Send Orders for Reprints to reprints@benthamscience.net

The Open Computer Science Journal, **2014**, *1*, **1-17**

Open Access

Character Segmentation for Automatic CAPTCHA Solving

Christos Makris and Christopher Town*

University of Cambridge Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK

Abstract: Many websites utilise CAPTCHA (Completely Automatic Public Turing tests to tell Computers and Humans Apart) schemes as human interaction proofs to grant access to their services only to people rather than spam bots. In this paper, we examine the security of six widely used types of CAPTCHA and present novel attacks against all of them, achieving success rates of up to 88%. We made improvements to three previously published attacks against the Hotmail, Wikipedia, and Slashdot challenges and devised novel and successful attacks against BotDetect's Wavy chess, reCAPTCHA, and a new variant of the Wikipedia scheme. Furthermore, we implemented a library that includes customisable segmentation algorithms and character recognisers. This library can serve as a tool for further investigating CAPTCHA security. Even though the difficulty and time needed to develop our CAPTCHA solver algorithms varied significantly between different schemes, none of these CAPTCHAS proved to be resistant to the attacks we devised. Based on our findings, we make recommendations for strengthening CAPTCHA methods to make them more resistant to automated attacks such as ours.

Keywords: CAPTCHA, character segmentation, human interaction proofs, optical character recognition, security.

1. INTRODUCTION

Online services such as webmail, social media, cloud storage, file sharing, and content creation platforms are often abused by bots. Websites are using CAPTCHAs (**C**ompletely **A**utomated **P**ublic **T**uring test to tell **C**omputers and **H**umans **A**part) as one of their main defence mechanisms against such bots. CAPTCHAs are challenges sent to users and permission is granted only to those that are able to solve them correctly within a certain time frame. The challenges are based on tasks that current state of the art algorithms do not perform well but that are fairly easy for people.

Challenges include recognition of distorted words, identification of the context of an image, logic questions, mathematical questions and understanding speech. A good candidate task is one such that

- challenges can be automatically generated
- there is a very large (ideally infinite) pool of challenges
- humans (even naive users) perform it easily
- bots perform the task poorly or only with substantial resource overheads

A CAPTCHA is secure if, in the long run, the total cost of automated attacks is higher than their expected gain. Hence the likelihood of a successful attacks is a measure of the security of a CAPTCHA. It was suggested that if more than 0.01% of the challenges can be successfully solved by a computer program then the scheme is broken [1], but in the literature a threshold of 1% [27, 6] is more commonly adopted. The threshold was decided based on the cost of the attack and the gains of the hacker for every successful attack. Since the use of CAPTCHAs, as well as the underlying security economics could have changed since these earlier studies were published, it would be useful to have a more recent and representative metric. In the absence of a widely used baseline metric, we will use the much more conservative 1% accuracy criterion that is becoming more widely used in the literature. Most schemes we consider in this paper were broken by our algorithms with a much higher precision, so the limit needs to be increased considerably for the schemes to be considered safe.

Good candidate tasks for CAPTCHAs are challenges where no artificial intelligence algorithm exists to solve them accurately. This creates a win-win situation since hackers are in effect forced to advance the field of Artificial Intelligence [20].

We focus on text-based CAPTCHAs because they were the first to be introduced [4] and remain the most widely used type. Even though Optical Character Recognition (OCR) has advanced substantially, solving text based CAPTCHA remains difficult [21, 15]. Major challenges for automated solution to CAPTCHAs include the fact that artificially created noise and distortions are added to make segmentation and recognition of characters difficult, the words do not necessarily belong to any lexicon, and the words are too few and too unpredictable for contextual disambiguation. We did not experiment with OCR algorithms but instead focused on algorithms that are commonly used against CAPTCHA challenges.

*Address correspondence to this author at the University of Cambridge Computer Laboratory, 15 JJ Thomson Avenue, Cambridge CB3 0FD, UK; Tel: +44(0)1223 763686; Fax: +44(0)1223 334678;
E-mails: christos.makris@cantab.net, cpt23@cam.ac.uk

2352-6270/14

The main aim of this paper was to check whether commonly used CAPTCHA schemes still suffer from known vulnerabilities and can be compromised by straightforward attacks using variations of known techniques. Additionally we implemented two novel attacks against reCAPTCHA and BotDetect's Wavy chess.

A major achievement of this paper is that we demonstrate that an attacker who implements an application with attacks documented in the literature can relatively easily develop attacks against new schemes by using combinations and alterations of those algorithms. It was out of the scope of our project to devise algorithms that would achieve the best possible accuracy against the CAPTCHA schemes. Instead, we demonstrate relatively straightforward techniques that achieve success percentages that already make a potential attack highly profitable.

We focused mostly on the segmentation part of the algorithms and we used a generic character recogniser that has been previously tested in the literature [6, 11]. While this is not an optimal state of the art algorithm, it offers the significant advantages of simplicity, robustness, and an ability of training an adequate set of classifiers in a small enough time period to allow us to check different variations of the segmentation algorithm against the validation test.

2. RELATED WORK

There are a number of published attacks against several CAPTCHA schemes, although some of these are still being used without major modifications [6, 23, 18, 3, 24]. One general benefit of such research is that it can offer guidelines for CAPTCHA designers in order to strengthen their algorithms. Specific prior art will be discussed in the context of our implementation.

A scheme is commonly regarded as broken if there exists an algorithm that can break more than 0.01% of the challenges [1]. For experiments simulating the behaviour of the CAPTCHA scheme in lab conditions, a stricter threshold of at least 1% should be achieved by a successful attack [27]. Since an attacker may only answer challenges when he is confident that the answer is correct, it is better to use the precision and coverage of the attacker's algorithm to assess the security of a scheme [6], which are defined as follows:

$$Precision = \frac{\#challenges\ answered\ correctly}{\#answered\ challenges}$$

$$Coverage = \frac{\#challenges\ answered}{\#total\ challenges}$$

In order to devise a technique to automatically solve text-based CAPTCHAs, one effectively needs to reverse engineer the methods that are used to produces the CAPTCHA scheme. This process can be broken down into three main stages [1]:

1. **Preprocessing**: Transformations such as image enhancements are carried out to make the work of the next stages easier

2. **Segmentation**: Segment text from the background and attempt to split it into single character blocks

3. **Recognition**: Classify each extracted character

Further post-processing stages can be added in order to improve the output of each step. A common practice is to write a specific algorithm for segmenting each scheme and use machine learning to derive classifiers to distinguish between individual letters. The classifiers are trained on data produced from the segmentation algorithm.

Computers can outperform humans at recognising individual characters [8] even under large distortion and hence the difficulty of solving text-based CAPTCHAs primarily depends on the difficulty of text segmentation. This problem can be made substantially more difficult through the use of hand-written characters [28], but all common CAPTCHA schemes currently use distorted versions of printed text with various anti-segmentation techniques.

2.1. Pre-processing

The purpose of pre-processing is to take the challenges to a state were the segmentation algorithms can extract the letters. Common pre-processing techniques include:

• **Background removal**: This is most useful against CAPTCHAs that use colour as a defence mechanism. It results in a greyscale image with the foreground pixels retaining their intensity and background pixels being white.

• **Up-sampling** Each pixel of the image is divided into subpixels, thus allowing finer control over the area that is affected by the segmentation algorithm.

• **Blurring**: The image is convolved with Gaussian, mean or median smoothing kernels to reduce the amount of noise in the image.

• **Thresholding**: Removes pixels of low intensity (they are treated as noise), resulting in a binarised image. This is very useful since most segmentation algorithms require a binarised input image.

• **Line removal**: Eliminate straight segments that are not part of characters. Lines are categorised as:

- lines with **smaller thickness** than the characters. They can be removed using erosions followed by dilations [23]. Black pixels in areas with small black count are removed from the erosion and dilation preserves the thickness of the characters in areas were pixels were not removed.

- lines with **similar or greater thickness** than the characters. If the lines are longer than the characters then line detection algorithms (*eg.*Hough transforms) are used to identify them. The area around them is then examined to decide which pixels to remove [6].

If the lines substantially overlap with characters, then they are not removed at this stage and instead are dealt with by the segmentation methods.

• **Thinning** of characters: a binary skeleton of the image is created using Zhang's thinning algorithm [26]. This is done in order to reduce the number of character pixels without removing any intrinsic information.

• Identification and correction of **global wave transformations** [12]. Useful when pattern matching segmentation will be performed [13]. Such transformations are not in general reversible and when they are, their reversal requires *a priori* assumptions about the transformations used.

• Identify the CAPTCHA scheme. A website can use more than one schemes, randomly select one and send a challenge from that scheme. Hence the attacker needs to identify the scheme in order to choose the attack algorithm. If the schemes differ significantly between they could be identified and only challenges of the weak scheme will be attacked, otherwise they can be treated as one scheme. The developed algorithm must be generic enough to attack all different challenge variations.

2.2. Segmentation

The goal of segmentation is to identify the location of the characters and extract them from the rest of the image. Published attacks use techniques that are specific to a particular CAPTCHA scheme. The most commonly used algorithms include:

• **Flood-filling segmentation** An object is a connected component created from neighbouring black pixels. Flood-filling segmentation takes a binarised image and returns the collection of objects found.

