Side-channel attack against the Capy HIP

Carlos Javier Hernández-Castro[1], María D. R-Moreno [2] and David F. Barrero [2]

[1] Universidad Complutense, Madrid, Spain.
chernandez@ucm.es

[2] Departamento de Automática. Universidad de Alcalá, Madrid, Spain
{mdolores, david}@aut.uah.es

***Abstract*—One of the first approaches to proposed to prevent automated attacks on Internet were the Human Interactive Proofs (HIPs). Since their invention, a variety of designs have been proposed, yet most of them have been successfully attacked. In this paper we focus on a new HIP, based on a puzzle solving scheme, created to increase both security and usability: the Capy CAPTCHA. We have analyzed its design, finding some important flaws. Based on them, we propose a low-cost, side-channel attack. Initial results show that the attack is able to break Capy with a 61% success ratio.**

I. Introduction

The abuse of web-based services is still a common activity on the Internet. These abuses take form by creating computer programs able to exploit those services much faster and many more times than humans. The first approach to solve this problem proposed several theoretical methods to prevent such attacks [9], under the general idea of using problems considered to be hard for computers, but easy for humans. Instances of these problems were used as remotely delivered tests, also called Human Interactive Proofs (HIPs). A few years later, CMU Researchers improved this idea with a program to tell apart bots from humans. They listed some desirable properties such a program should have. The term CAPTCHA was coined by them, and the problem improved its visibility in the academic world. During the 2000s decade there was abundant research on new techniques ([7], [12]) enabling the breaking of text-based word-image CAPTCHAs. All of these attacks made clever use of some very simple properties of the challenge images [2]. Added to some design flaws, this allowed attackers to *read* them. As a response, some companies hardened HIPs making them all too difficult [5]. At the same time many researchers started considering the broader AI problem of vision and image analysis ([10], [11], [4], [3]), but their proposals had either usability or practical problems, or were broken ([6], [12]). Currently there are new CAPTCHA proposals that try to find a sweet spot between being easy (and enjoyable) for humans, yet secure. Most of these proposals are presented by programmers on their own, or small programming teams. Others are proposed (sometimes by researchers) and never get to be implemented. Finally, others are publicly presented by researchers and/or backed by companies, like the one we focus on in this paper. Unfortunately there is no such thing as a CAPTCHA design guideline, apart from some basics ([1], [12]), and it is not uncommon to notice some recurrent design flaws in most of them.

In this article we introduce the Capy CAPTCHA, and proceed to analyze it from an attacker's poing of view. We present a side-channel attack that does not try to solve neither of the Artificial Intelligence problems used as a foundation for the CAPTCHA: it does not solve either the image recognition problem or the shape recognition problem in which the strength of this CAPTCHA is based. The proposed attack is a very low cost one, easy to implement by any attacker.

The structure of the paper is as follows. Section II presents Capy CAPTCHA, analyzing it from an attacker point of view. Next, in section III we present the rationale of our attack based on the found vulnerabilities and the attack in detail. After, section IV shows the preliminary results. Finally, conclusions are outlined, including some reflexions about the posibility to use our attack with other image-based CAPTCHAs that could be vulnerable to it.

II. Capy CAPTCHA

In this section we describe the general features and design structure of Capy in order to understand the proposed attack.

A. General Features

Capy CAPTCHA started in 2010 as an academic research project at Kyoto University and turned into a company in 2012. It has been quite well praised both in awards and in the press.

In the Capy web-page, their authors offer several types of CAPTCHA that basically fall into two categories: puzzle CAPTCHAs and text CAPTCHAs. We will focus on the first one, which is the truly innovative proposal, and not on the second. Many works have been done previously on text-based CAPTCHAs, and most, if not all can be considered either susceptible to attack or too difficult to solve even for humans. In the rest of this paper, when we refer to the Capy CAPTCHA, we will implicitly mean the Capy puzzle type. Capy works by creating a simple puzzle, one in which there is only one piece to place into an image (Fig. 1). The user should drag and drop the puzzle piece into the correct location within the challenge image.

