The Robustness of Hollow CAPTCHAs

Haichang Gao, Wei Wang, Jiao Qi,
Xuqin Wang, Xiyang Liu
Institute of Software Engineering
Xidian University, Xi'an, Shaanxi
710071, P.R.China
hchgao@xidian.edu.cn

Jeff Yan
School of Computing Science
Newcastle University, UK
Jeff.Yan@ncl.ac.uk

ABSTRACT

CAPTCHA is now a standard security technology for differentiating between computers and humans, and the most widely deployed schemes are text-based. While many text schemes have been broken, hollow CAPTCHAs have emerged as one of the latest designs, and they have been deployed by major companies such as Yahoo!, Tencent, Sina, China Mobile and Baidu. A main feature of such schemes is to use contour lines to form connected hollow characters with the aim of improving security and usability simultaneously, as it is hard for standard techniques to segment and recognize such connected characters, which are however easy to human eyes. In this paper, we provide the first analysis of hollow CAPTCHAs' robustness. We show that with a simple but novel attack, we can successfully break a whole family of hollow CAPTCHAs, including those deployed by all the major companies. While our attack casts serious doubt on the viability of current designs, we offer lessons and guidelines for designing better hollow CAPTCHAs.

Categories and Subject Descriptors

K.6.5 [**Management of Computing and Information Systems**]: Security and Protection—*authentication, unauthorized access*

General Terms

Security, Human Factors.

Keywords

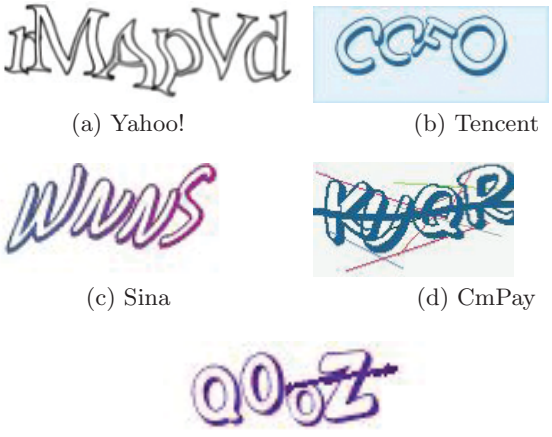
CAPTCHA; Convolutional Neural Network; Graph search; Security

1. INTRODUCTION

Since its invention, CAPTCHA has been widely deployed for defending against undesirable and malicious bot programs on the Internet [20]. The most widely used CAPTCHAs are text-based schemes [24], which typically require users to solve a text recognition task.

A good CAPTCHA should be friendly for humans to solve but hard for computers. It turns out that this balance between security and usability is hard to achieve. So far, many text CAPTCHAs have been broken, including those deployed by major companies such as Microsoft, Yahoo! and Google [8, 23]. However, as predicted in [23], CAPTCHAs are going through the same process of evolutionary development, just like cryptography and digital watermarking, with an iterative process in which successful attacks lead to the development of a next generation of systems.

In the last couple of years, Hollow CAPTCHAs have emerged as one of the latest text-based designs. They have been deployed by major websites such as Yahoo!, Baidu, Sina, Tencent and the online payment system (CmPay) of China Mobile, each serving tens of millions users on a daily basis. A main feature of such hollow CAPTCHAs is to use contour lines to form connected characters (see Figure 1) with the aim of improving security and usability simultaneously, as it is hard for state-of-the-art character recognition programs to segment and recognize such connected characters, which are however easy to human eyes.



(a) Yahoo! (b) Tencent

(c) Sina (d) CmPay

(e) Baidu

Figure 1: Hollow CAPTCHAs.

Given the high profiles of the companies that have deployed hollow CAPTCHAs, it is of practical relevance to examine the robustness of such hollow schemes, i.e. their resistance to automated attacks, which is an important se-

CCS'13, November 04 - 08, 2013, Berlin, Germany.

http://dx.doi.org/10.1145/2508859.2516732 .

curity property. On the other hand, as Hollow CAPTCHAs represent a new type of text scheme, it is also of academic interest to study their design and security. To our best knowledge, no such studies are available in the literature.

In this paper, we provide the first analysis of the robustness of hollow CAPTCHAs. We show that with a novel and generic attack, we can successfully break a whole family of hollow CAPTCHAs. The success rates of our attack on Yahoo!, Tencent, Sina, CmPay and Baidu schemes are 36%, 89%, 59%, 66% and 51%, respectively. It takes just seconds for our attack to break each of the schemes on a standard desktop computer. Therefore, our attack imposes a realistic threat, which might be misused by adversaries.

As a variety of design features are used in the hollow schemes, we also pinpoint which features contribute to security, and which do not. Our analysis provides a set of guidelines for designing hollow CAPTCHAs, and a method from comparing security of different schemes. We also discuss how to design better hollow CAPTCHAs.

This paper is organized as follows. Section 2 discusses related work. Section 3 provides an overview of five representative hollow CAPTCHAs. Section 4 gives an overview of our attack, and Section 5 describes the attack in details. Section 6 shows our attack results. Section 7 discusses lessons we have learnt, and how to design better hollow CAPTCHAs. Section 8 concludes the paper.

2. RELATED WORK

Moni Naor [15] first discussed the notion of Automated Turing Tests, but did not providing a formal definition or concrete designs. The first practical Automated Turing Test was developed by Alta Vista [12] to prevent bots from automatically registering web pages. This system was effective for a while but then was defeated by common OCR (Optical Character Recognition) technology.

In 2003, Mori and Malik [13] used sophisticated object recognition algorithms to break Gimpy (which used clutter interference) and EZ-Gimpy (which used texture backgrounds), achieving a success rate of 33% and 92%. Moy et al [14] developed distortion estimation techniques to attack EZ-Gimpy with a success rate of 99% and four-letter Gimpy-r with a success rate of 78%. In 2005, Chellapilla et al [19] successfully broke a range of CAPTCHAs with a success rate ranging from 4.89% to 66.2%. Early attack efforts also include the PWNtcha project [3].