• **Histogram segmentation**: the number of pixels at each column is counted and the minima of this value are chosen as possible segmentation points. This is especially useful when no column contains pixels of more than one character.

• **Pattern matching** techniques [10, 12, 13] the attacker searches for characteristic features of letters, including:

- the dot of `*i*', `*j*'
- the cross like intersection of `*f*', `*k*', `*r*', `*x*'.
- sigmoid shaped characters such as `*s*' and `5'

The categories are searched in a pre-specified order. Any characters that belong to the current category are identified and removed. The process is repeated for characters of the next category until there are no categories or characters left.

• **Snake Segmentation:** A line (snake) is created starting from the top of the image and moving downwards, left and right, without crossing any character pixels to create a segmentation line [24]. The snake ends at the bottom-most pixel of the image. Snakes at different horizontal positions are drawn and those that best describe a cut are selected. Each character is then contained in the area between two snakes.

2.3. Character Recognition

Characters are extracted from the CAPTCHA according to the results of the segmentation phase. A classifier is then used to recognise the letters. While early attacks against CAPTCHAs were using simple criteria such as the pixel count of the resulted objects, more recent attacks utilise machine learning classifiers that are trained on the output of the segmentation algorithm [6].

3. AUTOMATED CAPTCHA SOLVER FRAMEWORK

We developed a flexible framework for automated CAPTCHA solving. A modular design allows different pre-processing, segmentation and recognition algorithms to be deployed or new algorithms to be devised as required for any given scheme. Individual segmentation algorithms are used to extract the characters of the CAPTCHA challenge. We try to identify segmentation failures as early as possible and skip the challenge in order to minimise wasted effort.

Both pre-processing and segmentation steps required a number of hyper parameters to be tuned in order to achieve the best results. A subset of the evaluation set was used to manually select appropriate values for the hyper-parameters. We selected values that achieved reasonable performance against a set of a few hundred challenges. This number was regarded as small enough to allow manual inspection of segmentation results and large enough to ensure generalisation performance.

The set of parameters was selected on the rationale that we want to assess the ease with which attackers could attack the schemes. Hence we tried a few dozen to a few hundred parameter combinations and we stopped when a reasonable accuracy was reached (well above the success threshold defined above). This was in general not difficult as there were a number of parameters that could give good accuracy. The methods were thus shown to be reasonably stable with respect to those particular parameter values. Further parameter tuning would no doubt yield some small incremental performance gains, as would the usage of some more complex algorithms for particular image analysis subtasks, but as was mentioned earlier the main aim of this paper is not achieving maximal accuracy on a particular selection of data sets or indeed algorithms. Instead, we show how combinations and variations of well-known robust image processing and analysis techniques can break some of the most widely used CAPTCHA schemes currently in widespread usage. Our results present a challenge to the designers of CAPTCHA algorithms to make their methods more robust with respect to these kinds of solution frameworks.

3.1. Pre-Processing

The framework provided the ability to use up-sampling, blurring, thresholding, removing of certain intensities, and thinning of characters. We also developed custom methods for removing the lines in Slashdot CAPTCHAs and the chess effect in BotDetect's Wavy Chess. In the latter cases we used the fact that there was consistency in how those patterns were created to achieve better results. For each CAPTCHA scheme we selected both the needed methods and the order of applying them. Some methods were used more than once against the same scheme.

3.2. Segmentation

We implemented flood-filling, histogram segmentation, and a variation of histogram segmentation that uses further metrics from the pixel count in each column to decide the

segmentation lines. Additionally we developed a new algorithm that identifies the vertical lines of letters of reCAPTCHA challenges and then examines them in order to determine the segmentation line starting point and orientation. More implementation details will be given in the context of each scheme.

3.3. Character Recognition

Our character recognition module is structured into layers. In the first layer characters are grouped into categories of one or more characters and a Support Vector Machine (SVM) classifier is used to assign candidate character regions to relevant categories. If the chosen category contains more than one letter, then a second stage category specific SVM classifier is applied. This process continues until the letter has been unambiguously classified. In practice we have found two-layer classifier architectures sufficient.

The SVMs are trained using the real and imaginary parts of Zernike moments [11] up to and including order 10, and the width, height, aspect ratio, and inverse aspect ratio of the character's bounding box. We chose Zernike moments because their usage is well established in OCR (optical character recognition) tasks. Zernike polynomials are orthogonal to each other and hence can represent information compactly with very low redundancy while retaining enough of the characteristic information of shapes to allow efficient reconstruction. Used as classification features, this means that their usage provides higher representational accuracy than comparable methods at similar description length for fairly complex shapes. They also exhibit some invariance to simple rotations, translations and scale differences, which makes them especially useful for CAPTCHAs whose characters undergo various transformations.

We decided to use SVM since they can be quickly trained and they perform relatively well when given input Zernike Moments as input features. Other machine learning frameworks, such as convolutional neural networks, could have been used instead, but these typically require larger amounts of training data. Furthermore, [11] compares the performance of neural networks with that of SVMs training using Zernike features and strongly favours the latter approach. As the development of an optimal classification methodology is not a primary focus of this paper, we also adopted this approach, and our results demonstrate the efficacy of this algorithm and feature choice. Indeed, we gained character classification precision larger than 95% in all but the reCaptcha scheme. Characters in reCaptcha challenges were rotated and that caused a poor precision.

The Zernike moments are approximated for the discrete case of image processing by the formula [14]:

$$Z_{pq} = \lambda\left(p, N\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} R_{pq}(r_{xy})e^{-i\theta_{xy}} f\left(x, y\right)\right) \quad (1)$$

A better way to calculate them is by using the radial polynomials

$$R_{pq}(r) = \sum_{s=0}^{\frac{p-|q|}{2}} (-1)^s \cdot \frac{(p-s)!}{s!\left(\frac{p+|q|}{2}-s\right)!\left(\frac{p-|q|}{2}-s\right)} \cdot r^{p-2s} \quad (2)$$

There exist a number of fast and numerically stable (with a small number of erroneous cases) algorithms [9, 16, 19] to calculate these polynomials. We employ the q-recursive algorithm that uses the relations:

$$H_1 = \frac{q(q-1)}{2} - qH_2 + \frac{H_3(p+q+2)(p-q)}{8} \quad (3a)$$

$$H_2 = \frac{H_3(p+q)(p-q+2)}{4(q-1)} + (q-2) \quad (3b)$$

$$H_3 = \frac{-4(q-2)(q-3)}{(p+q+2)(p-q+4)} \quad (3c)$$

This algorithm has been shown to outperform other algorithms of similar complexity [17]. While its usage for high-order polynomials can give rise to numerical instabilities, this is not a problem in our case since we were able to restrict ourselves to polynomials of order 10 or lower without loss of accuracy. The polynomials are calculated and their values cached at the initialisation of the recogniser and hence we were able to investigate adding further polynomials using other (slower) methods to check that our usage of the q-recursive algorithm did not adversely affect classification performance. In practice we saw that polynomials of up to order 10 give good precision, and therefore our choice of the q-recursive algorithm was justified with minimal overhead for calculating the moments.

The image is transformed into polar coordinates and the feature vector is calculated. During training the maximum and minimum values of each feature are found and stored to be used to normalise each feature in the range $[-1, +1]$ using the formula:

$$\forall i \in \{1, 2, .., n\}.\ v_i \nwarrow\!\searrow v'_i = min_i + \frac{v_i}{max_i - min_i} \quad (4)$$

The SVMs were implemented using the GPU (Graphics Processing Unit) accelerated libsvm [2, 7] package that creates a SVM for each pair of classes. Multi-label classification is implemented by "voting": the outputs of all the SVMs are summed and the class with the most votes is chosen.

4. CAPTCHA-SPECIFIC ALGORITHMS

4.1. Wikipedia

Wikipedia challenges consist of two joined-up English words (each consisting of either four or five letters) that form a single character sequence. Highly distorted letters of a single font are used. Segmentation resistance is provided through local distortions of characters and a global wave

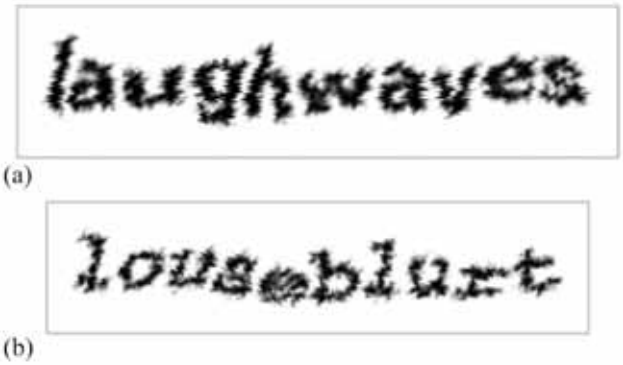

Fig. (1). Examples illustrating the difficulty of character segmentation in the case of the old **(a)** and new **(b)** Wikipedia CAPTCHAs.

