Breaking a Visual CAPTCHA

Anonymous Author(s)
Affiliation
Address
email

Abstract

Visual CAPTCHAs have recently become a practical mainstream online Turing test method to counter malicious or unauthorized access by automated scripts. However, naïve implementations of this system can create vulnerabilities that can be easily exploited with minimal computational cost. In this paper, I will demonstrate a simple attack against a popular open source web blog plugin, PHP CAPTCHA.

1 Introduction

Web sites offering services to a large public userbase are prime targets for spammers who advertise products and scams in an effort to entrap users. To increase posting efficiency, rather than manually post the content by hand, many spammers employ automated computer scripts to crawl through the web and post content on multiple sites en masse. Consequently, many site administrators have chosen to employ a Turing test approach to differentiate between legitimate users and bots [1].

The most popular form of the CAPTCHA, a *Completely Automated Public Turing test to tell Computer and Humans Apart*, is a visual image containing warped text characters with cluttering, distortion, and noise. The premise for this approach assumes that humans are more capable in decoding the text than the automated scripts.

Figure 1: Google CAPTCHA example

Modern CAPTCHAs, such those used for Google, are implemented specifically to thwart straightforward optical character recognition (OCR) techniques employed by typical document scanning software. Indeed, these systems are so designed in attempts to dissuade even legitimate software developers from bypassing their systems. However, there are no strict guidelines for designing CAPTCHA systems, which leads to several vulnerable implementations. In this paper, I will investigate one such implementation, PHP Captcha [2], and show how a straightforward combination of techniques from computer vision, graphics, and machine learning can be used to attack such a system.

2 Related Work

A recent security research paper [3] surveyed a variety of modern private CAPTCHA implementations and presented an anti-CAPTCHA implementation that effectively defeated most simple CAPTCHA ranging from eBay, Digg, and CNN. The paper outlined the weaknesses and common vulnerabilities of these systems and provided suggestions for improvement. My paper will attempt to verify some of the techniques described in this paper against a modern open-source CAPTCHA implementation. My implementation will also differ by using a computer graphics approach to segmentation, and a low-cost feature learning technique with K-means [9] for use in classification.

2 PHP Captcha

The CAPTCHA implementation I will attack in this paper is a popular open-source web page add-on running on PHP [2]. This software appears frequently as a plugin in web blogs to protect comment sections of pages from automated spammers. Prior to the rise in popularity in ReCaptcha [4], this CAPTCHA has seen much popularity in WordPress blogs.

The feature set of PHP Captcha allows site administrators to customize the distortion and clutter intensity of the CAPTCHA, the background, font, the color of the text and line, as well as the thickness and number of the lines. The software also provides an audio CAPTCHA, which has its own share of exploits [5], but this paper will focus on attacking the visual CAPTCHA.

3 Preprocessing

Images have to be preprocessed to remove unwanted clutter from the characters prior to segmentation. This clutter can interfere with processes further down in the pipeline, as they introduce extraneous artifacts that can then be mistaken as part of the character body. This section describes a few PHPCaptcha setups that require additional work.

4.1 Foreground/Background Segmentation

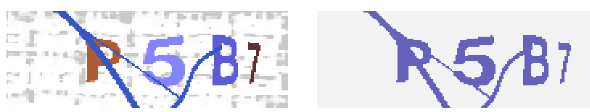

Figure 2: Input image (left) and two K-means color segments (right)

Many websites tend to customize the CAPTCHA with a background image of their own. Some may even have a variety to improve the aesthetics surrounding such a mundane setup. However, most backgrounds tend to be fixed and reused. This allows modeling the background for background subtraction. Simple backgrounds with pixel colors significantly different from the text in the foreground also can be segmented via K-means color segmentation with merely two or three clusters. Figure 2 shows segmentation the result with 2-means clustering.

4.2 Noise Removal

Figure 3: Input image (left) and denoised output (right)

Another scheme to avoid segmentation is via the addition of random salt and pepper noise with the same color as the text. This is vulnerable to an iterative removal of pixels that contribute poorly to the average energy within its surrounding patch until there are no further changes. This is essentially a crude execution of the Gibbs algorithm [6]. Removal of larger segments that have been collected can be done with an edge-preserving median filter. Note that very thin lines on the image would also be removed during this denoising.

