Using JPEG to Measure Image Continuity and Break Capy and Other Puzzle CAPTCHAs

Carlos J. Hernández-Castro • *Universidad Complutense de Madrid, Spain*

María D. R-Moreno and David F. Barrero • *Universidad de Alcalá, Spain*

Human interactive proofs (HIPs) are Internet security measures to avoid automated attacks. The authors focus on a new HIP: Capy CAPTCHA. Analyzing its design for flaws and weaknesses, they propose a low-cost, side-channel attack and extend their work to other image recomposition CAPTCHAs.

Abuse of free Web-based services abounds in today's cyberworld. A recent example is the possible abuse of cloud computing services (see www.wired.com/2014/07/how-hackers-hid-a-money-mining-botnet-in-amazons-cloud). Researchers have responded by tackling this problem in any number of ways. One approach proposed several theoretical methods to prevent such attacks,[1] using problems considered difficult for computers but easy for humans; these are called human interactive proofs (HIPs). A few years later, researchers improved this idea with a program to distinguish bots from humans, compiling a list of desirable properties that such a program should have. From these endeavors, the term *CAPTCHA* (Completely Automated Public Turing test to tell Computers and Humans Apart) was coined, and the problem improved its visibility in the academic world. During the 2000s, there was abundant research on new techniques[2,3] that could break text-based word-image CAPTCHAs. All of these attacks weren't really improvements on the state of the art in computer optical character recognition (OCR): instead, they typically made clever use of simple properties of the challenge images, either undoing part of the distortions or extracting enough information from them.[4] This, combined with some design flaws, allowed attackers the access they sought.

As a response, some companies rolled out much tougher HIPs; but they made them too difficult.[5] At the same time, many researchers started looking into the broader AI problem of vision and image analysis. One approach focused on CAPTCHAs based on labeling images (categorizing them), but a limitation of this approach is that it needs a large-enough database of labeled pictures to work. Luis von Ahn and Laura Dabbish proposed a way to build it by creating the "ESP game."[6] The site HotCaptcha.com was the first to propose using a large-scale, already human-labeled image database. Oli Warner suggested using photos of kittens to tell computers and humans apart.[7] Another proposal, the HumanAuth CAPTCHA, asked the user to distinguish pictures depicting either a nature-related image (such as a flower, grass, or the sea), or a human-generated one (for example, a clock, boat, or Big Ben). Assira used a similar approach, based on cat/dog images classification, using a large database.

Over time, each of these proposals have been broken. Recently, other types of image-based CAPTCHAs appeared that didn't need a database of labeled images, instead using those images' intrinsic properties. These approaches typically are based on some kind of image analysis,[8] such as face classification[9] or even cartoons.[10] And others are enhancing the typical OCR/text-based HIP,[11,12] much broken in Ian Goodfellow and his colleagues' work.[13] Unfortunately, there are no CAPTCHA design guidelines, apart from some

1089-7801/15/$31.00 © 2015 IEEE

basics,[3,14] that can be followed in future implementations.

Currently, various new proposals try to find a sweet spot between what's easy (and enjoyable) for humans, yet secure. Some are publicly presented by researchers and/or backed by companies, such as the one we examine here: Capy CAPTCHA. Here, we analyze Capy CAPTCHA for vulnerabilities from an attacker's point of view and present a side-channel attack that doesn't try to solve either the image recognition or shape recognition problems. We propose instead a very low-cost attack[15] based on a measure of the image's continuity, using the JPEG file size as the measuring instrument. We then test our approach on other puzzle HIPs, and discuss possible countermeasures against our attack.

Capy CAPTCHA

Capy CAPTCHA offers several types of CAPTCHAs on its webpage that basically fall into two categories: puzzle and text CAPTCHAs. We focus on the first, innovative one. Capy works by creating a simple puzzle, containing only one piece to place into an image. The user should drag and drop the puzzle piece into the correct location within the challenge image. The puzzle void within the challenge image isn't filled with just a random color; instead, it's filled with a portion from the same or another image. As we'll see, Capy sends to its server not only the puzzle piece's final position (where we drop it, within the challenge image) but also the log of the whole drag through the screen. This would allow them to further examine the complete pointer movement log in their servers.

In the production version, only one puzzle piece is present in each image. Nevertheless, in a video presentation, they show the possibility of more pieces per image. We'll focus on the production version, discussing later whether the found weaknesses extend to the multipiece version.

Capy presents an image of 400 × 267 pixels (the challenge image) and a puzzle piece of approximately 76 × 87 pixels — this size might vary, as the puzzle piece shape can change. It thus provides a security of $1/58320 \approx 0.0017$ percent against a random (brute force) attack. This result is pretty good and strong enough for a CAPTCHA.

Unfortunately, the first important design flaw of this CAPTCHA soon became evident: the puzzle piece only moves in discrete 10-pixel steps. This means a brute force attack would have a chance of 0.173 percent success against it. That appears a bit too high to put it into production.