Fig. (2). Top: Examples of correct (left) and erroneous (right) Wikipedia CAPTCHA component divisions. **Bottom:** *t* and *r* overlap vertically and are initially combined into a single component before being split.

transformation of the word string (see Fig. **1a**). The scheme was updated in Spring 2013 to further strengthen it (Fig. **1b**). The new scheme uses a different font with thinner letters, hence the applied transformations distort the letters significantly more. Our algorithm is able to break both schemes. It achieves 72% precision against the older scheme and 16% against the new scheme but the latter required a significantly smaller training set (just 210 samples compared to 2075 previously). Our discussion primarily focuses on the older scheme.

There are two causes of characters becoming conjoint and hence difficult to segment apart:

• Letters with long horizontal lines come very close to their neighbours because of the wave transformation. The formed joins are made of pixels of low intensities.

• Local distortions create areas of grey pixels around characters. These grey areas span through a lot of columns causing neighbouring characters to merge.

The scheme has been proven to be insecure by Bursztein *et al.* [6] who achieved a 25% attack success rate against it. The old scheme does not have significant differences from the one that they attacked. In their paper they do not present the specific details of their attack but we believe that they removed the noise, binarised the image and then they used flood-filling segmentation which is their main segmentation algorithm. We proceeded similarly although we have added some extra segmentation and post-segmentation steps. Our algorithm consists of the following main stages (see Fig. **3**):

1. The image is thresholded and connected components are extracted.

2. Connected components that are too small are removed as noise.

3. Connected components that are of typical dot dimensions are identified and candidate `*i*' or `*j*' are marked.

4. Connected components that belong to fragmented characters are merged.

5. Overly large connected components are split.

6. Characters are recognised using SVM classifiers.

7. If the final string contains the expected number of characters (currently 8-10) then return it, otherwise skip the challenge.

A number of common cases where two letters are not separated can be treated as a composite character (ligature) that is recognised explicitly by our classifiers. The most common conjoint character pairs are `ft', `rt' and `ry'.

4.1.1. `i', `j' Recognition

Objects of a certain size and shape are categorised as dots and their neighbours are examined. If exactly one of the neighbours is a long straight object and is in good horizontal alignment with the dot, then we mark those objects as a potential `*i*' or `*j*' character.

4.1.2. Join Connected Components

Two neighbouring objects are combined when at least one of the following conditions holds:

Fig. (3). Stages of Wikipedia CAPTCHA solving: (**a**) original character sequence, (**b**) thresholded, (**c**) connected components (**d**) small objects (such as noise near the n, e, t and i characters) are removed.

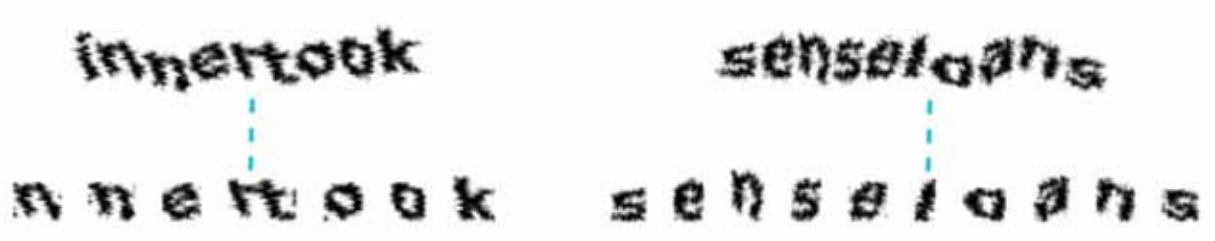

Fig. (4). Examples of characters extracted from Wikipedia challenges. The ligature `*rt*' can be recognised using a special classifier.

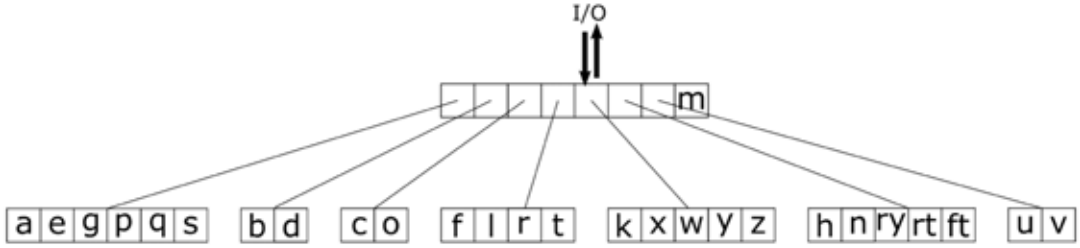

Fig. (5). Wikipedia recogniser consisting of a two-stage cascade of SVM classifiers. In addition to isolated letters, the recogniser is capable of recognising conjoint character pairs (ligatures) `*ry*', `*rt*' and `*ft*'.

• Three or more common columns

• More than one common column and both objects have more than four pixels per column

• If the distance between their bounding boxes is smaller than $2\,px$ (two pixels), they are searched to determine whether we can find three lines that do not intersect pairwise, that join the two components, and have lengths equal or less than $2\,px$.

Sometimes false joins are created and the next steps deal with them by combining fragmented objects.

4.1.3. Divide Objects

Any character with pixel count more than $500\,px$ is thresholded keeping only low intensities (≤ 90). The objects that are created are returned as separate characters. (See Fig. **2**) for examples.

4.1.4. Recogniser

The bounding boxes created by the preceding algorithms (see Fig. **4** for examples) are turned into features for recognition. Any `i', `j' found are returned according to the position of their dot. The structure of the classifiers is shown in Fig. **5**. Our solution algorithms for the other CAPTCHA schemes have similar recogniser structures, so details of most of these are omitted for the sake of brevity.

4.2. Slashdot

Slashdot CAPTCHAs consist of a single English word with a number of horizontal and vertical lines intersecting multiple characters and each other. The security of this CAPTCHA is founded upon the difficulty of segmenting characters apart (see Fig. **6a**). The characters are connected to each other by the lines and by clumping some characters together horizontally.

Characters are horizontally and vertically displaced but they have no distortions apart from in-plane rotations. They come from a single font, there are both hollow and solid characters and the line width of characters varies from a couple of pixels to more than 10. The character-set is such that line removal will cause damage to the characters and it is difficult to perform pattern recognition.

The scheme was attacked by Bursztein *et al.* [6] who broke it with precision 24% without using any post-processing and 35% by spell-checking the output with an English dictionary. They use a custom implementation of Hough Transforms to identify the lines and then they remove pixels by inspecting their surroundings. They do not provide further segmentation information but they probably use flood-filling segmentation in order to remove the connected components. We tried a similar algorithm but most of the connected components we extracted contained more than one character. We used a histogram based segmentation algorithm to segment the characters further, thus achieving a precision of 44% without the use of a lexicon.

We chose to remove pixels that were close to characters using histogram segmentation instead of removing the line points that were next to characters using other heuristics. Histogram segmentation has the advantage of not causing significant damage to the characters, whereas line point removal may lead to character break-up. Furthermore, histogram segmentation allowed us to segment apart characters that were erroneously connected by collapsing the distance between them. Since they were just touching each other in a single point and not overlapping, the histogram columns in which they were connected contained a small number of characters. Our algorithm consists of the following main stages (see Fig. **6**):

1. The image is thresholded to remove some noise and create a binary image (Fig. **6b**).

2. Identify the pixels of the horizontal and vertical lines

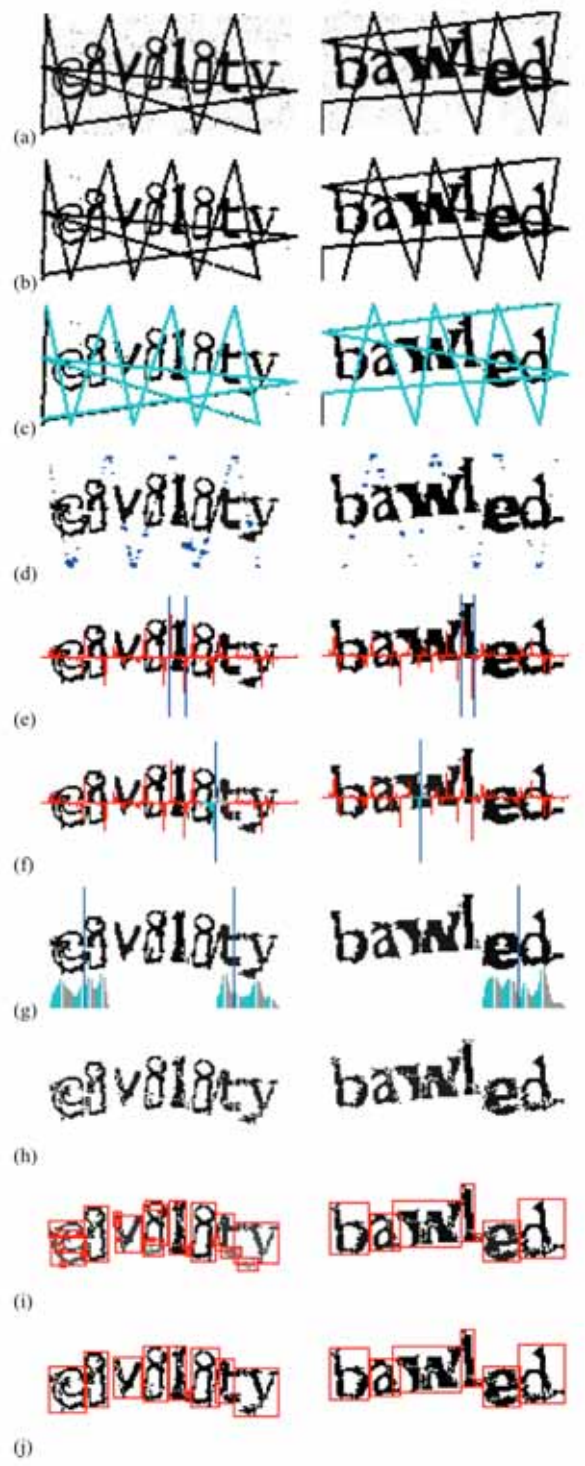

Fig. (6). Slashdot solution stages. (**a**) original images (**b**) thresholded images (**c**) line pattern is detected (**d**) the remaining line fragments (**e**) columns with large height gradient differences (**f**) segmentation columns identified by scanning Δh for positive runs that are followed by negative runs (**g**) segmentation lines between crests (**h**) images after segmentation finishes (**i**) first stage of merging characters (**j**) second stage of merging.