In 2006, Yan and El Ahmad [22] broke most visual schemes provided at Captchaservice.org, a publicly available web service for CAPTCHA generation, with a success rate of nearly 100% by simply counting the number of pixels of each segmented character, although these schemes were all resistant to the best OCR software on the market. New character segmentation techniques for attacking a number of text-based CAPTCHAs were developed by Yan and El Ahmad [23] in 2008, including the earlier mechanisms designed and deployed by Microsoft, Yahoo! and Google, and these have achieved a segmentation success rate of 92% against Microsoft CAPTCHA. In 2010, Yan's team broke the text-based CAPTCHAs that depend partially on the Gestalt Perception principle by merging black and shared white components to form individual characters [9].

In 2011, Bursztein et al. showed that 13 CAPTCHAs on popular websites were vulnerable to automated attacks [7], but they achieved zero success on harder schemes such as reCAPTCHA and Google's own scheme. In the same year, Yan's team published an effective attack on both of these schemes [8]. The CAPTCHA using moving-images in NuCaptcha which provided users with sloshing characters was analyzed by Xu et al in 2012 [21]. Table 1 lists some text CAPTCHAs that were successfully attacked.

We note that hollow CAPTCHAs have never been discussed in the literature prior to our current paper, and that they are distinct from other text-based CAPTCHAs discussed to date.

3. HOLLOW CAPTCHAS: POPULAR REAL WORLD SCHEMES

We choose to study 5 CAPTCHAs listed in Figure 1, which we consider represent the state of the art of hollow CAPTCHA designs, for two main reasons.

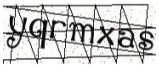
First, these schemes have been deployed by popular real world websites, and have an impact on hundreds of millions of users. For example, Yahoo!, Baidu, Sina and Tencent are all among the largest websites in the world; CmPay [2] is the online payment system of China Mobile, which enjoys a 70% share of the domestic mobile service market in China and has nearly 700 million users. Yahoo! use their hollow CAPTCHA in services such as their email system and password helper [6]. Sina use their scheme for services such as Weibo [4], the most popular micro blog platform in China with about 500 million users (also a Chinese equivalent to Twitter). The Baidu scheme is used for their account registration and maintenance [1]. With 600+ million users, Baidu is the largest search engine in China and one of the largest social network platforms. The Tencent scheme is used in their security center service [5].

Second, these schemes represent a range of different designs, with a variety of design features. For example, some schemes use interference arcs (e.g. CmPay and Baidu); others do not (e.g. Yahoo!, Tencent and Sina). Yahoo! uses character strings of a varied length, while others all use a fixed string length. Some schemes (e.g. Yahoo!) intentionally introduce a significant variation in the thickness of hollow portions across characters, and even in a single character; others do not vary this thickness much and it is more or less uniform.

A common feature is that all characters are presented as hollow objects. In general, hollow schemes appear to be a clever idea. First proposed by Google, Crowding Characters Together (CCT) has been widely adopted. This standard security mechanism for text CAPTCHAs improves security but has usability issues. For example, when characters are crowded together too much, confusing character pairs will appear and it is hard for people to recognize them [24]. However, hollow schemes allow characters connected or overlapped with each other, but maintain a reasonable usability. In a sense, this approach can be regarded as a clever variant of the CCT segmentation-resistant mechanism.

Since there are only (or mainly) randomly-generated contours in each CAPTCHA, it becomes difficult to detect each character's features using standard technologies. Common character recognition methods, such as template matching and other feature-based algorithms, that are effective in recognizing solid characters, are inapplicable to hollow characters. Moreover, characters' contour lines may connect or overlap with each other to prevent segmentation. When

Table 1: CAPTCHAs which have been attacked successfully

Origin	Samples	Success rate	Comments
Captchaservice.org [22]		92%	Weaknesses: constant pixel count of the same character, non-textured background, constant colors, no perturbation, only capital characters used
Clubic [3]		100%	Weaknesses: constant font, no rotation, no deformation, aligned glyph, constant background, weak color variation, weak perturbation
Slashdot [3]		89%	Weaknesses: constant font, no deformation, constant colors, weak perturbation, weak color variation
Microsoft [23]		60%	Weaknesses: easy to tell random arcs from valid characters by examining characteristics, easy to locate connected characters by 'even cut'
Google [8]		62%	Weaknesses: non-textured background, constant colors, no perturbation
NuCaptcha [21]		36.3%	Weaknesses: weak overlapping, constant color of 'codewords', short length of 'codewords', constant font
Megaupload [9]		78.3%	Weaknesses: shared components are white, non-textured background,constant font, short and fixed length, no perturbation

there are interference arcs, contour lines will be cut through or otherwise interrupted. Presumably, this will make it even harder for computers to recognize hollow characters.

We also note that two main segmentation-resistant mechanisms, namely CCT and interference arcs, have never been used simultaneously in a single CAPTCHA design before. However, some hollow CAPTCHAs apply the two mechanisms together, without introducing serious usability concerns in our experience.

4. OUR ATTACK: AN OVERVIEW

The key insight behind our attack is the following. We can manage to extract from hollow CAPTCHAs character strokes or components, and convert them to solid ones. This is not straightforward, as some contour lines are broken; we need to automatically repair them first. Furthermore, standard methods like Color Filling Segmentation (CFS, introduced in [23]) will pick up not just character strokes or components, but also those that do not belong to any character and which we call noise components. Therefore, it is essential to differentiate between legitimate character strokes and noise components automatically.

On the other hand, character strokes extracted this way are not linked with each other. Instead, they are scattered around. We implement a convolutional neural network (CNN) [11, 16, 18] as our recognition engine. CNN provides partial invariance to translation, rotation, scale and deformation. It extracts successively larger features in a hierarchical set of layers. Besides, as a stable and fast classifier, CNN has been successfully applied to character recognition; it takes only milliseconds to classify a character image.

Next, just like piecing together a jigsaw puzzle, we use our CNN recognition engine to try different combinations of adjacent strokes. With a graph search algorithm that we have designed, we can find the most likely combination as the right result with a good success rate.

