A Low-cost Attack on a Microsoft CAPTCHA

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

Abstract: CAPTCHA is now almost a standard security technology. The most widely used CAPTCHAs rely on the sophisticated distortion of text images rendering them unrecognisable to the state of the art of pattern recognition techniques, and these *text-based* schemes have found widespread applications in commercial websites. 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 analyse the security of a text-based CAPTCHA designed by Microsoft and deployed for years at many of their online services including Hotmail, MSN and Windows Live. This scheme was designed to be segmentation-resistant, and it has been well 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 a desktop computer with a 1.86 GHz Intel Core 2 CPU and 2 GB RAM. As a result, we estimate that this Microsoft scheme can be broken with an overall (segmentation and then recognition) success rate of more than 60%. On the contrary, its design goal was that "automatic scripts should not be more successful than 1 in 10,000" attempts (i.e. a success rate of 0.01%). For the first time, we show that a CAPTCHA that is carefully designed to be segmentation-resistant is vulnerable to novel but simple attacks. Our results show that it is not a trivial task to design a CAPTCHA scheme that is both usable and robust.

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 intends 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 programs. 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).

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].

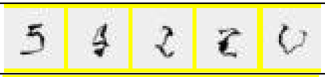
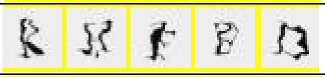
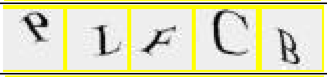
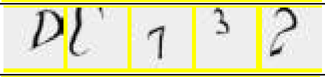
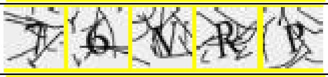
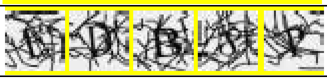
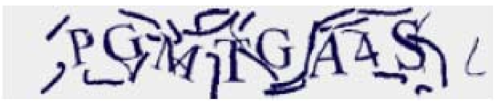
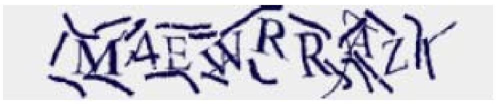
Characters under typical distortions	**Recognition rate**
	~100%
	96+%
	100%
	98%
	~100%
	95+%

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

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 examine the security of a Microsoft CAPTCHA, a scheme that is designed to be segmentation resistant. This scheme was designed by 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 Microsoft CAPTCHA has been deployed in many of their online services including Hotmail, MSN and Windows Live for years. Its first version was deployed in Hotmail's user registration system in 2002 [11], and 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 is carefully and well designed.

In this paper, we report a simple, low-cost segmentation attack that has achieved a success rate of higher than 90% on the latest version of this Microsoft CAPTCHA (as deployed in the summer of 2007). 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" attempts (i.e., a success rate of 0.01%) [4]. In fact, 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], it takes only slightly more than 80 ms in average for our attack to completely segment a challenge on a desktop computer with a 1.86 GHz Intel Core 2 CPU and 2 GB RAM. 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.

The rest of this paper is organised as follows. Section 2 discusses related work. Section 3 reviews the MSN scheme. Section 4 details our attack and Section 5 discusses the results of our attack. Section 6 discusses lessons we have learned, and Section 7 concludes this paper.

By attacking a well-designed, deployed CAPTCHA scheme, we learn how it could fail and could be improved. Overall, this paper contributes to the immediate improvement of the security of the widely deployed MSN CAPTCHA and schemes exhibiting similar weaknesses. It also contributes to furthering our understanding of the design of CAPTCHAs that are both secure and robust - the current collective knowledge on this topic is very limited.

2. Related work

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 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. This is one of the few examining the robustness of CAPTCHA from the security angle. In contrast to other work that relied on sophisticated computer vision or machine learning algorithms, we used only simple pattern recognition algorithms but exploited fatal design errors that we discovered in each scheme.

PWNtcha [7] is an excellent web page that aims to "demonstrate the inefficiency of many captcha implementations". It comments briefly on the weaknesses of a dozen visual CAPTCHAs. These schemes were claimed to be broken with a success rate ranging from 49% to 100%. However, no technical detail was publicly available (and probably as a consequence, at a prominent place of this web page, a disclaimer was included that it was not "a hoax, a fraud or a troll").

