A Low-cost Attack on a Microsoft CAPTCHA

Jeff Yan
School of Computing Science
Newcastle University, UK
Jeff.Yan@ncl.ac.uk

Ahmad Salah El Ahmad
School of Computing Science
Newcastle University, UK
Ahmad.Salah-El-Ahmad@ncl.ac.uk

ABSTRACT

CAPTCHA is now almost a standard security technology. The most widely deployed CAPTCHAs are *text-based* schemes, which typically require users to solve a text recognition task. The state of the art of CAPTCHA design suggests that such text-based schemes should rely on segmentation resistance to provide security guarantee, as individual character recognition after segmentation can be solved with a high success rate by standard methods such as neural networks.

In this paper, we present new character segmentation techniques of general value to attack a number of text CAPTCHAs, including the schemes designed and deployed by Microsoft, Yahoo and Google. In particular, the Microsoft CAPTCHA has been deployed since 2002 at many of their online services including Hotmail, MSN and Windows Live. Designed to be segmentation-resistant, this scheme has been studied and tuned by its designers over the years. However, our simple attack has achieved a segmentation success rate of higher than 90% against this scheme. It took on average ~80 ms for the attack to completely segment a challenge on an ordinary desktop computer. As a result, we estimate that this CAPTCHA could be instantly broken by a malicious bot with an overall (segmentation and then recognition) success rate of more than 60%. On the contrary, the design goal was that automated attacks should not achieve a success rate of higher than 0.01%. For the first time, this paper shows that CAPTCHAs that are carefully designed to be segmentation-resistant are vulnerable to novel but simple attacks.

Categories and Subject Descriptors

D.4.6 Security and Protection, H.1.2 User/Machine Systems.

General Terms

Security, Human Factors.

Keywords

CAPTCHA, robustness, segmentation attack, usability, Internet security

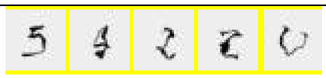
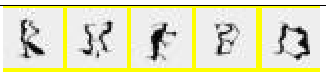
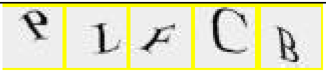
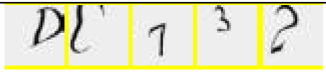
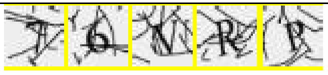
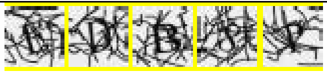
1. INTRODUCTION

A CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is a program that generates and grades tests that are human solvable, but intend to be beyond the capabilities of current computer programs [1]. This technology is now almost a standard security mechanism for defending against undesirable or malicious Internet bot programs, such as those spreading junk emails and those grabbing thousands of free email accounts instantly. It has found widespread application on numerous commercial web sites including Google, Yahoo, and Microsoft's MSN.

The most widely used CAPTCHAs are the so-called *text-based* schemes, which rely on sophisticated distortion of text images aimed at rendering them unrecognisable to the state of the art of pattern recognition methods. The popularity of such schemes is due to the fact that they have many advantages [4], for example, being intuitive to users world-wide (the user task performed being just character recognition), having little localization issues (people in different countries all recognise Roman characters), and of good potential to provide strong security (e.g. the space a brute force attack has to search can be huge, if the scheme is properly designed).

A good CAPTCHA must be not only human friendly, but also robust enough to resist to computer programs that attackers write to automatically pass CAPTCHA tests (or *challenges*). (CAPTCHA is an ideal research topic in the young interdisciplinary field of usable security, which has gained increasing attentions in the recent years.)

Table 1. Recognition rate for individual characters under different distortions (all data in this table are taken from [6])

Characters under typical distortions	Recognition rate
	~100%
	96+%
	100%
	98%
	~100%
	95+%

Early research suggested that computers are very good at recognising single characters, even if these characters are highly distorted [6]. Table 1 shows characters under typical distortions, along with success rates that a neural network can achieve to recognise them. It is established in [6] that if the positions of characters are known in challenge images generated by a CAPTCHA, then breaking this scheme is just a pure recognition problem, which is a trivial task with standard machine learning techniques such as neural networks [12].

However, when the location of characters in a CAPTCHA challenge is not known a-priori (e.g. in the following images taken from [4]), state of the art (including machine learning) methods do not work well in locating the characters, let alone recognising them.

The problem of identifying character locations in the right order, or *segmentation*, is still a challenging problem in the fields such as handwriting recognition and computer vision. In general, segmentation is computationally expensive, and often a combinatorially hard problem [4].

The state of the art of CAPTCHA design suggests that the robustness of text-based schemes should rely on the difficulty of finding where the character is (segmentation), rather than which character it is (recognition) [11, 3, 4, 5, 6]. That is, such CAPTCHAs should be *segmentation-resistant*. In other words, ***if breaking a (text-based) CAPTCHA can be successfully reduced to a problem of individual character recognition, then this scheme is effectively broken.***

In this paper, we report new character segmentation techniques of general value to attack a number of text CAPTCHAs, including the schemes designed and deployed by Microsoft, Yahoo and Google.

First, we present a novel segmentation attack on a high-profile Microsoft CAPTCHA. Designed to be segmentation resistant, this scheme was a collaborative effort of an interdisciplinary team of diverse expertise in Microsoft including document processing and understanding, machine learning, HCI and security. In fact, the widely accepted "*segmentation resistance*" principle was established by this team.

This CAPTCHA has been deployed in many of Microsoft's online services including Hotmail, MSN and Windows Live for years, with its first version used in Hotmail's user registration system in 2002 [11]. Ever since, the scheme has undergone extensive improvement in terms of both robustness [3, 4, 6] and usability [4, 5]. Microsoft has also filed three US patent applications to protect the underlying technology [8]. Clearly, this scheme was carefully designed.

However, our simple and low-cost attack has achieved a segmentation success rate of higher than 90% on the latest version of this Microsoft CAPTCHA (as deployed in the summer of 2007)[1]. For convenience, we will refer to this CAPTCHA as *the MSN scheme* in this paper. With the aid of this segmentation attack, we estimate that the MSN scheme can be broken with an overall (segmentation and then recognition) success rate of about 60%. In contrast, its design goal was that "automatic scripts should not be more successful than 1 in 10,000 (0.01%)" attempts [4]. Furthermore, although the MSN scheme was believed to be "extremely difficult and expensive for computers to solve" because of the difficulty of segmentation that its designers introduced [5], our attack completely segmented each challenge essentially instantly. To the best of our knowledge, *this for the first time shows that a CAPTCHA that was carefully designed by serious professionals to be segmentation-resistant is nevertheless vulnerable to novel but simple attacks.*

Next, we show that our attack is also applicable to other text CAPTCHAs, including the schemes designed by Yahoo and Google. In particular, a variant of our attack has achieved a high segmentation rate on a Yahoo CAPTCHA, which in theory can lead to the most successful attack to date on the scheme.