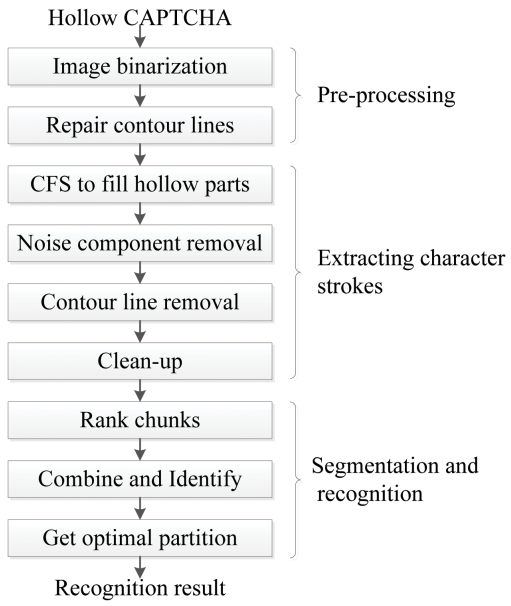


Figure 2: The pipeline of our attack.

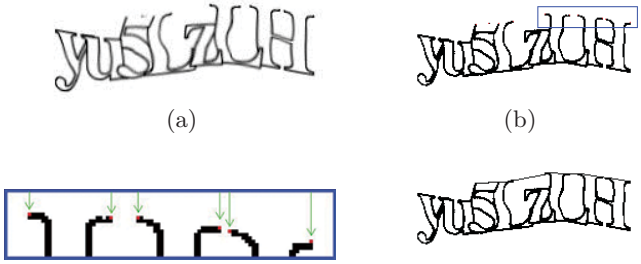
The high-level flow of our attack, shown in Figure 2, includes three main sequential steps: 1) pre-processing prepares each challenge image with standard techniques; 2) extracting character strokes; 3) segmentation and recognition with the CNN engine assisted graph search.

5. OUR ATTACK: TECHNICAL DETAILS

We choose Yahoo! and CmPay as examples to explain key techniques in our general attack. Note that these techniques are generically applicable to all the schemes, as evi-

(a) (b)

Figure 3: Binarized images. (a)Yahoo!, (b)CmPay.



(a) (b) (c) (d)

Figure 4: Repair contour lines. (a)The original image, (b)A blue rectangle indicates some identified break points, (c)The amplified blue rectangle where break points are highlighted in red, (d)The image with breakpoints connected.

denced later by our implementation and evaluation (details see Section 6).

5.1 Pre-processing

Image binarization. This is to covert a color or gray-scale image into black-and-white one. We use the standard Otsu's threshold method [17]. Figure 3 shows the binarized images.

In the cases of CmPay and Baidu, binarization does not just convert an image into black-and-white, but also removes thin interference arcs whose colors differ significantly from both hollow characters and thick arcs.

Repair contour lines. Some challenges use hollow characters whose contour lines are broken (e.g. Figure 4 (a)), for which CFS will fail filling the hollow parts. In the case of CmPay, binarization created broken contours, too (see Figure 3 (b)). In order to make CFS work, these broken contour lines must be repaired. We use Lee's algorithm [10] to automatically detect and then repair broken contours. This algorithm was initially designed for solving maze routing problems.

First, we identify break points in contour lines. Similar to the maze solving problem, a contour is regarded as the corridor, and break points as dead-ends in the maze. We use Lee's algorithm to traverse the contour line, and mark the dead-end pixel of each path in red (e.g. Figure 4 (c)). Then, we examine the relative positions of each pair of break points, and draw a one-pixel-thick line to connect those pairs that look valid (see Figure 4 (d)).

Incorrect connections may be created e.g. Figures 4(d) and 5, but our attack can cope with them, as explained later.

Figure 5: Contour repair: another example

(a) (b)

Figure 6: Images after CFS: (a)Yahoo!, (b)CmPay

5.2 Extracting Character Strokes

Fill hollow parts with CFS. This uses a flooding algorithm to detect connected non-black pixel blocks. It will pick up both character components and noise ones, if they have closed contours. We use a distinct color to fill, and thus identify, each component. After this step, the background color in an image is set to lightgray, but contour lines and other solid parts such as thick interference arcs remain in black. Figure 6 shows the images after CFS.

Noise component removal. As illustrated in Figure 7, there are at least three types of undesirable components (or chunks) which we consider as noise: 1) the closed part within a character, 2) those formed by two connected characters, and 3) those formed by wrong connections introduced by the contour repair algorithm.

We first define parameters θ and δ for each color component: $\theta = C_g/C$ and $\delta = C/S$, where C_g denotes the number of edge pixels that have a lightgray neighbor, C denotes the total number of edge pixels in this component, and S denotes the total number of pixels in this component.

With properly chosen threshold values a and b, we have the following: if $\theta < a$ for a component, it is a noise component of the first two types; if $\delta < b$, it is a noise component of the third type. We determine the values a and b via a learning algorithm that analyzes a small sample set of data.

In schemes like CmPay, an additional type of noise components was introduced by thick interference arcs. However, they are easy to remove by detecting such arcs' existence.

We have tested our noise component removal on all the 5 hollow schemes, and it works on all of them. Components surviving our noise removal are considered to belong to character bodies, and are thus preserved (see Figure 8).

Contour line removal. Removing contour lines is straightforward for CmPay, but requires more subtle techniques for the Yahoo! scheme. We have implemented both the straightforward and the more subtle approaches. Which ap-

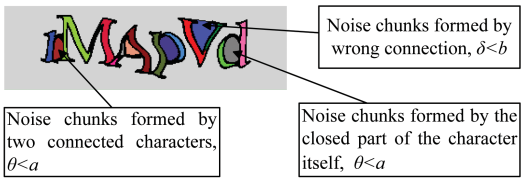


Figure 7: Three types of noise components.

(a) (b)

Figure 8: After removing noise components. (a)Yahoo!, (b)CmPay.

