CAPTCHA Security

A Case Study

A simple but novel attack can break some CAPTCHAs with a success rate higher than 90 percent. In contrast to early work that relied on sophisticated computer vision or machine learning techniques, the authors used simple pattern recognition algorithms to exploit fatal design errors.

Jeff Yan and Ahmad Salah El Ahmad
Newcastle University, England

Completely Automated Public Turing Tests to Tell Computers and Humans Apart (CAPTCHAs) are now an almost standard security mechanisms for defending against undesirable and malicious bot programs on the Internet (especially those bots that can sign up for thousands of accounts a minute with free email service providers, send out thousands of spam messages in an instant, or post numerous comments in blogs pointing both readers and search engines to irrelevant sites). CAPTCHAs generate and grade tests that most humans can pass but current computer programs can't.[1] Such tests—often called CAPTCHA challenges—are based on hard, open artificial intelligence problems.

To date, the most commonly used CAPTCHAs are text-based, in which the challenge appears as an image of distorted text that the user must decipher and retype. These schemes typically exploit the difficulty for state-of-the-art computer programs to recognize distorted text. Well-known examples include EZ-Gimpy, Gimpy, and Gimpy-r,[1] all developed at Carnegie Mellon University; Google, Microsoft, and Yahoo have also developed and deployed their own text CAPTCHAs. Many more schemes have been put into practice, but they're less visible in the literature. The "Related Work in CAPTCHA Design and Security" sidebar highlights additional efforts in the research community.

A good CAPTCHA must not only be human friendly but also robust enough to resist computer programs that attackers write to automatically pass CAPTCHA tests. However, designing CAPTCHAs that exhibit both good robustness and usability is much harder than it might seem. The current collective understanding of this topic is very limited, as suggested by the fact that many well-known schemes break. In particular, we recently found that we could break a widely deployed CAPTCHA—carefully designed and tuned by Microsoft—with a success rate of higher than 60 percent, even though its design goal was that automated attacks shouldn't achieve a success rate of higher than 0.01 percent.[2] We expect 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.

In this article, we study the strength of a CAPTCHA presented in a recent paper[3] and deployed on the Internet. We show that although this scheme effectively resisted one of the best optical character recognition (OCR) programs on the market, we could break it with a success rate of higher than 90 percent using a simple but novel attack that takes less than 50 ms on an ordinary desktop computer for decoding each challenge. In a nutshell, we found that simply counting the pixels in a CAPTCHA's characters can be a very powerful attack.

Our Target CAPTCHA

Our target CAPTCHA is the word-image scheme deployed at http://captchaservice.org, a publicly available Web service designed to generate CAPTCHA challenges. In a recent paper,[3] the scheme's creator,

Related Work in CAPTCHA Design and Security

Several pioneering efforts in the research community explore how to design text CAPTCHAs properly.[1–5]

Greg Mori and Jitendra Malik[6] broke two early simple CAPTCHAs—EZ-Gimpy (with 92 percent success) and Gimpy (33 percent success)—with sophisticated object recognition algorithms. Gabriel Moy and his colleagues[7] developed distortion estimation techniques to break EZ-Gimpy with a success rate of 99 percent and four-letter Gimpy-r with a success rate of 78 percent. Kumar Chellapilla and Patrice Simard[4] attacked several CAPTCHAs taken from the Web by using machine-learning algorithms, achieving a success rate of from 4.89 to 66.2 percent.

PWNtcha is an excellent Web page that aims to "demonstrate the inefficiency of many CAPTCHA implementations" (http://sam.zoy.org/pwntcha). It briefly comments on the weaknesses of roughly a dozen text-based CAPTCHAs, which this site claimed to break with a success ranging from 49 to 100 percent. However, no technical details about the attacks are publicly available.

Most recently, we developed effective low-cost attacks on CAPTCHAs deployed by Microsoft, Yahoo, and Google,[8] although PWNtcha considered two of the schemes "very good" and difficult to break.