3. Remove the pixels of the lines that are distant from characters

4. Remove pixels in columns where there is a likely character join

5. Join fragmented characters

6. Extract characters using the resulting bounding boxes of the characters (Fig. **6d**)

7. Skip the challenge if very wide (conjoint) characters are found

8. Recognise individual letters and form the result string

4.2.1. Line Pattern Identification

Line detection algorithms return a lot of false positives (characters are positioned so that their pixels create spurious lines). The Slashdot CAPTCHA lines are produced using a deterministic algorithm (Fig. **6c**) that seems to follow these basic steps:

• There are four to six vertical line starts at the top of the challenge and four to six ends at the bottom. Each start is connected to the two closest ends (or one if it is the first or last). Hence the challenge might contain seven to 11 vertical lines.

• One to three horizontal line starts on the left side and one to three ends on the right side. Each start is connected to the two closest ends and the top most start is connected to the vertical start or end that is closer to the right end.

• Occasionally there is a line starting at some point at the bottom of the challenge and connected to the last vertical end.

The horizontal starts and ends of the lines are points on the two sides where there are black pixels for three consecutive columns in two consecutive rows. Vertical points can be similarly identified by our attack algorithm. If an unexpected number of starts or ends is found then our algorithm skips the CAPTCHA.

4.2.2. Line Pattern Pixel Removal

All lines are exactly two black pixels wide and the area around them has much lower intensities than the area around characters. The area around each pixel of the intermediate stage line pattern is examined in the initial challenge (Fig. **6a**). If the pixels before and after the two black ones are white and no more than two of their common neighbours are grey, then the two black pixels are removed.

This algorithm is not able to remove some line pixels (blue areas in Fig. **6d**) because there is speckled noise added to the image and the pattern confuses line intersections with line pixels close to characters. Those pixels create small objects with very large density. The image is scanned for objects that meet those criteria which are then removed.

4.2.3. Removing Pixels from Columns where Characters are Joined

The heights h of the object at each column (difference between first and last black pixel in the column) are calculated and used to calculate the first derivative of heights

$$\Delta h[i] = h[i] - h[i+1]$$

and columns with large gradient are removed (Fig. **6e**).

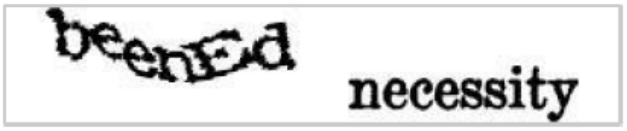

Fig. (7). Example of the most common variant of a reCaptcha challenge.

Table 1. Characteristics of the two most common types of reCAPTCHA words.

Single font used across challenges	Challenges have **different** fonts
Random sequence of characters	**English** word
Wavy transformation	Similar to **scanned printed** text
Collapsed (vertically squashed) characters	**Large distances** between characters

Character joins often have small '*v*' like shapes that are less common inside a character. The characters that have similar shape are characters that have thick lines. Hence when the line of a join is identified all the pixels of the line have independent probability of being removed. This tends to preserve thick characters since not all of their pixels are removed but disconnects character joins. The middle points of the joins are identified by scanning Δ. for positive runs that are followed by negative runs. The runs must be of medium size, three to six pixels (Fig. **6f**).

For each column the total number, c_{tot}, of pixels in the current column and in the two neighbouring columns is calculated. A run of increasing and then decreasing c_{tot} is called a crest. Most characters consist of exactly two crests and all segmentation points are found at points between the end of the previous and the start of the next crest.

For each object that is too long to consist of a single character the crests are examined. Objects that have four crests are segmented at the join between the second and the third crest. Objects that have three crests are segmented at the join between the widest crest and the two other crests. If an object with more than four crests is found the challenge is skipped. The two columns at the join are examined and the one that has the least pixels is chosen and its pixels are removed in a similar way as above (Fig. **6g**).

The line pixels that were not previously removed are iterated and each pixel has independent probability $\frac{7}{17}$ to be removed. This is done because character-to-character joins contain less pixels in a column than internal character joins (see Fig. **6h** for the final result of segmentation).

4.2.4. Join Character Fragments

Characters with lines of small thickness are often fragmented but the fragments overlap in a lot of columns. Objects whose bounding boxes overlap in more than $\frac{3}{10}$ of their columns are merged together (Fig. **6i**) and then neighbouring objects of the same height and density are also merged (Fig. **6j**).

Fig. (8). A newer reCAPTCHA challenge with a picture of a number.

4.3. reCAPTCHA

reCAPTCHA challenges consist of two words (Fig. **7**), with one being the word that needs to be entered correctly (the control word) and the answer given for the other word is used as an OCR solution for digitising books. If the submitted string for the control word is at edit distance one or zero from the actual string of the control word, the user is regarded as human and the answer to the other string is stored by the system [22].

Based on the characteristics summarised in (Table **1**) and an empirical analysis of reCAPTCHA samples (a few hundred challenges from the training set), we surmised that the string that has been distorted by a wave transform is used as the control word and we concentrated our efforts on implementing an algorithms that recognises it.

reCAPTCHA occasionally uses other versions of the challenges that consist of a wavy word and the image of a number (Fig. **8**). In some of those challenges, both words are required to solve the challenge which is the case for the challenges that are sent after a false submission. We only focused on the common reCAPTCHA challenges and we achieved a precision of 5% against the wavy word. Although this is not substantially above the target minimum of 1%, it shows that there are a number of potential vulnerabilities with this CAPTCHA.

The intuition behind our algorithm for reading the wavy word is that letters are constructed by connecting different lines with the great majority of the letters having lines that are perpendicular to the median line the word. The algorithm tries to find those lines and then by examining the area between them it decides where the end of the current character is. It constructs a divisions that will create the least possible damage to the characters.

4.3.1. Algorithm Outline

1. Identify the boundaries of the two words

2. Select the wavy word and discard the other

3. Check the inclination of letters of the challenge

4. **Up-sample** the image to allow more fine-grained processing. The image is scaled by a factor of 4 in each dimension.

5. **Dilate** it to reduce the thickness of character lines

6. **Blur** it using Gaussian kernels to remove the noise around characters

7. **Threshold** it to get binarised image

8. The positions of i,j are identified and the letters are removed from initial consideration

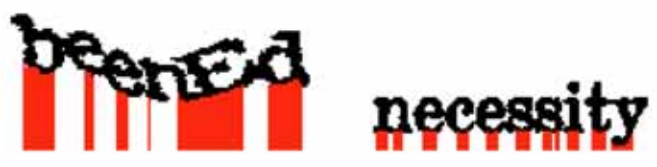

Fig. (9). The red lines represent the distances that are included in the calculation of the coefficients of variation of the two words.

9. Starting from the left, the end of the current letter is identified and a line is drawn removing the rest of the character:

(a) Find the current starting line

(b) Find up to three unique character lines

(c) Calculate the next segmentation line

(d) Remove all the pixels between the start line and the dividing line and save them for recognising the character

(e) If we did not reach the end continue until a dividing line has been computed

10. If at any point the next segmentation cannot be calculated, then fail

11. The letters are recognised, including any i, j that were previously identified

4.3.2. Find the Boundaries of the Words

The positions of the words are constant: the left-most word is positioned at the top of the image whereas the right-most word is positioned at the bottom of the image. Hence the end point of the first word can be recovered by moving from $x = \frac{2}{3} width$ backwards as long as there are empty columns or columns that have:

$$bottom\ distance > \frac{2}{3} height$$

$$\vee$$

$$top\ distance > \frac{1}{3} height$$

The start of the second word can be recovered similarly.

4.3.3. Select the Wavy Word and Discard the Other

If one of the words has more than three columns with no pixels we can safely assume that it is not the wavy word. In a different case, the word whose letters' distances have the largest coefficient of variation from the bottom of the image, is selected. Columns that have height (distance between the first and last pixel of the column) lower than the median height are not included in the calculation in order to discard character arcs. The included heights are shown in (Fig. **9**).