One last note: a survey on CAPTCHAs research can be found in [13].

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.

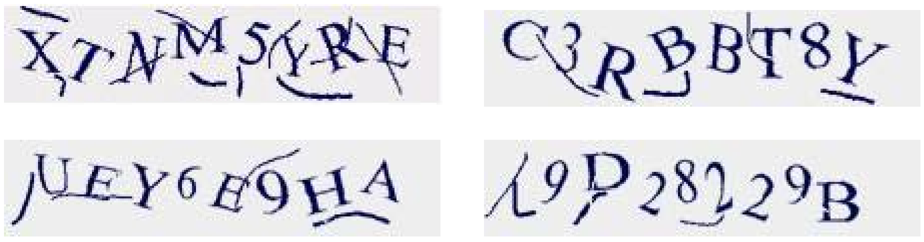


Fig 1. The MSN CAPTCHA sample challenges.

- 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. What is special in the design of the MSN scheme is the following: in contrast to many other schemes that use background textures and meshes in foreground and background colors as clutter to increase robustness, random arcs of different thicknesses are used as clutter in this scheme. The idea is that these arcs are themselves good candidates for false characters [5], and therefore such a design was expected to provide strong segmentation resistance. That is, current computer programs would fail to segment the distorted text into individual characters due to the introduction of random arcs.

4. A low-cost 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. Specifically, 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. (All the samples were collected in the summer of 2007.)

Our attack involves 7 consecutive steps, each of which is detailed in the following sections.

4.1 Binarization

We first convert a color challenge to a two-color image using a threshold method: pixels with intensity higher than a certain threshold value are converted to white, and those with a lower intensity to black (see Fig. 2)

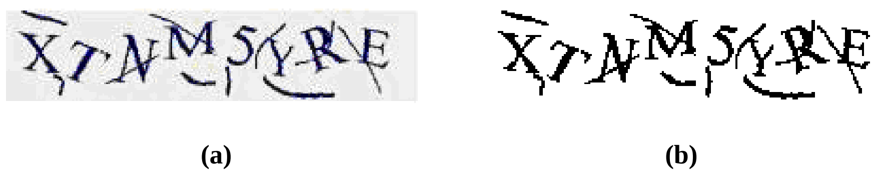


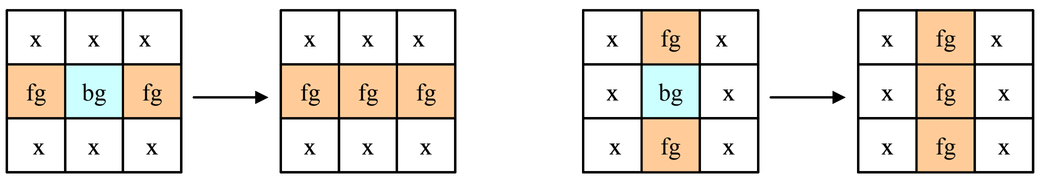
Fig 2. Binarization. a) original image, b) binarized image. (This sample is taken from [8], in which its resistance to segmentation is ranked by Microsoft as "hard", the highest level among all examples. We will use this sample to illustrate the whole process of our segmentation attack in this paper.)

4.2 Fixing broken characters

Thin background arcs remove some character content, and sometimes they also create a crack in characters. For example, the second character ('T') in Fig 2 is broken due to this reason. The current step attempts to fix broken characters for 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 an 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.



(a) (b)

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 4: 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 4, 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 this case if we chose to connect all two-pixel gaps.