This is a weak design idea that makes the CAPTCHA more susceptible to attacks. This in itself isn't an insurmountable problem for the idea behind this CAPTCHA, as it can be mitigated with bigger images, several puzzle pieces, and/or smaller step increments (5 pixels), among others. The major problem with this design decision is that it opens the door to attacks in which there's a noticeable difference between a correct solution and a solution 10 pixels away from it (and not just 1 pixel away).

Basis of the Side-Channel Attack

Capy and other image-recomposition HIPs are quite novel. To the best of our knowledge, there's no other attack that's directly applicable to them. There are other HIPs that are somewhat related, such as the ~~"Are you a human?"~~ CAPTCHA, broken by Manar Mohamed and his colleagues,[2] using an attack that learns the background and the moving images and creates a database of solutions by trial and error. PixelMap[16] is also a background-learning attack used against image classification HIPs, based on the fact that some weak image classification HIPs don't alter heavily reused backgrounds.

Most image-based CAPTCHAs have to do with labeling (or classifying) the images (such as Asirra or Imagination), or undoing a (typically progressive) distortion (such as a rotation). Attacks exist for many types of image classification CAPTCHAs, typically either using machine learning (ML) or using some side-channel information for classifying them.

If we look at the image recomposition CAPTCHA as a classification problem — that is, classifying correctly versus incorrectly solved challenges, the problem is that even if we can train a very good classifier for them (such as 95 percent accuracy), because there are more than 500 possible solutions, each challenge image will present approximately 28 false positives. Unless really extreme classification accuracy is possible, this doesn't seem like a feasible solution.

In our attack, we followed a different approach. We used an HTTP protocol analysis tool to understand and replicate the communications of the JavaScript client scripts with the Capy CAPTCHA server. In this phase, we learned that the communication protocol sends all the positions through which the piece travels while being dragged, encoded in base 32.

This would allow Capy to further examine the solution, detecting whether this pointer (mouse or finger) movement might correspond to a human, enhancing the human/bot discrimination. Unfortunately, after using the CAPTCHA, we detected that Capy isn't using this information further to discriminate between humans and bots (for example, using some ML clustering algorithm).

One important property of the correct solutions to the challenges of Capy is that the resulting images are more natural, in the sense that both colors and shapes are more continuous. The puzzle piece inside the challenge image must be visually disruptive for the human eye to be able to easily locate it. When it's covered

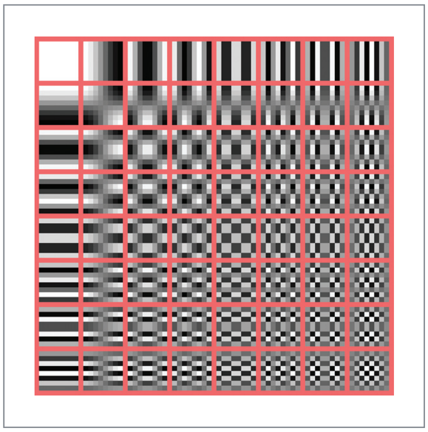

Figure 1. Representation of plane-waves corresponding to each discrete cosine transform (DCT) coefficient. Low-order terms represent the average color in the block, while the successive higher-order terms represent the strength of more and more rapid changes across the width or height of the block.

with the puzzle piece missing from the image, the resulting image (the original one) is better in terms of continuity — shapes around the figure are more continuous, as are colors and textures (shape repetitions).

This property of the correct solution leads us directly to think about the JPEG compression algorithm. It can compress any image but will do its best on photographs of realistic scenes with smooth variations of tone and color. JPEG defines a baseline of the standard that includes a compression algorithm, divided into the following steps:

- Transform the original image into a color space that separates luminosity from chrominance, best suited for the following steps.
- Downsample chrominance by averaging groups of adjacent pixels.
- Calculate the discrete cosine transform (DCT) of blocks of pixels for each channel (luminance and chrominance).
- Quantize each block of DCT coefficients using tables optimized for the human eye.
- Encode the resulting coefficients using a Huffman variable-length algorithm.

Knowing the basics about DCT is key to understanding how JPEG works. After the data are divided in blocks of 8 × 8 pixels, DCT converts the spatial image representation into a frequency map: the low-order terms represent the average color in the block, while the successive higher-order terms represent the strength of more and more rapid changes across the width or height of the block (see Figure 1). After this computationally intensive step, we obtain the coefficients of each frequency for the block.

Afterward comes the quantization step. Basically, we choose a quantization table, which has a value for each position of the 8 × 8 coefficient matrix. We apply integer division to each coefficient by the corresponding value in the table. Higher-end terms are divided by bigger numbers than the lower terms, giving more importance to the main color than to sudden changes, because human vision is much more sensitive to small variations over large areas than to high-frequency brightness variations. At the end of this process, we need fewer bits to represent these resulting coefficients, so we compress the bitstream in zig-zag order using a lossless algorithm.

The important aspect for us is that for JPEG, both light pattern regularity (texture) and color pattern regularity play a major role in the size of the resulting compressed image.