Protocol-level attacks on CAPTCHAs, together with their limitations of defending against bots, appear in the literature,[9] as does a recent survey on CAPTCHA research.[10]

References

1. H.S. Baird, M.A. Moll, and S.Y. Wang, "ScatterType: A Legible but Hard-to-Segment CAPTCHA," *Proc. 8th Int'l Conf. Document Analysis and Recognition*, IEEE CS Press, 2005, pp. 935–939.
2. M. Chew and H.S. Baird, "BaffleText: A Human Interactive Proof," *Proc. 10th Document Recognition & Retrieval Conf.*, SPIE, 2003, pp. 305–316.
3. A.L. Coates, H.S. Baird, and R.J. Fateman, "PessimalPrint: A Reverse Turing Test," *Int'l. J. Document Analysis & Recognition*, vol. 5, nos. 2–3, 2003, pp. 158–163.
4. K. Chellapilla and P. Simard, "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)," *Advances in Neural Information Processing Systems*, MIT Press, 2004, pp. 265–272.
5. K. Chellapilla et al., "Designing Human Friendly Human Interaction Proofs," *Proc. ACM Conf. Human Factors in Computing Systems* (CHI 05), ACM Press, 2005, pp. 711–720.
6. G. Mori and J. Malik, "Recognising Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, IEEE CS Press, 2003, pp. 134–141.
7. G. Moy et al., "Distortion Estimation Techniques in Solving Visual CAPTCHAs," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, IEEE CS Press, 2004, pp. 23–28.
8. J. Yan and A.S. El Ahmad, "A Low-Cost Attack on a Microsoft CAPTCHA," *Proc. 15th ACM Conf. Computer and Communications Security* (CCS 08), ACM Press, 2008, pp. 543–554.
9. J. Yan, "Bot, Cyborg and Automated Turing Test," to be published in *Proc. 14th Int'l Workshop Security Protocols*, Springer; www.cs.ncl.ac.uk/research/pubs/trs/papers/970.pdf.
10. C. Pope and K. Kaur, "Is It Human or Computer? Defending E-Commerce with CAPTCHA," *IT Professional*, vol. 7, no. 2, 2005, pp. 43–49.

Tim Converse (a technical manager at Yahoo), discussed the design of both this Web service and all the CAPTCHAs it supported.

Figure 1 shows the example challenges the target scheme generated. Using Converse's paper and the 100 random samples we collected for a sample set, we observed that this CAPTCHA had the following characteristics:

- The challenge was always a distorted image of a six-letter English word randomly chosen from a fixed set of 6,000 words.
- The distortion method used was a random shearing technique. As Converse described, "the initial image of text is distorted by randomly shearing it both vertically and horizontally. That is, the pixels in each column of the image are translated up or down by an amount that varies randomly yet smoothly from one column to the next. Then, the same kind of translation is applied to each row of pixels (with a smaller amount of translation on average)."[3]
- There were only two colors in each challenge image (PNG format), one for background and the other for foreground, which was the distorted challenge text. The choice of colors was either user specified or randomly created by the CAPTCHA.
- Each challenge used only capital letters.
- Letters in a challenge might overlap when projected vertically but rarely connected with each other.

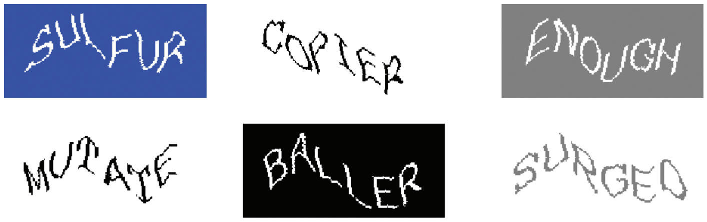

Figure 1. Example challenges. The target CAPTCHA generated English words in capital letters.