It so appears that the Capy designers have put some effort into its security. For instance, the puzzle void within the challenge image is not filled with just a random color; instead it is filled with a portion from another image. Not only that, in some (but not all) of the challenges, the other image has a color and texture similar to the challenge image and puzzle piece. According to Capy designers "[it] contains complicated details, edges and other factors which prevent bots from cracking the code"[1].

[1]Retrieved from http://www.capy.me/demo_puzzle, on June 14th 2014.

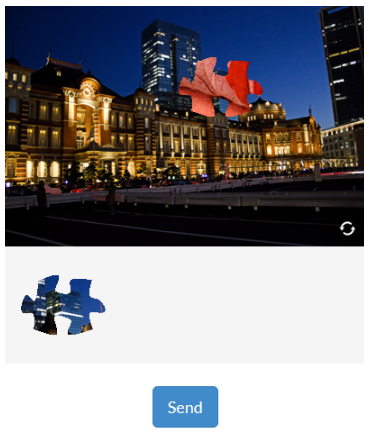


Figure 1: Puzzle CAPTCHA by Capy.

In the production version that appears on the Capy webpage, only one puzzle piece is present in each image. Nevertheless, in a video presentation of their idea, they show the possibility of more than a single puzzle piece per image. But we will focus on the production version.

B. *Design analysis*

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.

The solution space is thus roughly $400 - 76 \times 267 - 87 = 58.320$ possible answers (possible positions), and with no further information, will provide a security of $\frac{1}{58.320} = 0,0017\%$ against a random (brute force) attack. This is a decent result, strong enough for a CAPTCHA.

Unfortunately, after playing with this CAPTCHA, the first important design flaw was evident: the puzzle piece only moves in discrete 10-pixels steps. This means a brute force attack would have a chance of $\frac{1}{\frac{400-76}{10} \times \frac{267-87}{10}} \approx \frac{1}{32 \times 18} \approx 0,173\%$ success against it.

This is a weak design idea, that makes the CAPTCHA much more susceptible to attacks. The major problem with this design decision is that it opens the door to other attacks with a noticeable difference between a correct solution and a solution 10 pixels away from it (and not just 1 pixel away).

We have 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. They are encoded in base 32, adding the character *x* for separation, and are sent to the server as a string. For example, one possible solution string would be

```
ax8exax84xax7qxkx7gxkx76xkx...ixax1ixkx18x
```

This would allow them to further examine the solution sent once in their servers, detecting whether this pointer (mouse, finger) movement corresponds to a human, and thus enhance the human/bot discrimination. However, after some playing with the CAPTCHA, we detected that this is not the case. Capy CAPTCHA does not accept us to send only the final puzzle piece position, returning *False* (test not passed) if we just sent it. While playing with the CAPTCHA, we noticed that reducing the drag log size (*jumping* over positions) did not seem to affect its marking. Finally, we learned that sending just two positions, the initial one (always the same) and the final position (where the puzzle piece goes) gets *True* (test passed) marks every time the solution position was correct.

Capy CAPTCHA is not using this information to further discriminate humans and bots, maybe using some ML clustering algorithm would help to improve it. We think that not trying to take advantage of this information is also a minor weakness in its design.

III. The side-channel attack proposal

One fundamental property of the correct solutions to the challenges of this CAPTCHA is that the resulting images are complete and *more natural*, in the sense that both colours and shapes are more *continuous*. Compared to the challenge image, the solution image has more color and structure repetition.

When the puzzle piece *void* is covered with the puzzle piece missing from the image, the resulting image is better in terms of *continuity* - shapes around the puzzle figure are more continuous, as are colours and textures (shape repetitions).

A. *JPEG*

This crucial property of the correct solution leads us to think in the JPEG compression algorithm, that we will introduce now. To better understand our attack, it is important to understand how the JPEG baseline algorithm compresses the images, and thus what makes one image *more compressible* than another.

The Joint Picture Experts Group (JPEG) is an ISO combined committee established in 1987, product of the join of two research groups working in compression and transmission of images (a group from the CCITT, and the PEG from ISO). The result, the JPEG standard, is a toolkit of image compression techniques, that might be tailored and adjusted to the needs of the user [8]. JPEG is also primarily a lossy method of compression, not as other compressors like RLE, LZW, or formats like BMP, GIF, etc.