We conceived two attacks. The first one, *basic attack*, tries to find the correct answer by placing the puzzle piece in all possible grid locations, and for each resulting image, computing its JPEG size. The image that has a smaller size will be considered the correct one, and that position will be sent to the Capy server as our answer, encoded as the second and final position in the movement log.

The second one, *modal attack*, uses the JPEG compression algorithm with different quality settings (from 10 to 100, with 100 being the maximum). We then let each quality setting choose one *correct* solution, *voting* for it. The most-voted solution will be the chosen one. The idea behind this attack is, when our selection fails, to check whether it fails consistently (picking up most of the time the same wrong solution).

Experimental Results

Our initial estimation while designing the attack was that, in a good case scenario, using JPEG-size discrimination, we were going to be able to break Capy with an estimated success ratio of 3 to 5 percent. We thought that several problems, such as partial puzzle piece overlapping (thus reducing the image size with JPEG compression), small size variation (images in which the image chosen to fill in the puzzle void piece was already in harmony with the background's color and texture) and others would prevent this attack from getting a better result. We planned on this being a first stage of an attack, improved later with some additional information extracted from the images.

The attack is affected by the compression quality chosen for the JPEG algorithm, and therefore we tried with all compression levels, from 10 to 100 in 10-size steps. For each compression level, we performed 200 experiments. Then we proceed to launch 1,000 experiments with the best-performing setting (100). The full results of the experiments are available at https://github.com/carlos-havier/Capy-analysis.git.

Next, we describe the results of the two proposed attacks explained in the previous section, along with the reasons for the success or failure of the attacks in solving some of the challenges.

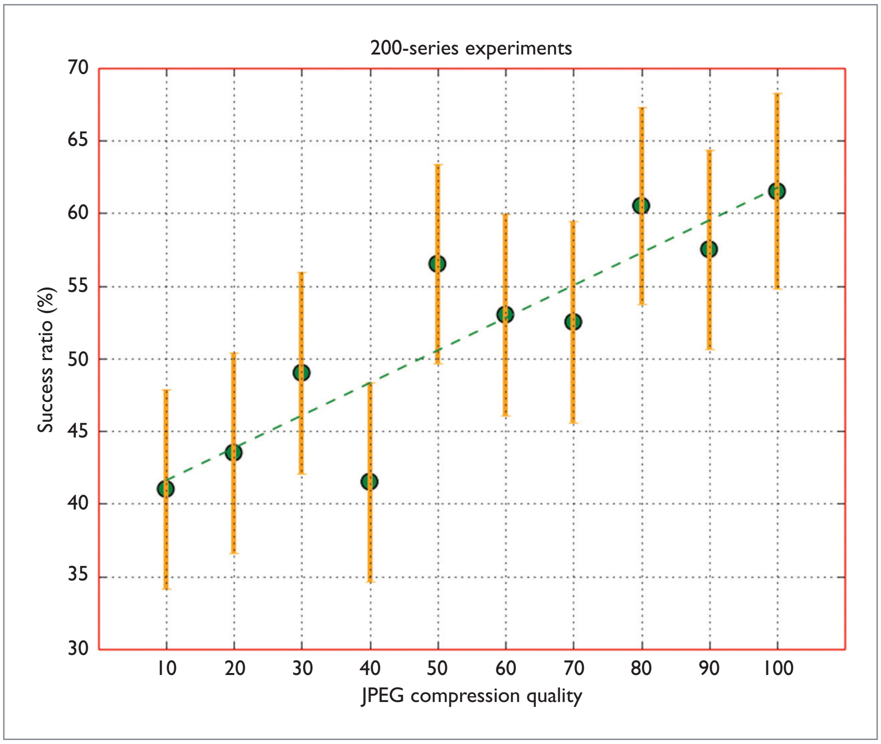

Figure 2. Percentage of success of the attack for different JPEG compression qualities, with 200 challenges in each quality setting. The first-order approximation is green, and error bars are orange.

Basic Attack Results

As JPEG image file size is dependent on the quality setting (that in turn affects the lossy compression algorithm), at first we performed our attack with different compression (quality) settings to discover which setting has a better success ratio. Because downloading a Capy challenge takes on average 4.33 seconds, we only performed this test for 200 challenges in each quality setting. We carried out an exhaustive search, using all quality settings from 10 to 100 in increments of 10.

After 4 hours and 42 minutes, we obtained the results shown in Figure 2, which depicts the attack's success ratio (correctly solved challenges, in percentage) depending on the JPEG compression quality ratio (from 10 to 100, again with 100 being the maximum). The dotted line represents the function's first order estimate. Even though the relation between a JPEG quality setting and the success ratio isn't linear, it's clear that there's a tendency to improve the attack's success ratio if we use a higher-quality JPEG setting. Also, the maximum compression quality was the one best able to differentiate the correct solution, with a 61.5 percent success ratio for these 200 experiments.

