# CAPTCHA

## Seminar Report

Submitted in partial fulfillment of the requirements

for the award of the degree of

**Bachelor of Technology**

*in*

**Computer Science Engineering**

*of*

**Cochin University of Science And Technology**

*by*


## Gibu Thomas Mathew




**DIVISION OF COMPUTER SCIENCE**

**SCHOOL OF ENGINEERING**

**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY**

KOCHI-682 022


AUGUST 2010

**DIVISION OF COMPUTER SCIENCE**

**SCHOOL OF ENGINEERING**

**COCHIN UNIVERSITY OF SCIENCE AND TECHNOLOGY**

**KOCHI-682022**

# Certificate

Certified that this is a bonafide record of the seminar entitled

**CAPTCHA**

presented by the following student

**Gibu Thomas Mathew**

of the VII semester, Computer Science and Engineering in the year 2010 in partial fulfillment of the requirements in the award of Degree of Bachelor of Technology in Computer Science and Engineering of Cochin University of Science and Technology.

**Mrs. ANUPAMA V.**
SEMINAR GUIDE

**Dr. DAVID PETER**
HEAD OF DIVISION

# ACKNOWLEDGEMENT

I thank **GOD** almighty for guiding me throughout the seminar. I would like to thank all those who have contributed to the completion of the seminar and helped me with valuable suggestions for improvement.

I am extremely grateful to **Dr. David Peter, Head of Division, Computer Science,** for providing me with best facilities and atmosphere for the creative work guidance and encouragement. I would also like to thank my coordinator **Mr.Sudheep Elayidom, Lecturer, Division of Computer Science** and my guide **Mrs. Anupama V.** for all the help and support extended to me. I thank all the Staff members of my college and friends for extending their cooperation during my seminar.

Above all I would like to thank my parents without whose blessings; I would not have been able to accomplish my goal.

**GIBU THOMAS MATHEW**

# ABSTRACT

CAPTCHA is an acronym for "**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part." It is a type of challenge-response test used in computing to ensure that the response is not generated by a computer. The process usually involves one computer (a server) asking a user to complete a simple test which the computer is able to generate and grade. Because other computers are unable to solve the CAPTCHA, any user entering a correct solution is presumed to be human. A common type of CAPTCHA requires that the user type letters or digits from a distorted image that appears on the screen.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

You're trying to sign up for a free email service offered by Gmail or Yahoo. Before you can submit your application, you first have to pass a test. It's not a hard test -- in fact, that's the point. For you, the test should be simple and straightforward. But for a computer, the test should be almost impossible to solve.

This sort of test is a **CAPTCHA**. They're also known as a type of **Human Interaction Proof** (**HIP**). You've probably seen CAPTCHA tests on lots of Web sites. The most common form of CAPTCHA is an image of several distorted letters. It's your job to type the correct series of letters into a form. If your letters match the ones in the distorted image, you pass the test.

CAPTCHAs are short for Completely Automated Public Turing test to tell Computers and Humans Apart. The term "CAPTCHA" was coined in 2000 by Luis Von Ahn, Manuel Blum, Nicholas J. Hopper (all of Carnegie Mellon University, and John Langford (then of IBM). They are challenge-response tests to ensure that the users are indeed human. The purpose of a CAPTCHA is to block form submissions from spam bots – automated scripts that harvest email addresses from publicly available web forms. A common kind of CAPTCHA used on most websites requires the users to enter the string of characters that appear in a distorted form on the screen.

CAPTCHAs are used because of the fact that it is difficult for the computers to extract the text from such a distorted image, whereas it is relatively easy for a human to understand the text hidden behind the distortions. Therefore, the correct response to a CAPTCHA challenge is assumed to come from a human and the user is permitted into the website.

Why would anyone need to create a test that can tell humans and computers apart? It's because of people trying to **game** the system -- they want to exploit weaknesses in the computers running the sites. While these individuals probably make up a minority of all

the people on the Internet, their actions can

affect millions of users and Web sites. For example, a free e-mail service might find itself bombarded

by account requests from an automated program. That automated program could be part of a larger attempt to send out spam mail to millions of people. The CAPTCHA test helps identify which users are real human beings and which ones are computer programs.

Spammers are constantly trying to build algorithms that read the distorted text correctly. So strong CAPTCHAs have to be designed and built so that the efforts of the spammers are thwarted.

## 1.2 Motivation

The proliferation of the publicly available services on the Web is a boon for the community at large. But unfortunately it has invited new and novel abuses. Programs (bots and spiders) are being created to steal services and to conduct fraudulent transactions. Some examples:

- Free online accounts are being registered automatically many times and are being used to distribute stolen or copyrighted material.
- Recommendation systems are vulnerable to artificial inflation or deflation of rankings. For example, EBay, a famous auction website allows users to rate a product. Abusers can easily create bots that could increase or decrease the rating of a specific product, possibly changing people's perception towards the product.
- Spammers register themselves with free email accounts such as those provided by Gmail or Hotmail and use their bots to send unsolicited mails to other users of that email service.
- Online polls are attacked by bots and are susceptible to ballot stuffing. This gives unfair mileage to those that benefit from it.

In light of the above listed abuses and much more, a need was felt for a facility that checks users and allows access to services to only human users. It was in this

direction that such a tool like CAPTCHA was created.

## 1.3 Background

The need for CAPTCHAs rose to keep out the website/search engine abuse by bots. In 1997, **AltaVista** sought ways to block and discourage the automatic submissions of URLs into their search engines. Andrei Broder, Chief Scientist of AltaVista, and his colleagues developed a filter. Their method was to generate a printed text randomly that only humans could read and not machine readers. Their approach was so effective that in an year, "spam-add-ons'" were reduced by 95% and a patent was issued in 2001.

In 2000, **Yahoo**'s popular **Messenger** chat service was hit by bots which pointed advertising links to annoying human users of chat rooms. Yahoo, along with Carnegie Mellon University, developed a CAPTCHA called EZ-GIMPY, which chose a dictionary word randomly and distorted it with a wide variety of image occlusions and asked the user to input the distorted word.

In November 1999, s*lashdot.com* released a poll to vote for the best CS college in the US. Students from the Carnegie Mellon University and the Massachusetts Institute of Technology created bots that repeatedly voted for their respective colleges. This incident created the urge to use CAPTCHAs for such online polls to ensure that only human users are able to take part in the polls.

