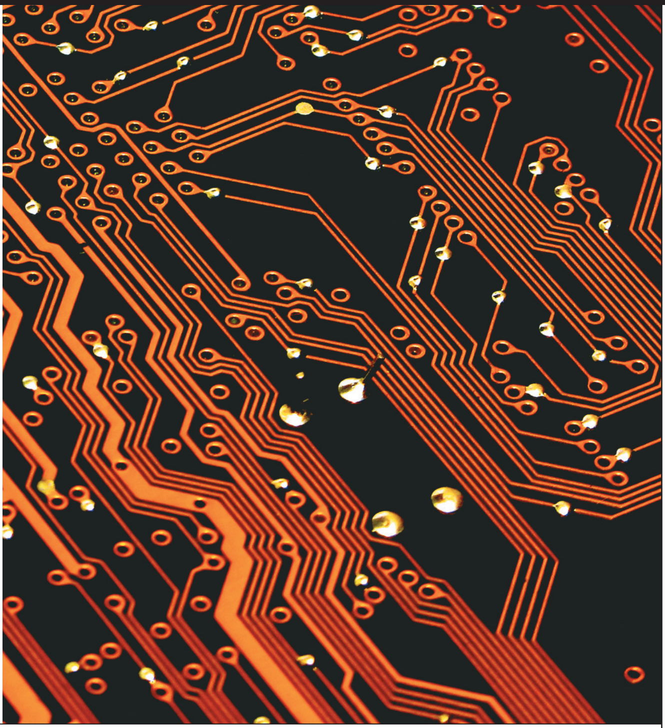
Captcha Robustness: A Security Engineering Perspective



Jeff Yan and Ahmad Salah El Ahmad
Newcastle University

Captchas are a standard defense on commercial websites against undesirable or malicious Internet bot programs, but widely deployed schemes can be broken with simple but novel attacks. Applying security engineering expertise to the design of Captchas can significantly improve their robustness.

A Captcha—completely automated public Turing test to tell computers and humans apart, also known as a human interaction proof—is a program that generates and grades tests that are human solvable but intended to be beyond current computers' capabilities.[1] This technology often makes use of a hard, open AI problem and is now a standard defense on commercial websites against undesirable or malicious Internet bot programs. For example, Google, Microsoft, and Yahoo have all deployed Captchas to make it more difficult for spammers to harvest free e-mail accounts.

In 1996, Moni Naor first proposed using automated Turing tests to verify that a human, rather than a bot, was making a query to a service over the Web.[2] AltaVista patented a similar idea in 1998. A research team at Carnegie Mellon University (CMU) led by Manuel Blum and Luis von Ahn coined the term Captcha in 2000, and they played a major role in popularizing the technology. To date, the most widely used Captchas are text-based schemes that prompt users to recognize distorted characters, which state-of-the-art pattern recognition programs supposedly cannot do.

Because a Captcha's role is effectively the same as a simple challenge-response protocol, Captchas are vulnerable to protocol-level attacks. For example, a spammer could shift the load of solving Captcha challenges to porn site visitors; a spammer could also outsource such a task to people in countries where cheap labor is available. System design is also important. For example, hackers could bypass some early Captchas simply by reusing a known challenge image's session ID.[3]

We have explored another aspect of Captchas' security—namely, their *robustness*, or the strength of their resistance to computer programs written to automatically solve Captcha tests. We found that numerous recent Captchas, including the schemes widely deployed by Microsoft, Yahoo, and Google as well as others less well known, could be broken with high success using simple but novel attack strategies that exploited fatal design errors in each scheme.

In contrast to current techniques used to improve Captchas' robustness, developed primarily by the computer vision and document analysis and pattern recognition communities, we advocate a security engineering approach that applies adversarial thinking skills. Although we have focused on text-based Captchas, some of the lessons we have learned also apply to other types.

