variety of bulletin board and blogging software which allows them to be easily deciphered [Figure 6] [24].



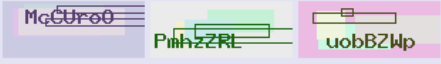
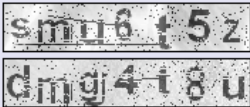


Origin	Samples	PWNtcha efficiency	Comments
Authimage		100%	Vendor site: http://www.gudlyf.com/index.php?p=376 Weaknesses: constant font, aligned glyphs, constant glyph position, constant rotation, no deformation, non-textured background, constant colours, no perturbation.
Clubic		100%	Weaknesses: constant font, no rotation, no deformation, aligned glyph, constant background, weak colour variation, weak perturbation.
linuxfr.org		100%	Weaknesses: constant font, aligned glyphs, no rotation, no deformation, non-textured background, weak colour variation, weak perturbation.
LiveJournal		99%	Weaknesses: constant font, constant character position.
lmt.lv		98%	Weaknesses: constant font, almost aligned glyphs, no rotation, no deformation, constant background, no colour variation, weak perturbation.
Ourcolony		100%	Weaknesses: constant font, no rotation, no deformation, no colour variation, no perturbation

Figure 6 - Chart outlining several types of easily broken CAPTCHA

Hocevar claims that his program is more of a toolkit for image filtering and manipulation than a general purpose decoder. He can not feed any CAPTCHA to it, but must custom tailor it to the type of CAPTCHA presented to it. His website provides examples of significantly harder CAPTCHAs that he still cannot break and is unsure if he will ever be able to [Figure 7].

Authimage (3)		A very good captcha, but not always human-solvable.
CFXCaptcha		A very good captcha.

Figure 7 - Chart outlining several types of hard CAPTCHA

Sam Hocevar originally published his results from the PWNtcha project in 2004, however he offered no source code. Unlike the aforementioned studies, he also did not publish detailed methodology on how he accomplished his results stating ethical reasons. Because of this, as opposed to more traditional studies, Hocevar's results were not reproducible. In 2008, he did release the source code to PWNtcha publicly, stating that the algorithms he used were for outdated CAPTCHAs that were no longer in use.

2.4 Atwood

Jeff Atwood, writer for the blog Coding Horror, described how his blog uses a "naïve CAPTCHA" meaning the CAPTCHA does not change. It uses the same challenge presented continually, yet the author claimed he has received fewer than ten SPAM messages and that the naïve approach was 99.9% effective against stopping SPAM [19]. Atwood's article isn't about naïve approaches specifically, but rather it explains why CAPTCHA isn't broken. He claims that although a few CAPTCHA-defeating proof of concepts have been published, many major websites such as Google, Yahoo and Hotmail still use CAPTCHA.

Furthermore, Atwood makes the argument that, "The real secret to CAPTCHA is

that it hits spammers where they are most vulnerable: in the pocketbook. The minute you put up a computational barrier, the entire economic model of SPAM comes crashing down [19]."

This argument works logically when examining the previously mentioned research of Chellapilla and Simard [21], Mori and Malik [22] and Hocevar [24]. In all of the mentioned examples where CAPTCHA was broken, the researchers had to design algorithms for a specific CAPTCHA. In the case of RMG vs. Ticketmaster [17], a situation can be seen where the financial benefits of breaking CAPTCHA outweigh the research costs.

In the case of smaller websites, it is not economically feasible to research algorithms for every possible CAPTCHA type. Rather it is easier to submit SPAM to every form that can be found and hope that some of them post to the website. I've experienced this on my personal website, <http://sumdog.com>, where I incorporated CaptchaPHP into the guestbook over two years ago to combat rising levels of SPAM. The amount SPAM I receive dropped from several messages a week to fewer than ten SPAM posts over the entire course of its implementation.

The importance of Atwood's work in regards to my own research is to show that SPAM can be prevented by using unsophisticated techniques. With Atwood's very simple naïve CAPTCHA, he was able to successfully reduce the amount of SPAM posted on his website. My research into both naïve and non-CAPTCHA based approaches show that there isn't an effective way to measure or compare the effectiveness of such approaches over regular CAPTCHA challenges. Still, this information is important to note as it is

significant to the general field of SPAM prevention.

2.5 Conclusions

From studying existing ventures into breaking CAPTCHA, there has been a considerable amount of research done on breaking CAPTCHA by independent researchers, major universities and even major corporations. Although many groups will publish results as well as complex methodology, few will post actual source code, most likely for ethical reasons.

What are considered easy cases have been solved for a considerable amount of time and require only simple image filtering in combination with shape recognition to solve. Harder cases may require additional work with segmentation and machine learning, yet some researchers have been able to get reasonable success rates even with such cases. The hardest challenges are images that use varying colors, intersecting lines, words layered upon words and various other techniques that make filtering the original characters very difficult while still maintaining easy visibility for human readers.

Even with all these innovations in segmentation, filtering and recognition, none of the aforementioned studies have a general case algorithm to work universally on all CAPTCHAs. The methodology must be customized and tailored for each type; therefore, attacks are most often targeted, either to specific types of challenges or to a single particular website of high interest to an attacker.

Chapter 3 – Research and Experiments

My research focuses on three distinct yet related problems. One is to attempt and break existing distorted word based CAPTCHA using freely available tools, the second is to create or improve a form of CAPTCHA and the third is to compare the effectiveness of using CAPTCHA based alternatives. The first problem involves writing an application to compare different approaches to breaking CAPTCHA. The second involves creating a script to implement a new form of photo identification base CAPTCHA. The third problem involves examining statistics from services which provider alternatives to CAPTCHA.

In this chapter I describe the Java based BMCB engine I designed to generate data sets of CAPTCHA images and run experiments against those sets using optical character recognition applications. I also cover FlickMeCaptcha, a PHP add-in I designed to be easy to integrate into existing websites that offers an improvement on photo-based CAPTCHA. Finally, I examine Akismet, an alternative to CAPTCHA which examines website submissions the same way e-mail filters examine messages for SPAM and compare its effectiveness to CAPTCHA.

3.1 BMCB Engine

The BMCB engine generates several sets of CAPTCHA images with known answers and stores those answers and their corresponding files in a database. The engine can then be used to apply image filtering, segmentation (separating individual characters

into sub images) and analysis techniques to the image sets and see how well the computer functions in determining the text in the CAPTCHA images. The engine is dynamic enough that it can be used for analyzing each image independently or using machine learning to train an analyzer with one of the data sets.

All of the generators, filters, segmentators and analyzers that come with the engine are based on existing open source technologies, however they can all easily be expanded to incorporate various technologies and algorithms. Most of the default analyzers are wrappers for open source Optical Character Recognition (OCR) software. By itself, the OCR software would have difficulty interpreting the highly distorted images, therefore the use of filtering and segmentation on the images was attempted to see if they would improve the OCR software's ability to correctly identify challenges.

3.1.1 Experiment Constraints

The engine is highly adaptable with the ability to set constraints within the workflow classes. In the experiments detailed within this paper, several open source CAPTCHA generators were modified to take in an argument from the command line in order to create a set of known CAPTCHA challenge images and responses. The following constraints were used:

- The CAPTCHAs consist of a set of random letters distorted in the common image based CAPTCHA
- All CAPTCHAs in a given data set are all a fixed length of five characters
- CAPTCHAs only contain letters, no numbers, with the challenge response being case insensitive

- Each CAPTCHA set consists of 1000 challenge response images

The constraints may seem restrictive, however they allow the design of the engine to focus on a very narrow scope and solve simple problems before progressing on to the general case.

3.1.2 Set Generation

In order to generate a set of known CAPTCHA challenges, common open source script need to be slightly modified. The engine has a default Command Line Generator class that will take any program given to it, pass that program the CAPTCHA letters as the first argument and the path to where the distorted image should be written as the second. Most generation scripts can easily be modified by altering the methods used to randomly generate the text within the image as well as the function used to display the image on the webpage.

Four different CAPTCHA generation scripts were used with the engine. Three of them are PHP based open source CAPTCHA tools: CaptchaPHP [25], Freecap [26] and Gotcha [27]. The final script is a custom one that creates an undistorted image in an easily readable font to be used for the trivial case to test the accuracy of the analyzers [Figure 8].

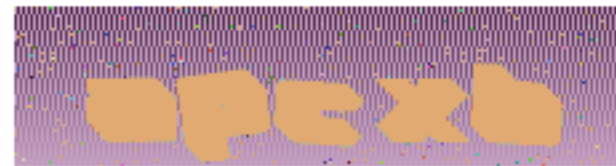
CaptchaPHP



Freecap



Gotcha



Trivial XUQAR

Figure 8 - Types of CAPTCHA

3.1.3 Image Filtering

The default filtering class that comes with the engine is a wrapper for the command line file manipulation tool ImageMagick. ImageMagick is an open source tool used to programmatically perform common image manipulation tasks such as brightness, contrast, color adjustment, edge detection, resizing, transformation, etc.

Simple effects such as brightness and contrast adjustment can greatly affect the readability of the image, both by humans and character recognition programs.

ImageMagick uses the “modulate” parameter to adjust brightness and contrast [Figure 9 - Figure 10].

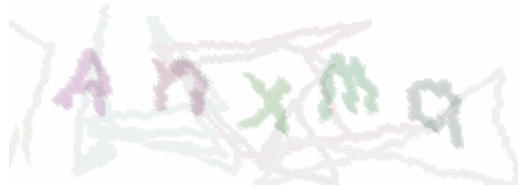


Figure 9 - Original CAPTCHA



Figure 10 - CAPTCHA with Modulate Filter 110,100 Applied

Noise reduction is another helpful filter when trying to examine images. The noise reduction filter helps to remove much of the distortion from areas of heavy changing contrast. Typically applied to photos to improve sharpness and remove imperfections, noise reduction can greatly improve the ability of an analyzer to recognize characters in a CAPTCHA [Figure 11 - Figure 12].



Figure 11 - Original CAPTCHA



Figure 12 - Noise Reduction Filter Applied

Filters can be used either by themselves or in combination with each other in order to remove noise from an image and make the image easier to process by shape recognition algorithms.

3.1.4 Analysis

Two basic analyzers are included with the engine. The first is GOCR, an open source OCR program released under the GNU General Public License, which was originally developed by Joerg Schulenburg who now leads a team of independent developers. The latest release of the software was in March of 2007 [28]. The second analyzer is OCRAD, an open source OCR program developed as a GNU project by the Free Software Foundation. Its latest release was in June of 2007 [29].

Both GOCR and OCRAD can be run directly from the command line. As their first argument, they take in a Portable Anymap (PNM) File. Their output is ASCII text in the standard western alphabet representing its recognition of the text [Figure 13].

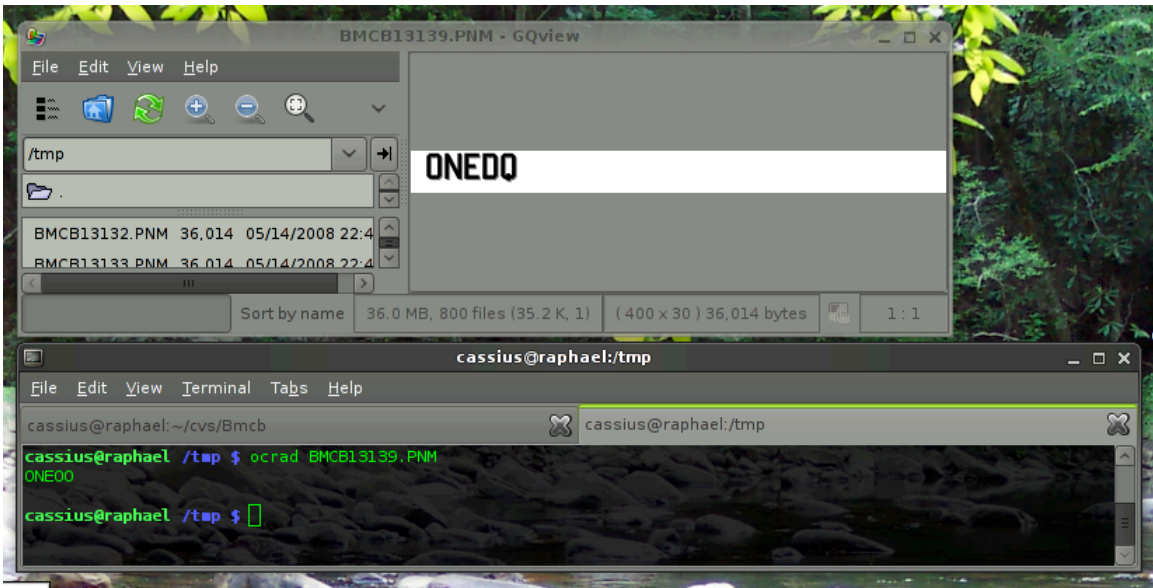


Figure 13 - OCRAD Analyzing a Trivial Image

The Guessing Analyzer is a trivial analyzer, which must know the length of the CAPTCHA (i.e. it must be called using a method that provides an array of image segments) and then randomly guesses the letters in the image without any analysis. Combined with the equally trivial Even Length Segementator, the Guessing Analyzer can be used to test the probability of correctly randomly guessing a CAPTCHA.

3.1.5 Experiments

The framework is designed so that experiments can be implemented in the form of workflows. Experiment workflows can be as simple as just taking raw unfiltered images and passing them directly to the analyzer or can be as complex as passing each image through a series of filters, before or after being segmented and eventually analyzed. The tools are chained together by utilizing a standard object for holding and

manipulating the image.