To benchmark how resistant this scheme was to OCR software attacks, we tested the sample set with ABBYY FineReader V.8, one of the best-quality commercial OCR products on the market. We performed two attacks: feeding each sample into the OCR software for automated recognition and manu-

Number of challenges						
All characters recognized	Partially recognized (no. of characters)					Zero characters recognized
	5	4	3	2	1	
0	0	3	6	8	16	67

(a)

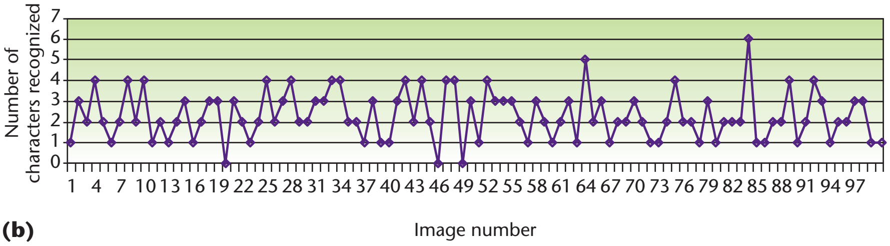

Figure 2. Resistance to optical character recognition attacks. In (a) the first attack, we fed each sample into the OCR software for automated recognition, and in (b) the second attack, we manually segmented each sample before letting the software recognize individual characters.

Table 1. A letter-pixel count lookup table for letters A to Z.*

LETTER	PIXEL COUNT	LETTER	PIXEL COUNT
A	183	N	239
B	217	O	178
C	159	P	162
D	192	Q	229
E	163	R	208
F	133	S	194
G	190	T	175
H	186	U	164
I	121	V	162
J	111	W	234
K	178	X	181
L	111	Y	153
M	233	Z	193

*J and L have the same pixel count; so do K and O, and P and V.

ally segmenting each sample and then letting the software recognize individual characters.

In the first attack, the OCR software couldn't completely recognize any of the samples. For 67 of them, it recognized no characters, whereas for the remaining 33, it partially recognized between one and four characters (see Figure 2a).[4] As Figure 2b shows, in the second attack, the software recognized 38 percent of the individual letters (128 out of 600) but recognized all six letters in only one sample (in one other and the only other sample, it recognized five of the letters). This isn't surprising, given that the recognition rate for individual letters theoretically implies a success rate of just 0.3 percent (.38) for breaking this scheme.[5,6] So, it seems that random shearing distortion provides reasonable resistance to OCR software attacks.

A Novel Attack

Although our target CAPTCHA appeared to be secure, as shown in the previous section, we broke it with a high success rate using a novel attack that consisted of a basic attack algorithm and several refinements.

Basic Attack

We based our attack on the following key insights that we obtained by analyzing the samples we collected:

- Most pixels in each challenge were of the background color—that is, the background color was dominant, and any pixel of a different color value in the image appeared in the foreground (as part of the challenge text). So, a program could easily and automatically extract the challenge text.
- Many challenges were vulnerable to a vertical segmentation attack (which we explain in more detail later); a program could vertically divide challenge images into segments that each contained a single character.
- Although a letter might be distorted into a different shape each time, it comprised a constant number of foreground pixels in the challenge image—that is, each letter had a constant pixel count. We worked out the pixel count for each letter from A to Z (see Table 1) and found that most had a distinct pixel count.

Our basic attack first used the vertical segmentation method, a standard technique that maps an image to a histogram representing the number of foreground pixels per column in the image. Then, we used vertical segmentation lines to separate the image by cutting through columns with no foreground pixels at all. Figure 3a shows that this method correctly cuts a challenge into six segments.

Once we segmented the challenge, our attack simply counted the number of foreground pixels in each segment. Then, we use the pixel count of each segment to look up Table 1 to determine the letter in the segment (see Figure 3b).

This basic attack achieved a success rate of 36 percent on the sample set—that is, it could completely recognize 36 out of the 100 sample challenges.

Dictionary Attack

Naturally, our basic attack failed to completely recognize some challenges. Figure 4 gives a failing example, in which the vertical segmentation method couldn't separate the letters S and K because a vertical slicing line couldn't split the letters without touching both of them. Our basic attack couldn't do anything more than give a partially recognized result "FRI★★Y" (we use an asterisk to represent each unrecognized character).