(a) (b)

Figure 9: After contour line removal. (a)Yahoo!, (b)CmPay.

proach is needed for handling a particular Cpatcha scheme can be automatically determined.

After removing the noise components in the CmPay scheme, black pixels are the interference arcs and contour lines, and all character strokes are in non-black colors. So it is straightforward to remove both contour lines and the interference arcs by switching all black pixels to the background color.

In the Yahoo! scheme, after removing noise components, black pixels in an image are character contours, incorrect connecting lines introduced by our contour repair algorithm, or a character stroke. We need to remove the first two types of black pixels, but keep the third. The second type is always of one-pixel thickness, and easy to remove first. Then we perform an image dilation on each stroke filled with a non-black color. The stroke is dilated to cover its surrounding contour line. That is, the dilation algorithm switches the contour line to the color of the stroke body to merge them. After this step, all remaining large blocks of black pixels have to be character strokes. Figure 9 shows images after contour removal.

Clean-up. Tiny pixel blocks might be created by contour line removal. In clean-up, they are either removed directly or merged with adjacent larger strokes via an automated algorithm. Figure 10 shows the resulting images. After this step, what remain in an image are all character strokes or components.

5.3 Segmentation and Recognition

The next step is to find how to form strokes into individual characters and recognize what the characters are. Now the problem is similar to a jigsaw puzzle in that each stroke is a piece of a master design. However, there is no fixed pattern for us to exploit for finding the right combination. The number of strokes is always larger than the number of characters to be formed, and the latter is a variable in schemes

(a) (b)

Figure 10: After clean-up. (a)Yahoo!, (b)CmPay.

(a) (b)

Figure 11: What remains are all character strokes, rank ordered. (a)Yahoo!, (b)CmPay.

such as Yahoo!; so there can be many possible combinations. Our approach is to combine adjacent strokes into possible characters and determine the most likely result.

First, all strokes in an image are numbered in an incremental order from the upper left to lower right (Figure 11).

Then we try to combine strokes or components following the incremental rank order. An $n \times n$ table is built for each image, where n is the total number of strokes in the image. A cell at the intersection of row i and column k in the table indicates whether it is feasible to combine strokes i, $i+1$, ..., k altogether to form a single component.

If such a combination is infeasible, the cell (i, k) will be set to NULL. This occurs in one of the following three scenarios: (1) when its row index i is larger than its column index k (we try combinations only in a monotonic order); (2) when the width of the combination is greater than the largest possible character width, which is empirically established with a simple analysis of the sample set of data; (3) when the width of the combination is less than the smallest possible character width, which is also empirically established with a simple analysis of the sample set.

If it is feasible to combine strokes i, $i+1$, ..., k to form a single component, we let the CNN decide which character this component is likely to be, and the cell (i, k) stores the neural network's recognition result, along with a confidence level the CNN feels about this result. This feasibility condition is met in all situations except the above three.

In our implementation, an image input to the CNN is normalized to the size of 28×28; the output confidence level is calculated after layer-by-layer forward propagation. Since the activation function used is a scaled version of the hyperbolic tangent [11], scaling causes the confidence level to vary between -1.7159 and 1.7159. The larger a confidence value is, the more likely the recognition result is correct.

Tables 2 and 3 show a $n \times n$ table for the Yahoo! sample and the CmPay sample, respectively. For example, in table 2, the cell (1, 1) indicates that the CNN recognize this single stroke as 'r' with a confidence level of .479; the cell (1, 3) indicates that the combination of strokes 1, 2 and 3 is recognized as 'M' with a confidence level of .432. Each empty cell (i, k) indicates that a combination of strokes i, $i+1$, ..., k is infeasible for one reason or another.

A $n \times n$ table gives all plausible stroke combinations for an image. Our task now is to use information in the table to find the most likely way of forming characters, i.e., finding the best segmentation or partition.

Each $n \times n$ table is actually equivalent to a directed and weighted graph. Figure 12 gives such graphs that are equivalent to Tables 2 and 3, respectively. In each graph, the nodes are numbered from 1 to $n+1$, and nodes 1, 2, ... n represent the corresponding strokes, respectively. An arc $< i, j >$ (i.e. a directed edge linking vertexes i and j) indicates that the combination of strokes from i to $j-1$ is feasible, and

Table 2: The $n \times n$ table generated by CNN for the Yahoo! sample in Figure 11.

	1	2	3	4	5	6	7	8	9	10	11
1	***r/0.479***	A/-0.157	M/0.432								
2		A/-0.027	M/-0.358	***M/1.025***							
3				T/0.01	W/-0.43						
4				L/0.255	W/0.216						
5					***A/0.482***	V/-0.2	V/-0.242				
6						V/0.238	***p/1.087***	A/-0.384			
7							y/-0.151	w/-0.462	V/-0.127		
8								V/0.358	***V/1.11***	H/-0.088	w/0.238
9									r/-0.035	6/-0.519	M/-0.255
10										c/0.554	***d/1.075***
11											

Table 3: The $n \times n$ table generated by CNN for the CmPay sample in Figure 11

	1	2	3	4	5	6	7	8	9
1		***K/0.935***	K/0.781	H/-0.229					
2		K/0.647	4/0.493	4/-0.445					
3					6/0.593	***U/0.692***			
4					6/0.06	U/0.688			
5					J/0.093	U/-0.332			
6									
7							***Q/1.102***		
8								R/0.229	***R/1.057***
9									S/1

the associated weight gives both the recognition result of this combination and the confidence level calculated by the neural network for this result. That is, the number of arcs in a graph equals to the number of non-empty cells in its corresponding table.

Now, our task of finding the best segmentation and thus the most likely recognition result is equivalent to the following graph search problem:

- Find a path that starts from node 1 and ends at node $n+1$, and in which each node is traversed only once, and a node always has a larger index number than all its predecessor(s).

- The path's length (i.e. the number of edges on the path) is exactly the same as the number of characters that are supposed to be in a CAPTCHA image.

  Note: the rationales are the following: *all* the strokes in an image will be used to form individual characters, but each stroke will be used only once and no stroke is shared in adjacent characters.