BACKGROUND

Specialized computer vision algorithms had success breaking some early text-based Captchas. For example, Greg Mori and Jitendra Malik designed sophisticated object-recognition algorithms to break EZ-Gimpy (92 percent success) and Gimpy (33 percent success), two early schemes created by the CMU team.[4] Gabriel Moy and colleagues later developed distortion estimation techniques to break EZ-Gimpy (99 percent success) and Gimpy-r (78 percent success).[5]

Microsoft researchers attacked numerous Captchas from the Web using machine-learning algorithms, largely neural networks, achieving a success rate ranging from 4.89 percent to 66.2 percent. They argued that breaking a challenge in which the characters' positions are known a priori is a pure recognition problem—a trivial task with standard machine-learning techniques; otherwise, such methods do not effectively locate the characters, let alone recognize them. In general, identifying character locations in the right order, or *segmentation*, remains an open problem that is computationally expensive and often combinatorially hard. Researchers therefore suggested that robust text-based schemes should rely on the difficulty of finding where each character is rather than what it is.[6-8] In other words, Captchas should be segmentation resistant.

A common method to estimate a Captcha's robustness is as follows: given the average percentage of challenges that can be entirely segmented correctly, s, and an achievable individual character recognition rate, r, the estimated overall success rate for breaking a scheme is $s \times r^n$, where n is the average length of text strings in the scheme.

A Captcha's character set is also relevant to the scheme's security. Given the character set's size, c, the chance for a blind guess to successfully recover a challenge that uses a random string of n characters will be $1/c^n$. If the scheme uses English words only, then an attack can try to collect all the words used to launch a dictionary attack. Given w number of words, the dictionary attack will have a success rate of $1/w$.