Lossy compression schemes throw *not important* data away during encoding. JPEG was designed specifically to discard information that the human eye cannot easily see. As an example, small changes in intensity (light vs. dark) are perceived better than similar changes in colour. That is why JPEG is more lossy when encoding colour.

One important aspect is that JPEG was designed to compress continuous-tone images of real-world subjects: photographs, video stills, or anything resembling natural subjects. Black and white documents, line art, etc. do not get the best results after JPEG compression, with visible artifacts.

Another important aspect is that the user can *tune* the quality of the JPEG encoder using a parameter called the *quality setting*, typically variable between 1 and 100, 1 given the biggest compression but worst quality.

JPEG defines a baseline or minimal subset of the standard that any JPEG implementation should include. The baseline uses an encoding scheme based on the Discrete Cosine Transform (DCT) as a compression step. The JPEG compression algorithm is divided into the following steps:

- transform the original image into a colour space that separates luminosity from chrominance, and is best suit for the following steps
- downsample chrominance by averaging groups of adjacent pixels
- apply the DCT to blocks of pixels, in each channel (luminance, chrominance)
- quantize each block of DCT coefficients using tables optimized for the human eye
- encode the resulting coefficients using a Huffman variable length algorithm

Key to understand how JPEG works is to understand the DCT. After the data is divided in blocks of 8×8 pixels, DCT converts the spatial image representation into a frequency map: the low-order terms represent the average colour in the block, where the successive higher order terms represent the strength of more and more rapid changes across the width or height of the block (Figure 2). The highest term represents the strength of a cosine wave alternating from maximum to minimum at adjacent pixels. After this computationally intensive step, we obtain the *coefficients* of each frequency, for the block.

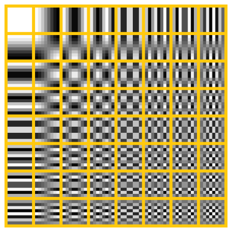


Figure 2: Representation of plane-waves for each DCT coefficient.

Once we have these coefficients, we represent them with less than complete accuracy. This is the quantization step. Basically, we choose a quantization table, which has a value for each position of the 8×8 frequency coefficients. We calculate the integer division of each coefficient by the corresponding value in the table. The higher-end terms are *quantized* more heavily than the lower terms, thus giving more importance to main color than to sudden changes. This is so because human vision is much more sensitive to small variations over large areas than to high-frequency brightness variations, so magnitudes of the high-frequency components are stored with a lower accuracy. At the end of this process, we need less bits to represent these resulting coefficients, so we are compressing the bitstream.

Separate quantization tables are used for luminance and chrominance data. Also, the tables used depend on the *quality setting* that the user chooses (higher compression meaning tables with bigger values). The ISO JPEG committee developed some tables, on which other groups have worked. The quantization table chosen for a particular image is saved in the bitstream, so the JPEG decompressor just reads it for every image.

The final bitstream, or resulting data from each block, run in a zig-zag order, is compressed with a lossless algorithm (a variant of a Huffman encoding).

The important aspect for us is that for JPEG, both light pattern regularity (texture) and colour pattern regularity play a major role in the size of the resulting compressed image. If an image is very regular in the sense that there is small variation in adjacent pixels, and the variation is somehow previsible, DCT and Huffman will work perfectly towards compressing it. If, in the other end, an image is random noise, JPEG will fail to compress it much, while loosing quality in the process.

As we recall from earlier, a fundamental property of the correct solutions to the challenges of this CAPTCHA is that the resulting images are complete and *more natural*, in the sense that both colours and shapes, are more *continuous*. This is so because the puzzle piece in the correct position brings the image back to its original, which will typically have less high-frequency information than the same image with one part abruptly substituted by another image.

For this reason, we will choose the image that, once compressed, uses the smallest size. Surprisingly, we will not need to use any other information from the image.

B. Attack proposal

We have created a program in Python that downloads a challenge, composed of the challenge image (that includes a portion of another image inside with the shape of the puzzle piece) and the puzzle piece. They are embedded together in the same *PNG* image. Our program, once the challenge is downloaded, tries to find the correct answer by placing the puzzle piece in all possible locations (within the 10×10 step grid), and for each resulting image, computing its size once compressed using JPEG. The image that has a smaller size will be considered the correct one, and thus that position of the puzzle piece will be sent to the Capy server as our answer, as the second and final position in the movement log.