The most basic experiments executed here include two control situations. One uses the analyzers to examine the accuracy of standard text. For the case of this experiment, the font used to render the unaltered text was *COLLEGE.TTF*. It was chosen because, although all the characters are uniform, unique and have mostly straight edges, the letters are somewhat non-standard and easily confusable with one another, allowing the control set to test the analyzers in less than unique conditions.

The second control situation involves the analyzers examining CAPTCHA images without first being filtered, segmented or altered in any way. The results of such a set would show the capabilities of analysis by itself without the benefit of pre-filtering, segmentation or any other noise reducing technique.

There is also a third, nondeterministic control test in which the analyzer randomly attempts to guess the letters in the CAPTCHA challenge without performing any filtering or analysis. It has the advantage of knowing the length of the CAPTCHA and shows the results of randomly guessing for that one ideal case and shows the feasibility of using such a brute force technique.

In addition to the control or trivial cases, there are several experiments which filter the images before passing them to the analyzers. Filtering experiments include changing various degrees of brightness and contrast as well as adding modulation. The specific settings for each filter were chosen based on viewing the raw images in the debugging tool and choosing the settings that did the best job of removing noise without bleaching out the letters embedded within the image for each individual CAPTCHA type.

The goal of the experiments is to easily break CAPTCHA. A high success rate is not necessary as automated software can make thousands of requests an hour. A success rate of 10% would be sufficient to post over 100 messages an hour, assuming over 1,000 requests could be made within an hour. Such a rate is more than feasible on a standard consumer grade high speed internet connection.

3.1.6 Results

The first experiment results show [Figure 14; Table 1] that randomly guessing, even on a small and known number of letters per image, proves to have a very low success rate. Only individual letters are guessed correctly and never with more than a 4% accuracy on a set size of 1,000. Randomly guessing solutions to CAPTCHA using this brute force technique is simply not a viable solution.

In the second set of experiments, the analyzers are tested against the raw CAPTCHA challenges without any filtering. GOCR by itself without any filtering does not correctly identify any full challenges except in the trivial case and even then, it only has a 68% success rate. In most cases, it can identify fewer than 1% of individual characters correctly [Figure 15; Table 2].

OCRAD does better in the raw test for individual characters, but not as well as GOCR for complete words. Counter-intuitive to the trivial results, OCRAD does do substantially better with the distorted images and is able to solve a very small percentage of challenges for both CaptchaPHP and Freecap [Figure 16; Table 3].

The third experiment involves adjusting the brightness of the CAPTCHA by a factor of ten while keeping the contrast constant. Using GOCR, adjusting the brightness

yielded a slight increase in word character for CaptchaPHP without increasing accuracy in any other categories [Figure 17; Table 4].

OCRAD's character analysis of CaptchaPHP and Freecap benefitted from a ten-point increase in brightness, as well as its word analysis of Freecap increasing its full word success rate to over 1% [Figure 18; Table 5].

The fourth experiment was similar to the third except the brightness was adjusted to 180 while the contrast stayed constant. By adjusting the brightness, GOCR was able to make significant gains in its analysis of CaptchaPHP's challenge. Other CAPTCHAs did not see an increase in correctness with the combination of GOCR and the filter [Figure 19; Table 6].

OCRAD also performed well against CaptchaPHP by adjusting the brightness to 180. It was also able to identify a very small percentage of characters in Gotcha as well as increase its accuracy against the trivial case [Figure 20; Table 7].

The fourth experiment not only adjusted the brightness to 140, but also reduced the contrast to 5. With the 140/5 filter, GOCR did better in its analysis of Freecap with a word accuracy of 0.7%, but decreased its effectiveness of CaptchaPHP [Figure 21; Table 8].

OCRAD did better against CaptchaPHP in correct letters with the 140/5 filter than GOCR gaining both a higher correct letter and word count, however, it was less effective against Freecap [Figure 22; Table 9].

In the fifth experiment, once again, only the brightness was increased. With a brightness adjustment of 160, this test places GOCR at the highest success rate of solving

CaptchaPHP with a correct word accuracy of 0.5%. The effect of brightness adjustment with CaptchaPHP seems to have the most significant effect on increasing its readability. Other CAPTCHAs did not see any significant rates of solvability [Figure 23; Table 10].

Increasing the brightness to 160 did help improve OCRAD's correct letters success rate against CaptchaPHP giving it a letter accuracy of 31.0%, the highest accuracy in individual letters out of any other experiment. However, other filters provided higher accuracy in correct words [Figure 24; Table 11].

In the final experiment, noise reduction is used with a radius of 1 to filter the images. GOCR did help with correctly identifying CaptchaPHP challenges, however the rate of success was not as significant as other filters with only a 0.3% word accuracy for CaptchaPHP and 0% for the other two non-trivial tests. [Figure 25; Table 12].

OCRAD performed slightly better with noise reduction than GOCR in accuracy of correct letters but did not do as well in correctly identifying words. As with GOCR, noise reduction was not as effective as other techniques [Figure 26; Table 13].

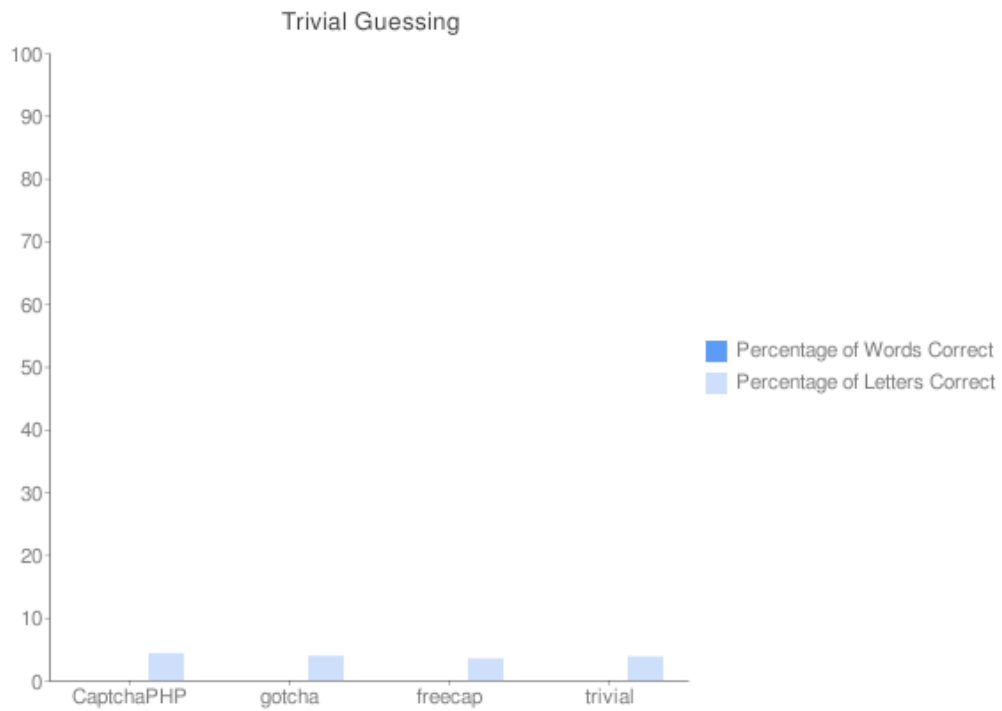


Figure 14 - Graph of Trivial Experiment Results from Guessing

Table 1 - Trivial Experiment Results from Guessing

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	4.3%	3.9%	3.5%	3.9%
Words Correct	0	0	0	0

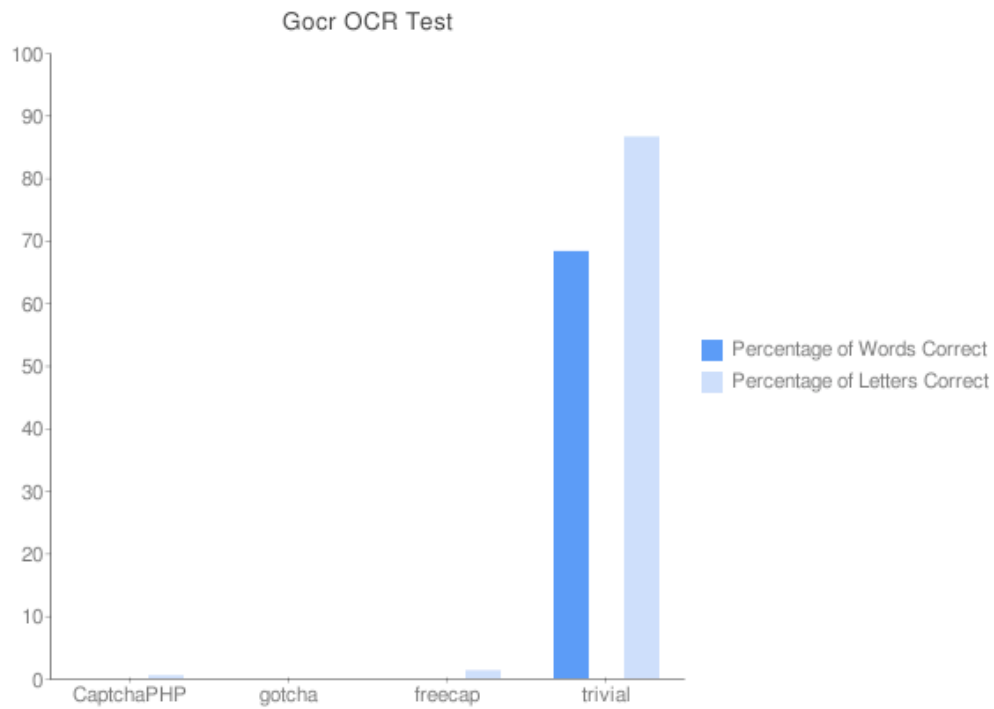


Figure 15 - Graph of Raw GOCR Test

Table 2 - Results from Raw GOCR Test

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	0.6%	<0.1%	1.3%	86.6%
Words Correct	0	0	0	68.3%

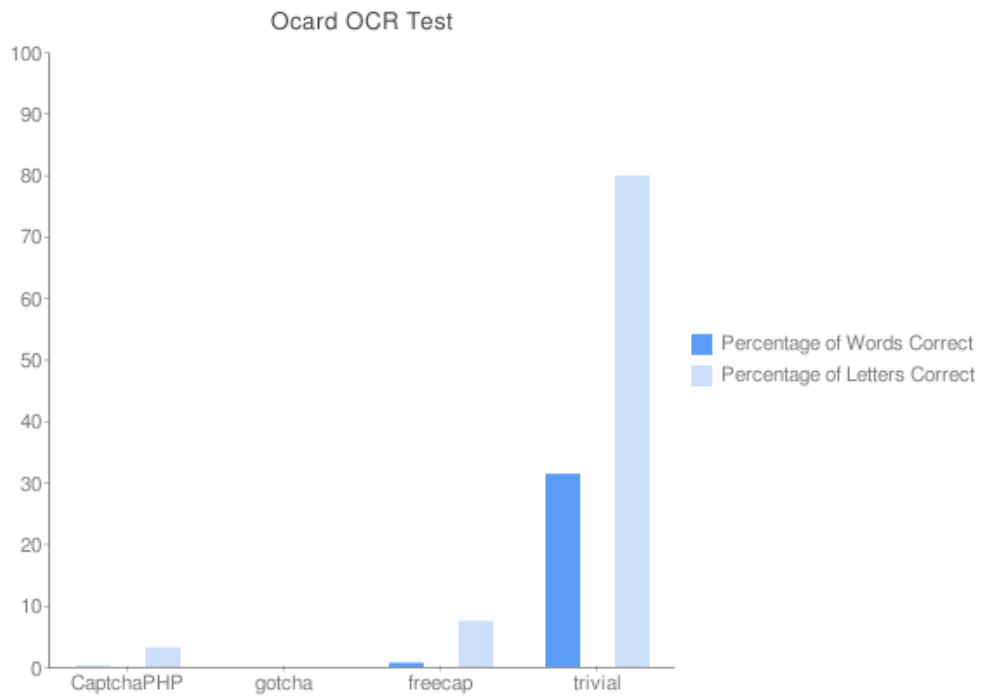


Figure 16 - Graph of Raw OCRAD Test

Table 3 - Results from Raw OCRAD Test

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	3.3%	<0.1%	7.5%	79.9%
Words Correct	0.1%	0	0.7%	31.5%

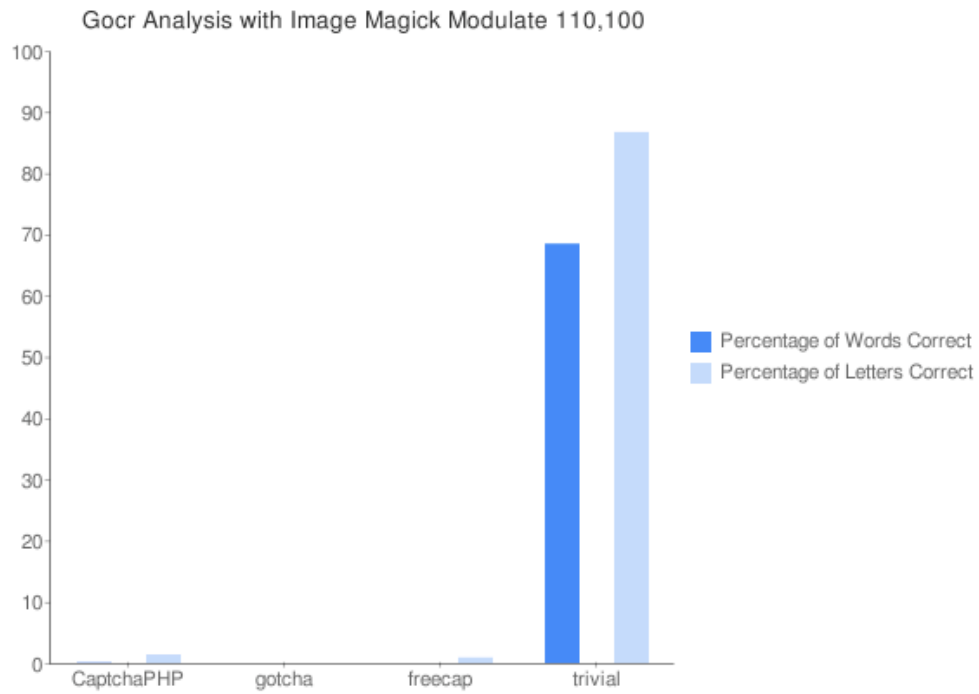


Figure 17 - Graph of GOCR Analysis of 110 Brightness Adjusted Image

Table 4 - Results from GOCR Analysis of 110 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	1.4%	<0.0%	0.9%	86.7%
Words Correct	0.1%	0%	0%	68.6%