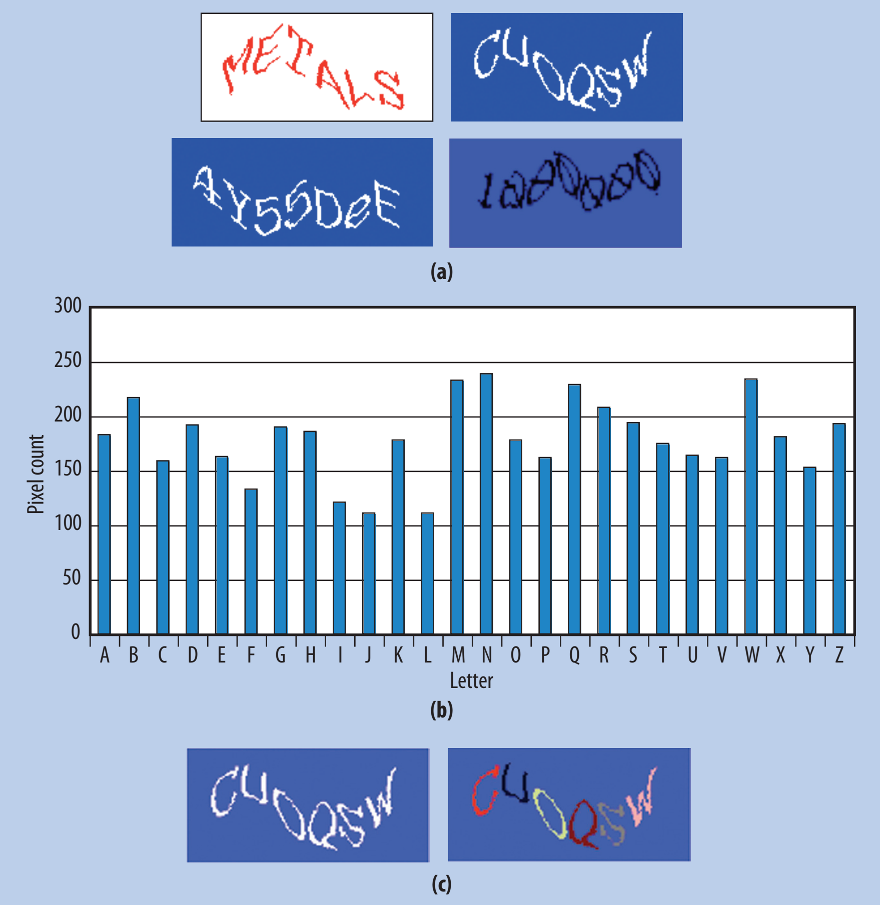


Figure 1. Four Captchaservice.org schemes. (a) Example challenges for each scheme (clockwise: word_image, random_letters_image, number_puzzle_text_image, user_string_image). The schemes used the same distortion technique but could differ in alphabet sets and text lengths. Each used two colors, with the challenge text being the foreground color. (b) Letters A-Z and their pixel counts. J and L, K and O, and P and V had the same pixel counts. (c) Color-filling segmentation. On the left is an original image, and on the right is the segmented image.

A commonly accepted goal for Captcha design is that automated attacks should not be more than 0.01 percent successful but that the human success rate should be at least 90 percent.[7] There is thus a tradeoff between Captcha robustness and usability.[9]

CAPTCHASERVICE.ORG SCHEMES

Captchaservice.org was a publicly available Web service that Tim Converse established in 2005 for the sole purpose of generating Captcha challenges.[10] We examined four of the Captcha schemes provided by this service: word_image, random_letters_image, user_string_image, and number_puzzle_text_image. Figure 1a shows a challenge example for each.

As deployed from 2006 to 2007, all four schemes based their robustness on a *random-shearing distortion* technique applied both vertically and horizontally to a challenge image. Converse explained that "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)."[10]

The main differences between the schemes were the alphabet set and text length. The word_image and random_letters_image schemes used six capital letters. The number_puzzle_text_image scheme used only numbers, which could be up to seven digits. The user_string_image scheme accepted any user-supplied string of at most 15 characters that consisted of digits and uppercase and lowercase letters.

Although a character was distorted into a different shape each time, it had the same pixel count in all challenges generated.

Random-shearing distortion provided all four Captcha schemes, with resistance ranging from reasonable to excellent in terms of being decoded by a top commercial optical character recognition product.[11] However, we usually could recognize all characters embedded in a challenge by exploiting common critical design flaws in the schemes.

One major vulnerability we identified is that although a character was distorted into a different shape each time, it almost always consisted of a constant number of foreground pixels—that is, it had the same pixel count in all challenges generated. Furthermore, as Figure 1b shows, most of the characters had a distinct pixel count.

A second critical flaw was that because each challenge used only two colors, with the challenge text being the foreground color, few characters connected with each other. This enabled us to use a simple *color-filling segmentation* (CFS) algorithm to identify each character in a challenge image in the right order. The algorithm first detected a foreground pixel and traced its foreground neighbors until it had traversed all pixels in a connected component. Next, it located a foreground pixel outside the detected component(s) and started another traversal process to identify an adjacent component. This process continued until all connected components in the challenge were located. Figure 1c shows the result of applying CFS to a challenge in which the number of colors used to fill the image is equal to the number of characters in the image.

Based on these two vulnerabilities, we designed the following six-step attack against the Captchaservice.org schemes:

1. Build a character-pixel count lookup table for the alphabet set used in a scheme.
2. Remove small noise dots, if any, in a challenge image—they are easily distinguishable as they have a pixel count much smaller than that of any legitimate character.
3. Divide the challenge into multiple segments with CFS.
4. Count the number of foreground pixels in each segment.
5. Look up the pixel count in the table to identify each candidate character. If a pixel count cannot be located in the table, the corresponding segment is very likely the component of a broken character. Combine this segment with its left and right neighbor segments, respectively, and treat the combination that returns a meaningful result in the lookup table as a single character. When both combinations are plausible, randomly choose one of them.
6. Distinguish characters with identical pixel counts such as J and L, K and O, and P and V by analyzing their geometric layouts with simple algorithms. (For the word_image scheme, which used an English word in each challenge, spell checking is an alternative method for distinguishing between characters with identical pixel counts.)