Then, we set our quality to a maximum and ran this attack for 1,000 experiments, to get a better estimate of the real maximum success ratio. In this case, our program was able to correctly solve the Capy CAPTCHA on 65.1 percent (±3.0 percent) of the occasions. This is a result two orders of magnitude above what's needed for a CAPTCHA to be considered broken (0.6 percent is enough[3]), which is quite extraordinary — especially for such a direct, low-cost attack.

Modal Attack Results

The idea behind this attack is that whenever the attack will fail and pick a wrong position, this wrong position isn't the same for different JPEG qualities. If that's indeed the case, then this attack might be able to show better results than our basic attack.

As this attack was much more time consuming, we limited it to 504 experiments. The results were disappointing, obtaining a success ratio of 56.5 percent. We noticed that when the attack failed, the JPEG's smaller size corresponded more than once to the same *wrong* position, thus rendering this approach seemingly useless.

We wondered whether the same voting scheme weighted by the JPEG image's quality (vote counting *quality*/10), would give better results. The results were better, correctly solving the Capy CAPTCHA 59.9 percent of the time; however, they were still inferior to the basic attack.

Results Analysis

Figure 3 shows some cases of failed solutions to the challenges. We named the challenge images from top to bottom as *rock, wood, and water, city at night, sandwiches,* and *lion*. In the *city at night* image, the puzzle void in the background has been filled with an almost plain color that happens to appear frequently in the background picture. The puzzle piece has two differentiated parts, with the biggest one being less-detailed. Our algorithm found that putting the highly detailed puzzle piece at the top part of the background leaves less information than if we put it in its correct place. The reason is that the algorithm is basically erasing highly detailed information. In the other challenges, we appreciate similar patterns: covering high detail parts of the background picture renders smaller images. Analyzing these failures, we noticed trends in them, depending on the properties of the background image used for each challenge, the filling of the puzzle void within the background, and also the filling of

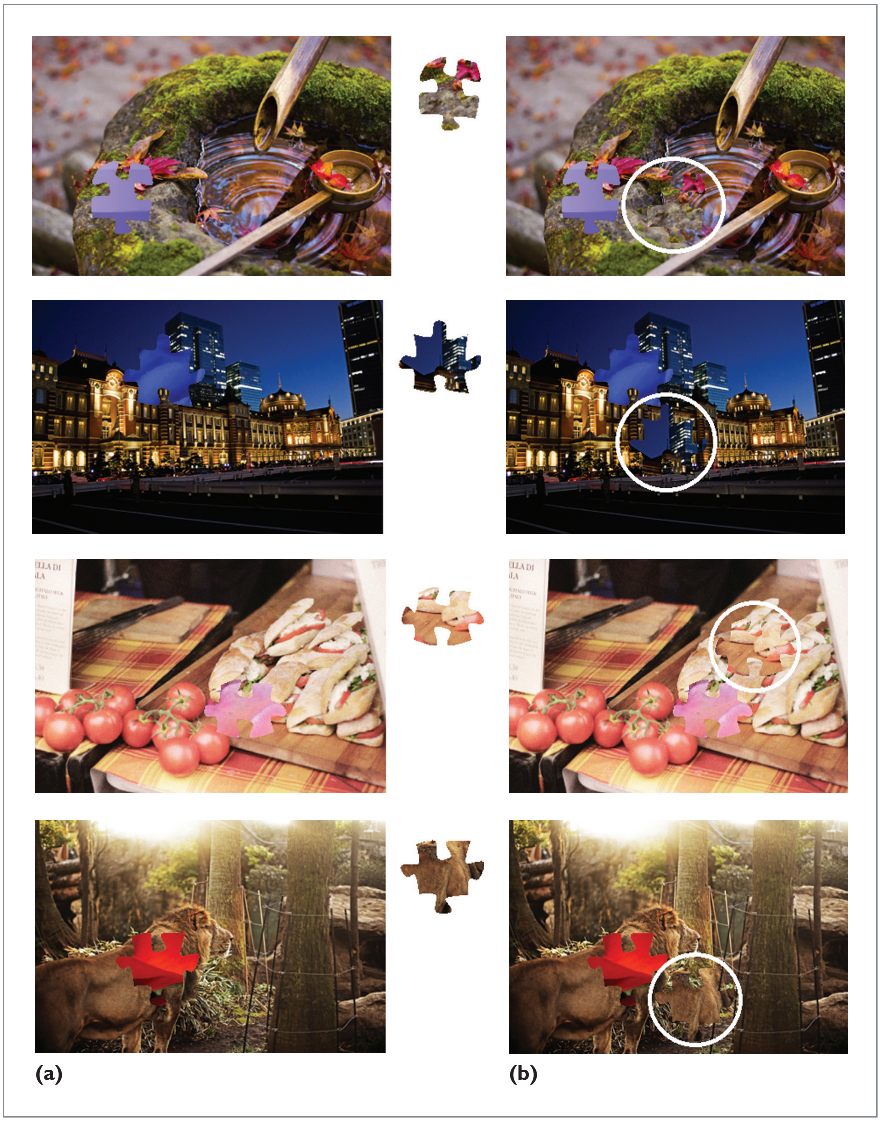

Figure 3. Wrongly solved challenges. (a) Challenge image and corresponding puzzle piece. (b) Piece placement by our algorithm (surrounded in white).

the puzzle piece. Then, we analyzed the four backgrounds available when we conducted these experiments. After reanalyzing our attack results, we got the results shown in Figure 4, where we can see that the background indeed affects the success ratio. Also, the JPEG compression settings differently affect each of the backgrounds.