The Capy server returns in turn an HTML answer including *True* only in the case that the correct position was sent. We log the response image sent (with the puzzle piece in the *guessed* position), the position itself (as x, y coordinates), the answer from the Capy server (that tells us if the answer was correct or not), and other details. We programmed it using the *PIL Image Library* of the Python language for image manipulation and compression, including the merge of the puzzle piece into the background, and the computation of the JPEG algorithm. We also used the libraries *httplib* and *urllib2* for HTTP communications with the Capy CAPTCHA server.

IV. Experimental results

Our initial estimation while designing this attack was that, in a good case scenario, using just this JPEG-size discrimination, we would be able to break Capy CAPTCHA with an estimated success ratio of over 3% to maybe 5%. 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 of colour and texture with the background) and others would prevent this attack from getting a better result. We planned for this to be the first stage of an attack, later improved with some additional information extracted from the images.

As the 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 for which one it seemed to have a better success ratio. As it takes on average 4.33 seconds to download a Capy challenge, we performed this test only for 200 challenges in each quality setting. We performed an exhaustive search, using all quality settings from 10 to 100 in steps of 10.

After 4:42 hours, we obtained the results in Fig. 3, that depicts the success ratio of the attack (number of correctly solved challenges, in percentage) depending on the JPEG compression quality ratio (setting from 10 to 100, the maximum). The dashed line is the second-grade approximation. Even though the relation between a JPEG quality setting and the success ratio is not linear, it is clear that there is a tendency to improve the success ratio of the attack if we use a higher JPEG quality setting. The maximum quality (minimum compression) was the one able to differentiate the correct solution best, with a 61.5% success ratio for these 200 experiments.

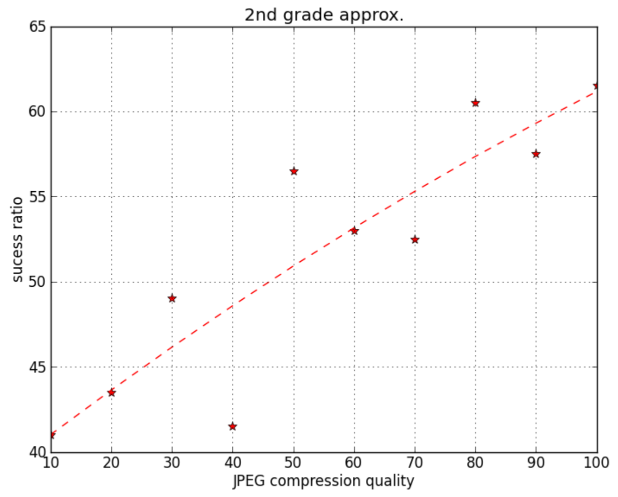


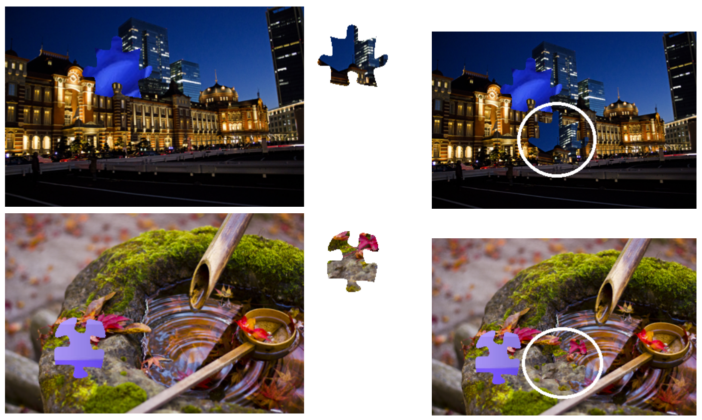
Figure 3: Success ratio by JPEG compression quality.

A. Results analysis

It is interesting to observe where our basic attack succeeds and where it fails, to better understand why it works so well, as well as figure out ways to make it work worse, making the CAPTCHA more resilient.