This simple attack was almost 100 percent successful at breaking each target scheme, and did so quickly. For example, it achieved 98 percent success breaking the random_letters_image scheme and took only around 16 milliseconds per challenge on an ordinary desktop computer with a Pentium 2.8-GHz CPU with 512 Mbytes of memory. This was much faster than previous attacks we used,[11] as CFS significantly outperformed older segmentation methods.

MICROSOFT CAPTCHA

Microsoft's Captcha is the product of an interdisciplinary team of experts in document processing and understanding, machine learning, human-computer interaction, and security, which established today's widely accepted segmentation-resistance principle and robustness criteria. The company first deployed the scheme in Hotmail's user registration system in 2002 and has continued to extensively improve its robustness and usability.[7,8] Microsoft online services including Hotmail, MSN, and Windows Live have used variations of the Captcha for years.

Figure 2a shows four example challenges generated by the Captcha as deployed in 2007. The main antisegmentation measure is the random use of nonintersecting and intersecting arcs of different thicknesses. Some arcs are as thick as the thick portions of characters and, to ensure usability, do not directly intersect with any characters. Other arcs are as thin as the thin portions of characters and intersect with thick arcs, characters, or both. Both types of arcs are the same color as challenge text. The designers' rationale was that the arcs themselves are good candidates for false characters, and thus the mix of random arcs and characters could confuse state-of-the-art segmentation methods.[7]

Segmenting the Microsoft Captcha requires the ability to distinguish arcs and valid characters, which we achieved in a simple attack illustrated in Figure 2b.

First, after some simple preprocessing including binarization, which converts a color challenge image to a black-and-white one, we applied a standard method to vertically segment the challenge into several chunks, each of which might contain one or more characters. *Vertical segmentation* involves mapping the image to a histogram that represents the number of foreground pixels per column in the image, then separating the image into chunks by cutting through columns that have no foreground pixels. Step 1 in Figure 2b shows a challenge divided into two chunks.

Next, as shown in Step 2 in Figure 2b, we applied CFS to each chunk to identify all the connected components, or *objects*, which can be arcs, characters, connected arcs, or connected characters.

Knowing the relative positions of objects in a chunk, we could discriminate arcs and real characters with high success. For example, characters were typically closer to the chunk's baseline, whereas arcs were closer to the top or bottom image borders. In addition, characters were horizontally juxtaposed, but never vertically. Based on these observations, we identified typical relative position patterns that could pinpoint which object was an arc in a chunk. Table 1 shows some of the patterns we identified along with real examples. This method of examining the relative