However, because the target CAPTCHA used English words, we found that we could enhance our basic attack by using a dictionary attack as follows. We first compiled a dictionary of roughly 6,000 six-letter English words. We used any partial result that the basic algorithm returned as a string pattern to identify candidate words in the dictionary that matched the pattern. Because multiple candidate words could exist, we introduced a simple solution to find the best possible result: for each dictionary entry, we precomputed (using Table 1) a pixel sum—the total number of pixels this word could have when embedded in a CAPTCHA challenge—and stored it along with the word in the dictionary. We also worked out, on the fly, a pixel sum for the unbroken challenge, which is the total number of all foreground pixels in the challenge. Our enhanced attack returned the first candidate word with the same pixel sum as the challenge as the final recognition result.

Figure 5 illustrates how the enhanced algorithm works. In this case, it used the partial result "FRI★★Y" that the basic algorithm obtained to identify all words that start with "FRI" and end with "Y" in the dictionary and found five candidate words: FRIARY, FRILLY, FRISKY, FRIZZY, and FRIDAY. However, it returned "FRISKY" as the best possible result because it was the only candidate that had a precomputed pixel sum of 987, which equals the unbroken challenge's pixel sum (133 + 208 + 121 + 372 + 153 = 987).

To make the dictionary attack work properly, it's crucial to create the correct string pattern after the vertical segmentation process. For example, when the vertical segmentation divided an image into only four segments, and the corresponding partial result was in the form of "B□B□," we needed to determine how many unrecognized letters were in each box "□." Otherwise, "B★B★★★," "B★★B★★," or "B★★★B★" would give totally different recognition results. On the other hand, if we used all these patterns, a lookup in the dictionary would likely find too many candidate words with an identical pixel sum.

This is exactly a problem of indexing letters in their correct positions; our solution involved a two-step method:

1. In some cases, it was trivial to work out a string pattern using contextual information. For instance, if a segmented image contained only one unrecognized segment, such as the example in

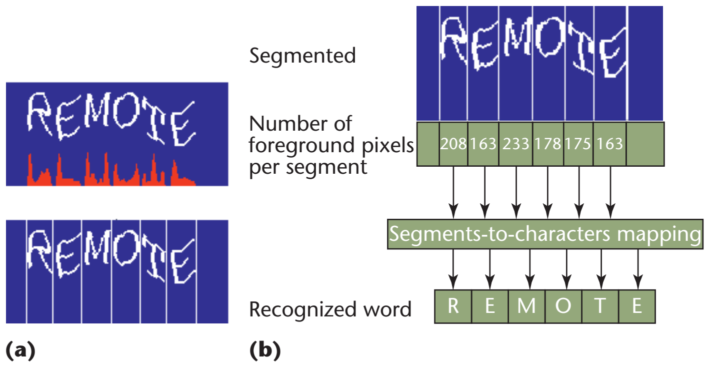

Figure 3. Our basic attack. We used (a) vertical histogram segmentation to cut a challenge image into six segments or letters and (b) table lookup to determine those letters.

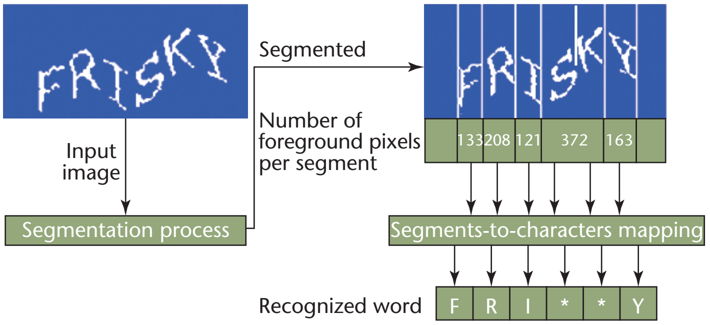

Figure 4. A failing example. Our basic attack couldn't completely recognize all the characters in some challenges. The partial line between S and K here is for illustration only; it didn't exist in the segmented result.