4.3.4. Further Pre-processing Steps

The resulting image (Fig. **13b**) contains a single object or multiple objects, not necessarily divided at character-to-character joins. This is because character-to-character joins are made to look like characters' internal points (darker pixels) and some internal character joins have very high intensities. This does not allow us to use the pixels' intensities to identify the joins and hence thresholding the image removes no useful information.

4.3.5. `i′,`j′ *Removal*

The removal of ` $i$ ',` j ' provides the later stages of the algorithm with useful information since it defines the boundaries of the nearby characters. Their positions can be

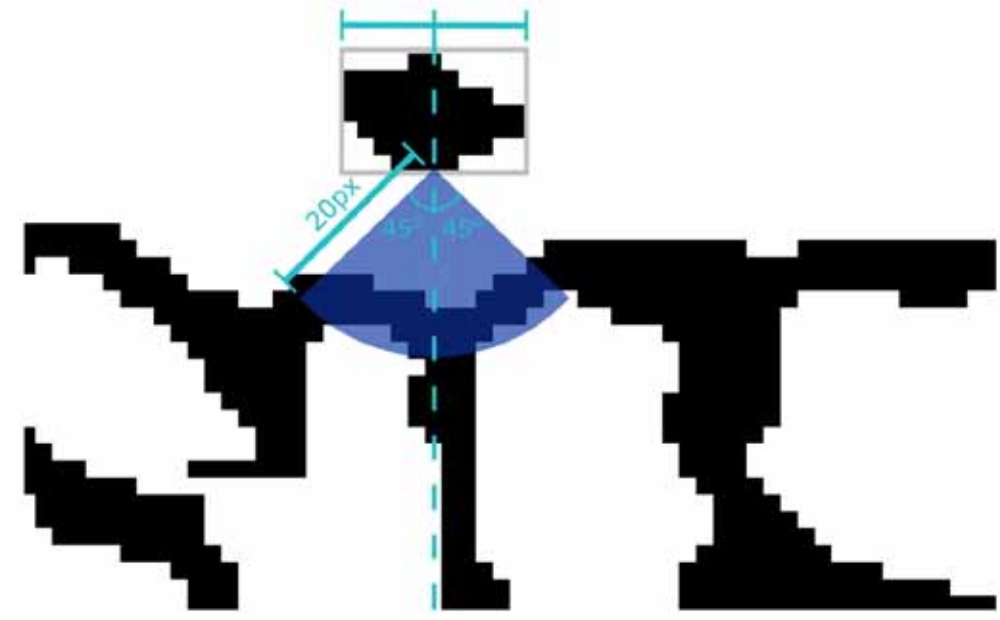

Fig. (10). Enlarged detail of the area that is scanned for the selection of the best line that describes the current ` $i$ ' or ` j '.

Fig. (11). The blue line is the best line to describe the line of the `*i*s as calculated from the above algorithm. The area between the red lines is the estimated area of the `*i*'.

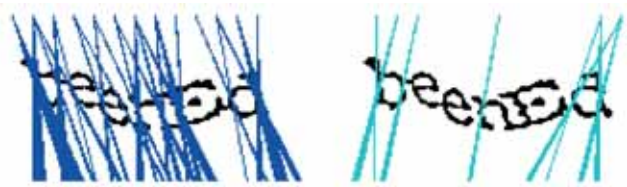

Fig. (12). Left: lines with positive orientation gradient. **Right:** lines with negative gradient.

identified since their dots almost always form a separate object that has specific dimensions.

The image is scanned for the dots of `i',`j' which are objects that have

$$width < 16\ px \wedge height < 12\ px$$

and for any column that they span there is no black pixel above them.

Starting from the middle of the dot (Fig. **10**) at its lowest row we test all the lines that have inclination smaller than 45 degrees from the perpendicular (dotted light blue line). A candidate line that describes the line of the `.' must have the first black pixel less than $20\ px$ from the start (the blue area). From those lines the one that contains more black pixels is selected to describe the `*i*'

The red lines in (Fig. **11**) are the parallels to the selected line that describes the `*i*' (the light blue line) and they start from neighbouring pixels of the dot's corners. The area between the two red lines is assumed to belong to the letter and it is removed from further consideration.

4.3.6. Check the Inclination of the Wavy Word

In order to increase the probability of correct decisions in subsequent steps and decrease the number of cases the algorithm needed to consider, only challenges with the majority of characters inclined left are examined (56% of total challenges).

Starting from the left top of the current image and going to the right, top lines at different inclinations are examined and two counters are maintained, representing lines that contain more than $20\ px$ and have positive and negative gradient respectively. (Fig. **12**) shows the lines that are counted in this calculation.

4.3.7. Find the Initial Line

The tangent to the left side of the first character line is the first guide line to be calculated. This is the line that has

Fig. (13). Major steps in solving a reCAPTCHA challenge: (**a**) Initial challenge (**b**) The wavy word after pre-processing is done (**c**) The calculated segmentation lines (**d**) The extracted characters of the challenge

the most black pixels. In the case of ties the last line to be encountered is selected.

4.3.8. Find Three Guide Lines

In a window equal to the maximum width of a character, all lines with inclination to the perpendicular of -5^o to + 70 are checked and the best lines that describe the characters' vertical lines are selected. Lines at small positive angles are checked first, then those at small negative angles and finally those at large positive angles. In each iteration the angles are checked in order of increasing magnitude. Lines that have more than 20 black pixels and fewer than 4 white pixels are compared against all the previously found lines. The resulting lines are sorted by x-coordinate and the first three are used as the guide lines.

4.3.9. Next Segmentation Line

The starting line is only used if there are enough black pixels between the start line and the first guide line. The areas between consecutive guide lines are examined to identify whether the area is a part of a character or is the join between two characters.

The first area that does not contain a part of a character is examined in order to find the most likely segmentation line that passes through it. If two areas between guide lines are examined the character can only be an `*m*', hence if the two areas behind it can be recognised as two arcs the segmentation line is just after the last guide line. If the algorithm is not able to identify a possible segmentation line it fails.

4.4. BotDetect (Wavy Chess)

BotDetect CAPTCHA challenges (Fig. **14a**) have two different layers, the first is a background that resembles a chess board with wavy dividing lines. The left-most top-

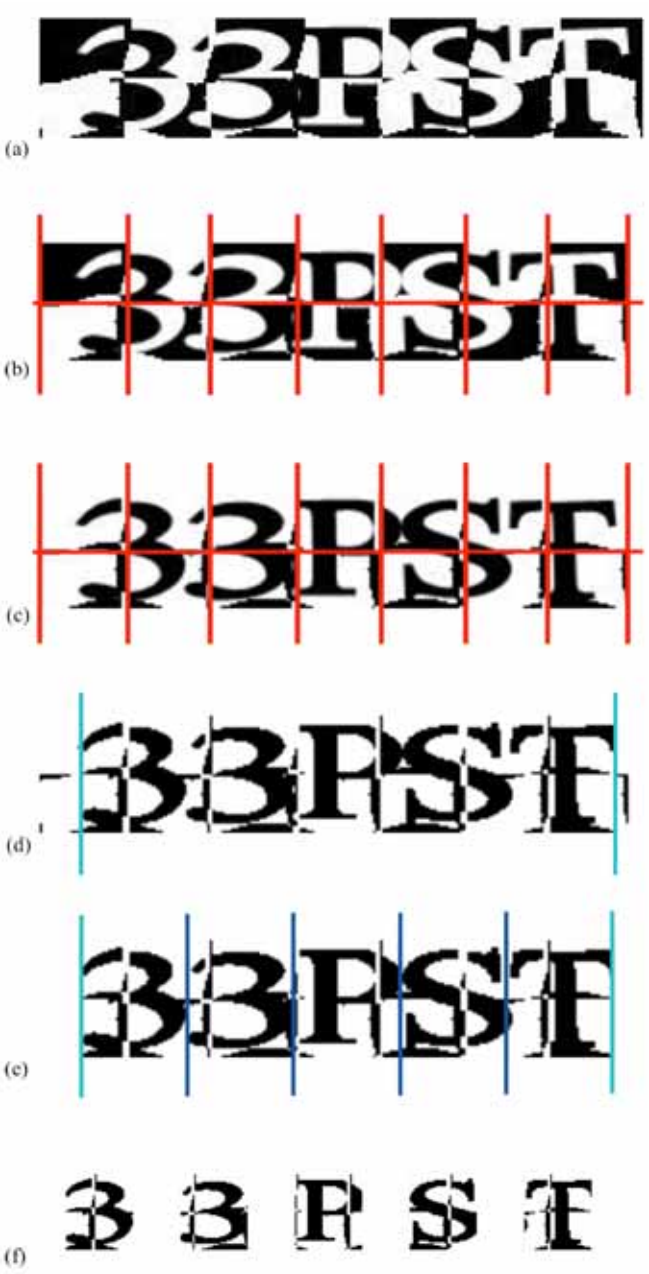

Fig. (14). The solution stages of our BotDetect (Wavy Chess) attack: (**a**) initial CAPTCHA (**b**) grid approximation (**c**) reversal of the chess effect (**d**) starting and ending lines (**e**) segmentation lines (**f**) Extracted characters.

most rectangle is sometimes black and sometimes white and the colour of neighbouring rectangles alternates. There are two rows of rectangles and a variable number of columns (four to eight). The text string is superimposed onto this background through inversion. The character-set consists of nineteen Latin letters and six Arabic digits. Three different fonts are used but each challenge has characters from a single font. Characters of the same challenge have the same width, but width differs between challenges.