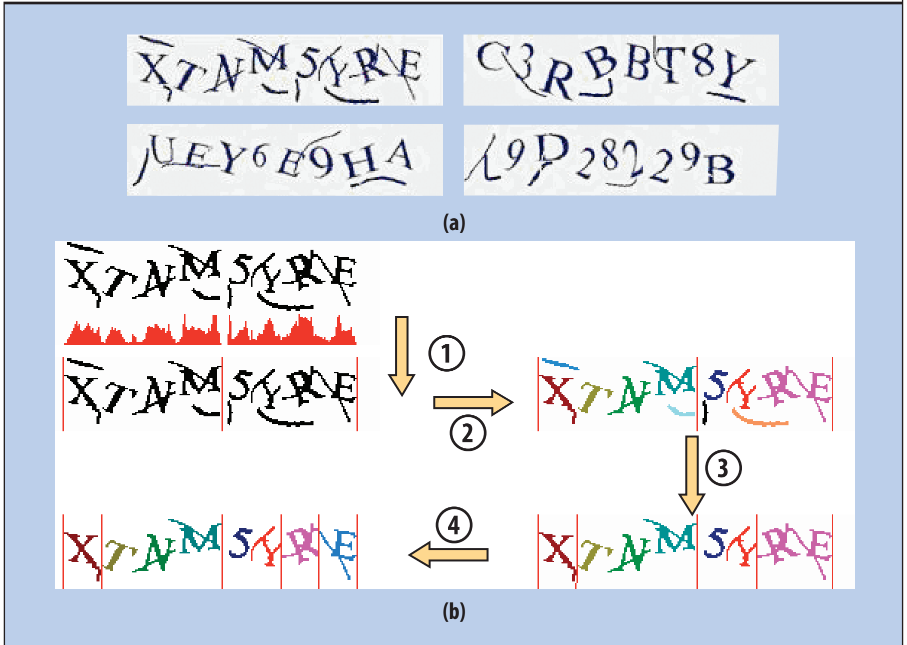


Figure 2. Microsoft Captcha. (a) Four example challenges. (b) A successful segmentation attack. Step 1: vertical segmentation, which divides a preprocessed image into chunks with the help of a histogram. Step 2: color-filling segmentation, which identifies separate objects in each chunk. Step 3: arc removal, which largely relies on relative position checking. After removing the arcs, we update the image's histogram and segment the image into more chunks. Step 4: identifying and then segmenting the remaining connected objects. The final result identifies eight valid characters in the right order, with each displayed in a different color and most arcs deleted.

Table 1. Objects' relative position patterns in Microsoft Captcha challenge image chunks.

Pattern	Description	Example	Implication
O1 O2 O3	Two objects more or less align with the baseline; a third object is under either of them		O3 is an arc and can be removed
O3 O1 O2	Two objects more or less align with the baseline; a third object is on top of either of them		O3 is an arc and can be removed
O1 O2 O3 O4	Three objects more or less align along the baseline; a fourth object is under any of them		O4 is an arc and can be removed
O3 O1 O2 O4	Two objects more or less align with the baseline; a third object is on top of either of them, and a fourth object is under either of them		O3 and O4 are arcs and can be removed
O1 O2	Two objects are vertically juxtaposed	or	The object that is less aligned with the baseline is an arc and can be removed

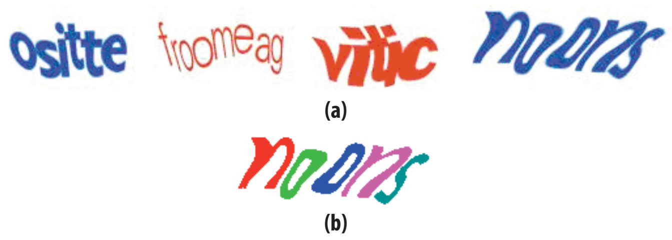


Figure 3. Google Captcha. (a) Example challenges. The far-right challenge is (b) vulnerable to CFS attack.

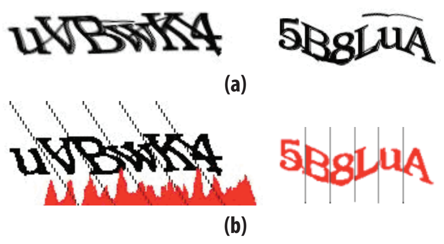


Figure 4. Yahoo Captcha. (a) Example challenges: angular (left) and regular (right). (b) Both types of challenges are vulnerable to segmentation.

position of objects, as Step 3 in Figure 2b shows, identified and removed most arcs in the challenge.

This Microsoft Captcha had other vulnerabilities: a valid character's pixel count was larger than that of an arc, and the length of text strings embedded in challenges was always eight. These flaws enabled us to accurately guess which object contained connected characters and how many such characters there were, and to properly segment the connected characters. Step 4 in Figure 2b gives an example of the final segmentation result.

Overall, our segmentation attack on this Captcha was more than 90 percent successful, and we estimated that Microsoft's scheme could be broken more than 60 percent of the time.[12]

GOOGLE AND YAHOO CAPTCHAS

Google and Yahoo have also been deploying Captchas to protect their online services.