## 1.4 CAPTCHAs and the Turing Test

CAPTCHA technology has its foundation in an experiment called the **Turing Test**. Alan Turing, sometimes called the father of modern computing, proposed the test as a way to examine whether or not machines can think -- or appear to think -- like humans. The classic test is a game of imitation. In this game, an interrogator asks two participants a series of questions. One of the participants is a machine and the other is a human. The interrogator can't see or hear the participants and has no way of knowing which is which. If the interrogator is unable to figure out which participant is

a machine based on the responses, the machine passes the Turing Test.

Of course, with a CAPTCHA, the goal is to create a test that humans can pass easily but machines can't. It's also important that the CAPTCHA application is able to present

different CAPTCHAs to different users. If a visual CAPTCHA presented a static image that was the same for every user, it wouldn't take long before a spammer spotted the form, deciphered the letters, and programmed an application to type in the correct answer automatically.

Most, but not all, CAPTCHAs rely on a visual test. Computers lack the sophistication that human beings have when it comes to processing visual data. We can look at an image and pick out patterns more easily than a computer. The human mind sometimes perceives patterns even when none exist, a quirk we call pareidolia. Ever see a shape in the clouds or a face on the moon? That's your brain trying to associate random information into patterns and shapes.

But not all CAPTCHAs rely on visual patterns. In fact, it's important to have an alternative to a visual CAPTCHA. Otherwise, the Web site administrator runs the risk of disenfranchising any Web user who has a visual impairment. One alternative to a visual test is an audible one. An audio CAPTCHA usually presents the user with a series of spoken letters or numbers. It's not unusual for the program to distort the speaker's voice, and it's also common for the program to include background noise in the recording. This helps thwart voice recognition programs.

Another option is to create a CAPTCHA that asks the reader to interpret a short passage of text. A contextual CAPTCHA quizzes the reader and tests comprehension skills. While computer programs can pick out key words in text passages, they aren't very good at understanding what those words actually mean.

# CHAPTER 2

# Type of CAPTCHAs

CAPTCHAs are classified based on what is distorted and presented as a challenge to the user. They are:

## 2.1 Text CAPTCHAs

These are simple to implement. The simplest yet novel approach is to present the user with some questions which only a human user can solve. Examples of such questions are:

1. What is twenty minus three?
2. What is the third letter in UNIVERSITY?
3. Which of Yellow, Thursday and Richard is a colour?
4. If yesterday was a Sunday, what is today?

Such questions are very easy for a human user to solve, but it's very difficult to program a computer to solve them. These are also friendly to people with visual disability – such as those with colour blindness. Other text CAPTCHAs involves text distortions and the user is asked to identify the text hidden. The various implementations are:

### 2.1.1 Gimpy

Gimpy is a very reliable text CAPTCHA built by CMU in collaboration with Yahoo for their Messenger service. Gimpy is based on the human ability to read extremely distorted text and the inability of computer programs to do the same. Gimpy works by choosing ten words randomly from a dictionary, and displaying them in a distorted and overlapped manner. Gimpy then asks the users to enter a subset of the words in the image. The human user is capable of identifying the words

correctly, whereas a computer program cannot.

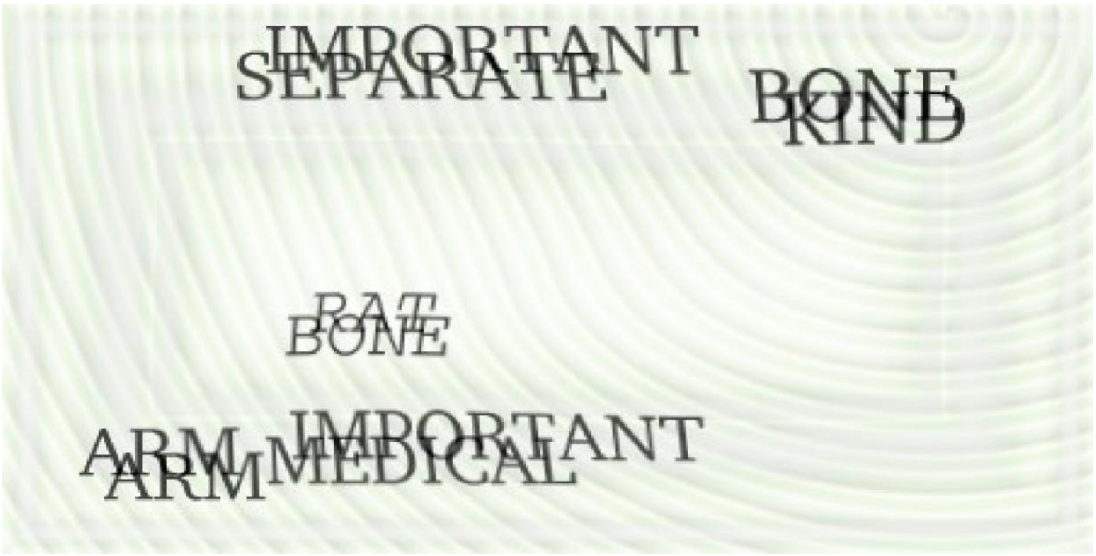



*Fig 2.1 Gimpy CAPTCHA*

### 2.1.2 Ez – Gimpy:

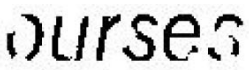
This is a simplified version of the Gimpy CAPTCHA, adopted by Yahoo in their signup page. Ez – Gimpy randomly picks a single word from a dictionary and applies distortion to the text. The user is then asked to identify the text correctly.

*Fig 2.2 Yahoo's Ez – Gimpy CAPTCHA*

### 2.1.3 Baffle Text

This was developed by Henry Baird at University of California at Berkeley. This is a variation of the Gimpy. This doesn't contain dictionary words, but it picks up random alphabets to create a nonsense but pronounceable text. Distortions are then added to this text and the user is challenged to guess the right word. This technique overcomes the drawback of Gimpy CAPTCHA because, Gimpy

uses dictionary words and hence, clever bots could be designed to check the dictionary for the matching word by brute-force.


ourses

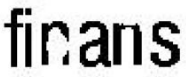

finans

*Fig 2.3 Baffle Text examples*

### 2.1.4 MSN CAPTHCHA

Microsoft uses a different CAPTCHA for services provided under MSN umbrella. These are popularly called MSN Passport CAPTCHAs. They use eight characters (upper case) and digits. Foreground is dark blue, and background is grey. Warping is used to distort the characters, to produce a ripple effect, which makes computer recognition very difficult.