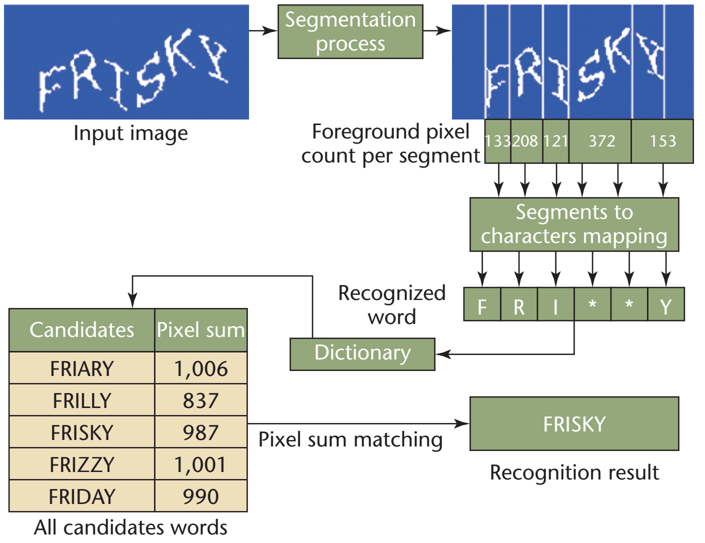

Figure 5. The basic attack with two enhancements. With the aid of a dictionary, the enhanced algorithm searches for six-letter English words that match up with unbroken challenge's pixel sum.

Figure 5, the number of unrecognized characters in the segment was six minus the number of all recognized characters. Another straightforward case was when our basic attack algorithm didn't recognize any characters in an image, which meant that the number of unrecognized segments in the image didn't really matter. For example, an image segmented into three unrecognized segments "□□□" would be no different than one for which the vertical segmentation completely failed.

2. When this first method didn't work, such as in the case of "B□B□," we relied on the number of pixels in each unrecognized segment to deduce how many characters the segment contained. For instance, when the number of pixels in a segment was larger than 239 (the largest pixel count in Table 1, which was for the letter N) but smaller than 2 × 239, this segment likely had two unrecognized letters. Some exceptions couldn't be handled this way; although the average pixel count for letters A to Z was 178.80, J, L, and I had a pixel count much smaller than the average. For example, the pixel sum of "ILL" or "LIL" was only 343 and a mere 232 for "LI" or "IL." We used an exception list to handle such cases. On the other hand, the combinations "LLL," "JK," and "KJ" never or rarely occur in English words.

An alternative to pixel sum matching was to use unrecognized segments only. In this way, we wouldn't store any pixel sums in the dictionary, but more computation would be required on the fly.

It's also worth noting that when the basic algorithm couldn't recognize any of the letters in a challenge, the dictionary attack's pixel sum matching method could serve as a last resort.

Further Enhancements

We experienced some "troublemakers" in the sample set that the previously described techniques couldn't break, so we devised some additional methods to handle them.

Letters with an identical pixel count. Letters having an identical pixel count could confuse our basic algorithm—for example, our algorithm initially recognized one challenge as "OELLEY," although the correct answer should be "KELLEY." Because O and K have the same pixel count, our basic algorithm had just a 50 percent of chance of breaking this challenge.

To overcome this problem, we relied on the following "spelling check": if a challenge includes a letter with a pixel count of 111 (J or L), 178 (K or O), or 162 (P or V), for each likelihood we generate a variant and then perform multiple dictionary lookups to rule out candidate strings that aren't proper words. In this case, our attack looked up both "OELLEY" and "KELLEY" in the dictionary, and because it only found "KELLEY," this result was returned as the best possible answer.

We also used this spelling check to enhance the dictionary attack algorithm's string pattern matching—for example, if the basic algorithm recognized a partial result as "V★B★IC," then both "V★B★IC" and "P★B★IC" would be valid matching patterns for identifying candidate words in the dictionary.

Broken characters. Some challenges contained broken letters that misled the segmentation algorithm—for instance, it could segment a letter into two parts instead of one, due to a crack in the letter.