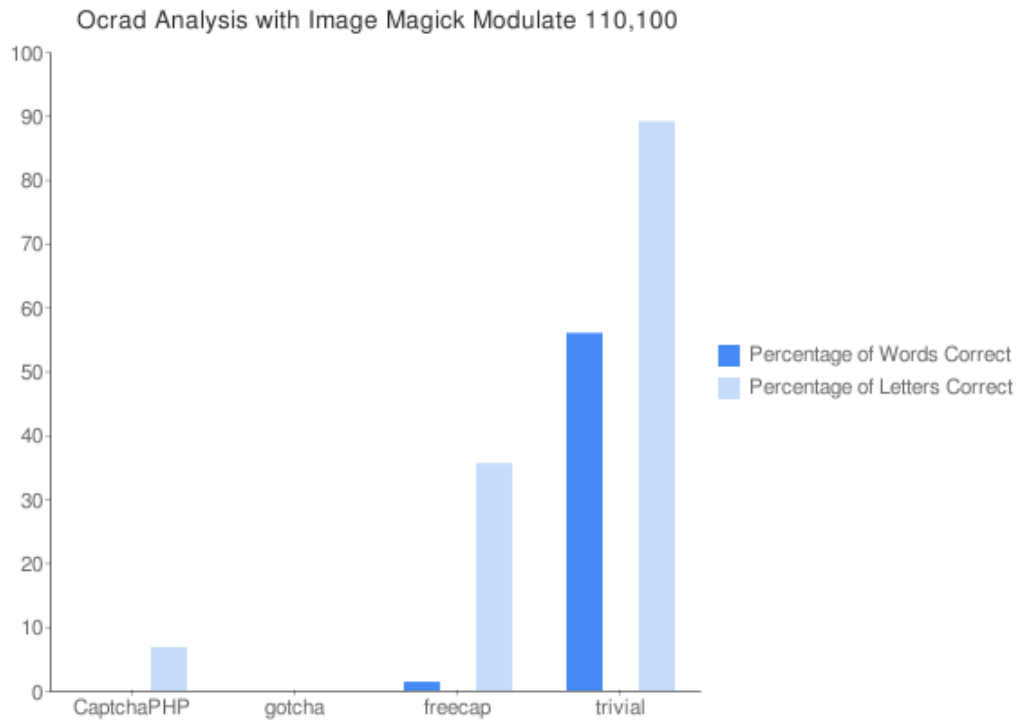


Figure 18 - Graph of OCRAD Analysis of 110 Brightness Adjusted Image

Table 5 - Results from OCRAD Analysis of 110 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	6.8%	<0.0%	35.6%	89.1%
Words Correct	0.0%	0%	1.5%	56.0%

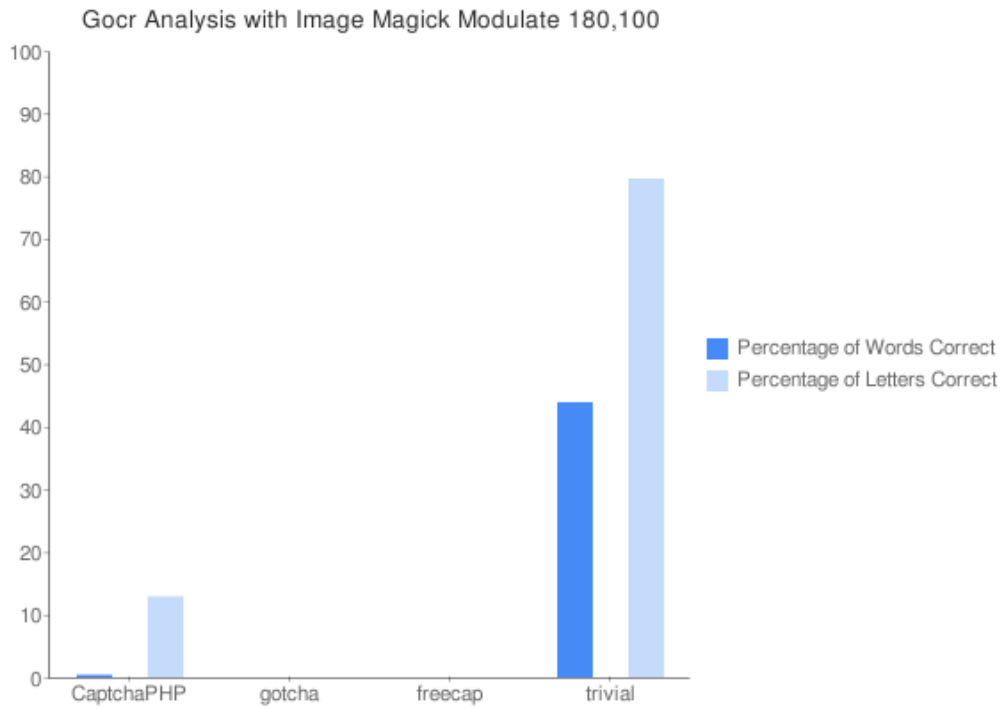


Figure 19 - Graph of GOCR Analysis of 180 Brightness Adjusted Image

Table 6 - Results from GOCR Analysis of 180 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	12.9%	<0.1%	0%	79.6%
Words Correct	0.4%	0	0%	44.0%

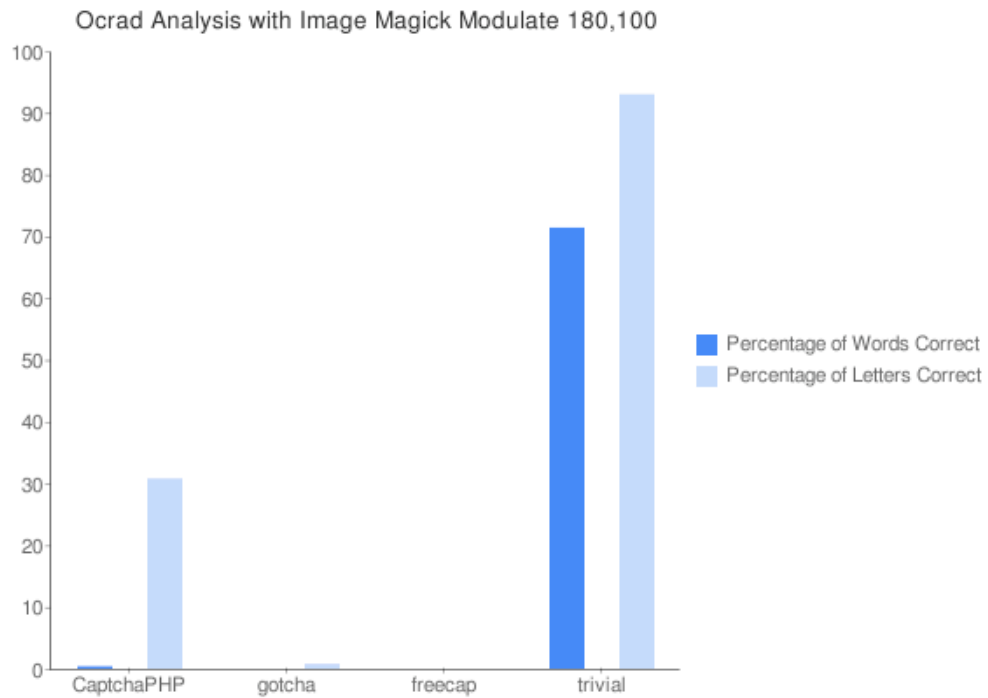


Figure 20 - Graph of OCRAD Analysis of 180 Brightness Adjusted Image

Table 7 - Results From OCRAD Analysis of 180 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	30.8%	0.8%	7.5%	93.1%
Words Correct	0.4%	0%	0.7%	71.5%

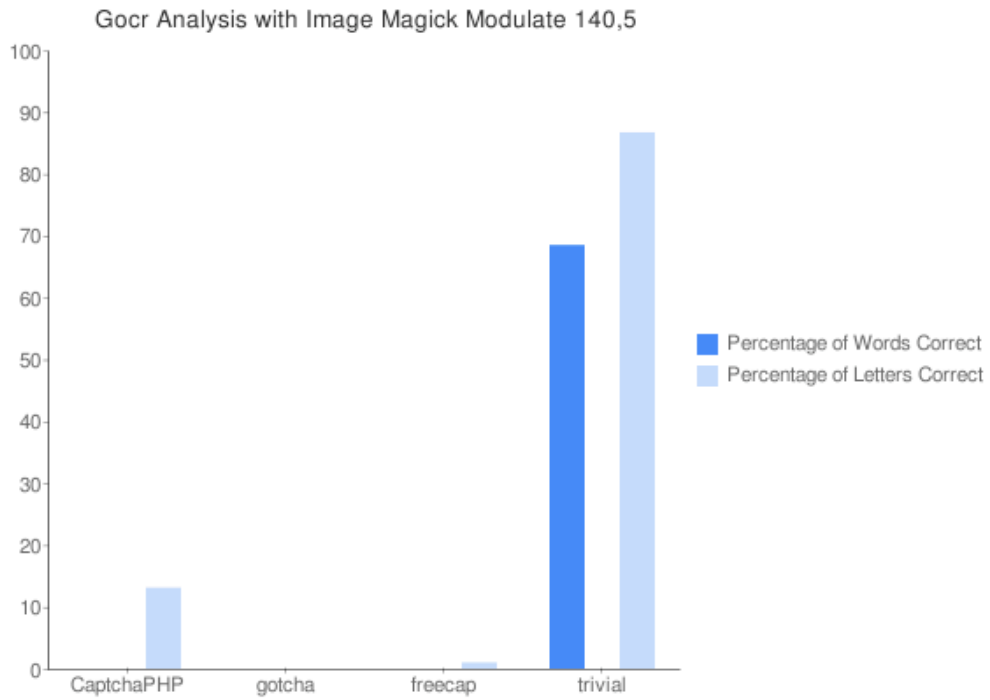


Figure 21 - Graph of GOCR Analysis of 140 Brightness, 5 Contrast Adjusted Image

Table 8 - Results From GOCR Analysis of 140 Brightness, 5 Contrast Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	13.1%	<0.0%	1.0%	86.7%
Words Correct	0.0%	0%	0.7%	68.6%

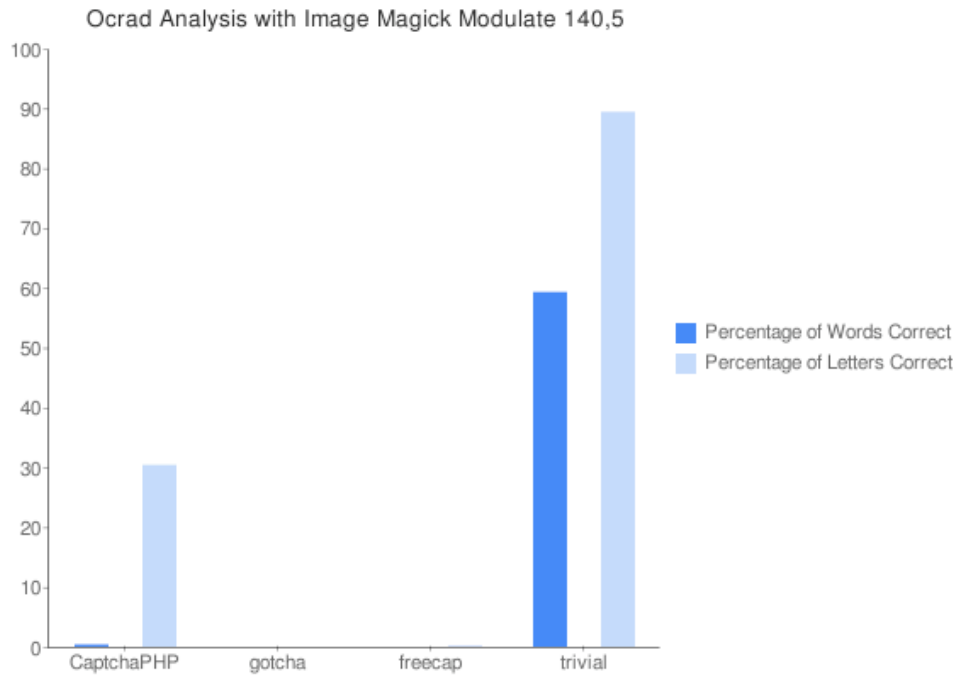


Figure 22 - Graph of OCRAD Analysis of 140 Brightness, 5 Contrast Adjusted Image

Table 9 - Results from OCRAD Analysis of 140 Brightness, 5 Contrast Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	30.5%	0.0%	0.2%	89.5%
Words Correct	0.4%	0%	0.0%	59.4%