The detailed structure of this paper is as follows. Section 2 discusses related work. Section 3 reviews the MSN scheme. Sections 4 and 5 detail our attack and its results respectively. Section 6 discusses the applicability of our attack. We highlight a variant of the attack that we have designed for the Yahoo CAPTCHA. We also show that a component of the attack is applicable to a Google CAPTCHA and multiple other schemes. Section 7 discusses representative "segmentation resistance" mechanisms implemented to date, uncovering more real-life examples of security and usability failures in this area. Section 8 summarises this paper and offer conclusions.

By attacking well-designed, deployed CAPTCHAs, we learn how they could fail and could be improved. Overall, this paper contributes to the immediate improvement of the security of the CAPTCHAs that were widely deployed by Microsoft, Yahoo and Google, as well as other schemes exhibiting similar weaknesses. It also contributes to furthering our understanding of the design of CAPTCHAs - the current collective knowledge on this topic is very limited - for example, which segmentation resistant mechanisms conceived to date are weak but which appears to be secure against currently available attacks.

2. RELATED WORK

It was reported on Feb 8, 2008 [17] that a surge of spam being sent from Windows Live accounts was observed, and a bot that could sign up Live Mail accounts was analysed by a security firm [18] to understand what was behind this phenomenon. However, in this reported case, the CAPTCHA decoding was not done by the bot, but at a remote server. It is unclear whether there was cheap human labor behind the scene feeding CAPTCHA answers manually. On the other hand, even if an automated attack was launched by the server, to date, no technical detail of this attack has been revealed at all. Moreover, the success rate observed for the bot was only about 30-35% [18].

The robustness of text-based CAPTCHA has so far been studied mainly just in the computer vision and document analysis and recognition communities. For example, Mori and Malik [9] have broken the EZ-Gimpy (92% success) and the Gimpy (33% success) CAPTCHAs with sophisticated object recognition algorithms. Moy et al [10] developed distortion estimation techniques to break EZ-Gimpy with a success rate of 99% and 4-letter Gimpy-r with a success rate of 78%. Chellapilla and Simard [3] attacked a number of visual CAPTCHAs taken from the web

[1] The work was done in the summer of 2007. We notified Microsoft the weakness of their CAPTCHA in Sept, 2007. Responding to their request, we held this attack confidential until April 10, 2008. To the best of our knowledge this is the first effective segmentation attack on the scheme.

with machine learning algorithms, achieving a success rate from 4.89% to 66.2%.

Our own early work [14] has broken a number of CAPTCHAs (including those hosted at *Captchaservice.org,* a web service specialised for CAPTCHA generation) with almost 100% success by simply counting the number of pixels of each segmented character, although these schemes were all resistant to the best OCR software on the market. In contrast to other work that relied on sophisticated computer vision or machine learning algorithms, this study used only simple pattern recognition algorithms but exploited fatal design errors that were discovered in each scheme. This is one of the few work examining the robustness of CAPTCHA from the security angle.

PWNtcha [7] is an excellent web page that aims to "demonstrate the inefficiency of many CAPTCHA implementations". It briefly comments on the weaknesses of about a dozen simple CAPTCHAs, which were claimed to be broken with a success ranging from 49% to 100%. However, no technical detail of the attacks was publicly available. Many more CAPTCHAs were also commented at this site. For example, both the MSN scheme and a Yahoo CAPTCHA that will be discussed in this paper (i.e. *Yahoo Scheme 1* in Section 6.1) were regarded by this site as "very good" and difficult to break.

Two interesting algorithms were proposed in [19] to amplify the skill gap between humans and computers. The algorithms could improve systems security for text-based CAPTCHAs, but are orthogonal to this paper. (In this paper, we do not discuss other types of CAPTCHAs such as image-based ones. For those who are interested, an overview of image-based CAPTCHAs can be found in [19].)

Usability and robustness are two fundamental issues with CAPTCHAs, and they often interconnect with each other. In [21], we examined usability issues that should be considered and addressed in the design of CAPTCHAs, and discussed subtle implications some of the issues can have on robustness.

One last note: a survey on CAPTCHAs research (including the design of most early notable schemes) can be found in [13], and the limitations of defending against bots with CAPTCHAs (including protocol-level attacks) were discussed in [15].

3. THE MSN SCHEME

Fig 1 shows some sample challenges generated by the MSN CAPTCHA scheme. We have no access to the codebase of the MSN scheme, so we collected from Microsoft's website 100 random samples that were generated in real time online at [16]. By studying [4, 5] and the samples we collected, we observed that the MSN scheme (as deployed) has the following characteristics.

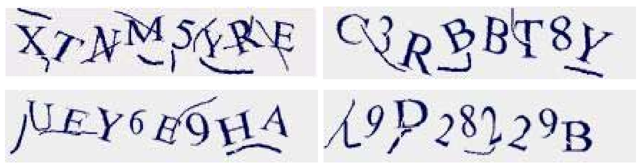


Fig 1. The MSN CAPTCHA: 4 sample challenges.

- Eight characters are used in each challenge;
- Only upper case letters and digits are used.
- Foreground (i.e. challenge text) is dark blue and background light gray.
- Warping (both local and global) is used for character distortion.

  Local warp produces "small ripples, waves and elastic deformations along the pixels of the character", and it foils "feature-based algorithms which may use character thickness or serif features to detect and recognise characters" [6]. Characters in the first and second rows of Table 1 are largely distorted by local warping.

  Global warp generates character-level, elastic deformations to foil template matching algorithms for character detection and recognition. Characters in the third and fourth rows of Table 1 are largely distorted by global warping.
- The following random arcs of different thicknesses are used as the main anti-segmentation measure.
  - Thick foreground arcs: These arcs are of foreground color. Their thickness can be the same as the thick portions of characters. They do not directly intersect with any characters, so they are also called "non-intersecting arcs".
  - Thin foreground arcs: These arcs are of foreground color. Although they are typically not as thick as the above type of arcs, their thickness can be the same as the thin portions of characters. They intersect with thick arcs, characters or both, and thus also called "intersecting thin arcs".
  - Thin background arcs: These arcs are thin and of background color. They cut through characters and remove some character content (pixels).

Both local and global warping is commonly used for distortion in text-based CAPTCHAs. Many schemes use background textures and meshes in foreground and background colors as clutter to increase robustness. However, random arcs of different thicknesses are used as clutter in the MSN scheme. The rationale was as follows. These arcs are themselves good candidates for false characters. The mix of random arcs and characters would confuse state of the art segmentation methods, providing strong segmentation resistance [5].

4. A SEGMENTATION ATTACK

We have developed a low-cost attack that can effectively and efficiently segment challenges generated by the MSN scheme. Specifically, our attack achieves the following:

- Identify and remove random arcs
- Identify all character locations in the right order; in other words, divide each challenge into 8 ordered segments, each containing a single character.

Our attack is built on observing and analysing the 100 random samples we collected – this is a "sample set". The effectiveness of this attack was tested not only on the sample set, but also on a large test set of 500 random samples – the design of the attack used no prior knowledge about any sample in this set. This methodology follows the common practice in the fields such as computer vision and machine learning[2]. (All the samples were collected in the summer of 2007.)

Our attack involves 6 consecutive steps, each of which is detailed in the following sections.