XTNM5YRE


L9D28229B

*Fig 2.4* ***MSN Passport CAPTCHA***

## 2.2 Graphic CAPTCHAs

Graphic CAPTCHAs are challenges that involve pictures or objects that have some sort of similarity that the users have to guess. They are visual puzzles, similar to Mensa tests. Computer generates the puzzles and grades the answers, but is itself unable to solve it.

### 2.2.1 Bongo

Bongo. Another example of a CAPTCHA is the program we call BONGO [2]. BONGO is named after M.M. Bongard, who published a book of pattern recognition problems in the 1970s [3]. BONGO asks the user to solve a visual pattern recognition problem. It displays two series of blocks, the left and the right. The blocks in the left series differ from those in the right, and the user must find the characteristic that sets them apart. A possible left and right series is shown in Figure 2.5

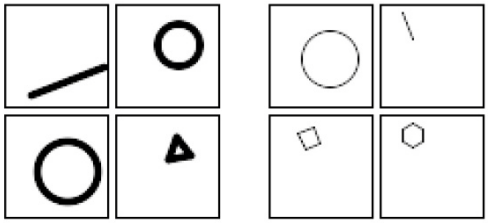


*Fig 2.5 Bongo CAPTCHA*

These two sets are different because everything on the left is drawn with thick lines and those on the right are in thin lines. After seeing the two blocks, the user is presented with a set of four single blocks and is asked to determine to which group the each block belongs to. The user passes the test if s/he determines correctly to which set the blocks belong to. We have to be careful to see that the user is not confused by a large number of choices.

### 2.2.2 PIX

PIX is a program that has a large database of labeled images. All of these images are pictures of concrete objects (a horse, a table, a house, a flower). The program picks an object

at random, finds six images of that object from its database, presents them to the user and then asks the question “what are these pictures of?” Current computer programs should not be able to answer this question, so PIX should be a CAPTCHA. However, PIX, as stated, is not a CAPTCHA: it is very easy to write a program that can answer

the question “what are these pictures of?” Remember that all

the code and data of a CAPTCHA should be publicly available; in particular, the image database that PIX uses should be public. Hence, writing a program that can answer the question “what are these pictures of?” is easy: search the database for the images presented and find their label. Fortunately, this can be fixed. One way for PIX to become a CAPTCHA is to randomly distort the images before presenting them to the user, so that computer programs cannot easily search the database for the undistorted image.

## 2.3 Audio CAPTCHAs

The final example we offer is based on sound. The program picks a word or a sequence of numbers at random, renders the word or the numbers into a sound clip and distorts the sound clip; it then presents the distorted sound clip to the user and asks users to enter its contents. This CAPTCHA is based on the difference in ability between humans and computers in recognizing spoken language. Nancy Chan of the City University in Hong Kong was the first to implement a sound-based system of this type. The idea is that a human is able to efficiently disregard the distortion and interpret the characters being read out while software would struggle with the distortion being applied, and need to be effective at speech to text translation in order to be successful. This is a crude way to filter humans and it is not so popular because the user has to understand the language and the accent in which the sound clip is recorded.

## 2.4 reCAPTCHA and Book Digitization

To counter various drawbacks of the existing implementations, researchers at CMU developed a redesigned CAPTCHA aptly called the reCAPTCHA. About 200 million CAPTCHAs are solved by humans around the world every day. In each case, roughly ten seconds of human time are being spent. Individually, that's not a lot of

time, but in aggregate

these little puzzles consume more than 150,000 hours of work each day. What if we could make positive use of this human effort? reCAPTCHA does exactly that by channeling t he effort spent

solving CAPTCHAs online into "reading" books. To archive human knowledge and to make information more accessible to the world, multiple projects are currently digitizing physical books that were written before the computer age. The book pages are being photographically scanned, and then transformed into text using "Optical Character Recognition" (OCR). The transformation into text is useful because scanning a book produces images, which are difficult to store on small devices, expensive to download, and cannot be searched. The problem is that OCR is not perfect.

reCAPTCHA improves the process of digitizing books by sending words that cannot be read by computers to the Web in the form of CAPTCHAs for humans to decipher. More specifically, each word that cannot be read correctly by OCR is placed on an image and used as a CAPTCHA. This is possible because most OCR programs alert you when a word cannot be read correctly.

But if a computer can't read such a CAPTCHA, how does the system know the correct answer to the puzzle? Here's how: Each new word that cannot be read correctly by OCR is given to a user in conjunction with another word for which the answer is already known. The user is then asked to read both words. If they solve the one for which the answer is known, the system assumes their answer is correct for the new one. The system then gives the new image to a number of other people to determine, with higher confidence, whether the original answer was correct.

Currently, reCAPTCHA is employed in digitizing books as part of the Google Books Project and its

Demonstration maybe represented as:

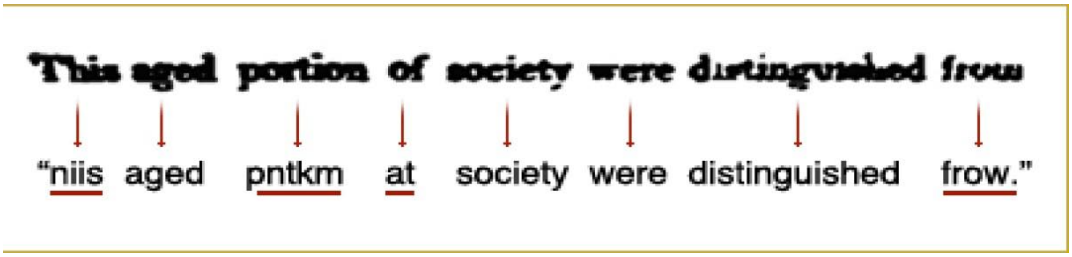



*Fig 2.6 First line shows scanned text, second line shows text read by OCR*

# Chapter 3

# Constructing CAPTCHAs

## 3.1 Things to know

The first step to create a CAPTCHA is to look at different ways humans and machines process information. Machines follow sets of instructions. If something falls outside the realm of those instructions, the machines aren't able to compensate. A CAPTCHA designer has to take this into account when creating a test. For example, it's easy to build a program that looks at **metadata** – the information on the Web that's invisible to humans but machines can read. If you create a visual CAPTCHA and the images' metadata includes the solution, your CAPTCHA will be broken in no time.

