Extended Security Analysis of Hollow Captchas

Haichang Gao[1], Ping Wang[1], Jeff Yan[2], Mengyun Tang[1], Fang Cao[1]

[1] School of Computer Science and Technology, Xidian University, China
[2] Department of Computer and Information Science, Linköping University, Sweden
hchgao@xidian.edu.cn, 497830219@qq.com, jeff.yan@liu.se,
MerryTangMengyun@gmail.com, 412602545@qq.com

Abstract

Text-based Captchas are now most widely used security technology for differentiating between computers and humans. Hollow Captchas have emerged as one of the latest designs, and they have been deployed by more and more major companies. Besides Yahoo!, Tencent, Sina, China Mobile and Baidu, some other websites, especially for higher security requirement shopping websites are also using this scheme. 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. It is hard for standard techniques to segment and recognize such connected characters, which are however easy for human eyes. In this paper, we provide a systematic security analysis of hollow Captchas. We show that with a simple but novel attack, we can break most hollow Captchas with a relatively high success rate, including those deployed by the major companies. Our attack for the first time combines segmentation and recognition in a single step. We also discuss lessons and guidelines for designing better Captchas.

Keywords: Captcha, Hollow, Graph search, Security

1 Introduction

Captcha (Completely Automated Public Turing Test to Tell Computers and Humans Apart) has been widely deployed for defending against undesirable and malicious bot programs on the Internet [1]. Current Captchas can be divides into three categories: text-based Captchas, image-based Captchas and audio-based Captchas. The most widely used Captchas are text-based schemes [2], which typically require users to solve a text recognition task.

If a Captcha is friendly for humans to solve but hard for computers, it can be considered as a good one. It turns out that the balance between security and usability is hard to achieve. So far, many text-based Captchas have been broken, including those deployed by major websites such as Microsoft, Yahoo! and Google [3-4]. However, [4] predicted that 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 next generation of systems.

Hollow Captchas, as one of the latest text-based designs, have emerged in the last couple of years They have been deployed by major websites such as Yahoo!, Baidu, Sina, Tencent and the online payment system of China Mobile (CmPay), each serving tens of millions of users on a daily basis. Such hollow Captchas 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.

Figure 1. Hollow captchas

Given the high profiles of the companies that have deployed hollow Captchas, it is of practical relevance

*Corresponding Author: Haichang Gao; E-mail: hchgao@xidian.edu.cn
DOI: 10.3966/160792642018081904011

to examine the robustness of such hollow schemes, i.e. their resistance to automated attacks, which is an important security property [12]. 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, this is the first systematically analysis of the security of hollow Captchas. Though recently some research teams (Vicarious [5], Elie Bursztein's team [6] and Gao [26]) have declared that they have found a generic method to break a whole family of latest Captchas, hollow schemes included. But neither technique details revealed nor did more hollow Captchas attack, we can't see more information on attacking the hollow Captchas.

In this paper, we provide the specific analysis of the robustness of hollow Captchas. With a novel and generic attack, we can successfully break a whole family of hollow Captchas (Except for Yandex, which will discuss in Section 6). The success rates received by our attack on the test set of Yahoo!, Tencent, Sina, CmPay, Baidu and Yhd schemes are 36%, 89%, 59%, 66%, 51% and 36%, respectively, and with average attack speed of 5.17, 1.14, 1.36, 3.70, 3.52, and 0.49 seconds respectively on a standard desktop computer (with 2.53 GHz Intel Core 2 CPU, 4 GB RAM). Therefore, our attack imposes a realistic threat, which might be misused by adversaries.

As a variety of design features 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 Captchas, and a method from comparing security of different schemes. We also discuss how to design better hollow Captchas.

An early version of this paper has been published in ACM CCS 2013 [22]. In this paper, we improved the graph search algorithm and compared the efficiency of the new algorithm and the previous in detail. We also tested our attack on two extra hollow Captcha schemes, YhD and Yandex. Especially the Yandex scheme which is deployed by the largest Russian search engine in its user password recovery mechanism, formed by seriously broken contour lines, is a new form of hollow Captcha that has not been analyzed in [22].

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 evaluates our attack process and analyses the failures. Section 7 discusses lessons we have learnt, and how to design better hollow Captchas. Section 8 concludes the paper.