4.2 Line Removal

Figure 4: Image vulnerable to line removal by color segmentation

Lines with thickness close to width of the character body are difficult to distinguish from the characters themselves. However, PHPCaptcha allows for the user to modify the color templates of the lines and text, which leads to a vulnerable case where the text, background, and line can be fully separated through once again by K-means clustering by color.

Figure 5: Vulnerable PNG input, line removal, and repair with inpainting

As of version 3.0 of the software, PHPCaptcha is also found to have an unpatched vulnerability due to the palette coding of PNG images generated by PHP graphics library. For vanilla CAPTCHA images presented in PNG format without a custom background, regardless of the amount of clutter by lines or noise or signature text, it is possible to completely separate the text and clutter by merely comparing the palette IDs in the image encoding. As in Figure 5, the lines can be removed entirely due to the palette ID over the pixels on the line (and over the text) being different from the palette ID of the text. A simple XOR operator over the values generates the center image.

Removing the lines creates gaps within the image that may impair segmentation and recognition pieces later in the pipeline. Consequently, it is necessary to fill in the missing pixels through inpainting [8].

4 Character Segmentation

Tightly spaced text create difficulties in segmentation due to difficulty distinguishing which given pixels belong to a character, especially if the characters are actually touching. In particular, it is difficult to segment by horizontal spacing alone. Here I introduce an alternative to segmentation inspired from computer graphics: seam-carving [7].

A seam is a path of connected pixels traveling from the top to the bottom of the image. Pixels must neighbor in location by exactly one pixel smoothly without ever becoming horizontal or a sharp changing in direction. These requirements ensure seams travel from the top of the image to the bottom as steeply and quickly as possible while spatially close.

Seams are produced differently from the seam-carving paper in that a penalty function is used in place of an energy function: Seams that pass through a pixel of any character (a black-colored pixel) accumulate penalty costs while passing through the background (a pixel of any other color) does not. Consequently, the seam will prefer to go around a character rather than through it when possible.

Figure 6: Clockwise from top left: input image, seam costs, seam carve, and binning

This property can be used for segmentation. A seam passing in between characters can act as the segmentation boundary that separates two clusters: the left side and the right side. Given that the background subtraction has already labeled that all characters have a black color, the set intersection of these black pixels and the pixels split by the boundaries produce the desired clustering.

The seam boundaries can then be taken as bins. Empty bins can be pruned to return only those that contain characters. Notice that with sufficiently small bins this approach can for used for CAPTCHAs that vary the number of characters. This approach also allows forced segmenting between touching characters, which still have a smaller accumulated penalty compared to seams crossing a full character body.

3 Crop and Resize

Pixels clustered via binning are then cropped and rescaled to a suitable template image of 40x60 pixels each. These letters are then grouped together for the training set. Since CAPTCHA by design fails if even one character is misidentified, this problem can be interpreted as standard OCR problem for individual characters after removal of clutter.

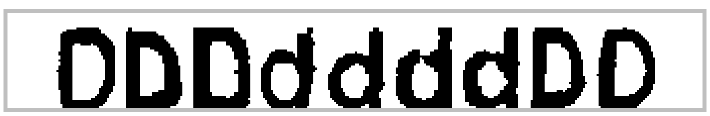

Figure 7: Typical characters samples extracted for a single letter.

PHPCaptcha by default installation settings do not distinguish between capital or lowercase variations of the letters. Therefore, we can train these together as a single unit, and thus reducing the loading of training, and the need to gather a sufficiently large training collection for each individual letter. This also allows combing similar characters like S, s, B, B, and W, w without additional costs in storage.

4 Feature Learning and Classifier Training

Applying techniques from Coate's [9], it is possible to generate basis or filters that represent parts of the collected image patches over the training set. The features on the left have PCA whitening applied rather than ZCA due to implementation restrictions of the library during the time of coding. However, we can observe quantitatively the Gabor-like resemblance of the features compared to the specific patches on the right image.