- The sum of confidence levels along the path is the largest possible in the graph.

We use a depth-first-search (DFS) algorithm to select the optimal partition which has the highest confidence sum. It is simple for our algorithm to cope with schemes like Yahoo!, where a varied string length is used.

For convenience, we utilize the $n \times n$ tables to implement the graph search algorithm. The search always starts from the first row in table, and progresses from the smallest index number to the largest in an incremental order.

The pseudo code below sketches the recognition process. $S_{i,j}$ denotes the resulting character which the CNN recognizes as the component combined from strokes $i, i+1, ..., j$; $a_{i,j}$ is the confidence level calculated by the CNN for this result; *step* represents the number of characters that has been recognized; *sum* represents the confidence value sum of strokes 1 to j, and S represents the recognition result of combined strokes numbered from 1 to j.

As generated by our attack program, tables 4 and 5 show all likely partitions and for each partition, its sum of confidence levels. The italicized items highlighted in red in each table indicate the optimal partition that has the highest sum of confidence levels and that matches a legitimate length of CAPTCHA strings. In both cases, 'rMApVd' and 'KUQR' are correct recognition results.

We note that simply choosing the cells with a relatively large confidence level does not work (well). The reason is that this greedy strategy skips many possible combinations, and thus the recognition success will be low. In comparison, our algorithm performs a thorough search of each graph, and will not miss any possible results. In Table 2, the confidence levels in the cells (1, 1) and (5, 5), both of which are on the optimal path, are actually smaller than that in the cell (10, 10), which is however not on the optimal path.

The sum value p of the selected partition is described as follows: $p = max\,(a_{1,i} + a_{i+1,j} + \cdots + a_{m+1,k} + a_{k+1,n})$, where $0 < i < j < m < k < n$, and we assume that the optimal partition consists of strokes numbered from 1 to i, then strokes i+1 to j, ..., m+1 to k, and finally strokes k+1 to n. The number of polynomial terms in the equation equals the string length in the CAPTCHA.

Table 4: Every likely result and its confidence sum for the Yahoo! Sample in Figure 11

Result	Confidence	Result	Confidence	Result	Confidence	Result	Confidence	Result	Confidence
rATAVyVM	1.135	rATAVyVd	3.216	rATAVyw	1.27	rATAVwrd	1.76	rATAVwM	0.465
rATAVVd	2.131	rATApVrd	3.431	rATApVM	2.136	rATApVd	4.218	rATApw	2.271
rATAArd	1.6	rATAAM	0.305	rATVyVrd	1.508	rATVyVM	0.213	rATVyVd	2.295
rATVyw	0.348	rATVwrd	0.838	rATVwM	-0.456	rATVVd	1.209	rATVVrd	1.618
rATVVM	0.323	rATVVd	2.405	rAVVyVrd	1.506	rAVVyVM	0.211	rAVVyVd	2.293
rAVVyw	0.346	rAVVwrd	0.836	rAVVwM	-0.458	rAVVVd	1.207	rAVpVrd	2.507
rAVpVM	1.212	rAVpVd	3.294	rAVArd	0.676	rMLAVyVM	1.056	rMLAVyVd	3.138
rMLAVyw	1.191	rMLAVwrd	1.681	rMLAVwM	0.386	rMLAVVd	2.052	rMLApVrd	3.352
rMLApVM	2.057	rMLApVd	4.139	rMLApw	2.192	rMLAArd	1.521	rMLAAM	0.226
rMLVyVrd	1.43	rMLVyVM	0.135	rMLVyVd	2.217	rMLVyw	0.27	rMLVwrd	0.76
rMLVwM	-0.534	rMLVVd	1.131	rMLVVrd	1.54	rMLVVM	0.245	rMLVVd	2.327
rMwVyVrd	1.349	rMwVyVM	0.053	rMwVyVd	2.135	rMwVyw	0.188	rMwVwrd	0.679
rMwVwM	-0.616	rMwVVd	1.05	rMwpVrd	2.35	rMwpVM	1.055	rMwpVd	3.136
rMwArd	0.519	rMAVyVrd	3.472	rMAVyVM	2.177	rMAVyVd	4.259	rMAVyw	2.312
rMAVwrd	2.802	rMAVwM	1.507	rMAVVd	3.173	rMApVrd	4.473	rMApVM	3.178
rMApVd	***5.26***	rMAArd	2.642	rMVyVrd	2.551	rMVyVM	1.255	rMVyVd	3.337
rMVwrd	1.881	rMVVrd	2.661	ATAVyVrd	1.821	ATAVyVM	0.526	ATAVyVd	2.608
ATAVyw	0.661	ATAVwrd	1.151	ATAVwM	-0.143	ATAVVd	1.522	ATApVrd	2.822
ATApVM	1.527	ATApVd	3.609	ATAArd	0.991	ATVyVrd	0.9	ATVyVM	-0.394
ATVyVd	1.686	ATVwrd	0.23	ATVVrd	1.01	AVVyVrd	0.897	AVVyVM	-0.397
AVVyVd	1.684	AVVwrd	0.227	AVpVrd	1.899	MLAVyVrd	2.663	MLAVyVM	1.367
MLAVyVd	3.449	MLAVyw	1.502	MLAVwrd	1.993	MLAVwM	0.698	MLAVVd	2.364
MLApVrd	3.664	MLApVM	2.369	MLApVd	4.45	MLAArd	1.833	MLVyVrd	1.741
MLVyVM	0.446	MLVyVd	2.528	MLVwrd	1.071	MLVVrd	1.851	MwVyVrd	1.66
MwVyVM	0.365	MwVyVd	2.447	MwVwrd	0.99	MwpVrd	2.661		

Table 5: Every likely result and its confidence sum for the CmPay sample in Figure 11

Result	Confidence	Result	Confidence	Result	Confidence
KUQR	***3.786***	KUQR	3.632	HUQR	1.598

DepthFirstSearch()
$i \leftarrow 1$
$step \leftarrow 0$
$p \leftarrow 0$
$sum \leftarrow 0$
$R \leftarrow NIL$
$S \leftarrow NIL$
$Traverse(i, step, sum, S)$
$print\ R$