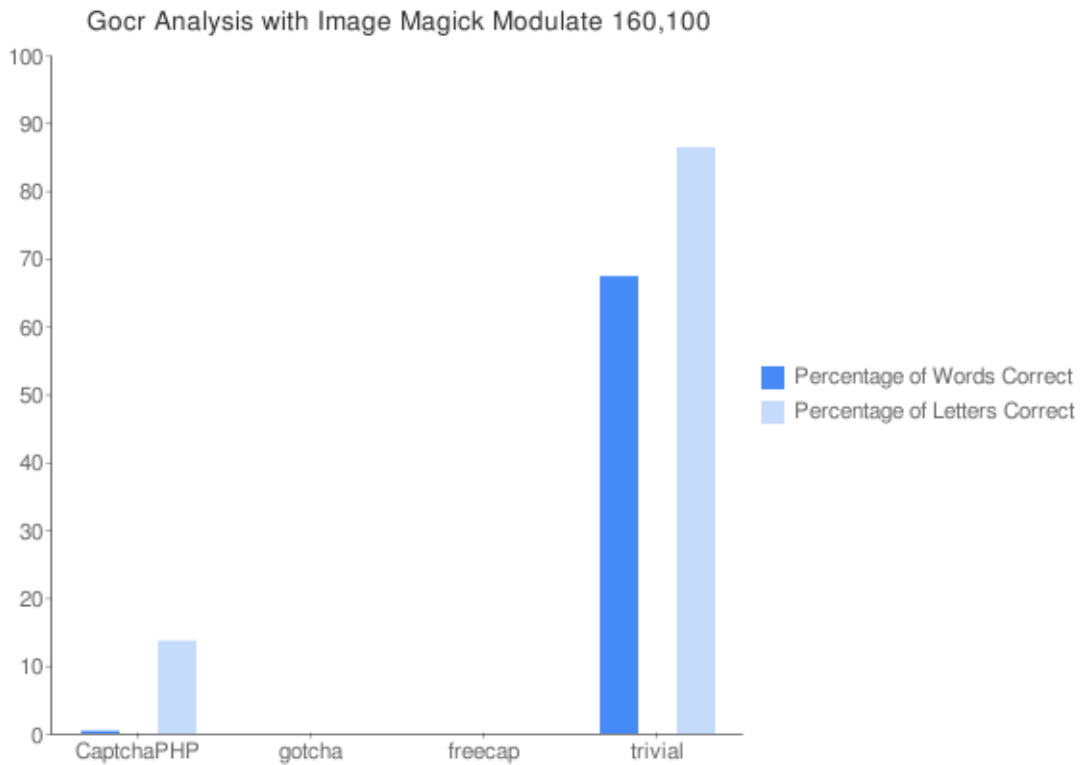


Figure 23 - Graph of GOCR Analysis of 140 Brightness Adjusted Image

Table 10 - Results from GOCR Analysis of 140 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	13.7%	<0.0%	0.0%	86.4%
Words Correct	0.5%	0%	0.0%	67.4%

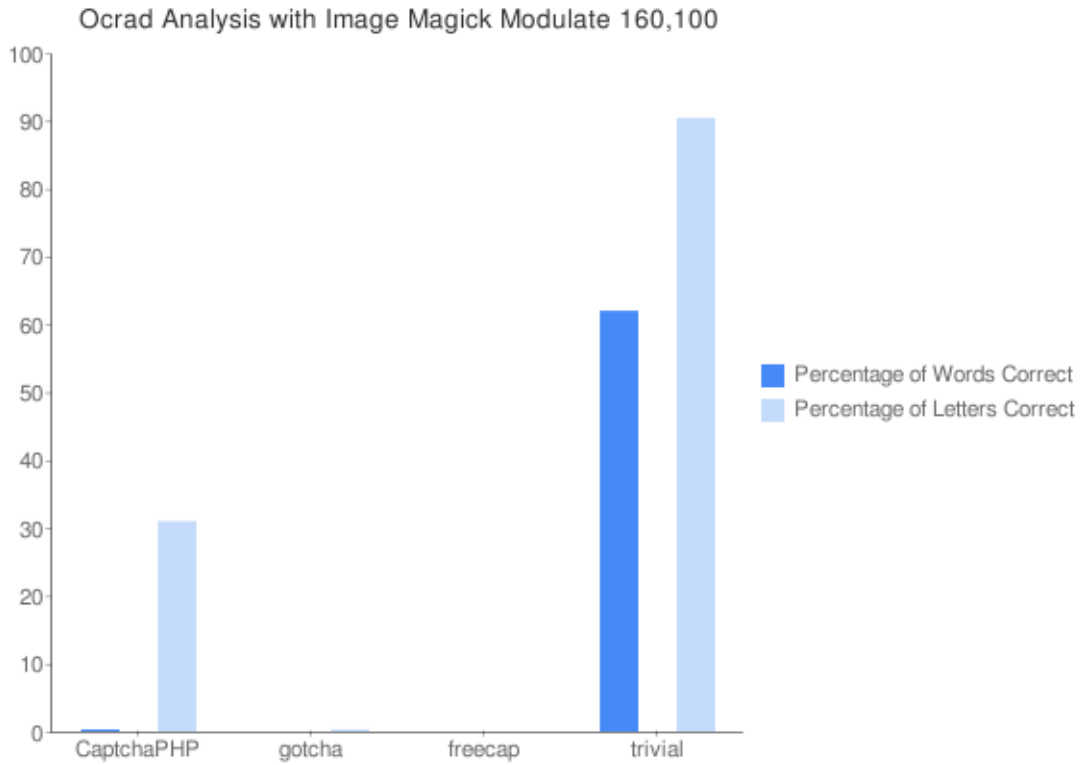


Figure 24 - Graph of OCRAD Analysis of 160 Brightness Adjusted Image

Table 11 - Results from OCRAD Analysis of 160 Brightness Adjusted Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	31.0%	0.3%	0.0%	90.5%
Words Correct	0.3%	0%	0.0%	62.1%

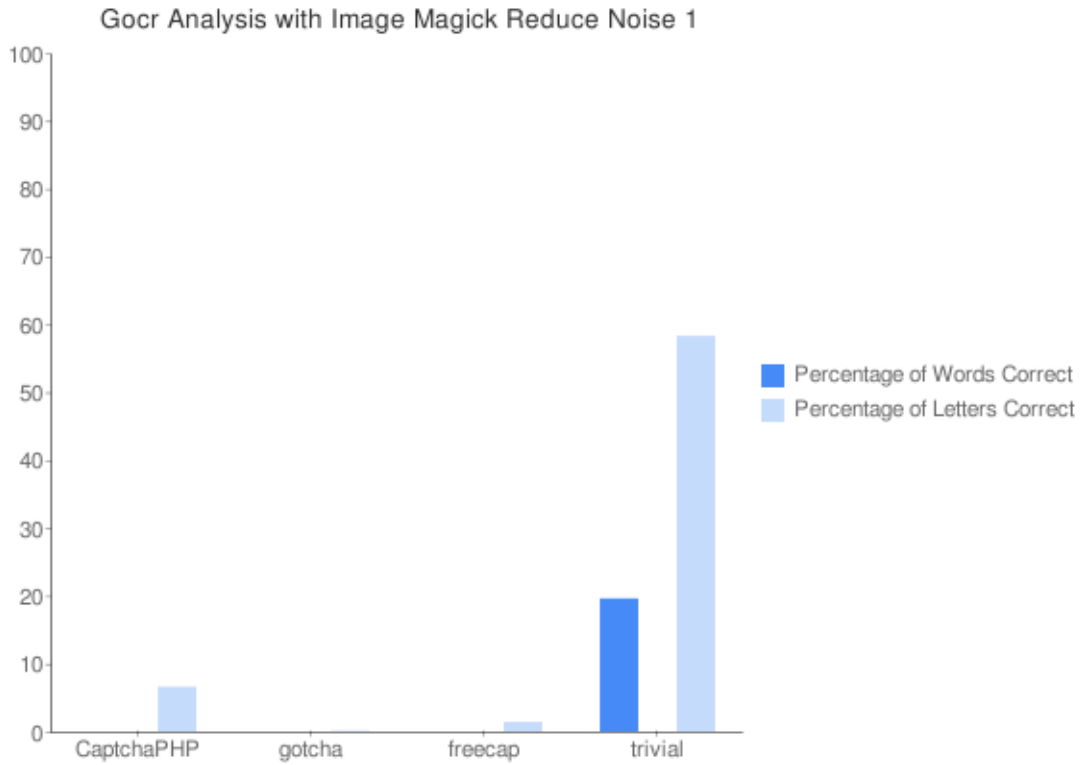


Figure 25 - Graph of GOCR Analysis of Noise Reduced Image

Table 12 - Results from GOCR Analysis of Noise Reduced Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	6.7%	<0.0%	1.5%	58.3%
Words Correct	0.3%	0%	0.0%	19.7%

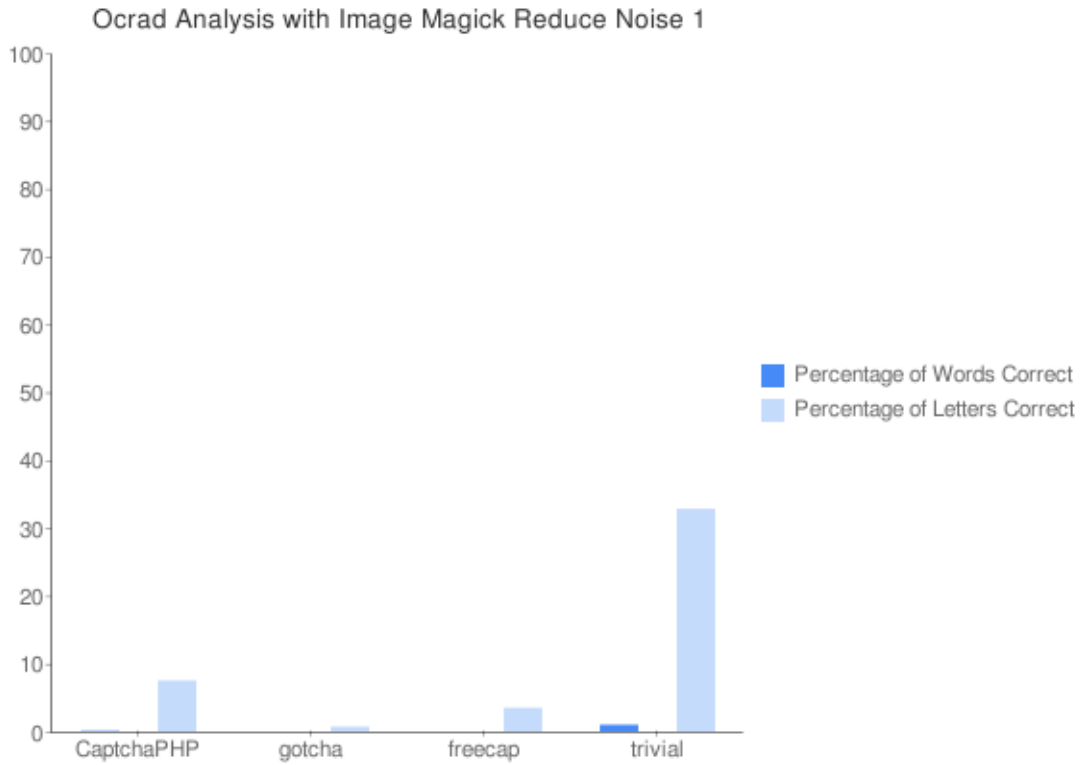


Figure 26 - Graph of OCRAD Analysis of Noise Reduced Image

Table 13 - Results from OCRAD Analysis of Noise Reduced Image

	CaptchaPHP	Gotcha	Freecap	Trivial
Letters Correct	7.6%	0.8%	3.6%	32.8%
Words Correct	0.1%	0.0%	0.0%	1.1%

3.2 *FlickMeCaptcha*

Part of the research not only deals with trying to break existing CAPTCHA, but also to creating a new CAPTCHA. Previously, in section 1.4.3 Photograph Identification, this paper had covered an existing type of visual CAPTCHA based on using photos where several of the images contain kittens. To prove one is human, the user must select all the images which contain kittens. The disadvantage to this technique is that a very large selection of images is required for this type of CAPTCHA to work. It also offers no alternative for visually impaired users.

The first disadvantage can be addressed by using publicly available images that have been pre-categorized. Rather than trying to construct an image repository that is static, the service Flickr from Yahoo offers a means by which regular people can upload photos to share over the Internet and assign them metadata known as 'tags' indicating information about the photos. Flickr offers a public application programming interface (API) to search and interact with the photo repository. With new photos being uploaded everyday and existing photos given new tags at the same rate, Flickr provides a very broad and dynamic base of photos to use in photo-based CAPTCHA.

The application I developed as proof of concept is called FlickMeCaptcha. It is written in PHP and uses Flickr's representative state transfer (REST) API in order to perform searches for photos. It creates a CAPTCHA challenge that displays images in a grid from which the user must pick out images which relate to a given tag. The amount of images displayed is configurable within the application, but by default it displays nine

images in a square grid. Four of the images match the given tag and at least three of them must be selected for the challenge to be answered correctly [Figure 27].