4.1 Pre-processing

We first convert a rich-color challenge to a black-white image using a threshold method: pixels with intensity higher than a threshold value are converted to white, and those with a lower intensity to black (see Fig. 2(a) and (b)). The threshold was manually determined by analysing the sample set, and the same value was used for each image in both the sample and test sets.

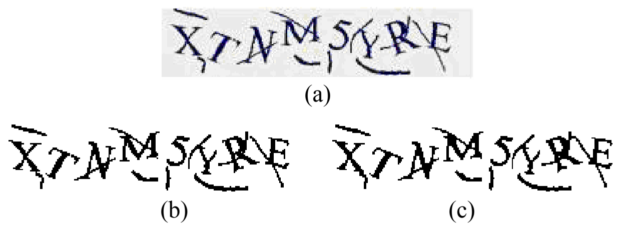


Fig 2. Pre-processing. (a) original image, (b) binarized image, (c) after fixing broken characters.

(This sample is taken from [8], in which its resistance to segmentation is ranked by Microsoft as "hard", the highest level among all examples. We will use this sample to illustrate the whole process of our segmentation attack in this paper.)

The second step of pre-processing is to fix broken characters: thin background arcs remove some character content, and sometimes they create a crack in characters (e.g., the second character 'T' in Fig 2(a) is broken due to this reason). This step serves two purposes: i) to keep a character as a single entity and consequently enhance our follow-up segmentation methods, and ii) to prevent small portions of characters from being removed as a noise arc later on.

We observed that thin background arcs are typically 1-2 pixels wide after binarization, and the following simple method works well to identify and fix broken characters caused by such arcs.

(1) Find pixels that are of background color and have left and right neighbours with foreground color (see Fig 3(a)).
(2) Find pixels that are of background color and have top and bottom neighbours with foreground color (see Fig 3(b)).
(3) Convert pixels identified above to foreground color.

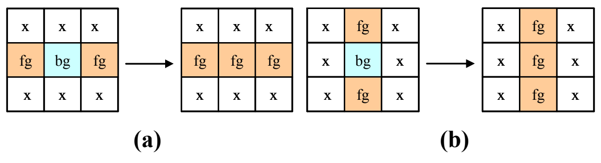


Fig 3. Connecting 1-pixel gap ('x' represents a pixel that is of either foreground or background color).

This method connects any 1-pixel gap that satisfies the conditions illustrated in Fig 3. Its effect is illustrated in Fig 2 (c): some missing pixels for character 'T' are recovered. A side effect of this method is that it might introduce additional foreground pixels that connect components that are initially disconnected. For example, in Fig 2 (c), a thin arc intersecting with 'R' is now connected with another arc intersecting with 'E'. But this drawback has proven a negligible issue in our study – that would not be the case if we chose to connect all two-pixel gaps.

4.2 Vertical Segmentation

A vertical segmentation method is applied to segment a challenge vertically into several *chunks*, each of which might contain one or more characters. The process of vertical segmentation starts by mapping the image to a histogram that represents the number of foreground pixels per column in the image. Then, vertical segmentation lines separate the image into chunks by cutting through columns that have no foreground pixels at all. Fig 4 shows that such vertical histogram segmentation cuts a challenge into two chunks.

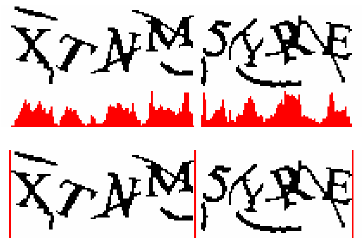


Fig 4. Vertical Segmentation

Typically, this vertical method not only achieves partial segmentation, but also contributes to our divide-and-conquer strategy, which is key to the success of our attack.

4.3 Color filling segmentation

In this step, a "color filling segmentation (CFS)" algorithm is applied to *each* chunk segmented in the previous step. The basic idea of this algorithm is to detect every connected component, which we call an *object*, in a chunk. An object can be an arc, character, connected arcs, or connected characters. The algorithm works as follows. First, detect a foreground pixel, and then trace all its foreground neighbours until all pixels in this connected component are traversed – that is, an object is detected. Next, the algorithm locates a foreground pixel outside of the area of the detected object(s), and starts another traversal process to identify a next object. This process continues until all objects in the chunk are located. This method is effectively like using a distinct color to flood each connected component, so we call it the "color filling" segmentation. In the end, the number of colours used to fill a chunk is the number of objects in the chunk.

With our CFS method, as shown in Fig 5 (a), we determine that there are six objects in the first chunk and five in the second.

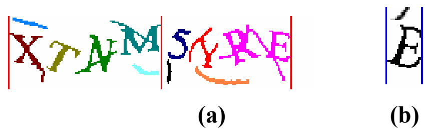


Fig 5. Color filling segmentation

Often, a challenge is divided into four or five chunks by vertical segmentation. It is worthwhile to mention that this color filling step is applied to each chunk, rather than only those wider chunks that probably contain more than one object. The reason is simply that thinner chunks might also contain more than one object (see Fig 5(b)), and we need to locate all objects in each chunk and

[2] In a related study [10] published at CVPR'04, the premier computer vision conference, the size of sample set was 564 and the size of test set 736.

track the number of objects for the follow-up arc removal and other steps.

CFS contributes to further segmentation by detecting objects that cannot be segmented by the vertical method, and gives the number of objects in each chunk. As will be discussed later on, CFS also contributes to further steps such as arc removal.

4.4 Thick arc removal

Thick arcs, if any, will be detected and removed after the above color filling process.

Characteristics of arcs. For the sake of usability, thick foreground arcs do not intersect with challenge characters, unless they are connected indirectly through a thin arc (thin arcs do intersect with characters) or are forced to connect with others due to the drawback introduced by the method of fixing broken characters in Section 4.1. We also observed that thick arcs have the following characteristics, which make it possible to identify and remove them automatically.

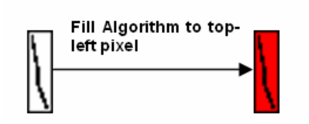
- **Pixel count**. Often, a thick arc has a relatively small pixel count (i.e., the number of foreground pixels in the arc).
- **Location**. Thick arcs are located close to or even intersect with the image border, something which rarely occurs with valid characters unless they are connected to the thick arc.
- **Shape**. Thick arcs do not contain circles. Characters such as A, B, D, P, Q, 4, 6, 8 and 9 all contain one or more circles.
- **Interplay between shape and location**. The position of thick arcs and their geometric shapes are somehow correlated. For example, thick arcs located at the start and end of a challenge are typically tall but narrow (that is, the ratio of height over width is large); thick arcs in the middle part of a challenge tend to be wide but short (that is, the ratio of width over height is large).

Arc removal algorithm. Our algorithm is largely based on the above observations, and includes the following steps.

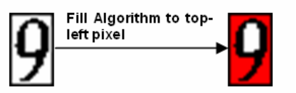
1) *Circle detection*, which detects if an object contains a circle. If an object contains a circle, we know it is definitely not an arc, and all other arc removal methods can be skipped. The circle detection method works as follows.