Google Captcha

To resist segmentation, Google's Captcha crowds characters together—that is, lets them touch or overlap—as Figure 3a shows. However, we correctly segmented 12 out of 100 random samples collected between December 2007 and February 2008 using CFS alone. Figure 3b shows one challenge vulnerable to such an attack. The average text length in the samples was 6.25, leading to an overall success rate of 8.7 percent in breaking this scheme.

Yahoo Captcha

In 2000, Yahoo became one of the first major websites to adopt Captcha technology, and since then it has upgraded its schemes numerous times. We examined a version that the company rolled out in March 2008 specifically designed to be segmentation resistant.

As Figure 4a shows, challenge texts in this Captcha were compacted and characters usually either touched adjacent characters or were connected by intersecting random lines. Just one week after its deployment, however, we discovered critical flaws in this scheme that could be easily exploited.

In this Yahoo Captcha, the length of text embedded in a challenge often varied and was intended to be unpredictable. This is a good design feature, as it is hard or even impossible for an automated attack to segment a challenge if the number of characters in the challenge is unknown. However, the scheme had a key vulnerability: we could estimate the number of characters in a challenge more than 68 percent of the time by measuring the width of the text in the challenge image.

Another vulnerability in the Captcha was that it generated two main types of challenges that could be differentiated by a simple program. As the left image in Figure 4a shows, *angular* challenges employed a transformation that shifts character pixels by an angle while maintaining the characters' shapes—a process similar to changing characters from a *regular* shape to italic form but in an opposite direction. Regular challenges like the right image shown in Figure 4a did not undergo such a transformation.

We designed two simple segmentation algorithms to deal with each type of challenge, together with associated rules to identify which algorithm to use, and achieved a success rate of around 33.4 percent for segmentation. We estimate that the Yahoo Captcha scheme can be broken 25.9 percent of the time.

Segmenting angular challenges. After preprocessing steps such as binarization, CFS, and arc removal, we first projected a challenge at an angle of 33.5 degrees to the vertical—we observed that the angle used for angular transformation was almost always the same—to create a histogram that represented the number of foreground pixels per projecting line in the image. We then used the histogram's span (length) to estimate n, the number of characters in the challenge. Next, we divided the histogram into n even chunks, which yielded $n + 1$ boundary points in the x-axis. Starting with each point, we drew a line at an angle of 56.5 degrees to the horizontal to cut the challenge image into n segments, each presumed to contain a single character.

The left image in Figure 4b is an example of angular segmentation that correctly estimated the number of characters in the challenge and successfully segmented them.

Segmenting regular challenges. After preprocessing, we directly estimated n using text width. If there was only a single connected component (object), we evenly and vertically cut it into n chunks, each being a segment. If there were two or more objects, we used their relative size to estimate the number of characters in each object i, denoted by n_i. For example, if we estimated that a challenge contained five characters and there were two objects in the challenge, we determined that the object with a larger width contained three characters and the other object contained two characters. We then evenly and vertically divided object i into n_i chunks, each being a segment.

The right image in Figure 4b is an example of regular segmentation that correctly estimated the number of characters in the challenge and, by simply dividing the text vertically into six even segments, successfully segmented the text.

INVARIANCE AS A 'MAGIC WEAPON'

In essence, all our attacks searched for and exploited invariants hidden in Captchas. For example, a major invariant in the Captchaservice.org schemes was character pixel count, which was usually distinct among different characters but remained constant for the same character under different distortions.

The invariants in the 2007 Microsoft Captcha included objects' relative position patterns in image chunks, which played a major role in differentiating arcs and characters. The invariant in the Google Captcha we examined was white space (gaps) between characters. The Yahoo scheme's invariants included a correlation between text length and width in the challenge image, a very limited number of patterns for global shape (regular versus angular), and a fixed angle for image transformation. We exploited other invariants to break an earlier Yahoo Captcha.[12]

We found that invariants, once identified, were easy to exploit. Alarmingly, some of the attacks we developed for simplistic Captchas were widely applicable to ones carefully designed by major companies. For example, the CFS method we used to segment multiple Captchaservice.org schemes contributed significantly to our successful attacks on the Microsoft, Google, and Yahoo Captchas. Similarly, the pixel-count attack not only achieved surprising success recognizing individual letters in many simplistic schemes,[11] but it also helped us to break the Microsoft and Yahoo Captchas by differentiating between valid characters and random noise.[12]

Exploiting invariants is a classic cryptanalysis strategy. For example, differential cryptanalysis works by observing that a subset of pairs of plaintexts has an invariant relationship preserved through numerous cipher rounds.[13] Our work demonstrates that exploiting invariants is also effective for studying Captcha robustness.