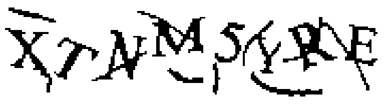
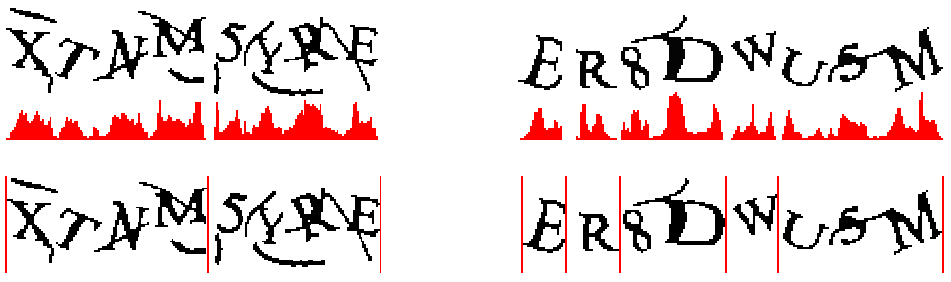


Fig 4. Fixed broken characters

4.3 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 5 shows that such vertical histogram segmentation cuts challenge (a) into two chunks, and the other (b) into five.



(a) (b)

Fig 5. Vertical Segmentation: two examples.

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.

4.4 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.

Our CFS method uses the notion of 8-connectivity for determining whether an adjacent pixel is a neighbor or not. In this notion, a pixel has eight neighbors (i.e., left, right, up, down, upper left, lower left, upper right and lower right). In contrast, in the notion of 4-connectivity, a pixel has only four neighbors (i.e., left, right, up, down). Fig 6 illustrates and compares both 8-connectivity and 4-connectivity. Of course, if a pixel and its neighbor are of the same color, then they are connected.

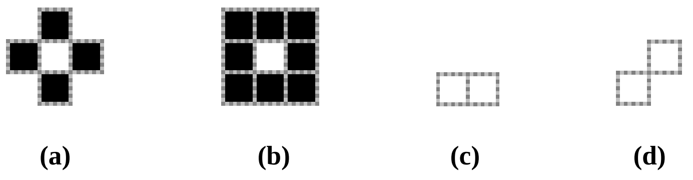


Fig 6. 4-connectivity vs. 8-connectivity. (a) 4-connectivity: a pixel (white) and its 4 neighbors (black), (b) 8-connectivity: a pixel (white) and its 8 neighbors (black), (c) two pixels that are both 4 connected and 8-connected neighbors (d) two pixels that are 8-connected but not 4-connected neighbors.

With our CFS method, as shown in Fig 7, we determine that there are six objects in the first chunk and five in the second.

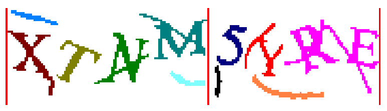


Fig 7. Color filling segmentation

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 (see Fig 8(a)), 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 8(b)), and we need to locate all objects in each chunk and track the number of objects for the follow-up arc removal and other steps.

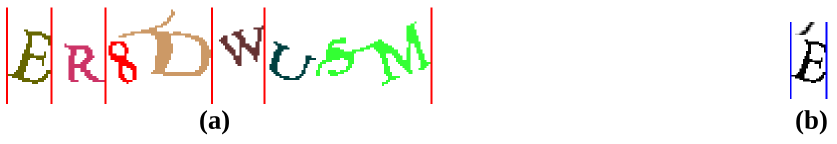


Fig 8. Why CFS is applied to each chunk. (a) The size of the 3rd and 5th chunks can suggest that the chunk contains more than one object; (b) two objects are in a thin chunk.

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

4.5 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 4.2. 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 challenge 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 9 shows two example cases. In Fig 9 (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 9 (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.)

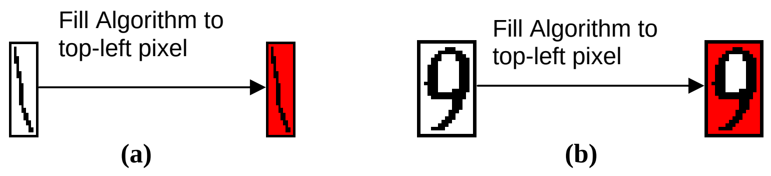

Fill Algorithm to top-left pixel
(a)
Fill Algorithm to top-left pixel
(b)

Fig 9. 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.) Fig 10 shows that an arc is removed in the 2nd chunk due to its small pixel count.