    - Draw a bounding box around an object, so that this bounding box does not touch any part of the object.
    - Apply the color filling algorithm to the top-left pixel, i.e., flood all background pixels that are connected to the top-left pixel, with a color that is different from foreground and background
    - Scan the bounding box for pixels of the background color. If such a pixel is found, then a circle is detected. Otherwise, no circle is detected.

    Fig 6 shows two example cases. In Fig 6 (a), there is no pixel of the original background color once the filling algorithm is applied. That is, we are sure this object does not contain any circles. In contrast, the filling algorithm cannot get rid of all pixels of the original background color in Fig 6 (b). Therefore, by detecting these pixels, the algorithm is sure that a circle exists in this object. (To improve the efficiency of the filling algorithm, the minimal gap between the object and the bounding box is just one pixel in both cases.)



(a)



(b)

Fig 6. Circle detection: examples

Then, we use the following 3 steps to detect and remove thick arcs as follows. At the end of each step, the histogram of the image is updated.

2) *Scan all objects that contain no circles for discriminative features* (other objects are safely ignored). Such discrimination is largely about pixel count checking. If an object has a pixel count smaller than or equal to 50, it is removed as an arc. (We observed that typically a character has a pixel count of larger than 50). When this step was applied to the challenge in Fig 5(a), an arc in the 2nd chunk was removed due to its small pixel count (see Fig 7).

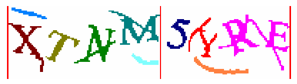


Fig 7. Arc removal - discriminative feature checking: an arc in the second chunk is removed.

3) *Relative position checking*. This step examines the relative position of objects in a chunk, and is applied to all chunks that contain more than one object (note that connected characters are considered as a single object). The basic idea behind this step is that the relative positions of objects can tell arcs and real characters apart. For example, typically characters are closer to the baseline (i.e. the horizontal central of a chunk) whereas arcs are closer to the top or bottom image borders. In addition, characters are horizontally juxtaposed, but never vertically. Once this step is completed, the histogram is updated.

    As shown in Fig 8, when this method was applied to the challenge in Fig 7, further arcs were removed. Meanwhile, the histogram was updated, and the image was further segmented.

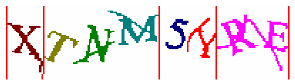
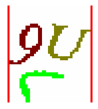
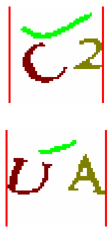
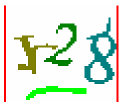
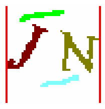
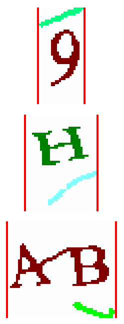


Fig 8. Arc removal - relative position checking: further arcs were removed and histogram was updated.

The relative position checking has proven the most effective in removing arcs in our attack. An incomplete list of typical relative position patterns is illustrated with real examples in Table 2.

Table 2. Typical relative position patterns

Relative position patterns			Decision
Layout	Description	Example	
O1 O2 O3	*Three objects in a chunk*: two objects more or less align along the baseline, the 3rd object under either of them		O3 is arc
O3 O1 O2	*Three objects in a chunk*: two objects more or less align along the baseline, the 3rd object on top of either of them		O3 is arc
O0 O1 O2 O3	*Four objects in a chunk*: Three objects more or less align along the baseline, the 4th object under any of them		O3 is arc
O1 O2 O3 O4	*Four objects in a chunk*: Two objects more or less align along the baseline, the 3rd and 4th objects under and on top any of them respectively		O1 and O4 are arcs
O1 O2	*Two objects in a chunk*: vertically juxtaposed		Either O1 or O2*

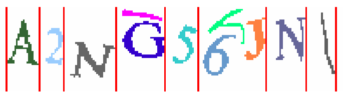
*First apply the circle detection result obtained before: if only one of the objects contain a circle, then the object without a circle is removed as an arc. If this does not work, then the object that is less aligned with the baseline is removed as an arc.

4) *Detection of remaining arcs*. The above steps do not necessarily identify all the arcs in an image. What is done in this step is as follows. First, count the number of remaining objects in the image (identified arcs are already removed and thus not counted). If this number is larger than 8, then there is at least one undetected arc in the image. A surprising observation about these undetected arc(s) is that they often are the first or last object in the current image. An ad-hoc method works for most of the cases by simply checking the first and last objects with the following rules:

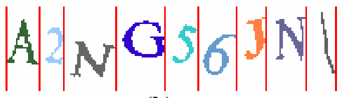
- If only one of them contains a circle, the object without a circle is removed as an arc.
- If neither of them contains a circle, then the object with a smaller pixel count is removed.

This process repeats until the image has exactly 8 objects remaining.

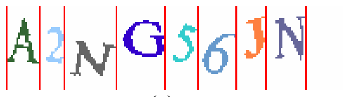
Another example illustrating the whole arc removal process is in Fig 9, where (a) was an image segmented by vertical and CFS segmentations. The discriminative feature checking failed to detect any arc, but relative position checking detected an arc in both the 4th and 6th chunks. Fig 9 (b) was the result after those two arcs are removed and the histogram was updated. Then, escaped arcs detection caught the last object as an arc. The final image at the end of the arc removal process is Fig 9 (c).



(a)



(b)



(c)

Fig 9. Arc removal: another example.

4.5 Locating connected characters

After removing arcs, an immediate step is to locate, if any, connected characters, which either vertical or color filling segmentation has failed to segment. Among *n* objects output by the previous step, if $n < 8$, then at least one of the objects contains two or more characters and these characters are connected (typically by thin intersecting arcs). This step estimates how many characters are connected and locates them.

The following design and implementation features of the MSN scheme all contribute to being able to estimate which objects contain how many connected characters.

- Fixed length: every challenge uses 8 characters.
- Connected characters in an object are horizontally but never vertically juxtaposed. Therefore, an object containing two or more connected characters is typically wider than other objects.
- On average a segmented chunk - by definition, a chunk cannot be further segmented by the vertical method but can by the CFS method - contains more than one character if the chunk is wider than 35 pixels. (This width was measured after the following normalisation process was applied to the chunk: the left segmentation line is adjusted to cross the left-most foreground pixel in the chunk vertically and similarly for the right segmentation line.)

According to the number of chunks, the width of each chunk, and the number of objects in each chunk, we can guess with a high success rate which chunk/object contains connected characters and the number of these characters (or in other words, guess how many characters exist in each chunk).

We use two examples to show how our algorithm works. The histogram for the image in Fig 8 indicates that it contains four chunks. Since there are exactly 8 characters in these chunks, we know there are the following five exclusive possibilities for the distribution of all the characters among the chunks[3]:

(a) There are four chunks, each having two characters.
(b) One chunk has three characters and there are two additional chunks each having two characters.
(c) One chunk has four characters and another two characters.
(d) There are two chunks each having three characters.
(e) One chunk has five characters.

Since the 2nd, 3rd and 4th chunks in the image were all wider than 35 pixels, the algorithm determines that there are at least three chunks each having more than one character. Consequently options (c), (d) and (e) are excluded - none of the options would allow more than two chunks that have more than one character. The algorithm also knows from the CFS algorithm that the 2nd chunk contains three objects, and therefore option (a) is also dropped. This leaves only option (b); thus the algorithm identifies that the 2nd chunk contains exactly three characters and the 3rd and 4th chunks contains two characters each.