Characters have black and white areas that are aligned along a different grid of straight lines at regular distances. The horizontal line is positioned at the centre of the image. The start of the grid does not match the start of the image but it has the same number of columns as the background grid. Characters are pasted at positions such that there is a small part that has the same colour as the background. The size of this patch is such that it does not interfere much with human readability but it is not possible to extract character parts with flood-filling segmentation.

There is no published attack for the current version of the scheme. El Ahmad *et al*. [25] attacked an earlier version of scheme that had straight rather than wavy lines by identifying the chess board and then, if the majority of pixels was black, they reversed the intensities of pixels. The designers of the scheme have addressed these issues since now the letters and the background change intensities at different positions. The algorithm would need to identify the chess board of both the background and the foreground in order to be effective. The wavy lines and the merge of the two patterns in unpredictable ways would make this difficult. Our algorithm finds an approximation of the background grid, reverses the grid, and then it uses the fact that all challenges have five letters of the same width to achieve precision 88%.

4.4.1. Algorithm Outline

1. Create an approximation of the chess grid

2. Identify the rectangles where pixel intensities are inverted

3. Correct the inversion

4. Find the start and end of the character's string

5. Divide the area between the start and end into five equal chunks

6. Recognise each letter and combine them to produce the final result

4.4.2. Grid Approximation

We scan the top-most row and the black pixels (intensity ≤ 100) whose neighbour is white (≥ 230) or vice versa are identified. The grid is created from vertical lines at those points and a horizontal line in the middle of the challenge.

The colour of the left-most top-most rectangle is the same as the value of left-most, top-most pixel. If the pixel is white the values of the pixels in the even rectangles of the first line and of the odd rectangles in the second row are flipped ($i_{new} = 255 - i$). If it is black, intensities of the complement rectangles are flipped.

4.4.3. Segmentation

Characters are connected to each other because of lines created from the previous steps. The start of the characters is the left-most column with more than $10\,px$ and the end of the characters is the right-most such column.

Each challenge has exactly five characters of the same width. Hence the start and end points of each character can be computed by dividing the total distance occupied by the character into 5 parts of equal width.

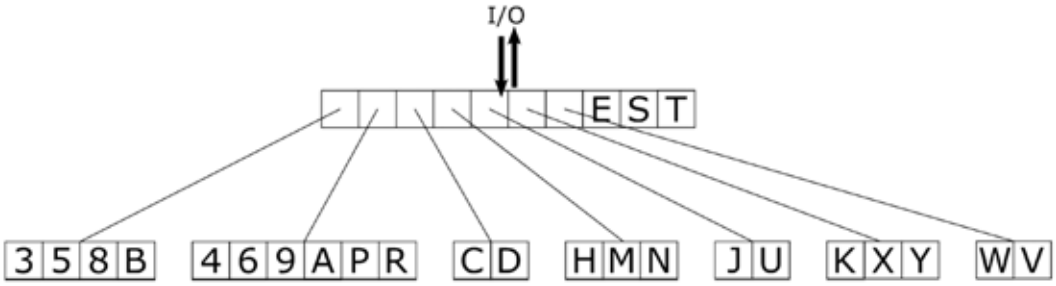

Fig. (15). Structure of the character recogniser for our BotDetect (Wavy Chess) attack.

$$start_i = start_0 + \frac{word\ width}{5} \times i$$

Sometimes characters cover the top most row and bogus points are found. Hence any challenges where more than seven such points are found are skipped.

The recogniser (Fig. **15**) uses a main classifier and several sub-classifiers of character classes. Characters of all three fonts used in BotDetect CAPTCHAs are recognised.

4.5. JCAPTCHA library - Hotmail and Gmail CAPTCHAs

We also examined the security of CAPTCHAs produced by the JCAPTCHA library that can be used at the server side of web applications and websites. A major disadvantage of the JCAPTCHA schemes is that for an accurate implementation of the challenges one needs to have the exact typeface that the developer of the scheme was using when he implemented the library.

We examined the Hotmail scheme, which was easily attacked using techniques similar to [29], and the Gmail scheme (see Fig. **16** for examples), for which no successful attack could thus far be developed. The challenges were most of the times not recognisable by humans as well.

Gmail challenges consist of a seven letter phonetic word. Characters are red, blue or green and they are merged with each other, forming a single object. Vertical lines of characters overlay each other. An algorithm similar to our reCatpcha attack cannot be used since there are neither thin line segments nor areas created by character joins, hence it is very difficult to accurately identify segmentation lines. A vertical morphology analysis similar to our Slashdot solver cannot be used either, since characters have similar morphology in both character-to-character joins and internal joins.

Fig. (16). Two Gmail challenges produced by the JCAPTCHA library. No sufficiently robust attack against Gmail CAPTCHAs has yet been devised. **Top:** abliter. **Bottom:** neethed.

The algorithm[1] for attacking the Hotmail CAPTCHAs is presented below. (Fig. **17**) shows the initial challenge and the results of segmentation.

Fig. (17). Top: The initial Hotmail challenge. **Bottom:** Segmented character are shown in different colours.

4.5.1. Algorithm Outline

1. Threshold the image in order to remove the background

2. Extract the connected components of the image

3. Remove connected components that are too small

4. Remove connected components that satisfy

$(a < 600 \wedge \rho < 0.22) \vee (a < 250 \wedge \rho < 0.35)$

where α is the pixel count and

$$\rho = \frac{a}{width \cdot height}$$

5. Examine wide objects and check if they have a column with small pixel count

6. If they do, then divide the object into two at that column, otherwise the challenge is skipped

7. If exactly 8 objects were found recognise the letters and return the string, otherwise skip the challenge

5. RESULTS

We evaluated our algorithms by checking the number of correctly solved challenges on a test set. The achieved

[1]http://jCAPTCHA.sourceforge.net/

Table 2. The number of samples used for training, validating, and testing each CAPTCHA breaker.

	Training Set	Validation Set	Test Set
Wikipedia	2075	411	800
Slashdot	2038	282	800
reCAPTCHA	2712	205	1044
BotDetect	2573	300	1000
Hotmail	2500	500	1000

precisions suggest that the CAPTCHA schemes we were able to break have obvious vulnerabilities that need to be addressed.

The time needed to solve a single challenge is very small compared to the time that humans need to solve the corresponding challenges. This allows for the use of even more complex transformation at different parts without worrying about time limitations.

The learnability of the algorithms was also examined in order to see the number of labelled challenges that is needed for training the algorithms to achieve the desired performance. In most cases that number was very small, allowing the rapid development of new algorithms when minor changes are made to the CAPTCHA scheme.

5.1. Data

Sample challenges for each scheme were downloaded from their websites using automated scripts. The scripts downloaded challenges and randomly divided them into three sets (Training, Validation, Test). Wikipedia and Slashdot challenges were collected from their sign-up page whereas reCAPTCHA and BotDetect challenges were downloaded from their online demo. Hotmail samples were generated using the JCAPTCHA library.

The servers were occasionally responding to our scripts with the same images, and such duplicates were discarded. All challenges were then manually labelled with groundtruth.

A subset of the training set was used to design the algorithms and the accuracy of the algorithms was refined against this validation set.

The sizes of the sets are outlined in Table **2**. Training sets of approximately 2500 samples were used and the performance of the algorithms was checked against a test set of 800 or more challenges.

5.2. Evaluation

We applied the following metrics:

$$Precision = \frac{\#true\ positive}{\#true\ positive + \#false\ positive}$$

$$Recall = \frac{\#true\ positive}{\#true\ positive + \#false\ negative}$$

$$Accuracy = \frac{\#correct\ solutions}{\#total\ solutions}$$

The performance of the character recognisers was evaluated against artificially created characters similar to those encountered in CAPTCHA challenges and characters extracted by the segmentation algorithms from training set images. All classifiers achieved accuracies of > 99%.

Table **3** summarises the precision, recall and accuracy levels achieved by our algorithms for each scheme. Table **4** summarises precisions reported by other published attacks, and (Fig. **18** and **19**) illustrates precision of our methods relative to others.

Our attacks against Hotmail, Slashdot and Wikipedia perform similarly or better than other published attacks, the attack against reCAPTCHA algorithm achieves precisions higher than the 1% threshold for a scheme being broken. With precision 88% on BotDetect, which is comparable to human precision [5], the use of the BotDetect scheme is unlikely to provide any security against attacks such as ours.

The attack against reCAPTCHA (5%) does not deal with book words, but after a wrong answer both words must be answered correctly and reCaptcha uses additional metrics that might reduce our precision. In any case, our results show that there are definite vulnerabilities in the reCAPTCHA scheme also.