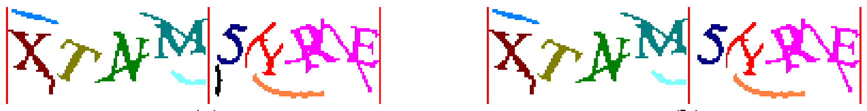


Fig 10. Arc removal - discriminative feature checking. (a) output from color filling; (b) an arc in the second chunk is removed.

3) *Relative position checking*. This step examines the relative position of objects in a chunk, and is applied to all chucks 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.

   Fig 11 shows that further arcs are removed by this method in the image processed by the previous step, and its histogram is updated.

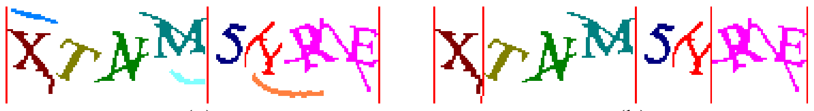


Fig 11. Arc removal - relative position checking. As shown in (b), further arcs in (a) are removed and histogram is updated.

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

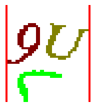
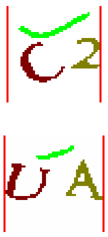
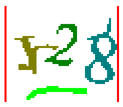
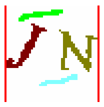
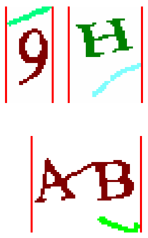
Relative position patterns			**Decision**
Layout	Description	Example	
O1 O2 O3	*Three objects in a chunk*: two objects more or less align along the baseline, the 3rd object under either of them		O3 is arc
O3 O1 O2	*Three objects in a chunk*: two objects more or less align along the baseline, the 3rd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

Table 2. Typical relative position patterns

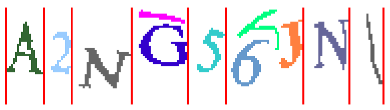
(*First apply the circle detection result obtained before: if only one of the objects contain a circle, then the object without a circle is removed as an arc. If this does not work, then the object that is less aligned with the baseline is removed as an arc.)

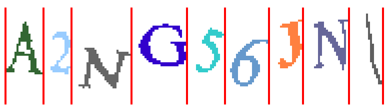
4) *Detection of remaining arcs*. The above steps do not necessarily identify all the arcs in an image. What is done in this step is as follows. First, count the number of remaining objects in the image (identified arcs are already removed and thus not counted). If this number is larger than 8, then 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.

Fig 12 shows the whole arc removal process with another example. Fig 12 (a) is 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 12 (b) is the result after those two arcs are removed and the histogram was updated. Then, escaped arcs detection catch the last object as an arc. The final image after all the process of arc removal is Fig 12 (c).


(a)


(b)

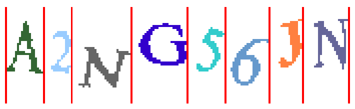

(c)

Fig 12. 3 Arc removal: another example

4.6 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).

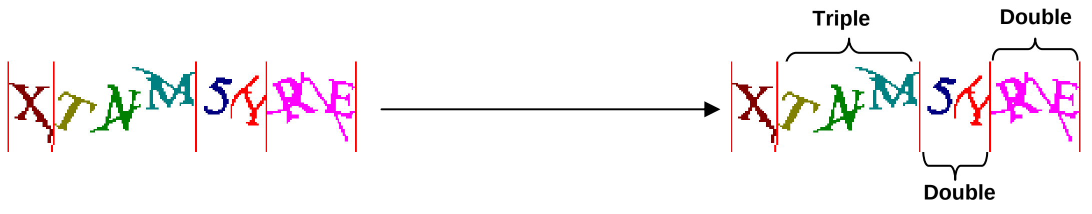
- Fixed length: every challenge uses 8 characters.
- Connected characters in an object are horizontally but never vertically juxtaposed. Therefore, an object containing two or more connected characters is typically wider than other objects.
- On average a segmented chunk – by definition, a chunk cannot 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).