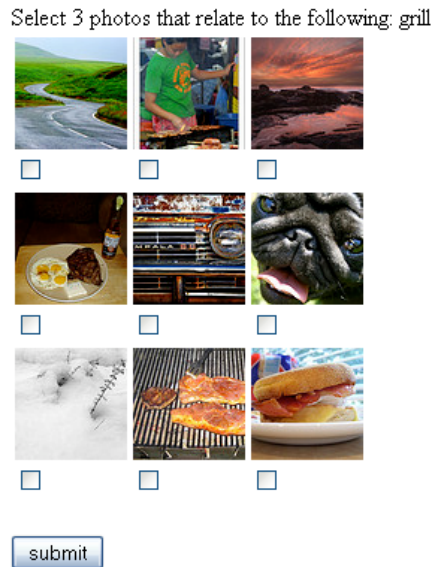


Figure 27 - FlickrMeCaptcha

FlickrMeCaptcha can retrieve the tags it uses in two ways. The first way is to use a predefined list stored in a text file with the application. The second is that it can retrieve the most popular tags on Flickr and randomly select from the set of popular tags. Using a text file has the distinct disadvantage of having a statically defined set of tags. Without a sufficiently large list, it may be possible to circumvent the CAPTCHA simply by examining all photos with a given tag. Popular tags may seem like an obviously better choice, however there are many cases where popular tags have nothing to do with the content of the photograph. For instance, one popular tag is "Canon10kn" which referees not to the photo itself, but to what camera was used to take it.

3.3 Non-CAPTCHA based SPAM Defense

Several services have arisen to combat SPAM, not by using CAPTCHA to ensure the poster is human, but by analyzing the content of the message in the same way e-mail filters work. Many of these services are free to use for non-profits and individual. Companies that provide the services pull in revenue to sustain themselves by offering commercial licenses. In my research I've examined the following three services: Akismet [13], Mollom [14] and Defensio [15].

Akismet is a service that analyzes blog and forum comments to determine if a particular message is SPAM or a legitimate interactive post from a human (what its designers refer to as "Ham"). Submissions to the service return a true or false indicating if the post is SPAM. Many popular content management systems such as WordPress and Drupal have plugins that run all user submitted content through Akismet's service.

According to Akismet's statistics page which is continually updated, as of February 2009, Akismet has caught over 9.8 billion SPAM messages while allowing 1.8 billion Ham messages [30]. The service interface allows site maintainers to mark false negatives and false positives, although quantities of such identifications do not appear on the overall statistics page. The inner workings of Akismet are kept mostly secret in order to prevent malicious users from "gaming the system [31]."

Criticism has been raised about Akismet flagging Ham comments as false positives for SPAM. A blogger, that goes by the pseudonym "timetheif," posted the following about Akismet in early 2008:

"Although I have never experienced this problem before, as of yesterday the comments that I leave on wordpress.com blogs where I have previously been approved did not appear immediately after I posted them. I became suspicious and did some investigation and found that the comments are ending up in the Akismet spam filter. Thus far raincoaster, sulz, thesacredpath and several other bloggers have found my comments in the Akismet filter and fished them out [32]."

An alternative to Akismet is Mollom, which also has plugins for several major content and blogging engines. As of February 2009, Mollom has claims to have caught 33.9 million SPAM messages with an average efficiency of 99.95%, meaning that only 5 in every 10,000 SPAM messages are not caught [33]. It is very similar to Akismet in the sense that it is a web based service that is free for non-commercial use and sustains itself with licensing for commercial and enterprise use.

Defensio is yet another direct competitor to Akismet and Mollom with a similar suite of plugins for major content engines and a free for non-commercial use license. In January of 2009, Defensio was acquired by Websense [34]. Unlike its competitors, Defensio does not offer a statistics page on its website to show the amount of SPAM its filter has blocked.

It is difficult to get accurate numbers for false-positives when it comes to CAPTCHA SPAM protection since oftentimes a user with visual disabilities may have several failed attempts before successfully being able to post a message or may give up

in frustration altogether. Although there are more statistics gathered for non-CAPTCHA based approaches, and such approaches have a more reliable way of determining false-positives (i.e. humans marking incorrectly flagged messages), there still seems to be no reasonable means by which to compare the two techniques.

The advantages of using a non-CAPTCHA based technique for preventing SPAM include the ability for visually impaired individuals to easily post messages as well as reducing the burden of the end users for proving that they are in fact human. The drawback is the possibility of false-positives and the requirement of website administrators to occasionally check responses marked as SPAM. This may become impractical for sites with heavy amounts of traffic such as bulletin boards or websites with significant amounts of comments. The advantages of either technique are not easily measurable or comparable due to the difficulty in creating controlled SPAM environments, difficulty in the collection of statistics on CAPTCHA based systems without human interactive studies and the closed source nature of non-CAPTCHA based alternatives.

Chapter 4 – Analysis

4.1 Degree of Success for the BMCB Engine

Examining the BMCB Engine, the best-case results with optimized filters show success rates ranging from 1 in 1000 to 3 in 1000. Although this may seem like a very low rate of success, the requests to web servers are automated and most web servers are designed to handle very large volumes of simultaneous connections. To determine if such a success rate is adequate to post large amounts of SPAM, the connection and transfer times must be examined as well.

Given that the average broadband connection within the United States is approximately 1.9Mbps [35], a CAPTCHA the size of 1 to 5 kilobytes would take less than a second to transfer at that speed, but the connection to the web server itself is expensive. Although most web servers support pipelining, that is the ability to stream multiple requests on a single connection, answering a CAPTCHA challenge requires a request for the image followed by a response to the server, which must then terminate.

Connection times can vary greatly depending on the server being access and network congestion. Assuming a modest average case connection time of two seconds, it is reasonable to assume at worst it would take five seconds (two per request and one to download the image itself) to request and respond to a website with a CAPTCHA challenge.

This would mean 12 requests could be completed per minute or over 700 per hour. With success rates using the engine at under 1 in 1000, this would mean at most,

only one SPAM message could be posted per hour. If the assumption is given that there is a very fast connection with minimal delay to a web server, and the entire request/response process only took one second, over 3,000 requests could be handled per hour allowing for the possibility of posting up to 3 messages an hour.

Since it's been established that the return rate for SPAM is very low, at best 1 in 1,000 and at worst 1 in 1,000,000 [19], a dedicated system in the best-case scenario of one response per second would be required to run for over 10 days in order to get a single return for SPAM posting efforts.

Although adding additional computers could increase the effectiveness of breaking CAPTCHA, the cost would still be prohibitive. The research given here shows that using open source and freely available image recognition and OCR analysis tools does not yield a high enough rate of return to be useful in producing SPAM messages.

Furthermore, examining the results shows that specific sets of filters can yield better results for one CAPTCHA while making results worse for another. For example, CaptchaPHP had the greatest success rate of 0.5% resulting from GOCR analysis combined with a 140 brightness adjustment. In the same experiment, Freecap had a success rate of 0%. Freecap's highest success rate was 1.5% using OCRAD analysis with a brightness adjustment of 110. In the same experiment CaptchaPHP had a success rate of 0%. This shows that filters and analysis must be specifically targeted to a given type of CAPTCHA.

In the case of the Gotcha CAPTCHA, not a single experiment was able to get even one challenge solved. Some experiments were able to correctly identify some letters

with success rates in the range of less than 0.1% to 0.8%, however the trivial test which, attempts to randomly guess the challenge without performing any type of analysis had a higher character success rate of 3.9%. This shows that for this one CAPTCHA, randomly guessing the challenge's answer has a higher rate of success than actually trying to analyze the image. Researchers have shown particular CAPTCHAs may be solvable, but to do so would require specific and directed effort into analysis of that particular CAPTCHA. There is, as of yet, no general algorithm that is effective against all challenges. Therefore small and moderately sized websites, blogs and message boards are reasonably protected from SPAM with CAPTCHA. Only large sites, where the potential for financial gain to outweigh the substantial research and development cost in breaking CAPTCHA, need to be concerned with such attacks.

4.2 Pros and Cons of FlickMeCaptcha

FlickMeCaptcha is an improvement over existing implementations of image based CAPTCHA such as KittenAuth. There are several advantages including the following:

- Availability of large selection of constantly growing images for challenges
- Ability to select random tags from the most popular set on Flickr
- Many configurable options including number of images to display and number of correct images that must be selected
- Ability to use a predefined wordlist

Although FlickMeCaptcha does solve some problems with photo-based

CAPTCHA, the largest concern being the base of images from which to choose from, it does bring up some new problems of its own. They include the following:

- Most popular tags on Flickr often do not represent anything in the image (e.g. a tag indicating the camera model used to take the photo)
- Copyright constraints restrict the available images to those licensed under the creative commons
- The current implementation makes several successive calls to the REST service instead of utilizing HTTP pipelines, causing a performance delay

Overall, the gains in photo-based CAPTCHA are significant and the script is more of a proof of concept that can be expanded upon and easily integrated into content management software and blogging engines.

4.3 Filtering versus CAPTCHA

Studying the alternatives to CAPTCHA resulted in finding a number of web service based filters with open programming implementations that can be used by many different types of content management software. Although many of the implementations provided general and overall statistics, none of them provided more detailed numbers that could be filtered or used to drill-down specifics. Due to this, studying the effectiveness of these approaches versus CAPTCHA becomes difficult. Akismet staff has even stated publicly that such studies would be ineffective [36].

Chapter 5 – Contributions to the Field

The crux of the research presented in this thesis comes from two programs specifically written for expanding knowledge of the field of CAPTCHA and SPAM prevention. Both programs are released as open source and will continue to be available for developers and researchers to expand upon. The first program is the BMCB Engine whose audience is specifically developers and researchers who want to test techniques for attaching and breaking CAPTCHAs. The second program is FlickMeCaptcha, a program intended for web developers and end users to provide a new image based CAPTCHA that can not easily be solved.

5.1 BMCB Engine

Although the BMCB Engine did not meet the measure of success proposed, that is the ability to solve CAPTCHA at a success rate necessary to post large amounts of SPAM in a reasonable amount of time using freely available filtering and analysis programs, it did provide some insights and contributions to others attempting the same type of research.

The results show that it is possible to use image filtering in combination with existing OCR tools to solve current CAPTCHAs automatically without the need of a human and in effect defeating the purpose of the test. However, since the percentages for success are so low, website administrators could protect themselves simply by installing software or hardware that tracks repetitive and continual network requests. Such software

is often used to prevent Denial of Service attacks. If such a system were in place, an attacker would need to use some type of distributed network or a set of computers compromised with a virus to effectively carry out an attack.

Furthermore, the attack must be targeted towards specific types of CAPTCHA. In the existing studies published, the algorithms used were specific websites and CAPTCHA scripts. The engine, along with results from existing research, show that filtering works best when it is target to a specific image challenge. There has been no evidence to show that it would be easy, nor is it currently possible, to generate an all-purpose analyzer that would be capable of decoding all types of CAPTCHA as well as a human could.

The analyzers and filters that come with the engine I developed are very basic and rely on existing, publicly available and free open source tools, yet still yield viable results. The advantage of using existing tools is that the engine itself is designed to be easily extendable, dynamic and can be adapted to facilitate other research with more complex algorithms. In its current state, it can be released to the public without large ethical concerns. It is my hope that this contribution to the open source community will help further future innovations and will be a useful tool for others who are interested in the field.

5.2 FlickMeCaptcha

FlickMeCaptcha provides a new tool that can be implemented by website designers to provide a new type of CAPTCHA to end-users. Although it uses the concept of photo-based challenges, which have been implemented before, the means by which it

acquires photos is innovative and provides a much larger and constantly changing base than previous implementations.

There are several advantages for using FlickMeCaptcha including a very large, constantly changing base of images that can be displayed in different sizes and are fairly easy to identify. The interface is fully customizable with Cascading Style Sheets (CSS) and the project is open source, so it is free to modify, and the tags used are stored in a plain text file that can easily be modified.

There are also disadvantages of using FlickMeCaptcha. The option to use popular tags instead of a list is not very useful due to many tags not relating to the photos directly. There is also the possibility that individuals have given photos tags that don't necessarily relate to them in an intuitive way, meaning the user may have to attempt to solve the CAPTCHA several times before being given a relevant set of images. FlickMeCaptcha is also limited to photos on Flickr that are copyright free, limiting the potential base of images. This can be changed in the setup but may cause legal problems. Finally, there is no built-in alternative for the visually impaired, so FlickMeCaptcha would have to be paired with another approach to be inclusive to those with disabilities.

5.3 Conclusion

Analyzing the problem of SPAM and the solution on CAPTCHA does not address the core problem that allows SPAM to propagate and be as invasive as it is. The core problem with SPAM is its success rate. Even with only a return of fifty responses to every million SPAM messages sent [37], the model is still economically viable because

the messages themselves cost virtually nothing to send. The cost to bandwidth, computer security and the massive infrastructure that must be put in place to deal with SPAM is detrimental to growth on the Internet.

Although CAPTCHA has been and will continue to be a viable solution, with new and innovative forms of CAPTCHA improving human success rates while being increasingly effective against automated attacks, the main problem is that SPAM will continue as long as there is a consumer response, no matter how minuscule. In order to remove the need for CAPTCHA and other SPAM prevention techniques, Internet users need to be more informed about identifying SPAM so as to not purchase products, and thereby grant legitimacy, to websites that use SPAM based promotions, to the degree that the return rate of SPAM becomes financially unviable.

Individuals have been running mass advertisements and scams using regular postal mail for decades before the age of the Internet. The greater problem of increasing awareness and social intelligence is well beyond the scope of this thesis. However, in the meantime, the innovations that have come about through combating unsolicited e-mail, viruses and malicious website attacks have been invaluable. Security research in both the fields of prevention and circumvention have led to powerful innovations in machine learning that have been of great benefit to the Computer Science community.

5.4 Future Work

Several areas of research exist which can be built on top of the results outlined in this thesis. These areas of research include enhancing the existing analysis engine,

expanding FlickMeCaptcha, and performing research on humans' ability to correctly respond to CAPTCHA challenges versus that of a machine.