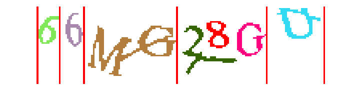


Fig 10. "Approximation" for locating connected characters

The second example (see Fig 10) is more subtle. The histogram for this image indicates it contains 5 chunks. Since there are exactly 8 characters in these chunks, we know there are the following three exclusive possibilities for the distribution of all the characters among the chunks:

(a) One of the chunks contains 4 characters
(b) One chunk has three characters and another two characters.
(c) There are three chunks each having two characters.

Since the 3rd and 4th chunks in the image were wider than 35 pixels, the algorithm determines that at least 2 doubles exists and consequently option (a) is excluded. Since there were only two such wider chunks, option (c) is also dropped. This leaves only option (b).

To determine which chunk contains a triple and which contains a double, the algorithm compares the width and the number of objects in both chunks. The algorithm find that the 3rd chunk "MG" is the widest chunk, however it also knows from the CFS algorithm that the 4th chunk "28G" contains 3 objects, this leaves only a maximum of 2 objects that can exist in the 3rd chunk; thus the algorithm identifies that the 3rd chunk contains two connected characters.

It is feasible to achieve the same results without using the number of chunks but relying more on the number of objects. However this alternative method requires keeping track of not only each object's position in the image but also the position with respect to its neighbors, which would make it much more complicated to implement the algorithm.

[3] In the general case, it is also trivial to enumerate all possibilities for distributing 8 characters across any given c (c is an integer between 1 and 8) chunks. On the other hand, in our experiments, scenarios where c=1, 2 or 3 have never occurred.

4.6 Segmenting connected characters

The previous step has identified any object(s) containing connected characters, as well as the number of these characters, denoted by c, contained in each object. We observed that often, a simple "even cut" method works to segment the connected characters in an object as follows.

1) Work out the width of the object by identifying its left-most and right-most pixels;

2) Vertically divide the object into c parts of the same width, each part being a proper segment.

For example, it was determined that the last object in Fig 8 and the 3rd object in Fig 10 contain two connected characters. For these objects, what our algorithm does is to evenly divide them into two segments, each being a character. Fig 11 shows the finalised 8 segments for both challenges.

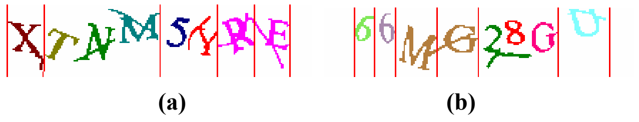


Fig 11. Completely segmented images

5. RESULTS

Success rate. Our segmentation attack has achieved a success rate of 91% on the sample set. That is, 91 out of 100 challenges were segmented correctly. To check whether it was generic enough, we ran our attack on a test set of 500 random challenges - our program had no prior knowledge about any sample in this set. Our attack achieved a success rate of 92% on the test set (the distribution of samples in the test set slightly favours our algorithm). For both the sample and test sets, the success rate was manually established.

We analysed all cases of failure of our segmentation attack in both the sample and test sets, and found that three types of failure occurred as follows.

- Failure of arc removal: some thick arcs were undetected.
- Failure of identifying connected characters. A typical case was: when a single character (e.g. 'W') was much wider than two connected characters, the former, rather than the latter, might be identified as the one containing connected characters. On the other hand, when thick arcs were not detected but treated as valid characters, they could also cause our algorithm to fail to detect connected characters.
- Failure of "even cut". It is unsurprising that this simple method does not always work to segment connected characters.

We also compared the percentage of each failure type in both the sample and test sets. The failure patterns in both sets are similar. The details of our failure analysis are in [22].

Attack speed. We implemented our attack in Java (little effort was spent in optimizing the run-time of code), and tested it on a desktop computer with a 1.86 GHz Intel Core 2 CPU and 2 GB RAM. The attack was run ten times on both the sample and test sets, and the average speed was taken (see Table 3). The figures in the table show that our attack is very efficient: on average, it takes only slightly more than 80 ms to completely segment a challenge in both sets.

Table 3. Attack speed

Speed (ms/challenge)	Average	Max	Min
Sample set	82.8	91.4	81.4
Test set	84.2	95.5	82.8

Implications. State of the art of machine learning can achieve a success rate of at least 95% for recognising individual characters in the MSN scheme, after they are segmented [5, 6]. However, this rate is a conservative estimate for recognising characters in samples we have collected for this study, for the following reasons.

- First, we checked all samples in our test set after we measured the success rate of our attack, and found that although the same types of distortion techniques were applied to characters in our samples and those listed in Table 1, the former were much less distorted than the latter. The same observation also applied to the sample set.
- Second, by manually inspecting all the samples that were correctly segmented by our attack, we observed no artifacts that would be introduced by any step of the attack to interfere with the final recognition step.
- Third, we have simple methods to get rid of some portions of "intersecting thin arcs" in each segmented character so that these characters are even less distorted and consequently easier to be recognised by standard machine learning techniques. For example, one of our methods is to guess the area of the real character inside an object by checking the density of foreground pixels for the object. As illustrated in Fig 12 (where the example is taken from the last segment in Fig 11 (a)), the majority of columns and rows inside the red box have a pixel count higher than a threshold value (3 in this case), while for portions outside of this box, the majority of columns and rows have a lower pixel count, which is in the range of the thicknesses of thin intersecting arcs. Thus, portions of such arcs are rightly recognised and removed as distortion.

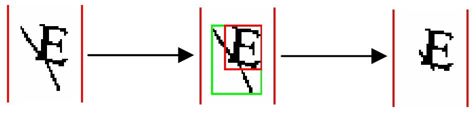


Fig 12. Thin arc removal using pixel-density based bounding box estimation.

As such, our segmentation attack suggests that the MSN scheme can be broken with at least an overall (segmentation and recognition) success rate of 61% (≈ .92*.95^8).

6. APPLICABILITY

Our attack on the MSN scheme is applicable to other CAPTCHAs. In this section, we discuss a few cases.

6.1 Yahoo CAPTCHA

We successfully applied a variant of our attack to a CAPTCHA that was deployed by Yahoo at their global websites until very recently - the last day that we observed this scheme was in active use (at Yahoo's site in China) was March 8, 2008. Our attack has achieved a segmentation rate of around 77% on this CAPTCHA. As a result, we estimate that this scheme could be broken with an overall (segmentation and then recognition) success rate of about 60% (≈.77*.95^5; the average text length in this scheme is 5). That is, in theory, our work can lead to the most successful attack to date on the scheme[4]. Alerted, Yahoo has ceased to use this CAPTCHA.

Fig 13 shows example challenges generated by this Yahoo CAPTCHA, which we call *Yahoo Scheme 1*. By analysing 100 random samples, we observed that the use of intersecting arcs was the main segmentation resistance mechanism in this scheme, and the arcs could have the same thickness as some portions of valid characters.