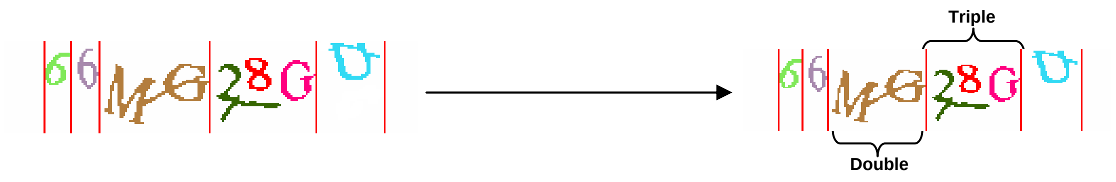
We use two examples (see Fig 13) to show how our algorithm works. The histogram for the image in Fig 13 (a) indicates that it contains four chunks. Since there are exactly 8 characters in these chunks, we know there are the following five exclusive possibilities for the distribution of all the characters among the chunks[1]:

(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.



(a)



(b)

Fig 13. “Approximation” for locating connected characters

The second example (see Fig 13 (b)) 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.

[1] 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.

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 neighbouring objects, which would make it much more complicated to implement the algorithm.

4.7 Segment 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 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 13 (a) and the 3rd object in Fig 13 (b) contain two connected characters. For these objects, what our algorithm does is to evenly divide them into two segments, each being a character. Fig 14 shows the finalised 8 segments for both challenges.

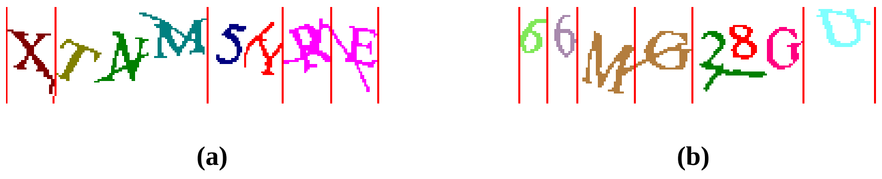


Fig 14. Completely segmented images

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 our attack was generic enough, we followed the practice in computer vision research (see, e.g. in [9, 10]). We collected another set of 500 random challenges, and then ran our attack on this test set – our program had no prior knowledge about any sample in this set. Our attack achieved a success rate of 92% (the distribution of samples in the test set slightly favours our algorithm).

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.

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

Table 3. Attack speed

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 properly 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.
- Second, 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 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 15 (where the example is taken from the last segment in Fig 14 (a)), the majority of columns and rows inside the red box have a pixel count larger than 3, while for portions outside of this box, the majority of columns and rows will range between 1 and 2 pixels – the thickness of thin intersecting arcs. Thus, portions of such arcs are rightly recognised and removed as distortion.

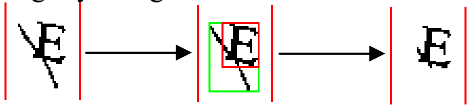


Fig 15. Thin arc removal using pixel-density based bounding box estimation.

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. Discussion

6.1 Analysis of our attack

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: in this case, not all thick arcs are detected and removed. In the following example, the 1st object in Fig 16 (a) is an undetected arc - it was treated as a valid character, and therefore our algorithm failed to recognise that the 5th object contained connected characters, leading to the failure to completely segment this challenge.

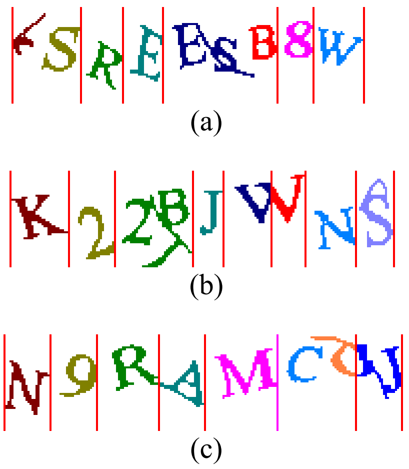


Fig 16: Typical failure cases: a) arc removal; b) "approximation"; c) "even cut".