To overcome this problem, we introduced a two-step method as follows. First, once the vertical segmentation process finished, our algorithm tested whether a segment was complete: if the number of foreground pixels in a segment was smaller than 111 (the smallest pixel count in Table 1), then this segment was incomplete; if the number of foreground pixels in a segment was larger than 111 but smaller than 239 (the largest pixel count in Table 1, that is, the letter N) and the algorithm couldn't find this number in the lookup table, then this segment was incomplete. Second, we would try to merge an incomplete segment with its neighboring segments. A proper merging was one for which the combined pixel count could lead to a meaningful recognition result—for instance, the combined count was equal to or less than 239, and appeared in the lookup table. When multiple proper combinations existed (for example, when segment 3 could be combined with either segment 2 or 4), the spelling check served as a last resort for finding the best possible result.

Additional pixels. In some cases, a letter might contain additional pixels against its pixel count in the lookup table. To address this problem, we relied on an approximate table lookup: when we couldn't find a pixel count in the lookup table for a segment, we recognized this segment as the most likely letter.

This method didn't succeed all the time because some letters had close pixel counts (such as V, E, and U; D, Z, and S; and M and W). However, sometimes, we could resort to the spelling check technique to get the correct answer. For example, when the approximate method returned multiple candidate answers, we could use spelling check to choose the best possible solution.

Results

With all these enhancements, our attack achieved a success rate of 92 percent on the sample set. To check whether our attack was generic enough, we followed a common practice in computer vision communities.[5,6] Specifically, we collected another set of 100 random challenges and then ran our attacks on this new test set; we didn't analyze any challenges in the test set, nor did we make any additional modifications to our program. Our attack completely broke 94 challenges in the test set (which gives a success rate of 94 percent). To put this in perspective, an adversary with access to the CAPTCHA's dictionary would have merely a 1/6,000 chance of guessing correctly without using our attack.

We implemented our attacks in Java (we spent little effort optimizing the code's runtime) and tested it on a computer with a Pentium 2.8-GHz CPU and 512 Mbytes RAM. It took us roughly 46 ms on average to break a challenge in both the sample and test sets.

Most failure cases in both sets occurred for the same reason: the failure of vertical segmentation led to partial results such as "S*****" and "******," which matched too many candidate words with the same pixel sum in the dictionary. The only exception occurred to a sample in the test set in which the spelling check failed to differentiate between the letters P and V because two alternative candidate words were both in the dictionary.

Discussion

Our attack exploited the following fatal flaws in the target scheme:

- It was easy to separate foreground text from background with an automatic program.
- The random shearing technique as implemented was vulnerable to simple segmentation attacks.
- Constant and (almost) unique pixel counts for each character often made it feasible to recognize a character by counting the number of foreground pixels in each segment.
- English words used in the scheme made it vulnerable to dictionary attacks.

All three other Captchaservice.org schemes that used the random shearing distortion method were also vulnerable to the pixel-count attack. Our experiments suggested that they could be broken with almost 100 percent success;[4] some of these schemes used a different alphabet set, so we had to compile a revised pixel-count character lookup table for each of them. We also identified seven other schemes deployed on the Internet that were vulnerable to the pixel-count attack,[4] including Bot Check, a popular Wordpress plug-in. The pixel-count method also contributed to our attack on the Microsoft CAPTCHA we described at the beginning of this article.[2] In that attack, a segment's pixel count didn't identify which character the segment was, but it helped us determine whether the segment was a valid character or random noise.

However, not every font type is vulnerable to the pixel-count attack—for example, we tested fonts such as Arial, Courier, and Bell MT with a size of 16, 26, and 36 points, and none of them were susceptible because many characters in these typefaces have the same pixel count. So, a CAPTCHA can use a single font, but its designer should exercise caution to evaluate that font's suitability.

CAPTCHA designers have several simple methods to defeat our attack:

- Make it hard to separate the text from the background—for example, use multiple colors for both foreground and background, leave no pattern that could help distinguish the foreground automatically, and include some foreground colors into the background and vice versa.
- Make it hard to segment each image—for example, connect characters with each other or add more cracks in each character.
- Make it impossible to distinguish a character by counting its pixels—for example, make all characters have the same pixel count all the time. Or make a character have very different pixel counts in different challenges (if the difference isn't large enough, an approximation method could probably determine each character).