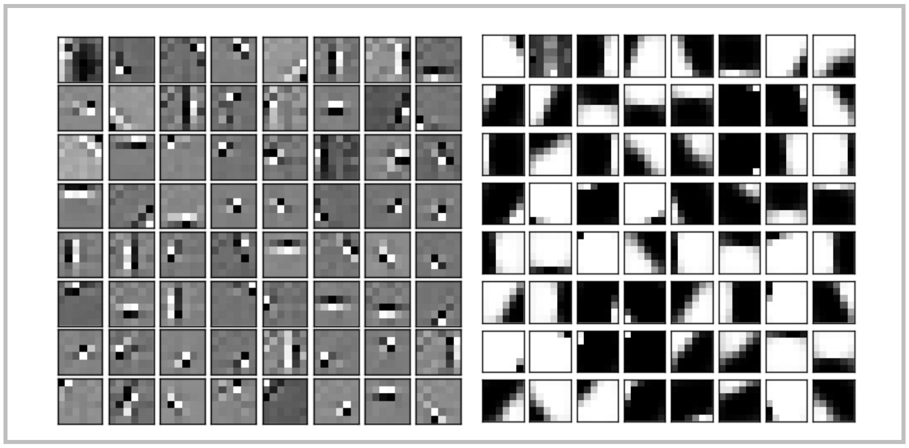

Figure 7: Dictionary features learned with K-means on the CAPTCHA training set with whitening PCA (left) and without (right).

5 Results

Using the code for PHP Captcha, 500 training samples and 200 test samples were generated. For this paper, the samples were generated with straightforward background subtraction, noise reduction, and line removal. Preprocessing and segmentation is applied for each sample.

Unfortunately, the implementation of the system through Python and the Scikit-learn library has not been fully successful due to the work-in-progress nature of the Coate's recent paper. As a result, only the bases can be computed via K-means but without feeding it into a convolution neural network or a simple linear SVM for training and classification. However, performance as indicates in Coates' paper suggest relatively high performance for classification to the extent of higher-end RBMs. We can expect similar performance here.

6 Limitations and Further Work

Line removal becomes particularly problematic when the colors cannot be segmented, and the image format is not vulnerable, and the lines are as thick as the characters. In this case, we can attempt to estimate the location of some of the lines by applying a Progressive Probabilistic Hough Transform [10] to the image gradient. In the particular OpenCV implementation, this returns endpoints of line segments each supported by the hypothesis that points in between contribute to that straight line or curve. However, wavy lines cutting across many characters horizontally prove difficult to remove without damaging the characters, and many line segments can remain undetected due to the characters themselves contributing as "noise" in Hough space.

In this scenario, the current approach would rely heavily on the seam carving segmentation to make the least costly cut around or through these lines. However, it may be possible to combine information from Hough estimations by reducing the cost of cutting through pixels on or close to a suspected line segment.

References

[1] M. Naor. Verification of a human in the loop or Identification via the Turing test. Available electronically: http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps, 1997.

[2] D. Phillips. SecurImage PHP Captcha 3.0. http://www.phpcaptcha.org, 2011.

[3] E. Bursztein, M. Martin, and J. Mitchell. Text-based CAPTCHA strengths and weaknesses. In Proceedings of the 18th ACM conference on Computer and communications security (CCS '11). ACM, New York, NY, USA, 125-138, 2011.

[4] Google Inc. reCAPTCHA. http://www.google.com/recaptcha

[5] A. Anonymous. Web App Security blog. http://www.idontplaydarts.com/2011/05/exploit-phpcaptcha-securimage/, 2011.

[6] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images*. Journal of Applied Statistics, 20(5):25–62, 1993.

[7] S. Avidan and A. Shamir, "Seam carving for content-aware image resizing," ACM Trans. Graph., vol. 26, no. 3, 2007.

[8] A. Telea, “An image inpainting technique based on the fast marching method,” In Proceedings of Journal of Graphics Tools, vol.9, no.1, pp.25–36, 2004.

[9] An Analysis of Single-Layer Networks in Unsupervised Feature Learning Adam Coates, Honglak Lee and Andrew Ng. In NIPS*2010 Workshop on Deep Learning and Unsupervised Feature Learning.

[10] J. Matas, C. Galambos, and J. Kittler, "Robust Detection of Lines Using the Progressive Probabilistic Hough Transform", presented at Computer Vision and Image Understanding, pp.119-137, 2000.