The engine is licensed under the GNU General Public License (GNU GPL) version 3, making it freely available for others to view, modify, enhance, learn from and redistribute. Although the engine itself is structured to be a comprehensive testing platform, the filters and analyzers themselves are not very strong. Since the engine is written in Java and has several extendable abstract base classes, it can easily be expanded to use new analysis algorithms and filtration techniques. Full guides for installing the engine and developing with it are located in Appendix A and Appendix B respectively.

Along with studying existing CAPTCHAs, additional research is also warranted in creating new CAPTCHA challenge techniques. FlickMeCaptcha is a wonderful proof of concept, but it still has room for development. Features that could be developed for it include an alternative CAPTCHA, either audio or word puzzle, for the visually impaired, using pipelined connections to speed up response time (versus making multiple connections to Flickr's REST service) and a better means for choosing relevant tags.

Since no human studies were conducted for this thesis, another area of research would involve human interaction with CAPTCHA and examine people's success rate in answering CAPTCHA challenges correctly in correlation to their age demographics and visual impairments. Such research, in conjunction with computer analysis, could show which techniques work better for humans and against automated programs.

The research in this thesis is one stepping stone in the complex fields of image filtering and analysis. My hope is that the research presented in this thesis will help other

researchers continue to examine and enhance CAPTCHA challenges and improve security and protection against SPAM.

References and Citations

References and Citations

- 1 Terms and Conditions for reCaptcha. Carnegie Mellon University. Retrieved from <https://admin.recaptcha.net/media/lic1.5.html> on June 26, 2008.
- 2 Why is Spam a Problem. Everett-Church. E-OTI. Retrieved from <http://www.isoc.org/oti/articles/0599/everett.html> on February 18, 2009.
- 3 PHP based CAPTCHA. Retrieved from <http://milki.erphesfurt.de/CAPTCHA/> on October 22th, 2007.
- 4 Telling Computers and Humans Apart Automatically. Ahn, Blum and Langford. Communications of the ACM. February 2004. Vol. 2 No. 2 p57-60
- 5 Protecting Websites with Reading-Based CAPTCHAs; Baird, Lik. Retrieved from <http://www.ece.cmu.edu/~mluk/CAPTCHA.pdf> on October 20th 2007
- 6 No Spam! - an alternative to CAPTCHA images. Retrieved from <http://www.vbulletin.org/forum/showthread.php?t=124828> on February 13, 2008
- 7 WP-Gatekeeper. Retrieved from <http://meyerweb.com/eric/tools/wordpress/wp-gatekeeper.html/> on February 13, 2008
- 8 Learn More about reCAPTCHA. Carnegie Mellon University. Retrieved from <http://recaptcha.net/learnmore.html> on June 26, 2008.
- 9 Kitten Auth. Warner. Retrieved from <http://www.thepcspy.com/kittenauth> on February 3, 2009.
- 10 Sahami, M., Dumais, S., Heckerman, D., & Horvitz, E. (1998). A Bayesian Approach to Filtering Junk E-mail. Learning for Text Categorization: Papers from the 1998 Workshop (pp. 55–62). Madison, Wisconsin: AAAI Technical Report WS-98-05.
- 11 Naïve Bayes Spam Filtering Using Word-Position-Bases Attributes. Hovold. Department of Computer Science. Lund University. Retrieved from

-
- <http://research.microsoft.com/users/joshuago/conference/papers-2005/144.pdf> on June 19, 2008
- 12 Image Based CAPTCHA Is Fast Losing It's Appeal. Phil Haack. Retrieved from http://haacked.com/archive/2005/01/20/Image_Based_CAPTCHA_Losing_Appeal.aspx on February 18, 2008.
- 13 Akismet. Retrieved from <http://akismet.com/> on March 17, 2009.
- 14 Mollom. Retrieved from <http://mollom.com> on March 17, 2009.
- 15 Defensio. Retrieved from <http://defensio.com> on March 17, 2009.
- 16 Color Vision. Cal Henderson. Retrieved from <http://www.iamcal.com/toys/colors/stats.php> on October 20th 2007
- 17 Hannah Montana Tickets on Sale! Oops, They're Gone. New York Times. Randall Stross. December 16, 2007. Retrieved from <http://www.nytimes.com/2007/12/16/business/16digi.html?ref=business> on February 18, 2008
- 18 Complex Image Recognition and Web Security. Baird. CSE Lehigh University. Bethlehem, PA. Retrieved From http://www.cse.lehigh.edu/~baird/Pubs/cirws_dcpr.pdf on June 26, 2008.
- 19 CAPTCHA Effectiveness. Atwood. Coding Horror. Retrieved from <http://www.codinghorror.com/blog/archives/000712.html> on June 26, 2008
- 20 Spammers Employ Stripper to Crack CAPTCHAs. Keizer. Computerworld. October 30, 2007. Retrieved from <http://computerworld.com/action/article.do?command=viewArticleBasic&articleId=9044779> on June 26, 2008.
- 21 Using Machine Learning to Break Visual Human Interaction Proofs (HIPs). Chellapilla, Simard. Retrieved from http://research.microsoft.com/~kumarc/pubs/chellapilla_nips04.pdf on June 19, 2008.

-
- 22 Breaking a Visual CAPTCHA. Greg Mori and Jitendra Malik. Retrieved from <http://www.cs.sfu.ca/~mori/research/gimpy/> on February 13, 2008
- 23 Breaking a Visual CAPTCHA: High Level Description. Greg Mori and Jitendra Malik. Retrieved from http://www.cs.sfu.ca/~mori/research/gimpy/high_level.html on March 5, 2008.
- 24 PWNTcha CAPTCHA Decoder. Sam Hocevar. Retrieved from <http://sam.zoy.org/pwntcha/> on February 20, 2008
- 25 CaptchaPHP. Milky. FreshMeat. Retrieved from <http://freshmeat.net/p/captchaphp> on March 10, 2009.
- 26 FreeCap. Retrieved from <http://www.freecap.ru> on March 10, 2009.
- 27 GOTCHA PHP Implementation of CAPTCHA. Smart Friend Network. Retrieved from <http://phpbtree.com/captcha/> on December 5, 2007.
- 28 GOCR. Retrieved from <http://jocr.sourceforge.net/> on May 13, 2008.
- 29 OCRAD. The Gnu Project. Retrieved from <http://www.gnu.org/software/ocrad/ocrad.html> on May 13, 2008.
- 30 Akismet Stats. Retrieved from <http://akismet.com/stats/> on February 25, 2009.
- 31 Akismet Frequently Asked Questions. Retrieved from <http://akismet.com/faq/> on February 25, 2009.
- 32 Akismet eating my comments. Timetheif. Wordpress.com Forums. January 1, 2008. Retrieved from <http://en.forums.wordpress.com/topic/akismet-eating-my-comments> on February 25, 2009.
- 33 Mollom's Daily Scorecard. Retrieved from <http://mollom.com/scorecard> on February 25, 2009.
- 34 Websense Acquires Defensio Extending Web 2.0 Security Capabilities. January 27, 2009. Retrieved from <http://defensio.com/pdf/acquisition.pdf> on February 25, 2009.

-
- 35 CWA survey: average broadband speed in US is 1.9Mbps. Cheng. Ars Technica. Retrieved from <http://arstechnica.com/tech-policy/news/2007/05/survey-average-broadband-speed-in-us-is-1-9mbps.ars> on February 11, 2009.
- 36 Make Commenting Easy. Akismet Blog. February 4, 2009. Retrieved from <http://blog.akismet.com/2009/02/04/make-commenting-easy/> on February 25, 2009.
- 37 The Economics of SPAM. Leyden. The Register. November 18, 2003. Retrieved from http://www.theregister.co.uk/2003/11/18/the_economics_of_spam/ on June 26, 2008.

Appendixes

Formatting Conventions

Filenames and paths to specific files will be listed in italics:

/usr/bin/vi

Commands that are intended to be typed in verbatim are represented in mono-type:

```
mysql
```

```
CREATE TABLE Bmcb;
```

Variables or options in commands will be specified using mono-type combined with brackets and underlines:

```
./runCommand.sh [dryRun | fullTest ]
```

Class names that are traditionally camel cased in code are separated out into individual words with each leading character capitalized and kept in the standard font:

Image Magick Filter Test Workflow.

Code inline with the paragraphs will be displayed in monotype with function names ended with parentheses.

```
myFunction()
```

Appendix A – Installing the Engine

The core of the engine is written in Java and it accesses several external programs to perform tasks. Although many of the external programs are portable and the application should be able to run on any system, it was primarily developed and tested on a Linux system. The requirements listed below are for the configuration I used during testing. BMCB may work with older or newer versions of the tools listed and customized versions may not require all the tools listed for filtering and analysis.

Requirements and Dependencies

- Java 1.5 or higher runtime environment and compiler
- PHP for CAPTCHA generation (any recent build of PHP 4 or 5 with gd support should work)
- MySQL 5.0.26 or higher
- ImageMagick 6.4 or higher
- Apache Ant 1.7.0 or higher

Downloading the Source Files

The latest version of the BMCB engine, as well as this installation document, can be found at <http://penguindreams.org/page/see/Bmcb>

Unpacking the Source Files

After downloading and saving the compressed source file, open a terminal and change to the directory it was saved to. Then run the tar command with appropriate parameters to extract the archive, replacing <version> with the appropriate version of the application.

```
tar xvfj Bmcb-<version>.tar.bz2
```

Compiling the Engine

An *ant* build file is included to easily compile the engine into a single jar file. To start the compilation, run the *ant* application in the currently working directory where the source code was extracted.

```
ant jar
```

Setting up the Database

An instance of MySQL must be running either on the system the engine is installed onto or on a remote server. For the purposes of this installation, it is assumed the engine and MySQL server are on the same computer.

A database must be created for the engine, permissions must be assigned to the database and the schema for the engine must be imported. This can all be done from the MySQL command line utility which can be run by simply running `mysql` from the command prompt. If a root password has been defined, you may have to run `mysql -u root -p` and then enter your password when prompted. From the prompt, run the create

and grant statements necessary to establish the database and issue permissions on it.

```
CREATE DATABASE bmcdb;
```

```
GRANT ALL ON bmcdb.* TO 'bmcdb'@'localhost' IDENTIFIED BY 'bmcdb';
```

The Configuration Files

Logging is provided using the Log4j libraries. The engine tries to locate the *log4j.property* file in the current working directory. The default configuration should be adequate for most users. To further tune the logging, documentation for log4j can be found at <http://logging.apache.org/log4j/1.2/>.

Program specific configuration settings are found in the *bmcdb.config* file located in the programs current working directory. It contains path names, database attributes and various other runtime configuration options needed for the engine. It will need to be adjusted for the particular environment on which it is installed.

Attribute	Description
db_host	Hostname of Database Server
db_user	Database Username
db_password	Database Password
db_database	Name of Database to use
db_port	Database port
cp_generators	Directory containing CAPTCHA generators
cp_setdirs	Directory to write dataset images
cp_tmpdirs	Directory to store temporary data
cp_results	Directory to store result graphs
cp_imagemagick	Full path to directory containing ImageMagick executables
an_ocrad	Full path to OCRAD executable
an_gocr	Full path to GOCR executable

Running the Program

The application can be run directly from the jar file so long as it is run from the directory which contains the */lib* directory. If the application jar needs to be relocated outside of the distribution directory, the *manifest.txt* must be modified to contain the path to the external libraries located in */lib* and the application must be recompiled. To run the

engine, execute the jar file.

```
java -java Bmcb.jar
```

Running the jar will produce a usage statement.

```
Usage: Bmcb [generate|trivialTest|magickTest|SegmentDebug]
```

By default, BMCB can run several of the built-in workflows included with the application. They include a generation workflow for creating CAPTCHA datasets and two testing workflows for analyzing CAPTCHA and generating results. Using the *SegmentDebug* option will start the graphical debugging tool displaying generated datasets.

Appendix B – High Level Overview of the Engine

Looking at the engine from the top down, all the tasks that are performed are in the form of workflows. Workflows can be used to generate sets of CAPTCHAs to test against or perform testing and analysis and gather statistics. A workflow is just a simple Java class that is used to call all the other components of the framework. They can be highly customizable to perform any type of experiment and have full access to all the other public components in the framework.

Components accessible from the workflows include a variety of tools for experiments including CAPTCHA generators, image filters, segmentators and analyzers. Storage of generated datasets can be handled using the database classes and a variety of static utility functions exist for the purpose of loading images, converting image types, logging and generating result graphs [Figure 28].