Similarly, it's unwise to build a CAPTCHA that doesn't distort letters and numbers in some way. An undistorted series of characters isn't very secure. Many computer programs can scan an image and recognize simple shapes like letters and numbers.

One way to create a CAPTCHA is to pre-determine the images and solutions it will use. This approach requires a database that includes all the CAPTCHA solutions, which can compromise the reliability of the test. According to Microsoft Research experts Kumar Chellapilla and Patrice Simard, humans should have an 80 percent success rate at solving any particular CAPTCHA, but machines should only have a 0.01 percent success rate [source: Chellapilla and Simard]. If a spammer managed to find a list of all CAPTCHA solutions, he or she could create an application that bombards the CAPTCHA with every possible answer in a **brute-force** attack. The database would need more than 10,000 possible CAPTCHAs to meet the qualifications of a good CAPTCHA.

Other CAPTCHA applications create random strings of letters and numbers. You aren't likely to ever get the same series twice. Using randomization eliminates the possibility of a brute-force attack

the odds of a bot entering the correct series of random letters are very low. The longer he string of characters, the less likely a bot will get lucky. CAPTCHAs take different approaches to distorting words. Some stretch and bend letters in weird ways, as if you're looking at the word through melted glass. Others put the word behind a crosshatched pattern of bars to break up the shape of letters. A few use different colours or a field of dots to achieve the same effect. In the end, the goal is the same: to make it really hard for a computer to figure out what's in the CAPTCHA.

Designers can also create puzzles or problems that are easy for humans to solve. Some CAPTCHAs rely on **pattern recognition** and **extrapolation**. For example, a CAPTCHA might include series of shapes and ask the user which shape among several choices would logically come next. The problem with this approach is that not all humans are good with these kinds of problems and the success rate for a human user can go below 80 percent.

## 3.2 Implementation

**Embeddable CAPTCHAs:** The easiest implementation of a CAPTCHA to a Website would be to insert a few lines of CAPTCHA code into the Website's HTML code, from an open source CAPTCHA builder, which will provide the authentication services remotely. Most such services are free. Popular among them is the service provided by www.captcha.net 's reCAPTCHA project.

**Custom CAPTCHAs** These are less popular because of the extra work needed to create a secure implementation. Anyway, these are popular among researchers who verify existing CAPTCHAs and suggest alternative implementations. There are advantages in building custom CAPTCHAs:

1. A custom CAPTCHA can fit exactly into the design and theme of your

site. It will not look like some alien element that does not belong there.

2. We want to take away the perception of a CAPTCHA as an annoyance, and make it convenient for the user.
3. Because a custom CAPTCHA, unlike the major CAPTCHA mechanisms, obscure you as a target for spammers. Spammers have little interest in cracking a niche implementation.
4. Because we want to learn how they work, so it is best to build one ourselves.

**CAPTCHA Logic**

The CAPTCHA image (or question) is generated. There are different ways to do this. The classic approach is to generate some random text, apply some random effects to it and convert it into an image.

1. The CAPTCHA image (or question) is generated. There are different ways to do this. The
   classic approach is to generate some random text, apply some random effects to it and convert it into an image.
2. Step 2 is not really sequential. During step 1, the original text (pre-altered) is persisted somewhere, as
   this is the correct answer to the question. There are different ways to persist the answer, as a server-side session variable, cookie, file, or database entry.
3. The generated CAPTCHA is presented to the user, who is prompted to answer.
4. The back-end script checks the answer supplied by the user by comparing it with the persisted (correct) answer. If the value is empty or incorrect, we go back to step 1: **a new CAPTCHA is generated**. Users should never get a second shot at answering the same CAPTCHA.
5. If the answer supplied by the user is correct, the form post is successful and processing can continue. If applicable, the generated CAPTCHA image is deleted.

Now, a sample implementation is presented. This is an implementation of a

CAPTCHA that resembles PIX. This uses JungleDragon, which is a wildlife image sharing site, as its image database. A first step in designing a custom CAPTCHA is to think about your user base in order to find a way to blend a CAPTCHA check into the user experience. Users are presented an image of an animal and then have to guess what it is. When they fail, a new random image is pulled up, as well as a fresh set of answers, their order and options shifted each time. The CAPTCHA back-end is implemented into a PHP class. To map the images to the answers, an associative array is used.

**Var $images = array (1=>"parrot", 2=>"panda", 3=>"lion", 4=>"dog", 5=>"cat");**

We may have more collections in the database. The method that generates the CAPTCHA is as follows:

```
1:  function generate_captcha($num_answers)
2:  {
3:  // get random image
4:  $image_num = rand(1,sizeof($this->images));
5:  $image_name = $this->images[$image_num];
6:
7:  // set the correct answer in the session
8:  $this->CI->session->set_userdata('captcha', $image_name);
9:
10:    // build up list of possible answers
11:    // we'll start by including the correct answer
12:    $answers = array();
13:    $answers[] = $image_name;
14:
15:    // next, we need to find num_answers - 1 additional options
16:    $count = 0;
17:    while ($count < ($num_answers-1)) {
18:      $currentanswer = rand(1,sizeof($this->images));
```

```
19:       if (!in_array($this->images[$currentanswer],$answers)) {
20:         $answers[] = $this->images[$currentanswer];
21:         $count++;
22:       }
23:     }
24:
25:     // shuffle the array so that the first answer is not
26:     // always the right answer
27:      shuffle($answers);
28:
29:     // build data array and return it
30:     $data = array(
31:       "image_num" => $image_num,
32:       "image_name" => $image_name,
33:       “answers”=>$answers
34:       }
35:       return $data;
36:       }
```

The relevant lines of the above code are explained below:

1. The method signature. Note how we can pass in $num_answer, to indicate how many possible answers are showed for each image.