Other CAPTCHAs Affected

The idea of presenting a puzzle piece HIP isn't entirely new; what's new is the way Capy CAPTCHA does it (using another image to fill the void, avoiding plain colors, and sometimes using similar textures). There are other CAPTCHAs currently in production that share this puzzle idea. Here, we describe the two that interested us most, along with the results we obtained using our attack with them.

KeyCAPTCHA

KeyCAPTCHA offers several versions that basically lie within two groups: *three* ones, with rectangular pieces without borders, draggable with 1-pixel precision, and the *magnetic* ones, with puzzle-like borders and with an n-pixel step movement, thus making it easier for humans to solve.

The voids in the background where the puzzle pieces should go are filled with plain white color. White areas are good for a small JPEG size, so our attack tends to respect them — that is, it doesn't put any puzzle pieces on top of these areas. For this reason, we shuffled two ideas: using a filter, so we only consider a valid position if the puzzle piece covers at least 90 percent white pixels; or adding some high-frequency noise to the white pixels.

After experimenting with KeyCAPTCHA, we were able to detect some repetitions in the background images. In the first 25 attempts, we only saw 20 different images, and 20 different ones in the next 25 attempts, with 8 of them seen previously in the first batch. We can use the mark and recapture method[17] to estimate the image library size as $N = 20 \times 20/8 = 50$, which isn't enough for a production CAPTCHA, given that once one background is solved, this information can be used to correctly solve it again.

Because we wanted to validate our attack — not the KeyCAPTCHA design limitations or internals — we decided to download 50 challenge images locally and try them using our attack. Of these 50 challenges, 18 were served as 3-puzzle-piece challenges, and the rest with 2 puzzle pieces. Ten challenges were correctly solved — that is, with a 20 percent success ratio on passing KeyCAPTCHA overall. Figure 5 shows some example results. Notice that even when KeyCAPTCHA presents non-continuity around the puzzle pieces (using borders and antialiasing), our attack is successful; this is a good indication of the JPEG's ability to measure continuity (and repetition) even with images that aren't fully continuous.

Garb CAPTCHA

The standard Garb CAPTCHA comes with 62 images. Garb divides one random image into four equal parts, and mixes their order. The user must interchange the puzzle pieces (using drag and drop) to their correct position.

Even though the Garb CAPTCHA takes away one pixel line in the border of each one of the four subimages, this isn't a major problem for our attack. A CAPTCHA with only 4! = 24 possible answers per image can be considered already too weak (4.16 percent brute-force attack success).

For our attack, for each library image, we created a random permutation. To solve it, we considered all possible 24 permutations, and we chose the one with the smallest JPEG file size (for the maximum JPEG image quality).

Our attack was able to correctly solve 61 of the 62 images in the standard library. We performed 1,000 sample tests, and obtained a 98.1 percent success ratio. The image that's incorrectly solved is consistently solved incorrectly, as we would expect (the first image in Figure 6). Note that in an improved version of our attack (using a larger image library), it would be easy to try the second, third, or fourth best solution to each failed challenge, once we learn that the first best solution of our JPEG attack is incorrect. This is possible because there's an order of *goodness* associated to each permutation (its JPEG file size).