Random warping provides another good defense against the pixel count attack—for example, local warp can introduce "small ripples, waves, and elastic deformations along the pixels of the character"[7] and global warp generates character-level, elastic deformations; both can make a character's pixel count less predictable.

Furthermore, local warp can foil "feature-based algorithms, which may use character thickness or serif features to detect and recognise characters,"[7] and global warp can foil template-matching algorithms for character detection and recognition. So, our work provides additional supporting evidence that it's good practice to use warping (both local and global) for character distortion in CAPTCHAs.

To sum up, a CAPTCHA that isn't resistant to OCR software attacks is definitely insecure, but a CAPTCHA that is resistant to such attacks could still be vulnerable to (simple) customized attacks. CAPTCHA creators should consider the simple but powerful attack presented in this article before deploying a CAPTCHA.

It's highly relevant at this time to develop a methodology for evaluating CAPTCHA robustness—that is, whether a scheme is robust enough to resist adversarial attacks. This is still an open problem, and it's a large part of our ongoing work. Given that hundreds of design variations exist, we doubt that a universal attack applicable to all CAPTCHAs is possible. Rather, a more realistic expectation is to create a toolbox that includes attacks such as the one presented in this article. This toolbox will be able to not only benchmark CAPTCHA strength but also prevent designers from making mistakes identified in early research. Many lessons and experiences that early researchers learned failed to be applied to more recent designs for a simple reason: they weren't readily accessible to current designers. An automated software tool will change this situation. □

Acknowledgments

We're grateful to Brian Randell and Lindsay Marshall for valuable comments. This work was partially supported by the EU-funded NoE ReSIST project.

References

1. L. Von Ahn, M. Blum, and J. Langford, "Telling Humans and Computers Apart Automatically," *Comm. ACM*, vol. 47, no. 2, 2004, pp. 56–60.
2. J. Yan and A.S. El Ahmad, "A Low-Cost Attack on a Microsoft CAPTCHA," *Proc. 15th ACM Conf. Computer and Communications Security* (CCS 08), ACM Press, 2008, pp. 543–554.
3. T. Converse, "CAPTCHA Generation as a Web Service," *Proc. 2nd Int'l Workshop on Human Interactive Proofs* (HIP 05), LNCS 3517, H.S. Baird and D.P. Lopresti, eds., Springer-Verlag, 2005, pp. 82–96.
4. J. Yan and A.S. El Ahmad, "Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms," *Proc. 23rd Annual Computer Security Applications Conf.* (ACSAC 07), IEEE CS Press, 2007, pp. 279–291.
5. G. Mori and J. Malik, "Recognising Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, IEEE CS Press, 2003, pp. 134–141.
6. G. Moy et al., "Distortion Estimation Techniques in Solving Visual CAPTCHAs," *Proc. IEEE Conf. Computer Vision and Pattern Recognition*, IEEE CS Press, 2004, pp. 23–28.
7. K. Chellapilla et al., "Designing Human Friendly Human Interaction Proofs," *Proc. ACM Conf. Human Factors in Computing Systems* (CHI 05), ACM Press, 2005, pp. 711–720.

***Jeff Yan** is on the faculty of computer science at Newcastle University, England, where he leads research on systems and usable security. His recent work includes the design of novel graphical passwords, the robustness and usability of CAPTCHAs, and security in online games. Yan has a PhD in computer security from Cambridge University, is a contributor to the O'Reilly book* Security and Usability: Designing Secure Systems that People Can Use *(2005), and was on the program committee for the first Symposium on Usable Privacy and Security (SOUPS) held at Carnegie Mellon in 2005. Contact him at jeff.yan@ncl.ac.uk.*

***Ahmad Salah El Ahmad** is a PhD student at Newcastle University, England. His research interests include computer security, particularly the design of secure and usable CAPTCHA systems. El Ahmad has an MSc in computing science from Newcastle University. He's a student member of the IEEE. Contact him at ahmad.salah-el-ahmad@ncl.ac.uk.*