4. Randomly select an image number based on the options in the associative array discussed earlier.

5. Get the name corresponding with the randomly selected image number from line 4

8. This is an important step. Here we are persisting the correct answer (image name) of the currently generated CAPTCHA. We need to persist this securely. In this case, using encrypted cookies, but server-side session variables, a file, or a database can also be used.

12. With the image selected and the correct answer persisted, we now need to

generate a set of possible answers. We'll store them in the $answers array.

13. Are set of options always has to contain the correct answer, so we'll include that in the array

16-23. Next, we will generate the additional answers, which are all wrong. We'll keep looping
until we have found the number of **unique** answers requested by $num_answers minus 1, since
we already included one answer: the correct one.

27. We do not want the correct answer to be at the same position in the answer list, therefore we shuffle the answer list.
30-36 . Here we are building up an array of values that the calling code needs to work with the CAPTCHA, and then return it.

There is another method check_captcha. This method checks if the answer that is passed to it corresponds to the persisted answer:

```
function check_captcha($answer)
{
  // check if captcha is correc
   return ($this->CI->session->userdata('captcha') === $answer) ? true : false;


}
```

That's it. We can now start using this class. From our script that renders our front-end

pages, we call:

```
// generate a new captcha $this-
>load->library('captcha');
$captchadata = $this->captcha->generate_captcha(5);
```

It is to be noted that this syntax of class loading and calling the method is specific for the CodeIgniter PHP framework. Alternatively the classic PHP syntax can be used if this framework is not used. Finally, in the postback code, we will call the check_captcha to see if the user has entered the correct answer based on the field value of captcha_answer. It depends on the validation library we use, how to call it, but we need to make sure that a new CAPTCHA is generated if the answer was empty or incorrect. Also, we need to insert validation messages that inform the user whether his is solution to the CAPTCHA was correct.

## 3.3 Guidelines for CAPTCHA implementation

If your website needs protection from abuse, it is recommended that you use a CAPTCHA. There are many CAPTCHA implementations, some better than others. The following guidelines are strongly recommended for any CAPTCHA code:

**Accessibility** CAPTCHAs must be accessible. CAPTCHAs based solely on reading text — or other visual-perception tasks — prevent visually impaired users from accessing the protected resource. Such CAPTCHAs may make a site incompatible with disability access rules in most countries. Any implementation of a CAPTCHA should allow blind users to get around the barrier, for example, by permitting users to opt for an audio or sound CAPTCHA.

**Image Security** CAPTCHA images of text should be distorted randomly before being presented to the user. Many implementations of CAPTCHAs use undistorted text, or text with only minor distortions. These implementations are vulnerable to simple automated attacks.

**Script Security** Building a secure CAPTCHA code is not easy. In addition to making

the images unreadable by computers, the system should ensure that there are no easy ways around it at the script level. Common examples of insecurities in this respect include:

1. Systems that pass the answer to the CAPTCHA in plain text as part of the web form.

2. Systems where a solution to the same CAPTCHA can be used multiple times (this makes the CAPTCHA vulnerable to so-called "replay attacks").

Most CAPTCHA scripts found freely on the Web are vulnerable to these types of attacks.

**Security Even After Wide-Spread Adoption-** There are various "CAPTCHAs" that would be insecure if a significant number of sites started using them. An example of such a puzzle is asking text-based questions, such as a mathematical question ("what is 1+1"). Since a parser could easily be written that would allow bots to bypass this test, such "CAPTCHAs" rely on the fact that few sites use them, and thus that a bot author has no incentive to program their bot to solve that challenge. True CAPTCHAs should be secure even after a significant number of websites adopt them.

**Should I Make My Own CAPTCHA?** In general, making your own CAPTCHA script (e.g., using PHP, Perl or .Net) is a bad idea, as there are many failure modes. We recommend that you use a well-tested implementation such as reCAPTCHA.

Beating spammers is not a hard science; you need to have multiple layers of defense. What you see above is in fact the most successful strategy against spammers: **obscurity**. Simply because we have a **CUSTOM CAPTCHA**, makes us less of a target. Spammers go for mass targets, as their success rate is typically extremely low. Even if we would have a single animal image with just two answers, and the answers would be in the same order each time, we would drastically reduce spam submissions.

An example of this effect is the site www.codinghorror.com/blog; it has a custom CAPTCHA check that simply lets you enter the word "Orange" to post a comment. Orange. Each time. Nothing is random, and there is only one answer. Still,

it has drastically reduced the spam on that blog, and it's a big blog. Obscurity works. Not for security, but against spammers.

It is extremely important to have CAPTCHAs based on a variety of sensory abilities. All CAPTCHAs presented here, except for the sound based CAPTCHA, rely on the user being able to see an image. However, since there are many visually impaired people using the Web, CAPTCHAs based on sound are necessary for accessibility. Unfortunately, images and sound alone are not sufficient: there are people who use the Web that are both visually and hearing impaired. The construction of a CAPTCHA based on a text domain such as text understanding or generation is an important open problem for the project.

## Chapter 4

# Breaking CAPTCHAs

The challenge in breaking a CAPTCHA isn't figuring out what a message says -- after all, humans should have at least an 80 percent success rate. The really hard task is teaching a computer how to process information in a way similar to how humans think. In many cases, people who break CAPTCHAs concentrate not on making computers smarter, but reducing the complexity of the problem posed by the CAPTCHA.

Let's assume you've protected an online form using a CAPTCHA that displays English words. The application warps the font slightly, stretching and bending the letters in unpredictable ways. In addition, the CAPTCHA includes a randomly generated background behind the word.

A programmer wishing to break this CAPTCHA could approach the problem in phases. He or she would need to write an algorithm -- a set of instructions that directs a machine to follow a certain series of steps. In this scenario, one step might be to convert the image in grayscale. That means the application removes all the color from the image, taking away one of the levels of obfuscation the CAPTCHA employs.

Next, the algorithm might tell the computer to detect patterns in the black and white image. The program compares each pattern to a normal letter, looking for matches. If the program can only match a few of the letters, it might cross reference those letters with a database of English words. Then it would plug in likely candidates into the submit field. This approach can be surprisingly effective. It might not work 100 percent of the time, but it can work often enough to be worthwhile to spammers.