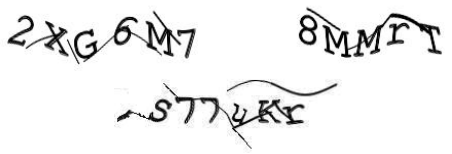
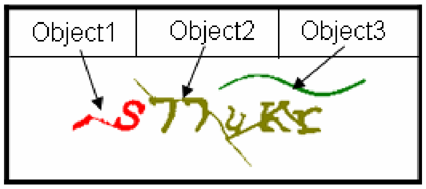


Fig 13. Yahoo Scheme 1: example challenges.

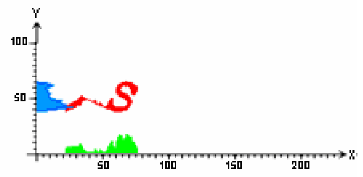
Our attack on this scheme works as follows. After binarizing an image, we segment it into a set of connected components (i.e., objects) by applying the CFS method – this method not only achieves partial segmentation, but also contributes to our divide-and-conquer strategy.

Then, for each object, we use a method, which is extended from the vertical segmentation in Section 4.2, to detect and remove arcs. This method is a major extension to our work on the MSN scheme, and its key technique is the following histogram analysis.

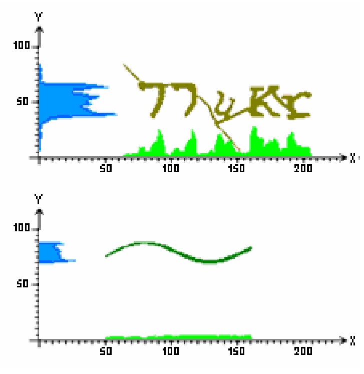
First, we map each object to two histograms, one representing the number of foreground pixels per column, and the other representing the number of foreground pixels per row in the object. We call them X- and Y- histograms, since they are created as if the object is projected to the X- and Y- axis respectively. Fig 14 (b) shows X- (in green color) and Y- histograms (in blue color) for each of the three objects identified in Fig 14 (a) by the CFS method.



(a)



[4] A Russian security team claimed that they have broken the same scheme with a success of around 35% [20]. No technical detail of their attack was publicly available, however.



(b)

Fig 14. (a) The result of CFS; (b) X- and Y-histograms for each identified object.

Then, our arc removal algorithm is mainly an ordered sequence of histogram analysis, and it works as follows.

First, the algorithm checks the highest peak value of each object's histograms. If the peak value of either its X- or Y-histogram is too small, then the object is either too flat or thin to be a valid character, and it is removed as an arc. When this step was applied, the third object in Fig 14 (a) was correctly removed as an arc, but the other two stayed.

Second, the algorithm examines each remaining object's Y-histogram to identify *low-density rows*, which have only a tiny number of pixels. When a sufficient number of such rows (at least 4 in our experiments) are consecutive, they typically constitute a region that has a low density of foreground pixels. Such region typically indicates that these rows contain only (portions of) arcs, and they can be safely removed.

As shown in Fig 15, this step correctly removed portions of arcs in both the top and bottom areas of the second object in Fig 14 (a), although it had no effect on the first object.

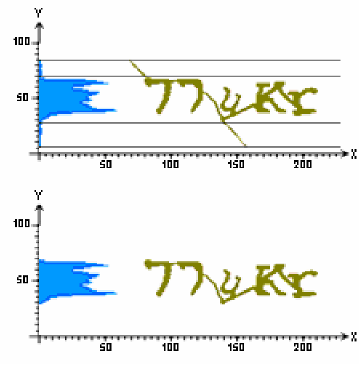
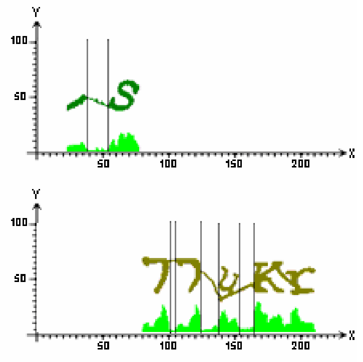


Fig 15. Arc removal: (a) low-density rows are identified, and (b) after step 2.

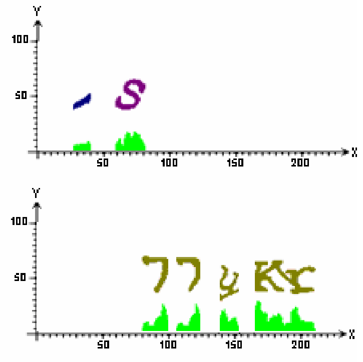
If any arc is removed from an object, the object's X-histogram should be updated at the end of this step (for the sakes of both efficiency and accuracy of further arc removal).

As the third step, the algorithm examines an object's X-histogram to identify *low-density columns*. When a sufficient number of such columns are consecutive, they constitute a region that has a low density of foreground pixels. Such region typically indicates that these columns are (portions of) arcs that can be safely removed.

As shown in Fig 16, this step successfully removed some horizontal portions of arcs in both objects.



(a)

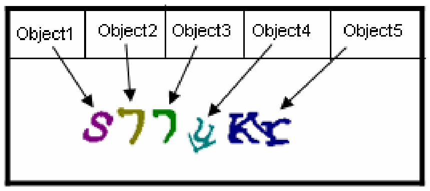


(b)

Fig 16. Arc removal: (a) low-density columns are identified, and (b) after step 3.

Lastly, clean up. Some small portions of arcs can still stay after the above steps, e.g. the first object in Fig 16 (b). However, these portions tend to have a much smaller pixel count than any valid characters, and therefore are easy to identify and remove.

Fig 17 (a) shows the challenge image after the whole arc removal process. Apparently, our algorithm not only removes standalone arcs, but also contributes to segmentation by removing portions of arcs that connect different characters.



(a)

(b)

Fig 17. (a) After arc removal, and (b) a segmented challenge.

After arc removal, we use a method that is very similar to Section 4.5 for locating remaining connected characters and estimating the number of such characters. Finally we use the same "even cut" method as in Section 4.6 to segment them.

For example, for the image in Fig 17 (a), our algorithm determined that the most likelihood was that Object5 had two connected characters because of its size, and thus the object was evenly segmented to two parts. Fig 17 (b) shows the final segmentation result, which is correct.

A detailed failure analysis for our attack on Yahoo scheme 1 is available in [2].

6.2 Google CAPTCHA

We also tested a CAPTCHA that is deployed by Google to protect their online services (see Fig 18) with our attack on the MSN scheme. We correctly segmented 12 out of 100 random samples we collected, leading to a success rate of 12%. This could lead to an overall success rate of 8.7% (≈ .12 * .95^6.25; the average text length in this scheme is 6.25). However, the segmentation success was exclusively contributed by the CFS method. At the time of preparing the camera-ready version of the present paper, it appears that Google have fixed this vulnerability.

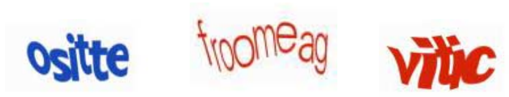


Fig 18. The Google CAPTCHA: sample challenges.