In table I we present some cases of failed solutions to the challenges. The first one (*city at night*) is particularly interesting: the puzzle void in the background has been filled with an almost plain colour, one that happens to appear frequently in the background picture. The puzzle piece has two differentiated parts, the bigger one also a low-detail one. Our algorithm finds that putting this piece on top of a high-detailed part of the background produces a smaller image (less information) than if we put it in its correct place. The reason for this is that the algorithm is basically *erasing* high frequency (detailed) information. In the other challenges, we appreciate similar patterns: covering high detail parts of the background picture renders smaller images.

Table I: Wrongly solved challenges.



After examining carefully other results, we note that this behaviour typically happens when the pattern/colour used for filling the void is a low frequency one (plain colour, with little changes). It also adds to it when the puzzle piece taken from the image does not contain many details (high frequency image data), but some parts of the challenge image do, and in high concentrations. Thus, images with both areas of very high detail and very low detail are more prone to this. This gives us a possible way to enhance the security of these types of CAPTCHAs, as we will discuss in section V-B.

V. Possible mitigation meassures

In this section we discuss possible ways to enhance the security of the Capy CAPTCHA, and similar CAPTCHAs based on puzzles, or some kind of challenge based on image recomposition.

First, we comment the possibilities of using a broader solution space, thus making it more resilient to brute force attack. Next, we present how to use our attack to filter out *weak* challenges, and discuss its possible drawbacks. Then we comment the benefits (and proper ways) to present a bigger image library, thus trying to avoid attacks that, once a particular challenge is solved, can recognize and solve it again. Finally, we analyze whether adding puzzle pieces (instead of just one) might or not be a proper solution to strengthen these CAPTCHAs.

A. Broader solution space

One possible solution we might think about would be to enlarge the solution space, and approximate it to the maximum possible, given the dimensions of the background image (a $width \times height$ solution space). The drawback is that this would make it much harder to be solved by humans, as placing the puzzle image exactly in its position is not an easy task with a mouse nor with a finger in a touch device.

One might think that we can check, once in the Capy server, that the solution given is *close enough* to the perfect solution, within a distance. The problem with this idea is that once the attacker determines that a x-pixel distance to the correct solution is accepted, then again, she can create a grid in x-pixel steps and try only those solutions.

A bigger image, and a smaller allowed distance to the correct solution, would force the attacker to try more positions, at least making the attack a bit more expensive in computational resources.

B. *Challenge pre-filtering*

As our attack does not correctly solve all the challenges presented, the Capy designers can use this to pre-filter the challenges served by their server, that is: offer only those challenges that are not solved by this attack. Note that this will not require a major modification to their CAPTCHA, and would make it resilient to our attack.

The negative point of this solution is that it will make it resilient to the attack we present here, but only to this particular attack.

Another type of pre-filtering is possible, this time, based on selecting images with very differentiated areas in terms of detail level. Also, when the puzzle piece void to be filled comes from an image that has similar patterns and colours to the real puzzle piece, and little detail (low frequencies). It also helps if we pick up a piece of the image with low detail for the puzzle piece. In section IV-A we learnt that we can, for instance fill the puzzle void in the background challenge with portions of the same image (with little detail), so colours and textures are alike.

These countermeasures might not only be able to prevent the attack we present here, but possibly other attacks based on image continuity. The problem with this is that the resulting CAPTCHA might then not be as user friendly, and a study on usability would be indicated in case of using this filtering measure. Also, tt remains to be seen if these filters would render the CAPTCHA weaker against other types of attacks.

C. *Bigger image library*

Having a small image library for the backgrounds of the challenges is a major problem. Here, the meaning of *small* is any number that we can download, store and analyze programmatically with a computer. Instead, what we need is a really large number of possible backgrounds, at least from the point of view of a computer algorithm.

Using a much larger image library, and also using some image distortion algorithms so the images are not pixel-per-pixel similar (in a way that the distortion is impossible for any other algorithm to undo, even if given several samples of the same image once distorted) may also help.