Failure of approximation. This failure is about incorrectly identifying connected characters. Fig 16 (b) gives such a failure case, in which a single character ('W') was wider than two connected characters and therefore the latter were not identified, leading to the final segmentation failure.

Failure of "even cut" (Segmentation of connected characters). It is not surprising that the simple "even cut" method we used for segmenting connected characters does not always work. Fig 16 (c) gives such a failure case, in which the first two connected characters ('RA') were properly segmented, but the second pair ('CJ') was not (due to the length of the intersecting arc).

Fig 17 compares the failure rate of all these scenarios in both the sample and test sets. In the sample set, 5% of the segmentation errors were due to failures of arc removal (versus 17 out of 500 samples, i.e. 3.4%, in the test set), 2% of the segmentation errors were due to failure to locate connected characters (versus 3.2% in the test set), and 2% due to inaccuracy of the crude "even cut" method for segmenting connected characters (versus 1.4% in the test set). Thus, the failure patterns in both sets are similar.

The combination of arc removal and "approximation" accounts for 77.8% (7/9) and 82.5% (33/40) of the failures in the sample and test sets respectively. To improve our attack, arc removal will be an obvious first-priority target. This is not only because arc removal is the largest failure category: 56.6% (5/9) in the sample set and 42.5% (17/40) in the test set, but also because if the failure rate of arc removal is decreased, the success rate of "approximation" will be proportionally increased.

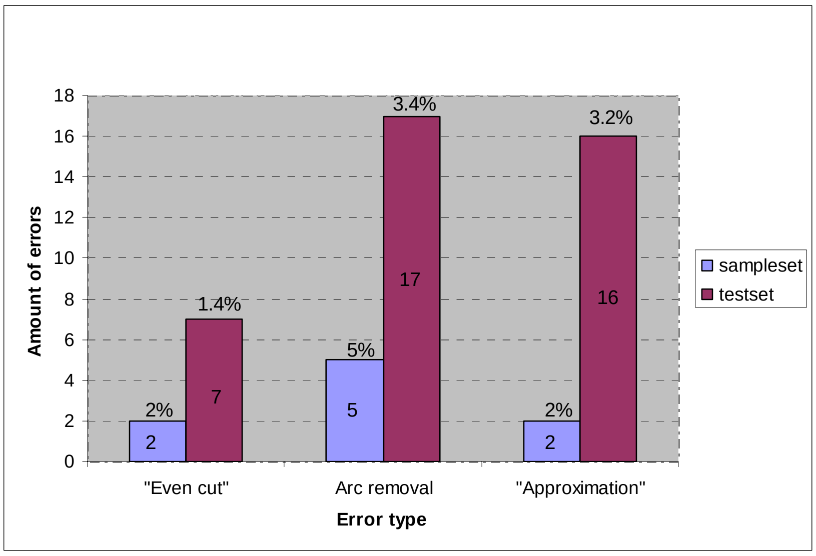


Fig 17: Analysis of failure cases

6.2 Defence

There are simple methods for defending against our segmentation attacks, for example,

- Letting characters touch or overlap with each other can provide extra segmentation resistance.
- Making it harder to tell characters and arcs apart (e.g. by juxtaposing characters in any direction).
- Randomly varied width for characters could confuse some parts of our attack.

6.3 Lessons

Strength of the MSN scheme

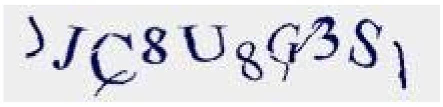
The designers recognised that both security and usability matter in the design of CAPTCHAs, and efforts were spent in addressing both issues in their design including. In particular, they attempted to address trade-off between security and usability – which is a tricky issue for CAPTCHA design.

Usability of this MSN scheme is reasonably good. For example, characters are not so distorted as to damage their recognisability by most human users. One particular good usability feature is that this scheme does not significantly disadvantage people whose mother tongue does not use the Latin alphabet. In some schemes, characters are distorted to be similar to handwriting - native speakers might find it easy to recognise them, but just imagine how difficult it would be for a user to recognise handwriting in a language that she knows nothing about.