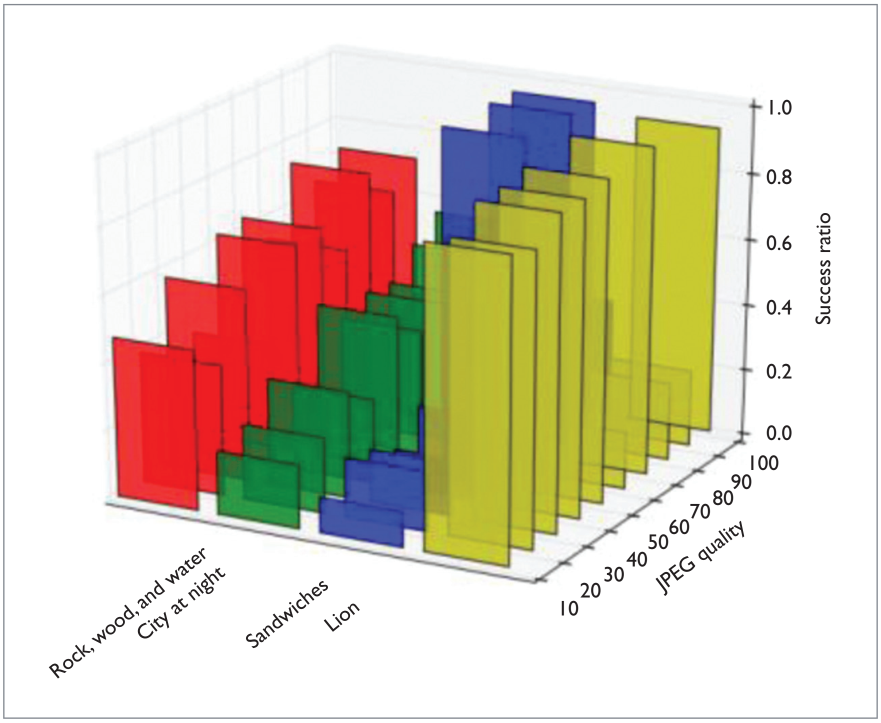

Figure 4. Success ratio per image type and JPEG quality setting. The background affects the success ratio.

JPEG versus Other Image Analysis Techniques

During our initial analysis of Capy, we used other image-processing techniques, including color histogram analysis, edge/shape detection (to detect within the image a shape similar to the puzzle piece's shape) and fast Fourier transform (FFT) analysis. These techniques were dependent on several parameters, and didn't perform as well. It remains to be seen if a more tailored technique would have been successful —for example, if there was some fuzzy detection of a puzzle shape within the image — but the cost of an attack is an extremely important aspect of IT security. Also, it remains to be seen whether image analysis techniques more tailored to Capy would generalize as well to other image-recomposition HIPs.

Possible Countermeasures against Our Attack

Now, let's discuss possible ways to enhance the security of Capy and related HIPs.

Broader Solution Space

A broader solution space would make the HIP more resilient to brute-force attack. Enlarging the solution space can mean using an *x*-pixel grid for possible solutions. This makes it much harder to be solved by humans, as placing the puzzle image (almost) exactly in its position isn't an easy task with a mouse or a finger.

A person might think that we can check in the Capy server that the solution given is close enough to the perfect solution (within a certain distance). But once the attacker determines that an *x*-pixel distance to the correct solution is accepted, she can create a grid in *x*-pixel steps and try only those solutions, if her attack algorithm provides her with a continuous measure of "goodness" of a solution.

We tested what would happen if we also considered puzzle positions that are almost correct — that is, with distances of 1 pixel to the correct one, 2 pixels, and so forth. We ran this experiment for all previously correctly solved challenges. Even with the solutions as near to the correct as 1 pixel distance, they can be detected as wrong by their larger JPEG size, for all occasions. Figure 7 gives a mean of the relative JPEG file size when the puzzle piece is put 1 or more pixels away from the correct position.

Challenge Prefiltering

Our basic attack doesn't correctly solve all the challenges presented. Thus, we can prefilter the challenges

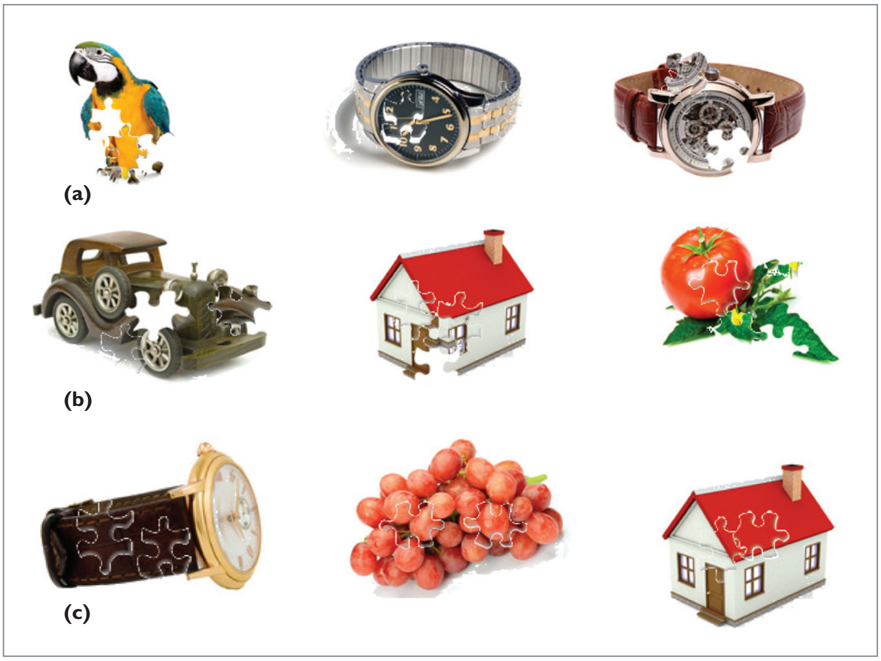

Figure 5. Examples of results, where we used our attack on KeyCAPTCHA puzzles. (a) Incorrectly, (b) partially, and (c) completely solved challenges.