Traverse(i, step, sum, S)
for $j \leftarrow i$ *to* n
if $a_{i,j} \neq NULL$
  then $sum \leftarrow sum + a_{i,j}$
    $S \leftarrow strcat(S, S_{i,j})$
    $step \leftarrow step + 1$
  else continue
    if $step \in CAPTCHA\ length$ **and** $p < sum$
      then $p \leftarrow sum$
        $R \leftarrow S$
        if $step \leq max\ CAPTCHA\ length$
          then $Traverse(j + 1, step, sum, S)$

Table 6: The success rate and speed of our attack

Scheme	Success on sample set	Success on test set (T)	Avg time per challenge (R)	Avg time per success (T/R)
Yahoo!	56%	36%	5.30 seconds	14.72 seconds
Tencent	93%	89%	1.23 seconds	1.38 seconds
Sina	63%	59%	1.77 seconds	3.00 seconds
CmPay	73%	66%	4.25 seconds	6.43 seconds
Baidu	57%	51%	3.87 seconds	7.58 seconds

6. EVALUATIONS

We have implemented our attack and tested it on all the 5 hollow schemes. We present our evaluations as follows.

Data Collection. For each of the 5 schemes, we collected from the corresponding website 1000 random CAPTCHAs as a sample set, and another 500 as a test set. For schemes like Yahoo!, Tencent and CmPay, the data were collected in August 2012; for schemes such as Sina and Baidu, the data were collected in April 2013.

Training Neural Network. The template library for training our convolutional neural network was prepared manually. We extracted 4244 characters from Yahoo! samples, 3754 characters from Tencent samples, 2680 characters from Sina samples, 3670 characters from CmPay samples, and 2940 characters from Baidu samples, to train the CNN.

Success Rate. We follow a common practice to evaluate our attack's success. For the Yahoo! scheme, we achieved a success rate of 56% on the sample set. That is, 56% of the challenges were entirely guessed correctly. Then we ran our attack on the test set, about which our program had

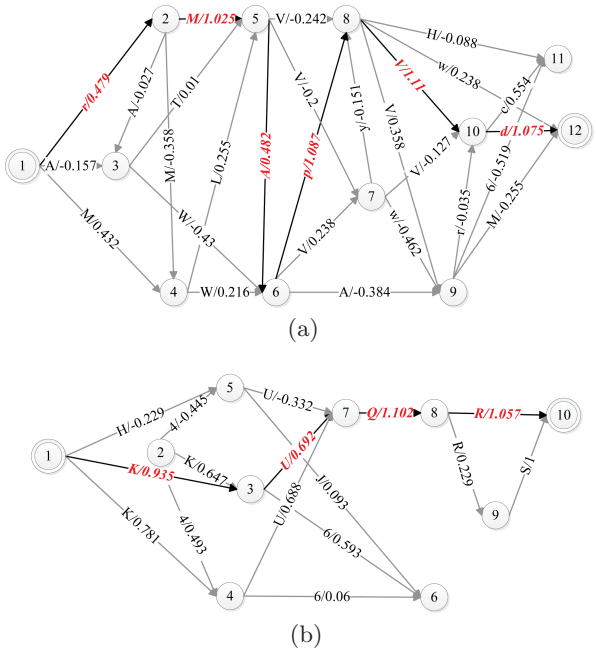


Figure 12: The equivalent graphs of Tables 2 and 3.

no prior knowledge about any particular sample within; we achieved a success rate of 36%.

Similarly, we ran our attack on all other schemes. The success rates are listed in Table 6. For the Tencent scheme, we achieved 93% success on the sample set and 89% on the test set. For Sina, we achieved 63% success on the sample set and 59% on the test set. For CmPay, our success was 73% on the sample set and 66% on the test set. For Baidu, our success was 57% on the sample set and 51% on the test set.

A commonly accepted goal for CAPTCHA robustness is to prevent automated attacks from achieving a success rate of higher than 0.01% [23]. Bursztein et al [7] considered this goal too ambitious, and they suggested that a CAPTCHA scheme is broken when the attacker achieves an accuracy rate of at least 1%. According to either criterion, all the five hollow schemes are successfully and terribly broken by our attack.

In reality, an attacker could achieve a success rate even higher than reported here, as he could simply skip a challenge if the confidence level for recognizing it is not large enough. Instead, he could keep requesting new challenges, and only when he is confident enough with a recognition result, he submits the answer to the CAPTCHA.

Attack Speed. We implemented our attack in C#, and tested it on a desktop computer with a 2.53 GHz Intel Core 2 CPU and 4 GB RAM. The attack was run ten times on each data set, and the average speed was recorded. Table 6 summarizes the speed of our attack on each scheme. On average, it takes only seconds to attack a CAPTCHA in any of these schemes. Besides, We also estimate an average time for successfully breaking a CAPTCHA in each scheme: on average, it takes 3 – 15 seconds. Clearly, our attack is efficient and poses a realistic threat to all the hollow schemes.

Other Classifiers. We also tested other classifiers such as Support Vector Machine, Back-Propagation Neural Network and template matching. The CNN engine has achieved the best overall performance for both attack speed and success rate.

Our DFS algorithm is not optimal. We did not optimize this algorithm for a simple reason: we measured the time consumption for each step in our attack. It turns out that 'contour line removal' and 'clean-up' take about 58% of the time of the whole attack, but 'segmentation and recognition' takes only about 13%.

7. DISCUSSIONS

7.1 Novelty

To our best knowledge, this is the first security analysis on the state of the art of hollow CAPTCHAs. We used some standard techniques in pre-processing, and also used CFS, which has been a standard method for analyzing text CAPTCHA robustness since its introduction in [23]. However, the key component of our attack, the graph search algorithm based segmentation and recognition, is absolutely novel. Overall, the combination of these and other techniques has led to a novel attack.

State of the art attacks on the CCT based CAPTCHAs, such as [7] and [8], do not work on the hollow CAPTCHAs studied in this paper.