What about more complex CAPTCHAs? The **Gimpy** CAPTCHA displays 10 English words with warped fonts across an irregular background. The CAPTCHA arranges the words in pairs and the words of each pair overlap one another. Users have to type in three correct words in order to move forward. How reliable is this approach?

As it turns out, with the right CAPTCHA-cracking algorithm, it's not terribly reliable. Greg Mori and Jitendra Malik published a paper detailing their approach to cracking the Gimpy version of CAPTCHA.

Mori and Malik ran a series of tests using their algorithm. They found that their algorithm could correctly identify the words in a Gimpy CAPTCHA 33 percent of the time [source: Mori and Malik]. While that's far from perfect, it's also significant. Spammers can afford to have only one-third of their attempts succeed if they set bots to break CAPTCHAs several hundred times every minute.

Another vulnerability that most CAPTCHA scripts have is again in their use of sessions; if we're on an insecure shared server, any user on that server may have access to everyone else's session files, so even if our site is totally secure, a vulnerability on any other website hosted on that machine can lead to a compromise of the session data, and hence, the CAPTCHA script. One workaround is by storing only a hash of the CAPTCHA word in the session, thus even if someone can read the session files, they can't find out what the CAPTCHA word is.

## 4.1 Breaking CAPTCHAs without OCR

Most CAPTCHAs don't destroy the session when the correct phrase is entered. So by reusing the session id of a known CAPTCHA image, it is possible to automate requests to a CAPTCHA-protected page.

**Manual steps**

Connect to CAPTCHA page

Record session ID and CAPTCHA plaintext

**Automated steps**

Resend session ID and CAPTCHA plaintext any number of times, changing the user data. The other user data can change on each request. We can then automate hundreds, if not thousands of requests, until the session expires, at which point we

just repeat the manual steps and then reconnect with a new session ID and CAPTCHA text.

Traditional CAPTCA-breaking software involves using image recognition routines to decode CAPTCHA images. This approach bypasses the need to do any of that, making it easy to hack CAPTCHA images.

## 4.2 Breaking a visual CAPTCHA

Greg Mori and Jitendra Malik of University of California at Berkeley's Computer Vision Group evaluate image based CAPTCHAs for reliability. They test whether the CAPTCHA can withstand bots who masquerade as humans.

**Approach:** The fundamental ideas behind our approach to solving Gimpy are the same as those we are using to solve generic object recognition problems. Our solution to the Gimpy CAPTCHA is just an application of a general framework that we have used to compare images of everyday objects and even find and track people in video sequences. The essences of these problems are similar. Finding the letters "T", "A", "M", "E" in an image and connecting them to read the word "TAME" is akin to finding hands, feet, elbows, and faces and connecting them up to find a human. Real images of people and objects contain large amounts of clutter. Learning to deal with the adversarial clutter present in Gimpy has helped us in understanding generic object recognition problems.

**Breaking an EZ-Gimpy CAPTCHA:** Our algorithm for breaking EZ-Gimpy consists of 3 main steps:


*Fig 5.1 Breaking CAPTCHAs*

1. **Locate possible (candidate) letters at various locations:** The first step is to hypothesize a set of candidate letters in the image. This is done using our shape matching techniques.

The method essentially looks at a bunch of points in the image at random, and compares these points to points on each of the 26 letters. The comparison is done in a way that is very robust to background clutter and deformation of the letters. The process usually results in 3-5 candidate letters per actual letter in the image. In the example shown in Fig 5.1, the "p" of profit matches well to both an "o" or a "p", the border between the "p" and the "r" look a bit like a "u", and so forth. At this stage we keep many candidates,

to be sure we don't miss anything for later steps.

2. **Construct graph of consistent letters:** Next, we analyze pairs of letters to see whether or not they are "consistent", or can be used consecutively to form a word**.**
3. **Look for plausible words in the graph:** There are many possible paths through the graph of letters constructed in the previous step. However, most of them do not form real words. We select out the real words in the graph, and assign scores to them based on how well their individual letters match the image. Similar algorithms are also devised by Mori and Malik to evaluate other image based CAPTCHAs like **Gimpy,** etc.

## 4.3Breaking an audio CAPTCHA

Recent research is suggesting that Google's audio capture is the latest in a string of CAPTCHA's to have been defeated by software. t has been theorized that one cost-effective means of breaking audio captures and image captures that have not yet had automated systems developed is to use a mechanical turk and pay low rates for per-CAPTCHA reading by humans, or provide another form of motivation such as access to popular sites for reading the CAPTCHA. However, it always required a significant level of resources to achieve.

The development of software to automatically interpret CAPTCHAs brings up a number of problems for site operators. The problem, as discovered by Wintercore

Labs and published at the start of March is that there are repeatable patterns evident in the audio file and by applying a set of complex but straight forward processes, a library can be built of the basic signal for each possible character that can appear in the CAPTCHA. Wintercore point to other audio CAPTCHAs that could be easily reversed using this technique, including the one for Facebook.

The wider impact of this work might take some time to appear, but it provides an interesting proof of breaking audio CAPTCHAs. At the least, it shows that both of Google's CAPTCHA tools have now been defeated by software and it should only be a matter of time until the same can be said for Microsoft and Yahoo!'s offerings. Even with an effectiveness of only 90%, any failed CAPTCHA can easily be reloaded for a second try.

## 4.4 Social Engineering used to break CAPTCHAs:

Spammers often use social engineering to outwit gullible Web users to serve their purpose. Security firm, **Trend Micro** warns of a Trojan called **TROJ_CAPTCHAR,** which masquerades as a strip tease game. At each stage of the game, the user is asked to solve a CAPTCHA. The result is relayed to a remote server where a malicious user is waiting for them. The strip-tease game is a ploy by spammers to identify and match solutions for ambiguous CAPTCHAs from legitimate sites, using the unsuspecting user as the decoder of the said images.

## 4.5 CAPTCHA cracking as a business:

No CAPTCHA can survive a human that’s receiving financial incentives for solving it. CAPTCHA are cracked by firms posing as Data Processing firms. They usually charge $2 for 1000 CAPTCHAs successfully solved. They advertise their business as “Using the advertisement in blogs, social networks, etc significantly increases the efficiency of the business. Many services use pictures called CAPTCHAs in order to prevent automated use of these services. Solve CAPTCHAs with the help of this portal; increase your business efficiency now!” Such firms help spammers in beating the first line of defence for a Website, i.e., CAPTCHAs.