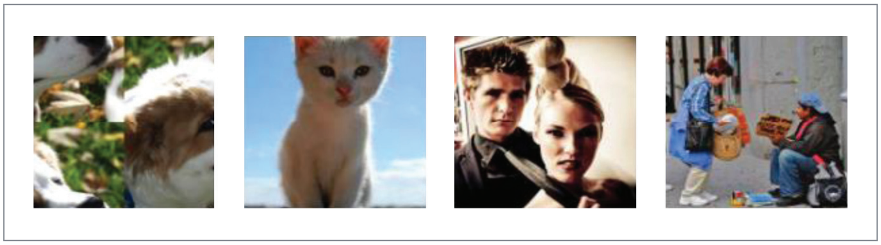

Figure 6. Several computed solutions for the Garb CAPTCHA. The first one is the only one that our algorithm fails to solve.

served by the Capy server, offering only those challenges that aren't solved by this attack. This might work only if we can't try the next-best solution after a failure. Yet this will make it resilient only to this particular attack.

Another type of prefiltering is possible based on selecting the puzzle piece void to be filled using another image that has similar patterns and colors to the real puzzle piece, according to the lessons we previously learned (in looking at our experimental results' analysis). These countermeasures might also be able to prevent other attacks based on image continuity, but the resulting CAPTCHA might not be as user-friendly.

Bigger Background Image Library and Transformations

An attacker shouldn't be able to recognize a background image. Using a much larger library of images, and also using some image-distortion algorithms so that the images are similar pixel-per-pixel may help. Angle transforms, local and global image warps, light changes, and color changes, along with added low-frequency noise will make it much harder for an algorithm to rebuild the original background. This could also apply to KeyCAPTCHA and Garb CAPTCHA.

Client Interaction Analysis

The client-side JavaScript libraries of Capy send to the server not just the final position where the user locates the puzzle piece, but the whole log of the mouse or finger drag. We can apply ML clustering algorithms to try to find clusters of typical drag movements for different sets of directions (either complete or partial). We can also add timing data. This is an example of security through obscurity, so we can't consider this as the main security discriminant.

Several Puzzle Pieces

The idea of using several puzzle pieces is already present in KeyCAPTCHA, which uses two or three puzzle pieces per challenge. If the general success ratio of the attack is X percent, a three-piece HIP would have a success rate of $(X/100)^3 * 100$ percent. For our basic Capy attack (65.1 percent), this would mean a success ratio of 27.5 percent for three pieces, which is still a very successful attack.

In this work, we presented our low-cost, side-channel attacks that break (bypass) Capy CAPTCHA. We also showed how these attacks, with minor modifications, can break other image recomposition CAPTCHAs that use the same puzzle ideas, albeit in slightly different ways.

To create a more robust defense, we provided some ideas to make Capy CAPTCHA and other HIPs resilient to such attacks. Although theoretically it should be possible to strengthen a CAPTCHA's abilities by correcting design flaws, the tradeoff is that some of these corrections could also compromise its usability — perhaps to a level that would make it overly difficult for humans.

Acknowledgments

Carlos Hernández-Castro thanks Лена Соколова, Carmen Castro, Declan J. H.-O., Женя Соколова, and Julio César H.-C. for their help and support. The JCLM project supported this work through grant PEII-2014-015A.

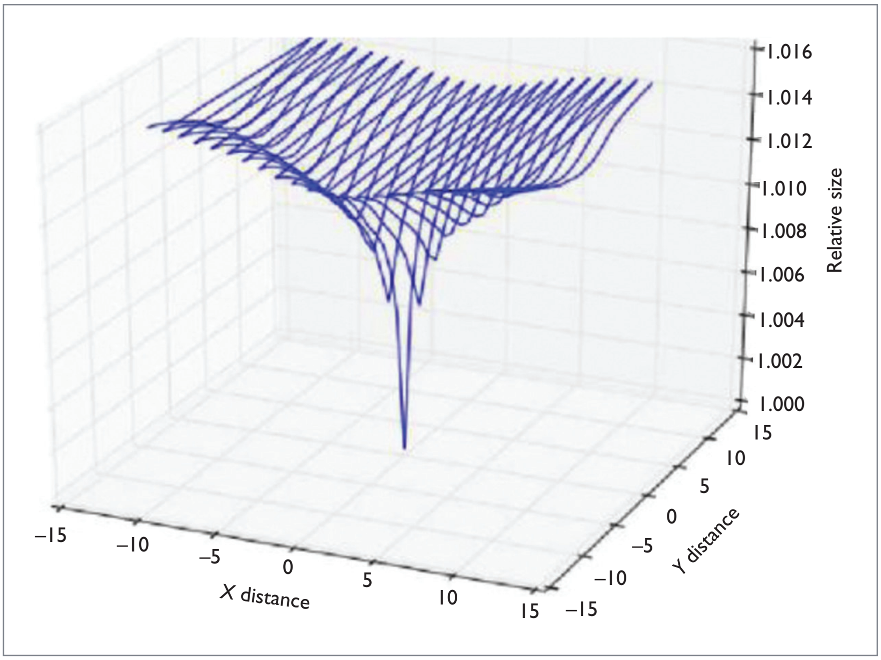

Figure 7. JPEG size proportions at different distances from the correct solution.