Among all the related work we have discussed earlier, the attack reported in [9] on a MegaUpload CAPTCHA is the only one that has a slight similarity to our attack. However, that attack was designed for a single CAPTCHA; it is ad hoc and not applicable to attacking hollow CAPTCHAs. Moreover, that attack focused on segmentation, and did not involve character recognition at all.

7.2 Generic Value

Our attack is applicable to a variety of hollow CAPTCHAs. It works on schemes with thin contours (e.g. Yahoo!) and on schemes with thick contours (e.g. Tencent); on schemes with interference arcs (e.g CmPay) and on schemes without such arcs (e.g. Yahoo!); on schemes with a fixed length (e.g. Sina) and on schemes with a varied length (e.g. Yahoo!). Table 7 summarizes the main features of these schemes. As they represent different designs, each with distinctive features, our attack is of some generic value.

7.3 Lessons

Our attack helps to identify which design features contribute to a hollow CAPTCHA's security, and which do not. Design features that *do* help security include the following.

Overlapped or connected characters are still the most crucial security feature, as by design it provides (some) segmentation resistance. It significantly contributes to the security of all the 5 schemes.

String length matters. This was first observed in [23] and then confirmed in [7]. First, it is good to use a relatively large length. The more characters used in a CAPTCHA image, the more components remain after preprocessing, and the larger the solution space will be. This will decrease an attack's success and speed. The more characters used, the harder for brute-force guessing, too. Second, it is good to use a varied length, which does not give away useful information to aid attackers. Attackers have to try multiple

Table 7: Main features of five hollow CAPTCHAs

Scheme	Interference Arcs	Broken Contour Line	Contour Line Thickness	Hollow Styles	String Length	Character Overlap	Alphabet Size
Yahoo!	No	Sometimes	Uniform	Varied	Varied (6-8)	Yes	28
Tencent	No	No	Varied	Uniform	Fixed (4)	Yes	25
Sina	No	No	Varied	Uniform	Fixed (4)	Yes	28
CmPay	Yes	After binarization	Varied	Uniform	Fixed (4)	Yes	30
Baidu	Yes	No	Varied	Uniform	Fixed (4)	Yes	51

(a) (b)

Figure 13: Yahoo!'s hollow styles: (a)thick strokes vs. (b)thin strokes (both after CFS).

possible lengths, which increases the search space for our graph algorithm, and could decrease its success and speed.

Broken contours considerably increase the difficulty level of designing and implementing an effective attack. In particular, they disable the otherwise powerful CFS. Broken contours introduced by design or after binarization are both good for security, but the former wins our recommendation as it is probably easier to control by a CAPTCHA generator.

Thick interference arcs cut across characters, not just dividing characters to fragments but also introducing noise components. This considerably increases the difficulty level of designing and implementing an effective attack.

Hollow styles. Varying thickness of hollow portions is an important style feature that contributes to security. Some hollow portions in the Yahoo! scheme were so thin that their contour lines were squashed together, which prevents the portions from being picked up by CFS. This not only increases the number of strokes in an image, but also makes it a challenge to cope with those squashed strokes.

In the Yahoo! scheme, another variation in hollow styles is heavily used. Namely, two font types are used, creating two styles: one we call the 'thick strokes', and the other the 'thin strokes' (see Figure 13). Until now, what is explicitly discussed in this paper is the 'thin strokes'. In the 'thick strokes', character strokes can be completely picked up by CFS, leading to not just fewer components than in the 'thin strokes', but also a simplified treatment by the follow-up attack procedures. That is, 'thick strokes' is a weaker design than 'thin strokes' in terms of security. Note that our attack is applicable to both types, and our program automatically handles both the types.

Having multiple designs and deploying them alternately in a random order is good for security, as first suggested in [26] and then confirmed in [7]. However, randomly alternating two hollow styles in the Yahoo! scheme is probably only marginally useful for improving security, as the alternatives are not equally strong.

Varying width of individual characters is a design feature that is somehow related to hollow styles but beyond that. It contributes to security for the following reasons. The larger the width difference between the thinnest character and the fattest one, the larger a search space faces our graph search algorithm, and the more likely it will give inaccurate results.

Design features that *do not* help security include:

Complete contours, which help CFS to pick up character strokes.

Contour thickness. Thinning [25] contour lines to a uniform thickness is useful for our attack, but not essential. Therefore, we do not consider variations in contour thickness contributes much to security.

Thin interference arcs, which are easily removed by binarization.

Fixed string length, which gives away useful information to aid segmentation.

Short string length, which reduces an attack's search space and increase its chance of success.

This set of 'Does' and 'Don'ts' constitutes a set of design guidelines for the security of hollow CAPTCHAs. It also provides a method for comparing different designs, as well as explaining and pinpointing why one scheme is better than the other in terms of security.

For example, although broken, the Yahoo! scheme is apparently the best among all the 5 schemes, and the following features contribute to its unique strength: 1) Using a relatively large string length, and the length is not fixed. 2) Variations in hollow portions' thickness. 3) Variations in the width of individual characters width.

The second best is Baidu. Its unique strength lies in its large alphabet set, which have 58 characters including digits and upper- and lower-case letters. This is the largest alphabet set among the five schemes. It significantly increases the solution space for individual characters, and makes it harder for the CNN engine to accurately recognize the characters. In particular, with a limited number of training data, it is unsurprising that the CNN engine will decrease our attack's success. On the other hand, interference arcs marginally contribute to the security of this scheme, too.

To the contrary, Tencent is the worst design, as it made the worst choices for almost all security features.

7.4 Towards Better Designs

Although broken, the Yahoo! scheme has some good security features. It is unwise to throw away its current design and start from scratch. Instead, we explore how to evolve this scheme into a better design. Plausible options include the following.

Increasing the alphabet size, which is currently among the smallest across the 5 schemes. This is a simple but effective solution, and with little negative impact on usability (if confusing characters are excluded from the alphabet).

Using broken contours more often. In its current design, not all of the Yahoo! challenges contain broken contours. It is useful to make challenges with broken contours occur

more often, since this will likely decrease our attack's success rate and increase the time it takes to succeed.