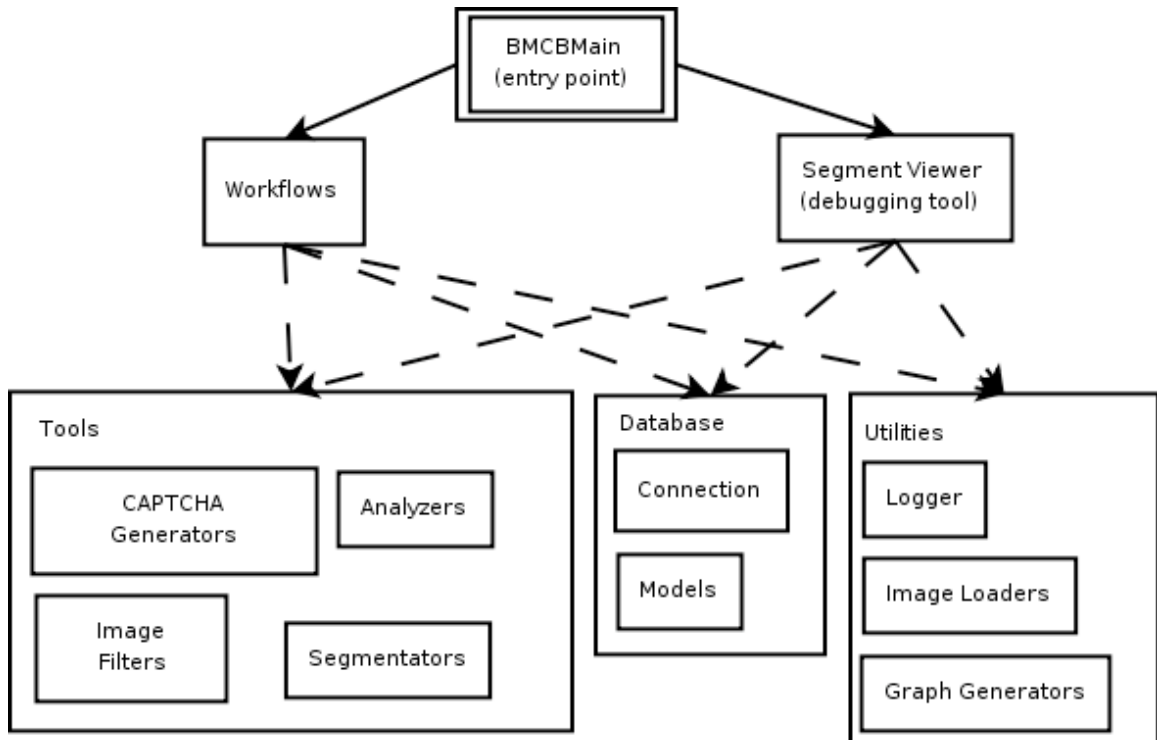


Figure 28 - High Level Overview of BMCB Engine

Generators

The Abstract Generator class provides the basis for any type of generator. The primary included generator is the Command Line Generator, which calls an external program with two arguments, the first being the text to be placed in the image and the second being the directory to write the image to. The Abstract Generator can also be extended for other purposes such as generation via an HTTP request, in the case where the CAPTCHA is generated on a non-UNIX machine such as from an active server page (ASP), or a Java based generator which uses existing Java classes to generate a CAPTCHA.

CAPTCHA programs typically generate their challenge image text randomly.

Within the engine the text to be generated needs to be controlled in order to determine if the analysis of the text is correct. The generated text is still random, but the correct response needs to be stored in a database for comparison. Because of this, CAPTCHA generators need to be modified slightly in order to work correctly with the generator classes. For this reason, it is best to use open source CAPTCHA programs which can easily be modified and adapted to work with the engine.

Segmentators

Segmentators are based off the Abstract Segmentator class. They are given the argument of an image and are expected to return either a series of axes where a break between letters would occur or an array of image objects that have been pre-segmented. A given segmentation algorithm will extend this class and implement all the abstract methods.

Filters

All filters are based on the Abstract Filter class. A filter is a preprocessor for an image before it is used with a segmentator or analyzer. It takes in an image object and applies some algorithm to the image to attempt to clean up general noise and distortion. The filter is applied directly on the image passed into this object, so if an original copy is needed for any purpose, it should be made before the filter is applied.

Analyzers

Analyzers are based off the Abstract Analyzer class. They perform the core of the analysis process and can act upon either an entire image or an array of image segments.

Abstract functions are also defined for learning and training the analyzer if a learning style algorithm is used. Implementations of this class can chose to throw an Analysis Not Supported Exception in cases where a particular algorithm may not have the ability to learn or to accept image segments.

Workflows

Workflows are based on the Abstract Workflow class. They have an execution method and are used to knit together all the steps and individual pieces necessary to perform a task. For example, the Generator Workflow is executed to look through every available CAPTCHA generator and create datasets for testing. Workflows can be used to run a particular set of tasks for an experiment using a given set of images, segmentators, filters and analyzers, as well as calculate the results.

Utility Classes

Several utility classes also exist to help deal with miscellaneous tasks required through the course of the program. They include common application classes such as those needed for reading configuration files and application logging as well as more program specific classes for loading and storing images in various formats and rendering charts and graphs from result sets.

Appendix C – Developing with the Engine

The BMCB Engine is comprised of several components including the Generators, Segmentators, Image Filters, Analyzers and Utilities. All these components are called within a Workflow. New workflows must be added to the entry point of the program. This guide is intended for developers who want to modify the engine for their own analysis techniques and build new experiments into the engine.

This document follows a bottom up approach focusing on the individual components and building them into a full experiment or workflow. Many useful examples are included with the engine itself and should be read alongside this document to gain a full understanding of how to build new experiments into the existing framework.

Utility Classes

There are several independent utility classes contained within the framework that are used throughout the application. Many of these classes contain static standalone methods for basic tasks such as image conversion, logging, configuration, result graphing and various other common tasks [Figure 29 - Diagram of Utility Classes].

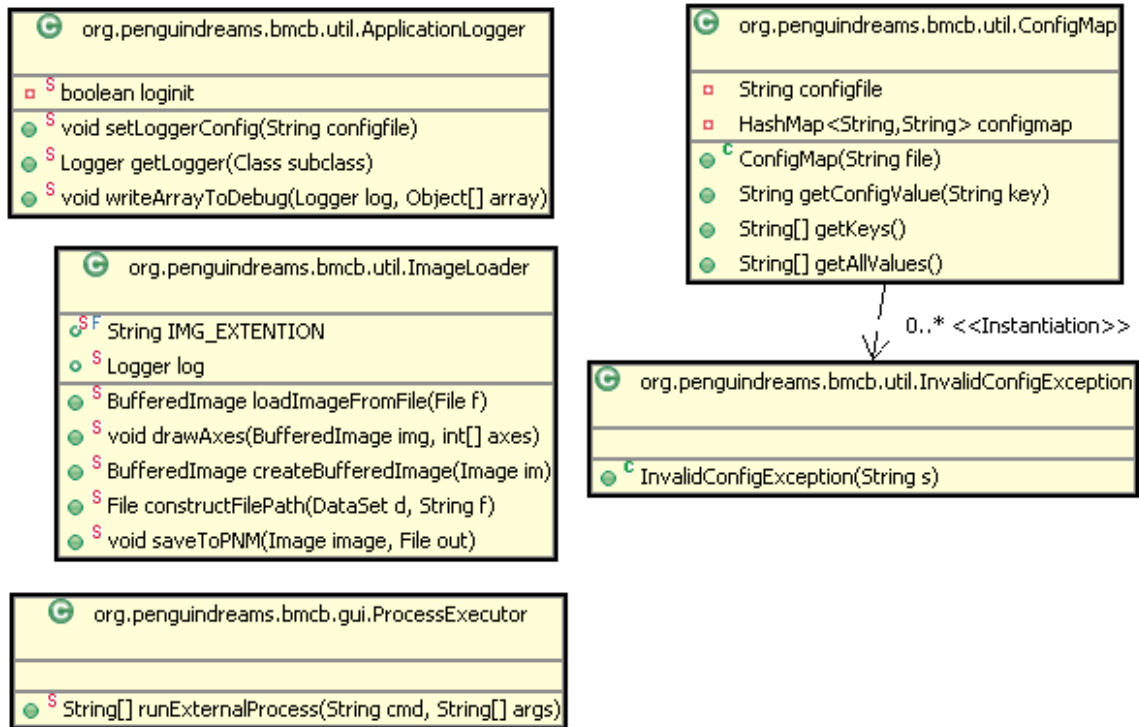


Figure 29 - Diagram of Utility Classes

Some of the more important utilities are as follows:

- **Application Logger:** This class is called from many of the abstract classes in the framework to initialize a protected log variable. Developers shouldn't need to call this directly unless they create a new class from scratch, as the existing log variable is accessible in nearly every abstract class. Examples for creating new instances are located in the abstract classes.
- **Image Loader:** This class contains several functions to assist with image manipulation including image loading, converting between Images and Buffered Images, determining the path of an image from a dataset, converting images to

PNM files used by OCR programs, and drawing lines on segment boundaries on images to be used with the Segment Viewer debugging tool.

- Process Executor: A simple class to handle the basic task of running an external process
- Config Map: Used to read settings from the bmcb.property file

Generators

The engine comes with a command line generation class which passes two arguments to a command line application, the first being the CAPTCHA to be produced and the second being the location for the output file. This process can be seen in the built-in generation workflow where random challenge/responses are generated, stored in a database and then generated for each CAPTCHA type.

Typically, a CAPTCHA application generates its challenge randomly and does not accept a challenge as an argument. Therefore, the CAPTCHA script needs to be modified. For the purposes of using this engine, it is best to use open source CAPTCHAs which can easily be modified. The following changes may be necessary before using a CAPTCHA script with this engine:

- Modify the program to take in the CAPTCHA phrase as the first argument
- Write the output to a file given by the second argument
- Adjust the front path to be independent from the script's location

The following examples detail how these three modifications can be performed in a typical PHP based CAPTCHA script. Each CAPTCHA script a developer wants to

incorporate will require different modifications to be compatible with the built in Command Line Generator, or it may require a custom generator.

Modifying the CAPTCHA application to take the input from a command line, rather than generating it randomly, can be done in several ways. The developer can modify the function that generates the random CAPTCHA or the point at which the key is saved to the session can be modified [Figure 30].

```
#added by Sumit
#$_SESSION[CAPTCHA_SESSION_KEY] = $text = substr($t, rand(0, (strlen($t)-6)), rand(3,6));
$text = $argv[1];
```

Figure 30 - Modification to Gotcha to Take Challenge Input

Most CAPTCHA scripts have a function used to generate a random phrase or set of letters. This section is what must be modified in order to use the script with the engine. The type of modification will vary depending on the programming language used and may require the creation of a customized generation class.

Typically, most CAPTCHA scripts are designed to output directly to a web browser. This behavior must also be modified to write the file, with the challenge solution as the filename, to a directory [Figure 31]. As with the previous modification, this will vary heavily depending on the programming language of the script and the script itself.

```

/*
switch($output)
{
    // add other cases as desired
    case "jpg":
        header("Content-Type: image/jpeg");
        ImageJPEG($pic);
        break;
    case "gif":
        header("Content-Type: image/gif");
        ImageGIF($pic);
        break;
    case "png":
    default:
        header("Content-Type: image/png");
        ImagePNG($pic);
        break;
}*/
//Modified by Sumit to write captcha file out
global $argv;
imagepng($pic,$argv[2]);

```

Figure 31 - Modification of Freecap to Output Challenge to a File

Another modification that may or may not be necessary involves modifying the path for included and dependent files. For many of the scripts included with the engine, this involves modifying the statement that declared which font to use to be working directory independent [Figure 32].

```

//ADDED by Sumit -- Adjusting Font Paths
$FONT_DIR = dirname(__FILE__);
$fonts = Array();
$fonts += glob($FONT_DIR."/*.ttf");
foreach($fonts as $f) {
    $t->addFont($f);
}
//$t->addFont('SFTransRoboticsExtended.ttf');
//$t->addFont('arialbd.ttf');

```

Figure 32 - Modification of Gotcha for Font Path

The above modification may or may not be necessary depending on the way the script loads its fonts and dependencies.

The above modifications are only some of the changes that may need to be made to a CAPTCHA application in order to get it to work with the engine. Developers may run into other challenges, however most CAPTCHA should be adaptable, either by extending the Abstract Command Line Generator or by creating a custom generator class, so long as the CAPTCHA application provides some means for manually inputting the challenge response.

Image Filters

Image Filters extend the Abstract Filter class. They must modify a Buffered Image that is passed to the filter by reference. If the calling class requires an unaltered version of the Buffered Image, it must clone a copy before passing it to the filter.

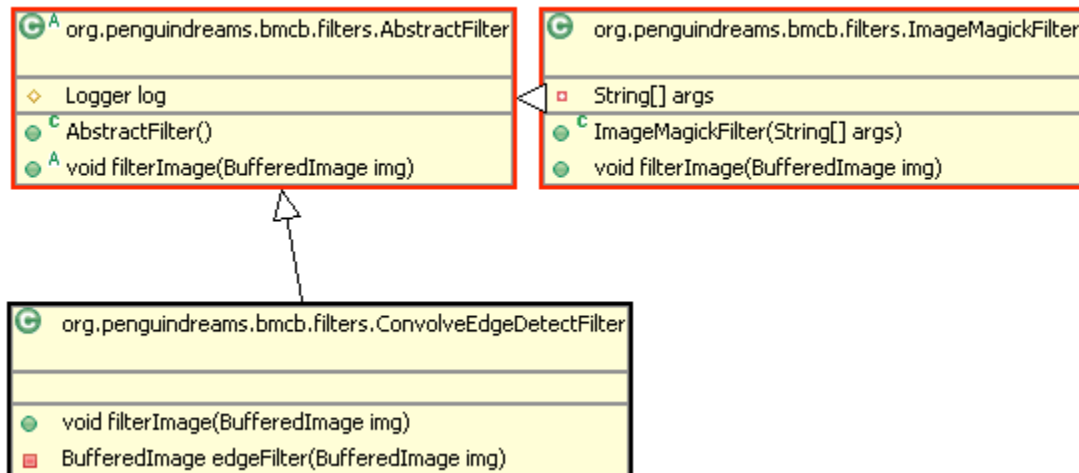


Figure 33 - Class Diagram for Filters

In addition, if a new Buffered Image is created during the filtering process, it can be copied into the passed in argument. An example is the function for edge filtering which returns a new Buffered Image object[Figure 34].

```
public void filterImage(BufferedImage img) {  
  
    BufferedImage t = edgeFilter(img);  
    img.getGraphics().drawImage(t,0,0,null);  
}
```

Figure 34 - Copying One Buffered Image to Another

Segmentators

Segmentators are based on the Abstract Segmentator class. An instance of all the segmentator objects is created specifically for an image. Derived classes must implement the abstract `getSegmentAxes()` function which returns the x coordinate where the image is split into separate vertical segments. Various other functions in the base class can be called to split the image into individual arrays of Buffered Image objects to be used within the workflows [Figure 35].