6.3 Other CAPTCHAs

It is worthwhile to note that both the Yahoo and Google schemes we discussed above were designed to be segmentation resistant. For CAPTCHAs that do not follow the principle of segmentation resistance, it would be trivial for the CFS method to segment them correctly. For example, the CFS method would be a more efficient and effective way of attacking *Captchaservice.org* schemes that were broken in our earlier work [14].

7. ON SEGMENTATION RESISTANCE

The Microsoft, Yahoo and Google CAPTCHAs discussed above represent three mainstream styles of segmentation resistance mechanisms implemented to date, which are summarised as follows.

- The Microsoft style: random arcs as false characters.
- The Yahoo style: random angled connecting lines.
- The Google style: crowding characters together.

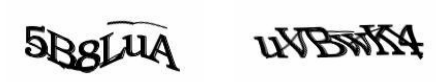
Applying our novel segmentation attack, we identified that these mechanisms, as currently implemented, have security flaws. However, we do not claim that the segmentation resistance principle is overturned. For example, it is feasible to defend against our attack on the Google scheme by removing gaps between adjacent characters to stick the latter together – this would entirely defeat our attack. (However, this might make it worse a usability issue that, as discussed later on, already exists in the current implementation of the scheme, if care is not taken).

There are also simple methods for improving the MSN scheme, for example:

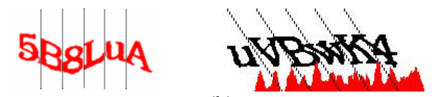
- Adopting the "crowding characters together" method, e.g. letting characters touch or overlap with each other.
- Making it harder to tell characters and arcs apart (e.g. by juxtaposing characters and arcs in any direction).
- Using randomly varied widths for characters could also confuse some parts of our attack.

Although there is no conclusive technical evidence yet, the method of "crowding characters together", if implemented properly, does appear to have more merit than other methods in providing segmentation resistance. For example, as discussed above, it can be applied to improve both the Microsoft and Google schemes.

Probably motivated by the same observation, Yahoo rolled out their new CAPTCHA in March 2008. As shown in Fig 19 (a), challenge texts in this scheme are more compacted than before, and characters are usually connected - they either touch with each other, or are connected by intersecting random lines. We use this latest Yahoo scheme as the last cautionary example in this paper to show how a seemingly sound principle can go wrong in practice.



(a)



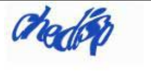
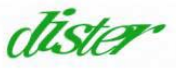
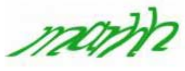
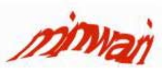
(b)

Fig 19. Yahoo's latest scheme (a) example challenges; (b) segmented images

We discovered a number of elementary but fatal flaws in this latest Yahoo scheme. For example, it would be difficult or even impossible for an automated attack to segment a challenge if the number of characters in the challenge is unknown. Unlike the MSN scheme, the Yahoo's new CAPTCHA uses a varied text length, which is a good design feature. However, we observed that the number of characters (n) in a challenge can be estimated with a high success rate by measuring the width of the text in the challenge. Furthermore, this scheme is vulnerable to either a simplified version of our attack on the previous Yahoo scheme, or a new "angular segmentation" attack that segments a challenge properly with angled lines. The first example in Fig 19 (b) shows an extreme case, where a challenge is vulnerable to the first attack: an "even cut" worked after n was estimated. The second example in Fig 19 (b) shows that a challenge was correctly segmented by angled lines. Using two such simple segmentation algorithms with associated rules to identify which algorithm to use, we achieved a segmentation success rate of around 33.4% on the latest Yahoo scheme. As a result, we estimate that this scheme can be broken with an overall (segmentation and recognition) success rate of 25.9% (≈ .334*.95^5; the average text length in this scheme is 5). Our detailed security analysis of this Yahoo scheme is discussed in [2]. We have informed Yahoo this attack as well as the attack described in Section 6.1. Responding to their request, we kept our work confidential to allow them time to fix the vulnerabilities.

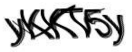
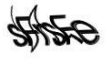
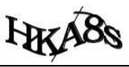
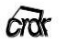
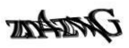
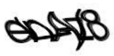
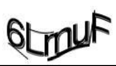
On the other hand, while the "crowding characters together" method, if implemented properly, appears to provide better security, it can introduce a usability shortcoming that has been long ignored, namely a new type of confusing characters. For example, under some distortions in the Google scheme, "vv" resembles "w"; "cl" resembles "d"; "nn" resembles "m"; "rn" resembles "m" ; "rm" resembles "nn"; "cm" resembles "an", and so on (see Table 4 for a few examples). In 2007, we observed that 6% of challenges generated by the Google scheme contained such characters, and would barely be usable for human users, or at least they would create confusion so that the users could not be sure what the right answers should be.

Table 4. Confusing characters in the Google CAPTCHA

Image	Confusing characters
	the middle part is 'd' or connected "cl"?
	Another case of "cl" or "d" confusion.
	the starting part is 'm' or connected 'rn"?
	A real headache: is the first part "m" or "rn", the middle part "inv" or "nw"?

A similar usability problem also exists in the latest Yahoo scheme, which adopts the "crowding characters together" method (see Table 5 for some examples). We observed that about 10% of challenges generated by this scheme contain such confusing characters, and thus would be human unsolvable or at least cause confusion. However, this problem was rarely observed in Yahoo Scheme 1.

Table 5. Confusing characters in the latest Yahoo scheme

Challenge image	Answer
	*yKKV5y* or *yKKT5y*?
	*SFrsFe* or *sFrsEe*?
	*HZKA8S* or *HKA8S*?
	*crar* or *crdr*?
	*znAzwG* or *zn4zwG*?
	No idea what the second character is
	*6LmuF* or *6LrnuF*?

Given the large number of unusable challenges observed, we recommend that any scheme that implements this "crowding characters together" mechanism treat confusing character pairs with special care when distorting them. Moreover, for CAPTCHAs such as the latest Yahoo scheme, it appears that not using intersecting lines at all would further improve the scheme's usability without sacrificing its security.

8. SUMMARY AND CONCLUSION

For the first time, we have shown that although the Microsoft's MSN CAPTCHA intentionally bases its robustness on segmentation resistance, it is vulnerable to a simple, low-cost segmentation attack. Our attack has achieved a segmentation success rate of 92%, and this implies that the MSN scheme can be broken with an overall (segmentation and then recognition) success rate of more than 60%. Therefore, our work shows that the MSN scheme provides only a false sense of security.

Tested by its designers, the MSN scheme was resistant to prior art segmentation attacks. However, for the first time, we used a color filling method for segmenting characters in a CAPTCHA. Together with traditional vertical histogram analysis, this method has proven powerful. We also found that it is easy to automatically tell random arcs (which were used as false characters in the scheme to confuse automated attacks) from valid characters by examining characteristics such as pixel counts, shapes, locations, relative positions, and distances to baseline. We also designed a novel method for locating connected characters and estimating the number of such characters.

The attack on the MSN scheme was also tested on other CAPTCHAs. In particular, a variant of the attack has achieved a high segmentation rate on a CAPTCHA that was widely deployed by Yahoo until early this year. In addition, a component of the attack, i.e. the CFS segmentation, is applicable to the Google CAPTCHA and multiple other schemes.