References

1. M. Naor, "Verification of a Human in the Loop or Identification via the Turing Test," paper, 1996; www.wisdom.weizmann.ac.il/~naor/PAPERS/human.pdf.
2. M. Mohamed et al., "A Three-Way Dissection of a Game – CAPTCHA: Automated Attacks, Relay Attacks, and Usability," *Proc. 9th ACM Symp. Information, Computer, and Comm. Security*, 2014, pp. 195–206.
3. B.B. Zhu et al., "Attacks and Design of Image Recognition CAPTCHAs," *Proc. 17th ACM Conf. Computer and Comm. Security*, 2010, pp. 187–200.
4. E. Bursztein, M. Martin, and J. Mitchell, "Text-Based CAPTCHA Strengths and Weaknesses," *Proc. 18th ACM Conf. Computer and Comm. Security*, 2011, pp. 125–138.
5. C.A. Fidas et al., "On the Necessity of User-Friendly CAPTCHA," *Proc. ACM Conf. Human Factors in Computing Systems*, 2011, pp. 2623–2626.
6. L. von Ahn and L. Dabbish, "Labelling Images with a Computer Game," *Proc. ACM Conf. Human Factors in Computing Systems*, 2004, pp. 319–326.
7. O. Warner, "KittenAuth," *ThePCSpy*, blog, 2009; www.thepcspy.com/kittenauth.
8. S. Vikram, Y. Fan, and G. Gu, "SEMAGE: A New Image-Based Two-Factor CAPTCHA," *Proc. 27th Ann. Computer Security Applications Conf.*, 2011, pp. 237–246.
9. D. D'Souza, P.C. Polina, and R.V. Yampolskiy, "Avatar CAPTCHA: Telling Computers and Humans Apart via Face Classification," *Proc. IEEE Int'l Conf. Electro/Information Technology*, 2012, pp. 1–6.
10. T. Yamamoto, T. Suzuki, and M. Nishigaki, "A Proposal of Four-Panel Cartoon CAPTCHA," *Proc. IEEE Int'l Conf. Advanced Information Networking and Applications*, 2011, pp. 159–166.
11. M.A. Kouritzin, F. Newton, and Biao Wu, "On Random Field Completely Automated Public Turing Test to Tell Computers and Humans Apart Generation," *IEEE Trans. Image Processing*, vol. 22, no. 4, 2013, pp. 1656–1666.
12. S.A. Alsuhibany, "Optimising CAPTCHA Generation," *Proc. 6th Int'l Conf. Availability, Reliability and Security*, 2011, pp. 740–745.
13. I. Goodfellow et al., "Multi-Digit Number Recognition from Street View Imagery Using DCNNs," talk, *Int'l Conf. Learning Representations*, 2014; www.youtube.com/watch?v=vGPI_JvLoN0.
14. P. Baecher et al., "CAPTCHAs: The Good, the Bad, and the Ugly," *Proc. Sicherheit*, 2010, pp. 353–365.
15. C.J. Hernández-Castro, M.D. R-Moreno, and D.F. Barrero, "Side-Channel Attack against the Capy HIP," *Proc. 5th Int'l Conf. Emerging Security Technologies*, 2014, pp. 99–104.
16. C. Fritsch et al., "Attacking Image Recognition CAPTCHAS," LNCS 6264, *Proc. 7th Int'l Conf. Trust, Privacy and Security in Digital Business*, 2010, pp. 13–25.
17. G.A.F. Seber, *The Estimation of Animal Abundance and Related Parameters*, Blackburn Press, 1982.

Carlos J. Hernández-Castro is a PhD candidate in the Computer Engineering Department at the Universidad de Alcalá, Spain. His research interests include IT security, machine learning, and the fields that bring them together, especially CAPTCHAs. Hernández-Castro has an MSc in computer security from the Universidad Complutense de Madrid, Spain. Contact him at chernandez@ucm.es.

Maria D. R-Moreno is an associate professor in the Computer Engineering Department at the Universidad de Alcalá, Spain. Her research focuses on AI, in particular AI planning and scheduling, evolutionary computation, machine learning, and intelligent execution applied to real applications such as aerospace, robotics, e-learning, or CAPTCHAs. R-Moreno has a PhD in computer science from the Universidad de Alcalá. Contact her at mdolores@aut.uah.es.

David F. Barrero is a senior lecturer in the Computer Engineering Department at the Universidad de Alcalá, Spain. His research interests include genetic programming and evolutionary algorithms, educational data mining, lightweight cryptography, and electronic government. Barrero has a PhD in computing from the Universidad de Alcalá. Contact him at david@aut.uah.es.