Only the Gmail CAPTCHA has not yet proved vulnerable to our attacks.

5.3. Learnability

To evaluate how the efficacy of our solvers depends upon the amount of available data, challenges from the training sets were randomly divided to create smaller training subsets of 100, 200, 300, 500, 750 and 1000 challenges. Similarly, validation sets equal to $\frac{1}{5}$ of the corresponding training corpus were created. In order to create the training and validation letter corpora the following procedure was used:

1. For each challenge set iterate over the challenges and solve them

2. If the number of characters return is equal to the number of characters of the groundtruth

(a) Assume that the challenge is segmented correctly.

Table 3. Performance statistics of our CAPTCHA attack algorithms.

	Precision (%)	Recall (%)	Accuracy (%)	Answered (%)
Wikipedia	72 ± 3	100	67	93
Slashdot	44 ± 4	88	26	53
reCAPTCHA	5 ± 3	100	0.11	25
BotDetect	88 ± 2	96	81	91
Hotmail	68 ± 5	42	50	31

Table 4. Precisions reported by other published attacks.

Scheme	Precision	Published Source
3*Hotmail	3*0.61	Segmentation accuracy 0.95, all challenges have eight characters, each character is recognised correctly with probability 0.95 [29].
Wikipedia	0.25	2*Precision recorded in [6]
Slashdot	0.35	

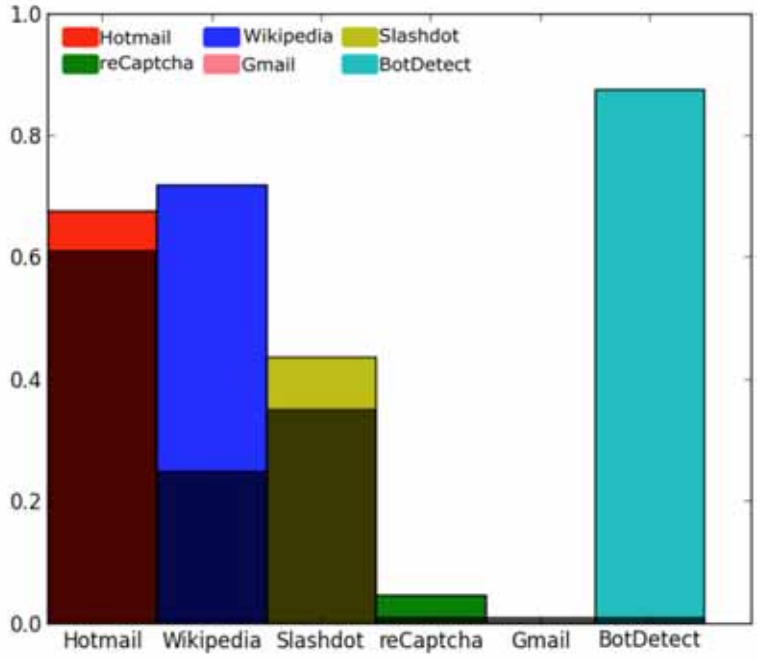

Fig. (18). Precisions achieved by our attacks relative to previously published attacks and 1% target benchmark (the darker area of each bar).

(b) Extract the characters.

(c) Label each character according to its position.

Then the character recogniser of the scheme was trained using training, validation pairs. The precision was calculated against the full test corpus of the respective CAPTCHA scheme.

All algorithms apart from reCAPTCHA achieve precisions higher than 7% when trained with no more than 100 challenges and their performance increases as the training set increases. It can be seen that those schemes are not only weak but they require minimal data collection to be broken. The first correctly solved reCAPTCHA challenge occurs when the training set is 1000. The expensive data collection and labelling increases the total cost of the attacker and hence it increases the percentage above which the breaker will be profitable.

5.4. Time

Each challenge must be answered in less than 30s [27] which is close to the human average [5]. The time taken by our program (total of segmentation and recognition) are significantly lower than that limit. Median solution times are lower than 250 ms for all schemes and there is no challenge

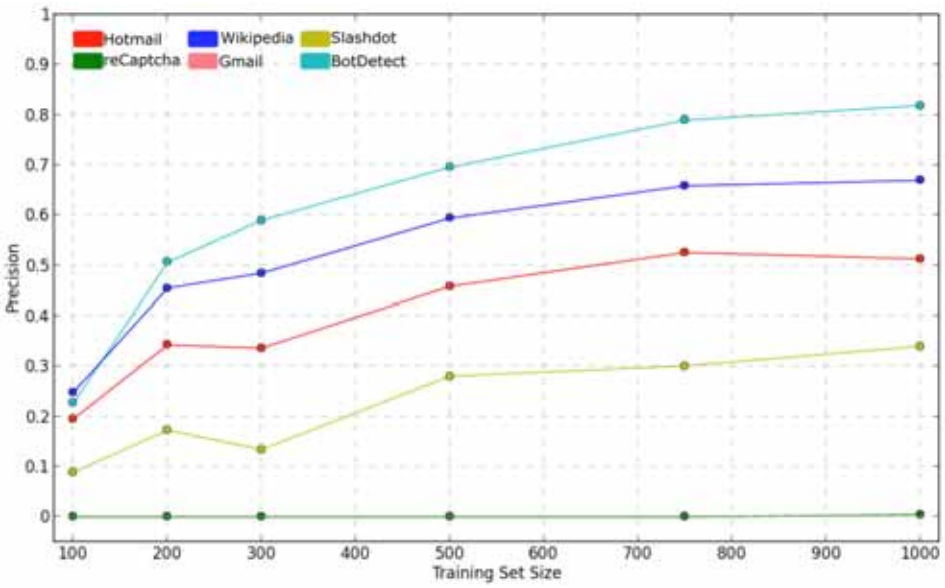

Fig. (19). Analysis of learnability (dependence of precision on training set size).

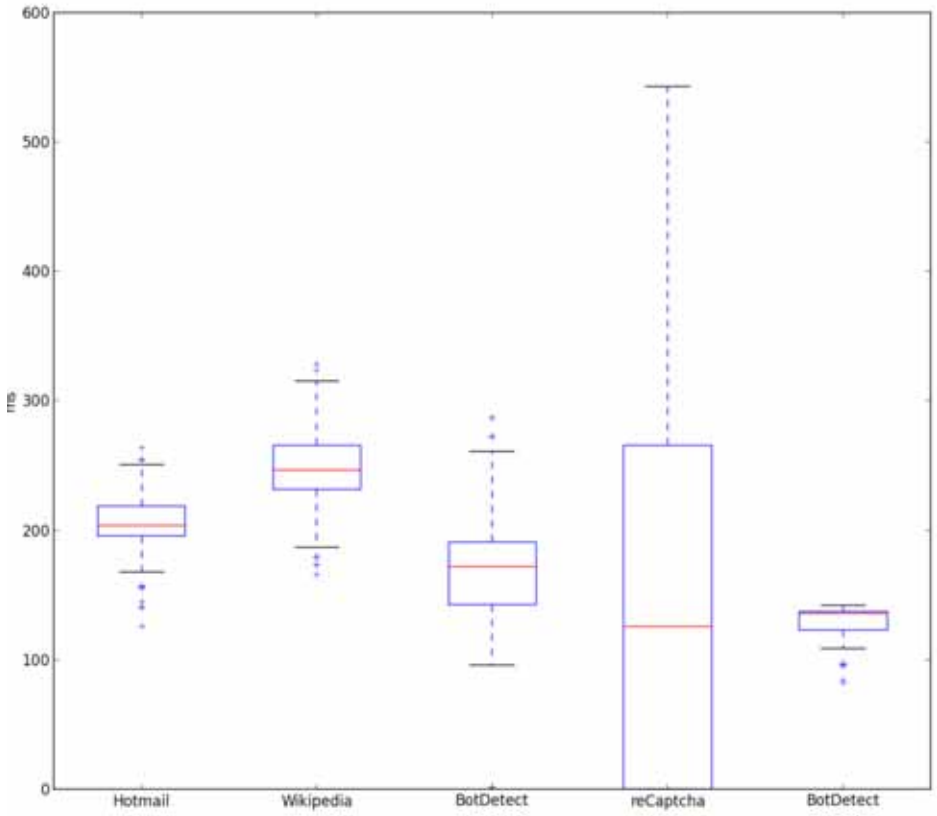

Fig. (20). Time distribution box plots of our attacks algorithms for each CAPTCHA scheme.

that requires more than 600 *ms*. Fig. (**20**) shows box charts of time overheads for each algorithm.

A lot of reCAPTCHA challenges (44%) return almost immediately because the characters are inclined to the right and the challenge is skipped. This is the reason for the low median time. Other schemes have much lower variability mainly because of changes in the size of the image or more connected/disconnected letters that need to be corrected. Bottom outliers in Slashdot and BotDetect are due to failure in grid reconstruction. It is clear that there are no issues with time and algorithms can safely use more time consuming methods.

CONCLUSION

Achievements

We have achieved the following main tasks:

• Improvements made to known attacks against Hotmail, Wikipedia, and Slashdot challenges

• Successful attacks against BotDetect's Wavy chess, reCAPTCHA and the new Wikipedia scheme

• Implementation of a library that includes customisable segmentation algorithms and customisable character recognisers. The library can serve as a tool for further investigating CAPTCHA security.