The Microsoft, Yahoo and Google CAPTCHAs we have analysed represented three major segmentation resistance mechanisms implemented to date. While the mechanisms used in the MSN and the Yahoo (scheme 1) CAPTCHAs were broken by our attacks, it appears that the "crowding characters together" mechanism advocated by the Google CAPTCHA could provide better security against currently available attacks.

However, this mechanism was not worry free. For the first time, we identified some flaws of this mechanism as implemented in the Google and the latest Yahoo schemes. Furthermore, we identified a long ignored usability problem introduced by this increasingly popular segmentation resistant mechanism. We also discussed countermeasures for addressing these security and usability concerns. We expect that with all the enhancements learnt from previous failures, the "crowding characters together" mechanism will become more robust and user friendly.

Overall, all these contribute to furthering current understanding of the design of better CAPTCHAs, in particular the design and implementation of segmentation resistance mechanisms.

To conclude this paper, we have the following. CAPTCHA design is an interdisciplinary topic where expertise from multiple domains plays an important role. As demonstrated in this paper, security engineering expertise and experience, in particular adversarial thinking skills (i.e. identifying what can go wrong), can make a unique and significant contribution to the improvement of the robustness of CAPTCHAs, but were not in place when either Microsoft, Yahoo or Google were designing their schemes.

Another important lesson is that even if segmentation resistance is a sound principle, the devil is in the details. The techniques we have reported in this paper, in particular those used on the MSN and two Yahoo CAPTCHAs, demonstrate new methods for evaluating the strength of segmentation resistance mechanisms.

The relatively wide applicability of our attack on the MSN scheme is encouraging. However, we doubt that there is a universal segmentation attack that is applicable to all text

CAPTCHAs, given that hundreds of design variations exist [19]. Instead, a more realistic expectation is to create a toolbox (i.e. a collection of algorithms and attacks, ideally organized in a composable way) for evaluating the strength of CAPTCHAs – this is our ongoing work.

Designing CAPTCHAs that exhibit both good robustness and usability is much harder that it might appear to be. The current collective understanding of this topic is still in its infancy. To evolve the design of CAPTCHA, a young but important topic, from an art into a science still requires considerable study. Our experience suggests that CAPTCHA will go through the same process of evolutionary development as cryptography, digital watermarking and the like, with an iterative process in which successful attacks lead to the development of more robust systems.

9. ACKNOWLEDGMENTS

We are grateful to Brian Randell for proofreading an early version of this paper and many useful comments. Comments and suggestions from Philippe Golle and anonymous reviewers also helped to improve this paper.

10. REFERENCES

[1] L von Ahn, M Blum and J Langford. "Telling Humans and Computer Apart Automatically", CACM, V47, No2, 2004.

[2] J Yan and A S El Ahmad. "Is cheap labour behind the scene? - Low-cost automated attacks on Yahoo CAPTCHAs", School of Computing Science Technical Report, Newcastle University, England, 2008.

[3] K Chellapilla and P Simard, "Using Machine Learning to Break Visual Human Interaction Proofs", Neural Information Processing Systems (NIPS), MIT Press, 2004.

[4] K Chellapilla, K Larson, P Simard and M Czerwinski, "Building Segmentation Based Human-friendly Human Interaction Proofs", 2nd Int'l Workshop on Human Interaction Proofs, Springer-Verlag, LNCS 3517, 2005.

[5] K Chellapilla, K Larson, P Simard and M Czerwinski, "Designing human friendly human interaction proofs", ACM CHI'05, 2005.

[6] K Chellapilla, K Larson, P Simard, M Czerwinski, "Computers beat humans at single character recognition in reading-based Human Interaction Proofs", 2nd Conference on Email and Anti-Spam (CEAS), 2005.

[7] Sam Hocevar. PWNtcha - captcha decoder web site, http://sam.zoy.org/pwntcha/, accessed Jan 2008.

[8] Microsoft Corporation. "Human Interaction Proof (HIP) -- Technical and Market Overview", 2006. Available at http://download.microsoft.com/.../Human_Interaction_Proof_Technical_Overview.doc. Accessed Jan 2008.

[9] G Mori and J Malik. "Recognising objects in adversarial clutter: breaking a visual CAPTCHA", IEEE Conference on Computer Vision & Pattern Recognition (CVPR), 2003.

[10] G Moy, N Jones, C Harkless and R Potter. "Distortion estimation techniques in solving visual CAPTCHAs", IEEE CVPR, 2004.

[11] P Simard, R Szeliski, J Benaloh, J Couvreur and I Calinov, "Using character recognition and segmentation to tell computers from humans", International Conference on Document Analysis and Recognition (ICDAR), 2003.

[12] P Simard, D Steinkraus, J Platt. "Best Practice for Convolutional Neural Networks Applied to Visual Document Analysis", International Conference on Document Analysis and Recognition (ICDAR), IEEE Computer Society, Los Alamitos, pp.958-962, 2003.

[13] C Pope and K Kaur. "Is It Human or Computer? Defending E-Commerce with CAPTCHA", IEEE IT Professional, March 2005, pp. 43-49

[14] J Yan and A S El Ahmad. "Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms", in *Proc. of the 23rd Annual Computer Security Applications Conference (ACSAC'07).* FL, USA, Dec 2007. IEEE computer society. pp 279-291.

[15] J Yan. "Bot, Cyborg and Automated Turing Test", the Fourteenth International Workshop on Security Protocols, Cambridge, UK, Mar 2006. Also available at http://www.cs.ncl.ac.uk/research/pubs/trs/papers/970.pdf.

[16] https://signup.live.com/hmnewuser.aspx?mkt=en-us&revipc=CN&ts=3970181&sh=WsBO&hm=1&ru=http%3a%2f%2fmail.live.com%2f%3fnewuser%3dyes&rx=http%3a%2f%2fget.live.com%2fmail%2foverview&rollrs=04&lic=1

[17] Dan Goodin, "Automated Automated crack for Windows Live captcha goes wild", The Register, Feb 8, 2008. http://www.theregister.co.uk/2008/02/08/microsoft_captcha_buster/

[18] Websense Security Labs, "Streamlined anti-CAPTCHA operations by spammers on Microsoft Windows Live Mail", Feb 6, 2008. http://securitylabs.websense.com/content/Blogs/2907.aspx

[19] J Elson, JR Douceur, J Howell and J Saul. "Asirra: a CAPTCHA that exploits interest-aligned manual image categorization". ACM CCS'07.

[20] "Yahoo! CAPTCHA is broken", available at http://network-security-research.blogspot.com/2008/01/yahoo-captcha-is-broken.html.

[21] J Yan and A S El Ahmad. "Usability of CAPTCHAs - Or, Usability issues in CAPTCHA design", the fourth Symposium on Usable Privacy and Security, Pittsburgh, USA, July 2008.

[22] J Yan and A S El Ahmad. "A Low-cost Attack on a Microsoft CAPTCHA", School of Computing Science Technical Report, Newcastle University, England, 2008.