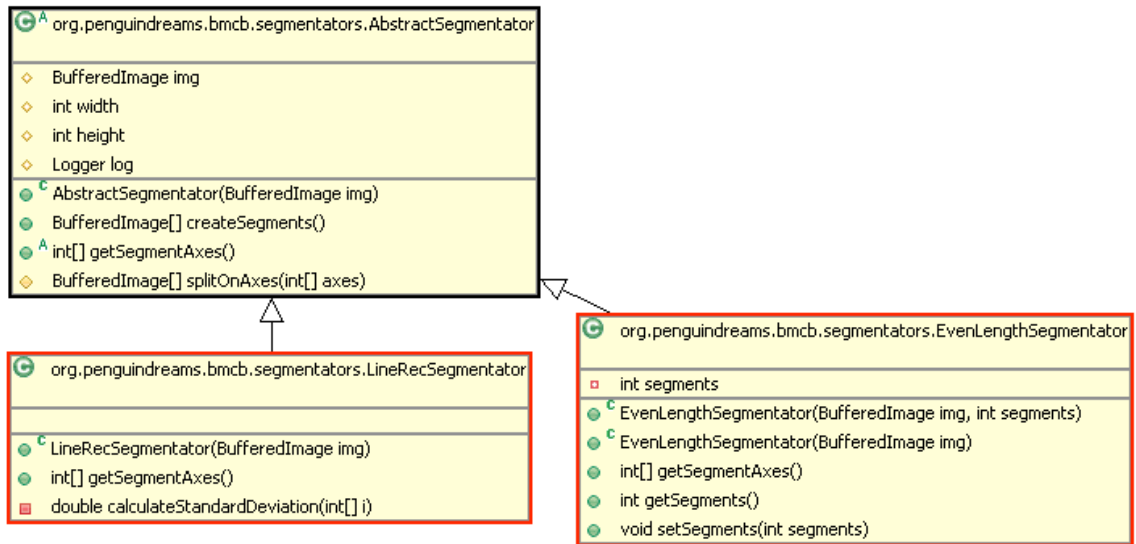


Figure 35 - Class Diagram for Segmentators

Analyzers

Analyzers are what actually try to solve the CAPTCHA challenge after appropriate filters and segmentation have been applied. The Abstract Analyzer has been designed to contain several functions in the case of learning and non-learning algorithms as well as different functions for analyzing segments as opposed to full images. Analyzers which do not support a given abstract function can choose to throw an Analysis Not Supported Exception. An abstract Command Line Analyzer has been included to assist in the process of calling an external application for analysis [Figure 36].

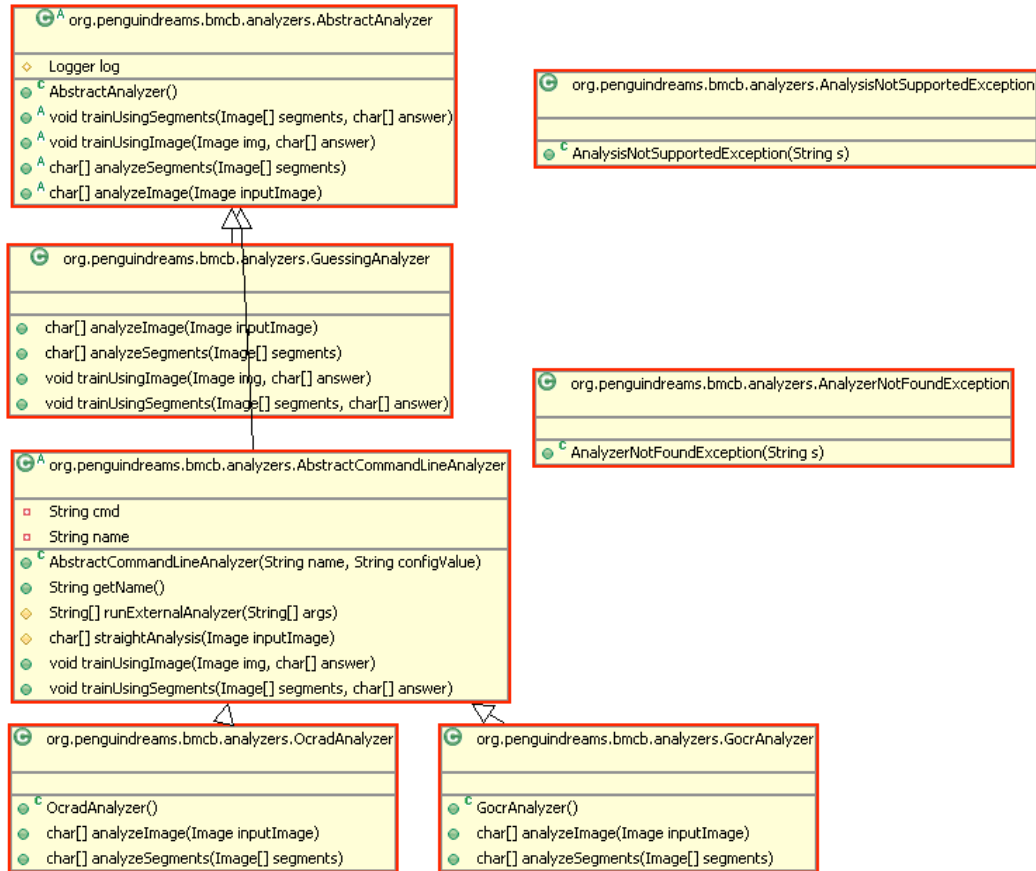


Figure 36 - Class Diagram for Analyzers

Workflows

The piece that ties all the individual components together is the workflow. The workflow calls all the individual pieces listed so far and can be used for generation of set data, analysis or any various other tasks. The workflows included with the engine are used for data generation, analysis and testing. Developers will want to either modify existing workflows or create new workflows for whatever experiments they may wish to perform [Figure 37].

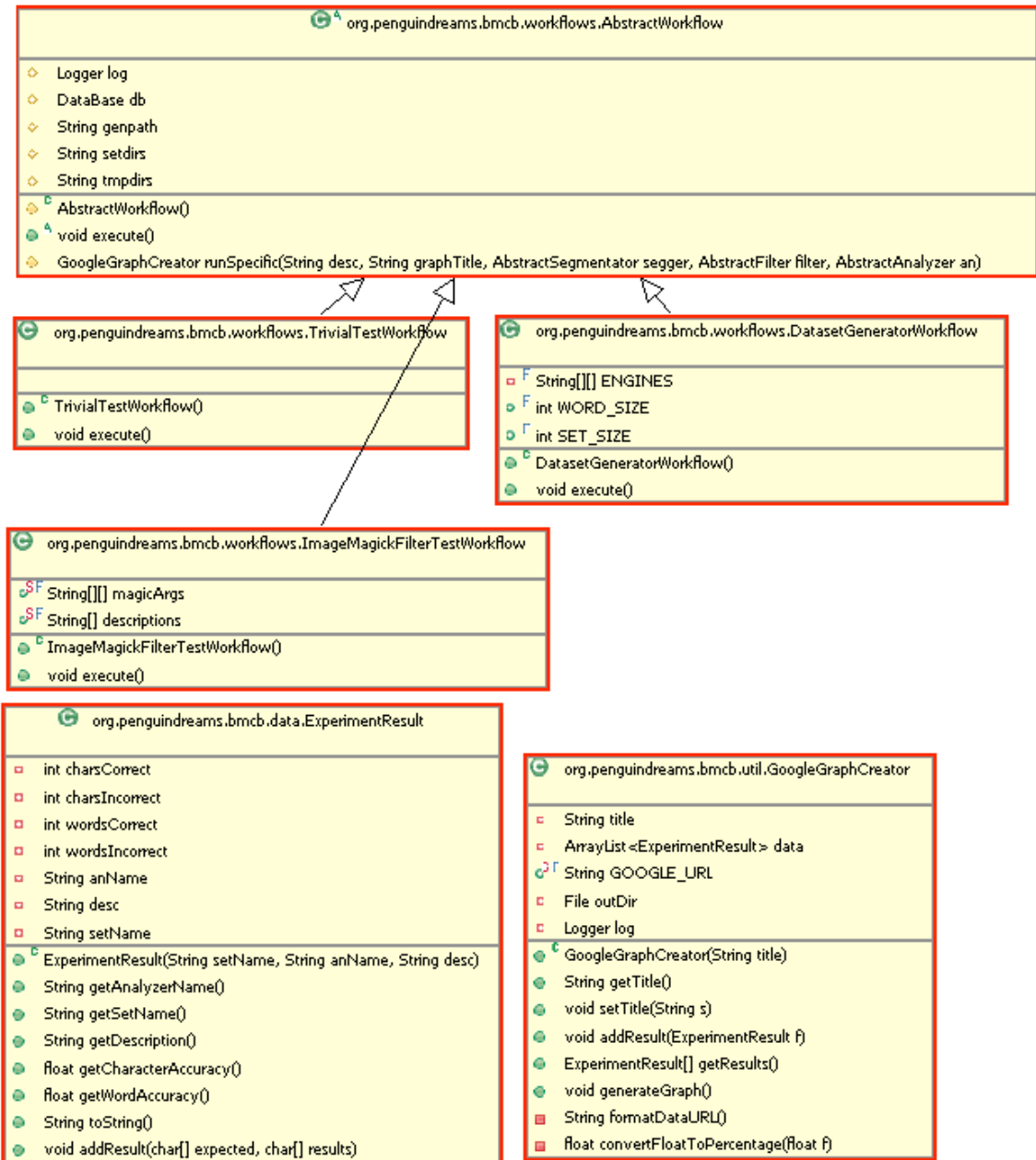


Figure 37 - Class Diagram for Workflows

The Abstract Workflow class contains a protected `runSpecific()` function which takes in all the individual components including a Segmentator, Filter and

Analyzer, along with a description for results and logging purposes, and runs them against all the currently available datasets.

Entry Point

The BMCBMain class provides the primary entry point for the application. After creating additional workflows, an appropriate command line argument or set of command line arguments will be need to be added to the main function to kick off the workflow.

Visual Debugging Tools

There is also a Segment Viewer packaged with the engine. It is a graphical tool to view data sets as well as the results from segmentators, image filters and analyzers [Figure 38]. Newly created Segmentators, Analyzers and Image Filters can easily be added to the Segment Viewer.

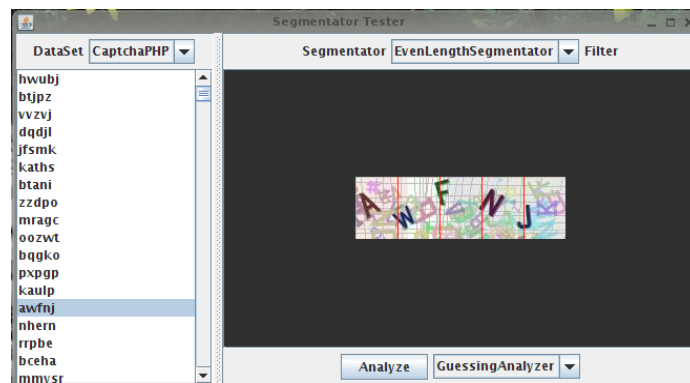


Figure 38 - Visual Debugger

Appendix D – FlickMeCaptcha

Installation

The most current version of FlickMeCaptcha can be found at <http://penguindreams.org/projects/flickmecaptcha> in the download section. Simply download the appropriate tar file and extract it to a working directory.

```
tar xvfj flickmecaptcha-<version>.tar.gz
```

The compressed tar file contains *FlickMeCaptcha.php* which is the primary script and only one necessary to integrate the application into an existing website. There is also a script named *example.php* which shows how to integrate the application into a form and provides a means for testing the application on a web server.

There are several configuration options,, which can be set in the *FlickMeCaptcha.php* file. Most of these options can be left at their default. The only setting which must be changed is the `scriptFile` variable which must point to the web path where the script is located. Other variables purposes and settings are documented within the script file itself.

Integration

To present the CAPTCHA within a web form, the *FlickMeCaptcha.php* script must be included in the current PHP file followed by a call to the `getChallenge()` function [Figure 39]. Please note that the session must be started using the

`session_start()` function in PHP before creating or verifying the CAPTCHA.

```
<form id="test" method="post" action="">
<p>
  <label>Name:</label><br />
  <input type="text" name="name" /><br />
  <label>Message:</label><br />
  <textarea name="message" rows="20" cols="20"></textarea><br/>
  <input type="hidden" name="cmd" value="check" />
  <br />
  <? echo Captcha::getChallenge(); ?>
  <br />
  <input type="submit" value="submit" />
</p>
</form>
```

Figure 39 - Adding FlickMeCaptcha to a Form

Once the form has been submitted, the CAPTCHA can be verified for correctness using the `checkChallenge()` function [Figure 40].

```
<?php
if($_POST['cmd'] == 'check') {
  if(Captcha::checkChallenge()) {
    echo '<div id="correct">You Answered the Challenge Correctly.</div>';
  }
  else {
    echo '<div id="incorrect">You Answered the Challenge Incorrectly.</div>';
  }
}
?>
```

Figure 40 - Verifying FlickMeCaptcha Challenge