# Chapter 5

# Issues with CAPTCHAs

There are many issues with CAPTCHAs, primarily because they distort text and images in such a way that, sometimes it gets difficult for even humans to read. Even the simplest, but effective CAPTCHA, like a mathematical equation "What is the sum of three and five?" can be a pain for cognitively disabled people.

## 5.1 Usability issues with text based CAPTCHAs:

Are text CAPTCHAs like Gimpy, user–friendly? Sometimes the text is distorted to such an extent, that even humans have difficulty in understanding it. Some of the issues are listed in table 6.1

| Category | Usability issue | |
|---|---|---|
| Distortion | Distortion method and level | |
| | Confusing characters | |
| | Friendly to foreigners? | |
| Content | Character set | |
| | String length | How long? |
| | | Predictable or not? |
| | Random string or dictionary word? | |
| | Offensive word | |
| Presentation | Font type and size | |
| | Image size | |
| | Use of colour | |
| | Integration with web pages | |

*Table 6.1 Usability issues in text based CAPTCHAs*

**Distortion** becomes a problem when it is done in a very haphazard way. Some characters like 'd' can be confused for 'cl' or 'm' with 'rn'. It should also be easily

understandable to those who are unfamiliar with the language.

**Content** is an issue when the string length becomes too long or when the string is not a dictionary word. Care should be taken not to include offensive words.

**Presentation** should be in such a way as to not confuse the users. The font and colour chosen should be user friendly.

## 5.2 Usability of audio CAPTCHAs:

In audio CAPTCHAs, letters are read aloud instead of being displayed in an image. Typically, noises are deliberately added to prevent such audio schemes from being broken by current speech recognition technologies.

**Distortion:** Background noises effectively distort sounds in audio CAPTCHAs. There is no rigorous study of what kind of background noises will introduce acceptable sound distortion. However, it is clear that distortion methods and levels, just as in text based CAPTCHAs, can have a significant impact on the usability of audio CAPTCHAs. For example, an early test in 2003 showed that the distorted sound in an audio CAPTCHA that was deployed at Microsoft's Hotmail service was unintelligible to all (four) journalists, with good hearing, that were tested. Due to sound distortion, confusing characters can also occur in audio CAPTCHAs. For example, we observed that it is hard to tell apart 'p' and 'b'; 'g' and 'j', and 'a' and '8'. Whether a scheme is friendly to non-native speakers is another usability concern for audio CAPTCHAs.

**Content:** Content materials used in audio CAPTCHAs are typically language specific. Digits and letters read in a language are often not understandable to people who do not speak the language. Therefore, unlike text-based schemes, localisation is a major issue that audio CAPTCHAs face.

**Presentation:** The use of colour is not an issue for audio CAPTCHAs, but the integration with web pages is still a concern. For example, there is no standard graphical symbol for representing an audio CAPTCHA on a web page, although many schemes such as Microsoft and reCAPTCHA use a speaker symbol. More importantly, what really matters for visually impaired users is that the html image

alternative text attached to any of the above symbol should clearly indicate the need to solve an audio CAPTCHA.

When embedded in web pages, audio CAPTCHAs can also cause compatibility issues. For example, many such schemes require JavaScript to be enabled. However, some users might prefer to disable JavaScript in their browsers. Some other schemes can be even worse. For example, we found that one audio scheme requires Adobe Flash support. With this scheme, vision-impaired users will not even notice that such a CAPTCHA challenge exist in the page, unless Flash is installed in their computers - apparently, no text alternative is attached to the speaker-like Flash object, either.

# Chapter 6

# Applications

CAPTCHAs are used in various Web applications to identify human users and to restrict access to them.

Some of them are:

1. **Online Polls** As mentioned before, bots can wreak havoc to any unprotected online poll. They might create a large number of votes which would then falsely represent the poll winner in spotlight. This also results in decreased faith in these polls. CAPTCHAs can be used in websites that have embedded polls to protect them from being accessed by bots, and hence bring up the reliability of the polls.

2. **Protecting Web Registration** Several companies offer free email and other services. Until recently, these service providers suffered from a serious problem – bots. These bots would take advantage of the service and would sign up for a large number of accounts. This often created problems in account management and also increased the burden on their servers. CAPTCHAs can effectively be used to filter out the bots and ensure that only human users are allowed to create accounts.

3. **Preventing comment spam** Most bloggers are familiar with programs that submit large number of automated posts that are done with the intention of increasing the search engine ranks of that site. CAPTCHAs can be used before a post is submitted to ensure that only human users can create posts. A CAPTCHA won't stop someone who is determined to post a rude message or harass an administrator, but it will help prevent bots from posting messages automatically.

4. **Search engine bots** It is sometimes desirable to keep web pages unindexed to prevent others from finding them easily. There is an html tag to prevent search engine bots from reading web pages. The tag, however, doesn't guarantee that bots won't read a web page; it only serves to say "no bots, please." Search engine bots, since they usually belong to large companies, respect web pages that don't want to allow them in. However, in order to truly guarantee that bots won't enter a web site, CAPTCHAs are needed.

5. **E-Ticketing** Ticket brokers like TicketMaster also use CAPTCHA applications. These applications help prevent ticket scalpers from bombarding the service with massive ticket purchases for big events. Without some sort of filter, it's possible for a scalper to use a bot to place hundreds or thousands of ticket orders in a matter of seconds. Legitimate customers become victims as events sell out minutes after tickets become available. Scalpers then try to sell the tickets above face value. While CAPTCHA applications don't prevent scalping; they do make it more difficult to scalp tickets on a large scale.

6. **Email spam** CAPTCHAs also present a plausible solution to the problem of spam emails. All we have to do is to use a CAPTCHA challenge to verify that a indeed a human has sent the email.