D. *Several puzzle pieces*

It is possible to argue that the idea mentioned by Capy designers of using more than one puzzle piece at a time for a single challenge might improve its security. This will be very possibly the case, but remains to be seen to what extent. For example, if just placing one puzzle piece (of, say, three pieces) in its correct position gives the correct properties to the resulting image - which in the case of our attack can be described as *continuity*, then the attack can solve first one piece, then the second, then the third. If the general success ratio of the attack is $X\%$, the three-piece success rate would be $\frac{X^3}{10^6}\%$. For example, for our basic attack with a success ratio of 65.1%, for a three piece puzzle it would solve it 27.5% of the time, being still a very successful attack.

VI. Conclusions

The Capy CAPTCHA, although seemingly well designed, user friendly, and praised by press and prices, presents important design weaknesses. They are so important that they completely compromise its security.

We have presented a low-cost, side-channel attack that is easy to be implemented, and that is able to break (bypass) this CAPTCHA with a 61% success ratio. We find thus that a simple metric is able to correctly classify the solutions to this CAPTCHA.

Capy CAPTCHA is a great CAPTCHA in terms of usability. Unfortunately, that usability is related to some extent to some weak design decisions, that render the CAPTCHA extremely weak against attacks like the one presented here. It is not straightforward to improve the security of this CAPTCHA.

We feel that this low-cost attack, with minor modifications, will be able to bypass some other image recomposition CAPTCHAs or *puzzle* CAPTCHAs that use the same puzzle ideas - although in slightly different ways. Among these are KeyCAPTCHA and the Garb CAPTCHA. We look forward to examining the security of those CAPTCHAs, and extend our experiments to them to validate this possibility.

Acknowledgements

The first author wants to especially thank Declan, Женя and Julio.

References

[1] Paul Baecher, Marc Fischlin, Lior Gordon, Robert Langenberg, Michael LÃijtzow, and Dominique SchrÃűder. CAPTCHAs: The Good, the Bad, and the Ugly, 2010.

[2] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based CAPTCHA Strengths and Weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, pages 125–138, New York, NY, USA, 2011. ACM.

[3] Ritendra Datta, Jia Li, and James Z. Wang. Imagination: a robust image-based captcha generation system. In *MULTIMEDIA '05: Proceedings of the 13th annual ACM international conference on Multimedia*, pages 331–334, New York, NY, USA, 2005. ACM.

[4] Jeremy Elson, John R. Douceur, Jon Howell, and Jared Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security*, pages 366–374, New York, NY, USA, 2007.

[5] Christos A. Fidas, Artemios G. Voyiatzis, and Nikolaos M. Avouris. On the Necessity of User-friendly CAPTCHA. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 2623–2626, New York, NY, USA, 2011. ACM.

[6] Philippe Golle. Machine learning attacks against the Asirra CAPTCHA. In *Proceedings of the 5th Symposium on Usable Privacy and Security, SOUPS 2009, Mountain View, California, USA, July 15-17, 2009*, ACM International Conference Proceeding Series. ACM, 2009.

[7] Manar Mohamed, Niharika Sachdeva, Michael Georgescu, Song Gao, Nitesh Saxena, Chengcui Zhang, Ponnurangam Kumaraguru, Paul C. van Oorschot, and Wei bang Chen. Three-Way Dissection of a Game-CAPTCHA: Automated Attacks, Relay Attacks, and Usability. *CoRR*, abs/1310.1540, 2013.

[8] J.D. Murray and W. VanRyper. *Encyclopedia of graphics file formats*. O'Reilly Series. O'Reilly & Associates, 1996.

[9] M. Naor. Verification of a human in the loop or Identification via the Turing Test, 1996.

[10] Luis von Ahn and Laura Dabbish. Labeling images with a computer game. In ACM Press, editor, *CHI '04: Proceedings of the 2004 conference on Human factors in computing systems*, pages 319–326. ACM Press, 2004.

[11] Oli Warner. Kittenauth. http://www.thepcspy.com/kittenauth, 2009.

[12] Bin B. Zhu, Jeff Yan, Qiujie Li, Chao Yang, Jia Liu, Ning Xu, Meng Yi, and Kaiwei Cai. Attacks and design of image recognition CAPTCHAs. In *Proceedings of the 17th ACM conference on Computer and communications security*, CCS '10, pages 187–200, New York, NY, USA, 2010. ACM.