The simple techniques we used, including the pixel-count method, CFS, and histogram analysis by projecting an image vertically, at a particular angle, or both vertically and horizontally (as in our previous work on an earlier Yahoo scheme[12]), are effective across the board. Therefore, we believe such techniques provide a good first set of tools for examining the strength of text-based schemes, in particular their segmentation resistance.

All the invariants described thus far are *pixel-level invariants*, so named because they are exploitable by image processing methods at the pixel level. In addition to pixel-level invariants, there are invariants inherent in Captcha text strings that can also be exploited—these *string-level invariants* are independent of any pixel-level features. Typically, the linguistic model a scheme employs to generate text strings creates string-level invariants. For example, in Microsoft's 2007 Captcha, the length of text strings embedded in the challenge image was constant, and this vulnerability in the design helped us to identify and segment connected characters.

The ultimate defense against attacks is to remove exploitable invariance by employing proper techniques such as randomization.

If a Captcha uses only dictionary words, this will create another string-level invariant, as the dictionary limits the choice of possible strings. Exploiting such a string-level invariant alone could lead to an attack. For example, when a Captcha only uses a small collection of English words, a dictionary attack could be highly successful.

String-level invariance can be exploited in other ways. For example, in attacking the Captchaservice.org text_image scheme, a pattern-recognition algorithm we designed earlier achieved only limited success in terms of character recognition, but a dictionary attack complemented the algorithm well to obtain a good overall result; that is, with the help of a dictionary, we exploited a weakness in the construction of challenge text strings—all were six-character English words—to successfully guess the remaining characters initially unrecognized by the pattern-recognition algorithm.[11] However, our new attack exploiting pixel-level invariance alone turns out to be more efficient.

In sum, the failures of the Captchas we analyzed are largely due to invariants at the pixel level, string level, or both. The ultimate defense against attacks like ours is to remove exploitable invariance by employing proper techniques such as randomization. For example, pixel-level invariants can be removed in the following ways. In the Captchaservice.org schemes, an option is to vary a character's pixel count in different challenges or simply make all characters have the same pixel count all the time. In the Yahoo Captcha, it is possible to introduce more types of global shape patterns and have them occur in random order, thus making it harder for computers to differentiate each type. And in the Google Captcha, an option is to simply remove the white space between characters (this might cause usability issues and thus requires careful treatment). To remove string-level invariants, an option for Captcha designers is to avoid using dictionary words and set a varied, unpredictable length for embedded text strings.

Although the Captchas we broke were better designed than earlier ones deployed from 2000 to 2004, Captcha robustness did not fundamentally improve. As a consequence of our work, Captchaservice.org ceased to offer its service, and Microsoft, Yahoo, and Google all modified their schemes.

There are two main explanations for the lack of robustness in the Captchas we analyzed. First, their design was almost exclusively based on research in computer vision, document recognition, and machine learning. However, our attacks did not rely on sophisticated, specialized algorithms. Instead, we applied our training in security engineering to identify critical vulnerabilities in each of the schemes, especially invariants at the pixel and string levels, and then design simple but novel methods to exploit those flaws. Second, a good Captcha requires striking the right balance between robustness and usability, which often have subtle influences on each other. This makes Captcha design a challenging task. Current understanding of how to do this is limited, and more research is needed.