Suggested CAPTCHA Improvements

Based on the results of our implementation and evaluation and the knowledge we have gained from the research in this area, we have the following suggestions for improving CAPTCHA security.

CAPTCHA providers need to avoid any regularities in the algorithm that they use to construct the challenges. The lines of Slashdot were easily recreated once the pattern used to create them was identified and the darker area around characters helped in removing lines without damaging characters. Similarly the regularities of BotDetect's chess effect and the same width of characters in a challenge helped break it. Additionally they need to vary the challenges more significantly. This would force the attackers to have a very generic algorithm and this will make tuning of the hyper-parameters difficult. In all but reCAPTCHA, finding a set of hyper-parameters that was giving sufficient precision against a scheme did not require more than a couple of hours. Because of the in-class variability of reCAPTCHA challenges, even though the precision of the algorithm was 22% against the validation set, it dropped to 5% against the test set.

CAPTCHA providers can also try to identify possible data collection from attackers and try to introduce greater latency or present potential bots with unforeseen difficulties. A category of challenges that fail even if a correct answer is given but have characteristics that are different from the regular challenges (*eg.* different font, characters not used in regular challenges) can be used for this purpose. Those challenges must not be easily spotted in order not to be excluded from the training set.

Furthermore, challenges that are very difficult (or impossible) for human users and likely to be unsolvable by automated techniques can be sent to potential bots. Humans will be expected to skip the challenge whereas bots will attempt an answer. Challenges that contain nonsense letters or pictograms can serve this purpose.

Future Outlook

It is unclear whether some of the currently used CAPTCHA schemes serve their purpose. An attacker with a similar tool to ours that implements CAPTCHA breaking methods can attack several schemes with minimal work. The number of required challenges before a successful algorithm is produced is small and the number of parameters to be tuned is not large. CAPTCHAs need to use more randomness and variability in all features.

Of the three schemes (Wikipedia, Slashdot, BotDetect) that were broken with very high precision, only Wikipedia recently changed their scheme but it does not defend against our algorithm. Our algorithms show that reCAPTCHA, the most commonly used CAPTCHA, is not invincible either. Its security is provided not only from the challenge but from the metrics that they keep about website users. Alternative CAPTCHAs that use non-standard, easier to use challenges are gaining credibility [2].

It is likely that challenges will get easier to solve (with varying difficulty for different users) and bots will be categorised by their behaviour against challenges and not only by their ability to solve them.

CONFLICT OF INTEREST

The authors confirm that this article content has no conflict of interest.

ACKNOWLEDGEMENTS

Declared none.

REFERENCES

[1] K. Chellapilla, K. Larson, P.Y. Simard, and M. Czerwinski, "Building segmentation based human friendly human interactive proofs", In: *Proceedings of the Second International Workshop on Human Interactive Proofs*, Springer-Verlag, 2005, pp. 1-26.

[2] A. Andreas, D. Anastasios, M. Vasileios, and K. Ioannis, "GPU acceleration for support vector machines", In: *12th International Workshop on Image Analysis for Multimedia Interactive Services (WIAMIS)*, Delft, The Netherlands, April 2011.

[3] P. Baecher, M. Fischlin, L. Gordon, Ro. Langenberg, M. Lützow, and D. Schröder, "CAPTCHAs: The good, the bad, and the ugly", *Sicherheit. LNI*, vol. 170, pp. 353-365, 2010.

[4] H.S. Baird, A.L. Coates, and R.J. Fateman, "Pessimal Print: a reverse Turing test", *Int. J. Doc. Anal. Recognit.*, vol. 5, no. (2-3), pp. 158-163, 2003.

[5] E. Bursztein, S. Bethard, C. Fabry, D. Jurafsky, and J.C. Mitchell, "How good are humans at solving CAPTCHAs a large scale evaluation", In: *Security and Privacy*, May 2010.

[6] E. Bursztein, M. Martin, and J.C. Mitchell, "Text-based CAPTCHA strengths and weaknesses", In: *Computer and Communications Security(CCS)*, October 2011.

[7] C.C. Chang and C.J. Lin, "LIBSVM: A library for support vector machines", *ACM Trans. Intell. Syst. Technol.*, vol. 2, no.27,pp. 1-27, 2011.

[8] K. Chellapilla, K. Larson, P. Simard, and M. Czerwinski, "Computers beat humans at single character recognition in reading based human interaction proofs (HIPS)", In: *Proceedings of the 2nd Conference on Email and Anti-Spam*, pp. 21-22, 2005.

[9] C.W. Chong, P. Raveendran, and R. Mukundan, "A comparative analysis of algorithms for fast computation of Zernike moments", *Pattern Recognit.*, vol. 36, no. 3, pp.731-742, 2003.

[10] A.S. El Ahmad, J. Yan, and M. Tayara, "*The robustness of Google CAPTCHA*s", Technical Report 1278, School of Computing Science, University of Newcastle upon Tyne, September 2011.

[11] L. Fedorovici and F. Dragan, " A comparison between a neural network and a SVM and Zernike moments based blob recognition modules", In: *Applied Computational Intelligence and Informatics (SACI), 6th IEEE International Symposium 2011*, pp. 253-258.

[12] H. Gao, W. Wang, and Y. Fan, " Divide and conquer: An efficient attack on Yahoo! CAPTCHA", In: *IEEE 11th International Conference on, Trust, Security and Privacy in Computing and Communications (TrustCom), 2012*, pp. 9-16.

[13] G. Mori, and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual CAPTCHA", In: *Proceeding of Computer Vision and Pattern Recognition, 2003 IEEE Computer Society Conference*, 2003, vol. 1, pp. 1-134.

[2] BBC News, 2013: "Ticketmaster dumps 'hated' CAPTCHA verification system." http://www.bbc.co.uk/news/technology-21260007

[14] R. Mukundan, and K.R. Ramakrishnan, "Moment functions in image analysis: theory and applications", World Scientific, vol. 100, 1998.

[15] M. Naor, "Verification of a human in the loop or identification via the turing test", Weizman Institute report, 1996.

[16] G.A. Papakostas, Y.S. Boutalis, C.N. Papaodysseus, and D.K. Fragoulis, "Numerical stability of fast computation algorithms of Zernike moments", *Appl.Math.Comput.*, vol.195, no.1, pp. 326-345, 2008.

[17] C. Singh and E. Walia, "Fast and Numerically Stable Methods for the Computation of Zernike Moments', *Pattern Recogn*., vol. 43, no.7, pp. 2497-2506, 2010.

[18] P.Y. Simard, "Using machine learning to break visual human interaction proofs", in Advances in Neural Information Processing Systems 17, Neural Information Processing Systems *NIPS 2004*, pp. 265-272, MIT Press, 2004.

[19] C. Singh, and E. Walia, "Algorithms for fast computation of Zernike moments and their numerical stability", *Image Vision Comput.*, vol. 29, no.4, pp. 251-259, 2011.

[20] L. von Ahn, M. Blum, and J. Langford, "Telling humans and computers apart automatically or how lazy cryptographers do AI", *Computer Science Department*, p. 149, 2002.

[21] L. von Ahn, M. Blum, N.J. Hopper, and J. Langford, "CAPTCHA: Using hard AI problems for security". *In: Proceedings Of Eurocrypt*, pp. 294-311. Springer-Verlag, 2003.

[22] L. von Ahn, B. Maurer, C. McMillen, D. Abraham, and M.Blum, "reCAPTCHA: Human-based character recognition via web security measures", *Science*, vol. 321,no. 5895, pp. 1465-1468, 2008.

[23] J. Wilkins, "Strong CAPTCHA guidelines vo1. 2", Technical report, 2009.

[24] Yan and A.S. El Ahmad, " Breaking visual CAPTCHAs with naive pattern recognition algorithms", *In: Computer Security Applications Conference*, 2007. ACSAC 2007. 23rd Annual, pp. 279-291, December. 2007.

[25] A.S.El Ahmad, J. Yan, and W.Y. Ng, " CAPTCHA Design: Color, Usability, and Security", *Internet Comput.*, IEEE , vol.16, no.2, , pp. 44-51, March 2012.

[26] T. Y. Zhang, and C. Y. Suen, "A fast parallel algorithm for thinning digital patterns", *ACM Commun.*, vol. 27, no. 3, pp.236-239, March 1984.

[27] B.B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai, " Attacks and design of image recognition CAPTCHAs", *In: Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pp. 187-200, New York, NY, USA, 2010. ACM.

[28] A. Rusu, A. Thomas, and V. Govindaraju, "Generation and use of handwritten CAPTCHAs", *Int. J. Doc. Anal. Recog*., vol. 13, no. 1, pp. 49-64, 2010.

[29] J. Yan, and A.S. El Ahmad, "A low-cost attack on a Microsoft CAPTCHA", *In: Proceedings of the 15th ACM conference on Computer and communications security*, pp. 543-554, 2008.

Received: February 13, 2014 Revised: February 28, 2014 Accepted: June 28, 2014