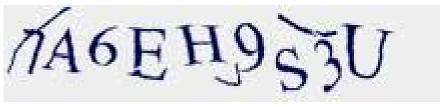
Weakness of the MSN scheme

Security. A major problem of this scheme is that it is vulnerable to our simple segmentation attack. The segmentation resistance built into this scheme seems to be largely about preventing bounding-box based segmentation, and apparently its designers never realised that a simple color filling process can be used to do segmentation effectively and that a combination of vertical and color filling segmentation can be powerful. Moreover, it is easy to tell arcs from characters by examining characteristics such as pixel counts, shapes, locations, relative positions, and distances to baseline. In addition, the use of a fixed number of characters per challenge also aids our segmentation attack.

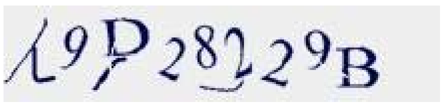
Usability. Some distortions still cause usability concerns. For example, it is difficult to tell an arc from a character such as 'J', '7' and 'L' in Fig 18. In particular, the confusion between an arc and 'J' was observed regularly in our study.



(a)



(b)



(c)

Fig 18. 1st object in (a), (b) and (c) looks like 'J', '7' and 'L' respectively.

And since a thin arc can connect thick arcs and characters, it is possible that random shapes created by this connection can look similar to other valid characters.

On the design of CAPTCHAs

- It is not a trivial task to design a CAPTCHA scheme that is both usable and robust.

- Size matters. The size of the character set to be used in a CAPTCHA matters, and so does the length of a string used in each challenge. If the character set size and the string length are too small, bots would have a high chance of using random guesses to pass the CAPTCHA test. On the other hand, the longer a string that is used in a challenge, the more security it can achieve. For example, assume that individual character recognition rate is r (<1), the chance of recognising the whole challenge of n characters is typically r^n, which decreases as n grows.

- The string length issue has interesting usability implications. If strings composed of random combinations of characters are used in a scheme, then the longer the string is, the more difficult the scheme is to use. The reason is it is more demanding for users to correctly decode and enter their answers. For example, users might tend to make recognition mistake, e.g. due to distorted character looking like others. However, this is not necessarily the case in schemes where English words are used. For example, it was observed for the reCAPTCHA scheme [15] that the longer the string is, the higher the pass rate users have [2]. A likely explanation is that the longer the word is, the more information people can gather, and thus Gestalt psychology effectively helps people to decode the word correctly. However, from short words that are too distorted to recognise immediately, users would not be able to gather enough information to decode them correctly.

- Whether the length of strings used in a scheme is predictable or not can also be a design issue that has implications on both security and usability. We observed that although the design choice of using a fixed string length in the MSN scheme aided our segmentation attack, it can improve the scheme's usability. For example, the knowledge that each challenge contains exactly 8 characters can ensure users realise that the first object in each challenge in Fig 18 is a random arc, rather than a character 'J', '7' or 'L'.

- On the contrary, if the MSN scheme used a variable and unpredictable string length for each challenge, it would be much harder or even impossible for users to recognise that the above-mentioned objects are indeed arcs. At the cost of this decrease in usability, however, this design choice would make it much harder or even impossible to perform an automatic segmentation attack such as ours.

7. 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. An important lesson is that even if segmentation resistance is a sound principle for designing secure text-based CAPTCHAs, the devil is in the details. It is critical to make sure that a design is not vulnerable to any known (and ideally unknown) segmentation method.

On the other hand, designing CAPTCHAs that exhibit both good robustness and usability is much harder that it might appear to be. First, current collective understanding of this topic is still in its infancy. Second, the requirements, tools and methodologies for assessing CAPTCHA designs are almost non-existent. 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.

Acknowledgments

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 paper confidential until 10 April, 2008. We are grateful to Brian Randell for proofreading this paper and many useful comments.

Postscript

While this paper was held confidential, it was reported [17] on Feb 8, 2008 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].

Reference

1. L von Ahn, M Blum and J Langford. “Telling Humans and Computer Apart Automatically”, CACM, V47, No2, 2004.
2. Luis von Ahn, Personal Communications, Oct 2007.
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. http://recaptcha.net/

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