As an aside, home-brewed Captchas seem to be a bad idea, just like home-brewed cryptography and security systems. Properly designing a Captcha requires considerable skill, and even experienced designers make mistakes. The devil is in the details. Any scheme should be carefully crafted and implemented by highly qualified people, and publicly and independently vetted before deployment.

Our experience suggests that Captchas will evolve in a process similar to cryptography, digital watermarking, and the like, with successful attacks leading to the development of more robust systems. Security engineers will play a major role in this process. C

Acknowledgment

We thank the anonymous reviewers whose comments helped improve this article.

References

1. L. von Ahn, M. Blum, and J. Langford, "Telling Humans and Computer Apart Automatically: How Lazy Cryptographers Do AI," *Comm. ACM*, vol. 47, no. 2, 2004, pp. 57-60.
2. M. Naor, "Verification of a Human in the Loop, or Identification via the Turing Test," unpublished manuscript, 1996; www.wisdom.weizmann.ac.il/~naor/PAPERS/human.pdf.
3. H. Yeend, "Breaking CAPTCHAs without Using OCR," blog entry, 2005; www.puremango.co.uk/cm_breaking_captcha_115.php.
4. G. Mori and J. Malik, "Recognising Objects in Adversarial Clutter: Breaking a Visual CAPTCHA," *Proc. 2003 IEEE Conf. Computer Vision and Pattern Recognition* (CVPR 03), vol. 1, IEEE CS Press, 2003, pp. 134-141.
5. G. Moy et al., "Distortion Estimation Techniques in Solving Visual CAPTCHAs," *Proc. 2004 IEEE Conf. Computer Vision and Pattern Recognition* (CVPR 04), vol. 2, IEEE CS Press, 2004, pp. 23-28.
6. K. Chellapilla and P.Y. Simard, "Using Machine Learning to Break Visual Human Interaction Proofs (HIPs)," *Advances in Neural Processing Systems 17* (NIPS 04), MIT Press, 2004, pp. 265-272.
7. K. Chellapilla et al., "Building Segmentation Based Human-Friendly Human Interaction Proofs (HIPs)," *Proc. 2nd Int'l Workshop Human Interaction Proofs* (HIP 05), LNCS 3517, Springer, 2005, pp. 1-26.
8. P.Y. Simard et al., "Using Character Recognition and Segmentation to Tell Computers from Humans," *Proc. 7th Int'l Conf. Document Analysis and Recognition* (ICDAR 03), vol. 1, IEEE CS Press, 2003, pp. 418-423.
9. J. Yan and A.S. El Ahmad, "Usability of CAPTCHAs, Or Usability Issues in CAPTCHA Design," *Proc. 4th Symp. Usable Privacy and Security* (SOUPS 08), ACM Press, 2008, pp. 44-52.
10. T. Converse, "CAPTCHA Generation as a Web Service," *Proc. 2nd Int'l Workshop Human Interactive Proofs* (HIP 05), LNCS 3517, Springer, 2005, pp. 82-96.
11. J. Yan and A.S. El Ahmad, "Breaking Visual CAPTCHAs with Naïve Pattern Recognition Algorithms," *Proc. 23rd Ann. Computer Security Applications Conf.* (ACSAC 07), IEEE CS Press, 2007, pp. 279-291.
12. J. Yan and A.S. El Ahmad, "A Low-Cost Attack on a Microsoft CAPTCHA," *Proc. 15th ACM Conf. Computer and Comm. Security* (CCS 08), ACM Press, 2008, pp. 543-554.
13. E. Biham and A. Shamir, *Differential Cryptanalysis of the Data Encryption Standard*, Springer, 1993.

Jeff Yan *is a lecturer in the School of Computing Science at Newcastle University, UK, where he is a founding research director of the Center for Cybercrime and Computer Security, and currently holds a visiting faculty post with the Department of Information Engineering, Chinese University of Hong Kong. His recent research includes systems security and human aspects of security. Yan received a PhD in computer security from Cambridge University. Contact him at jeff.yan@ncl.ac.uk.*

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