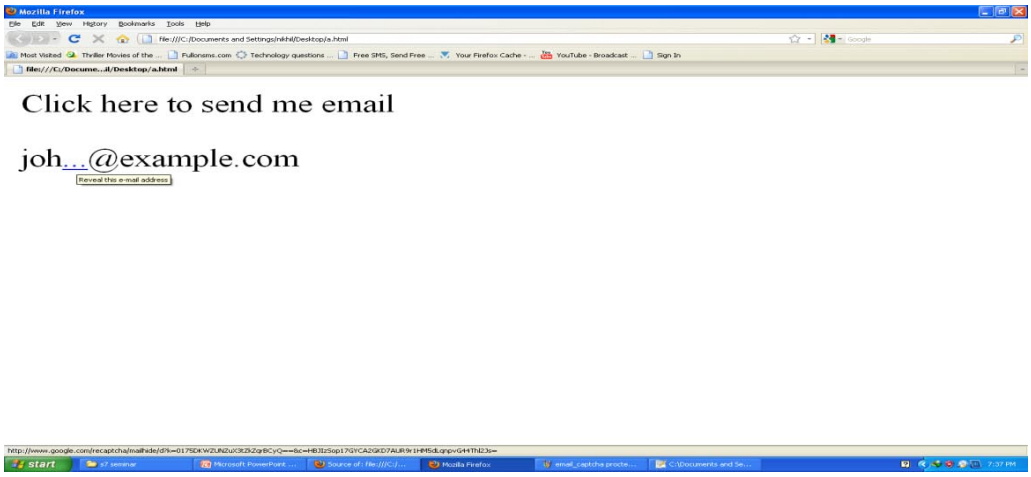



Fig: Captcha Protected e-mail

7. **Preventing Dictionary Attacks** CAPTCHAs can also be used to prevent dictionary attacks in password systems. The idea is simple: prevent a computer from being able to iterate through the entire space of passwords by requiring it to solve a CAPTCHA after a certain number of unsuccessful logins. This is better than the

8. **As a tool to verify digitized books** This is a way of increasing the value of CAPTCHA as an application. An application called reCAPTCHA harnesses users responses in CAPTCHA fields to verify the contents of a scanned piece of paper. Because computers aren't always able to identify words from a digital scan, humans have to verify what a printed page says. Then it's possible for search engines to search and index the contents of a scanned document. This is how it works: The application already recognizes one of the words. If the visitor types that word into a field correctly, the application assumes the second word the user types is also correct. That second word goes into a pool of words that the application will present to other users. As each user types in a word, the application compares the word to the original answer. Eventually, the application receives enough responses to verify the word with a high degree of certainty. That word can then go into the verified pool.

9. **Improve Artificial Intelligence (AI) technology** Luis von Ahn of Carnegie Mellon University is one of the inventors of CAPTCHA. In a 2006 lecture, von Ahn talked about the relationship between things like CAPTCHA and the field of artificial intelligence (AI). Because CAPTCHA is a barrier between spammers or hackers and their goal, these people have dedicated time and energy toward breaking CAPTCHAs. Their successes mean that machines are getting more sophisticated. Every time someone figures out how to teach a machine to defeat a CAPTCHA, we move one step closer to artificial intelligence. As people find new ways to get around CAPTCHA, computer scientists like von Ahn develop CAPTCHAs that address other challenges in the field of AI. A step backward for CAPTCHA is still a step forward for AI – "every defeat is also a victory".

# Chapter 7

# CONCLUSION

CAPTCHA is now in frequent use in the comment areas of message boards and personal weblogs. Many bloggers claim that CAPTCHA challenges are successful in eradicating comment spam, but below a certain threshold of popularity, any other method of comment spam control should be reasonably successful -- and more accessible to users with disabilities.

Sites with attractive resources and millions of users will always have a need for access control systems that limit widespread abuse. At that level, it is reasonable to employ many concurrent approaches, including audio and visual CAPTCHA, to do so.

## 7.1 Present Scenario

Conclusion of CAPTCHAs are associated with its usefulness and deciphering ability of HUMAN and BOTS. In present scenario of web world millions of website is using this protocol to minimizes the exploitation of resources. CAPTCHA-based Code Voting is easy to use

- has a good scalability
- Protection against Malware voting fraud
- Security depends on the premise that computers cannot read the CAPTCHAs
- CAPTCHAs may be solved by humans
- Malware is able to cast random votes

## 7.2 Future of CAPTCHAs

The future of Captcha is also interesting. There's no doubt that image processing software and computers themselves will become more powerful and eventually will be able to automatically decipher today's Captcha images. captcha will be obsoleted by verified id's but may have a future in answering student's homework questions. Captcha project is growing and more and more websites deploy Captcha tests, managed by the central Captcha server, I am really afraid it to grow up to another medical test, eye-chart

test, global scientific experiment, to test one's visual impairment,as central server of Captcha is processing responses identified by IP address, assigned to real persons. So those living on the Internet with static IP addresses may have each of their respective Captcha response processed, analyzed and saved. At first glance this seems like an good solution, but it too has major problems. The first is sample set size: although Asirra has a set of around three million photos This isn't big enough to provide a completely new image for every time one is presented. Given a bit of time a spammer (possibly co-operating with other spammers) could easily build a database of photos to animal (analogous to rainbow tables in password cracking). This can beworked around by programmatically.

# Chapter 8

# REFRENCES

1: von Ahn, L., Blum, M., Hopper, N. and Langford, J. „CAPTCHA: Using Hard AI Problems for Security”

2: Greg Mori and Jitendra Malik. “Breaking a Visual CAPTCHA.”, Unpublished Manuscript, 2002.

3: Yang, M.H., Kriegman, D.J., Ahuja, N. (2002) “Detecting Faces in Images”, IEEE-PAMI 24:1

4: Lienhart, R., Hartmann, A. (2002) “Classifying images on the web automatically”, J. Electron. Imaging Vol 11, 445

5: Bajaj, Vikas (April 25, 2010). "Spammers Pay Others to Answer Security Tests"

6: Sergei, Kruglov. "Defeating of some weak CAPTCHAs"

7: "Latest Status of CAPTCHA Trademark Application"

8: http://www.captcha.net/

9: http://www.blogtoplist.com/rss/captcha.html

10: http://www.gnu.org/licenses/gpl.html

11: http://www.white-hat-web-design.co.uk/articles/php-captcha.php

12. http://en.wikipedia.org/wiki/Fiber_Channel