More fundamentally, introducing more broken points in each challenge will likely significantly increase security. Breaking contour lines badly or at least creating a large number of broken points at random locations will make it hard to repair the contours. If the repair algorithm does not work, attackers will be unable to rely on the otherwise effective CFS method to extract character strokes any more.

Introducing thick interference arcs to cut through characters so that when adjacent strokes are combined for tests, combined characters contain either extra strokes or lack essential parts. This will confuse the CNN engine, leading to incorrect recognition results. Also, cutting-through arcs can be used to create a large number of components in an image. The more components the CNN engine has to try to combine, the lower the attack's success rate and speed.

Increasing the length of each CAPTCHA string helps, too, for the same reason as above.

Increasing the variation in character widths. The larger the gap between the smallest and largest width of individual characters in an image, the larger a solution set our graph search algorithm is required to go through. This might significantly slow down our attack.

Some of these measures (such as increasing the string length) only have a linear effect on the search space, but methods such as 'using broken contours more often' theoretically would be much more effective in thwarting attacks.

We note that careful studies are needed to establish how well these advised measures will work, and more importantly, some measures may decrease recognition success for humans. It is important to strike the right balance between security and usability. It remains an open problem what design will be eventually both secure and usable, and whether this design is mission impossible. Nonetheless, our discussions offer practical suggestions for improving hollow CAPTCHA designs.

8. SUMMARY AND CONCLUSION

We have shown that hollow CAPTCHAs, a new type of text scheme, have serious security problems. With a simple attack, we have broken the hollow schemes deployed by Yahoo!, Tencent, Sina, CmPay and Baidu with a success rate of 36%, 89%, 59%, 66%, and 51%, respectively. As these schemes are different from each other, and each with distinctive design features, our work casts serious doubt on the current generation of hollow CAPTCHAs.

Hollow CAPTCHA is a clever idea in that it improves usability while keeping characters connected or touching each other, but this idea has unexpected security consequences. A key issue in text CAPTCHA design is to find a segmentation-resistant mechanism that is secure and user-friendly simultaneously. The hollow CAPTCHA approach does not achieve this goal yet.

By comparing representative designs of popular hollow CAPTCHAs, we have identified good design features for better security. We have also discussed how to create next generation of better designs. However, it remains an open problem how to design CAPTCHAs that are both secure and usable, and this is our ongoing work. Overall, our work contributes to advancing the current collective understanding of CAPTCHA design.

9. ACKNOWLEDGEMENTS

We thank Lindsay Marshall, Feng Hao and anonymous reviewers for helpful comments. Xidian authors are supported by the National Natural Science Foundation of China (60903198) and the Fundamental Research Funds for the Central Universities.

10. REFERENCES

[1] Baidu. `https://passport.baidu.com/reg`.

[2] China mobile. `https://cmpay.10086.cn/`.

[3] Pwntcha - captcha decoder web site. `http://sam.zoy.org/pwntcha/`.

[4] Sina weibo. `http://weibo.com/signup/signup.php`.

[5] Tencent security center. `http://aq.qq.com/cn2/findpsw/findpsw_index`.

[6] Yahoo! password helper. `https://edit.yahoo.com/forgot?stage=fe100&src=&intl=us&done=http://www.yahoo.com&partner=reg`.

[7] 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*, pages 125–138. ACM, 2011.

[8] Ahmad S El Ahmad, Jeff Yan, and Mohamad Tayara. *The Robustness of Google CAPTCHAs.* Computing Science Technical Report CS-TR-1278, Newcastle University, UK, 2011.

[9] Ahmad Salah El Ahmad, Jeff Yan, and Lindsay Marshall. The robustness of a new captcha. In *Proceedings of the Third European Workshop on System Security*, pages 36–41. ACM, 2010.

[10] Jeffrey H Hoel. Some variations of lee's algorithm. *Computers, IEEE Transactions on*, 100(1):19–24, 1976.

[11] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[12] Mark D Lillibridge, Martin Abadi, Krishna Bharat, and Andrei Z Broder. Method for selectively restricting access to computer systems, February 27 2001. US Patent 6,195,698.

[13] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, volume 1, pages I–134. IEEE, 2003.

[14] Gabriel Moy, Nathan Jones, Curt Harkless, and Randall Potter. Distortion estimation techniques in solving visual captchas. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–23. IEEE, 2004.

[15] Moni Naor. Verification of a human in the loop or identification via the turing test. *Unpublished draft from http://www. wisdom. weizmann. ac. il/˜ naor/PAPERS/human abs. html*, 1996.

[16] M. O'Neill. Cnn. `http://www.codeproject.com/Articles/16650/Neural-Network-for-Recognition-of-Handwritten-Digi`.

[17] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *Automatica*, 11(285-296):23–27, 1975.

[18] Patrice Simard, David Steinkraus, and John C Platt. Best practices for convolutional neural networks applied to visual document analysis. In *ICDAR*, volume 3, pages 958–962, 2003.

[19] PY Simard. Using machine learning to break visual human interaction proofs (hips. *Advances in neural information processing systems*, 17:265–272, 2005.

[20] Luis Von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Communications of the ACM*, 47(2):56–60, 2004.

[21] Y Xu, G Reynaga, S Chiasson, JF Frahm, F Monrose, and PC Van Oorschot. Security and usability challenges of moving-object captchas: decoding codewords in motion. In *21st USENIX Security Symposium*, 2012.

[22] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*, pages 279–291. IEEE, 2007.

[23] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM conference on Computer and communications security*, pages 543–554. ACM, 2008.

[24] Jeff Yan and Ahmad Salah El Ahmad. Usability of captchas or usability issues in captcha design. In *Proceedings of the 4th symposium on Usable privacy and security*, pages 44–52. ACM, 2008.

[25] TY Zhang and Ching Y. Suen. A fast parallel algorithm for thinning digital patterns. *Communications of the ACM*, 27(3):236–239, 1984.

[26] 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*, pages 187–200. ACM, 2010.