2 Related Work

Moni Naor first discussed the notion of Automated-Turing-Tests [7], but they did not provide a formal definition or concrete designs. Alta Vista [8] developed the first practical Automated-Turing-Test 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 [9] utilized sophisticated object recognition algorithms to break Gimpy (which used clutter interference) and EZ-Gimpy (which used texture backgrounds) with a success rate of 33% and 92% respectively. Moy et al. [10] developed distortion estimation techniques to attack EZ-Gimpy and achieved a success rate of 99% and four-letter Gimpy-r with a success rate of 78%.

In 2005, Chellapilla and Simard [11] 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.

In 2006, Yan and El Ahmad [13] broke most visual schemes provided at Captchaservice.org by simply counting the number of pixels of each segmented character and have achieved a success rate of nearly 100%, although these schemes were all resistant to the best OCR software on the market. In 2008, new character segmentation techniques for attacking a number of text-based Captchas were developed by the same team [4], including the earlier mechanisms designed and deployed by Microsoft, Yahoo! and Google, and these have received a segmentation success rate of 92% against Microsoft Captcha. In 2010, they broke the text-based Captchas that depend partially on the Gestalt Perception principle by merging black and shared white components to form individual characters [14].

In 2011, Bursztein et al. [15] carried out a systematic study of existing visual Captchas based on distorted characters and showed that 13 of the 15 Captchas on popular websites were vulnerable to automated attacks, 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 [3]. The Captcha using moving-images in NuCaptcha which provided users with sloshing characters was analyzed by Xu et al. in 2012 [16].

In 2014, Burszteins team [6] proposed a machine learning algorithm to score all possible ways to segment a Captcha and decide which combination is the most likely to be the correct one. They break 8 different Captchas used by real world popular web sites.

In 2015, Karthik et al. [25] proposed two methods to automatically classify Microsoft Captcha samples. One was based in a fine-grain segmentation combined with template matching, and another was based on CNN and used state-of-the-art recognition techniques. The former achieved a success of 5.56%, and the later achieved a success of 57.05%.

More recently, at NDSS'16, Gao et al. [26] reported a simple generic attack that firstly used Gabor filters to extract character components along four different directions and then try different combinations of adjacent character components to form individual characters. It is the most recent research on Captcha robustness analysis and is effective for many text-based Captchas.

We note that hollow Captchas have never been specifically 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

To evaluate the effectiveness of our attack, we choose to study 7 hollow Captchas listed in Figure 1 (including Yahoo!, Tencent, Sina, CmPay, Baidu, Yhd and Yandex), which 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 For example, Baidu, Yahoo!, Tencent and Sina, ranked by Alexa.com as top websites in the world respectively, are all among the most popular websites worldwide. Sina use its scheme on Weibo.com, the most popular micro blog platform in China with about 600 million users (also a Chinese equivalent to Twitter). 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. Yandex is the largest Russian search engine, it uses its Captcha in user password recovery mechanism. Yhd is a popular online shopping website in China.

Second, these schemes are with distinct design features, and represent a range of different designs. For example, some schemes use interference arcs (e.g. CmPay, Baidu, and Yandex); others do not. Except for Yahoo! which uses character strings of a varied length, others all use a fixed string length. Some schemes (e.g. Yahoo! and Yhd) 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 in all the Captchas above is that all characters are presented as hollow objects. Generally speaking, hollow schemes seem to be a clever idea. Crowding Characters Together (CCT), firstly proposed by Google, has been widely adopted. This standard security mechanism for text Captchas improves security but has usability issues. For example, confusing character pairs will appear when characters are crowded together too much and it is hard for people to recognize them [2]. 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 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 extract character strokes or components from hollow Captchas and convert them to solid ones. As for some schemes whose contour lines are broken, this process is not straightforward; we need to automatically repair them first. Furthermore, standard methods like Color Filling Segmentation (CFS, introduced in [4]) 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. Note that character strokes extracted this way are not linked with each other. Instead, they are scattered around.

Next, we try different combinations of adjacent strokes, and use convolutional neural network (CNN) [17-19] recognition engine to determine which character a combination most likely to be. It has been already shown that CNN can work on classification [29-30]. 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.

The high-level workflow of our attack includes three main sequential steps:

- Pre-processing, processes each challenge image with standard techniques, like image binarization and contour lines repairing.
- Extracting character strokes, relies on a number of techniques, such as using CFS to fill hollow parts, noise component removal, and contour line removal and clean up.

- Segmentation and recognition, which use CNN engine assisted graph search.

Note the sub procedures listed may not necessary in some schemes.

5 Our Attack: Technical Details

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 [20]. The first step of Figure 2 (a) to Figure (g) show the binarized images.

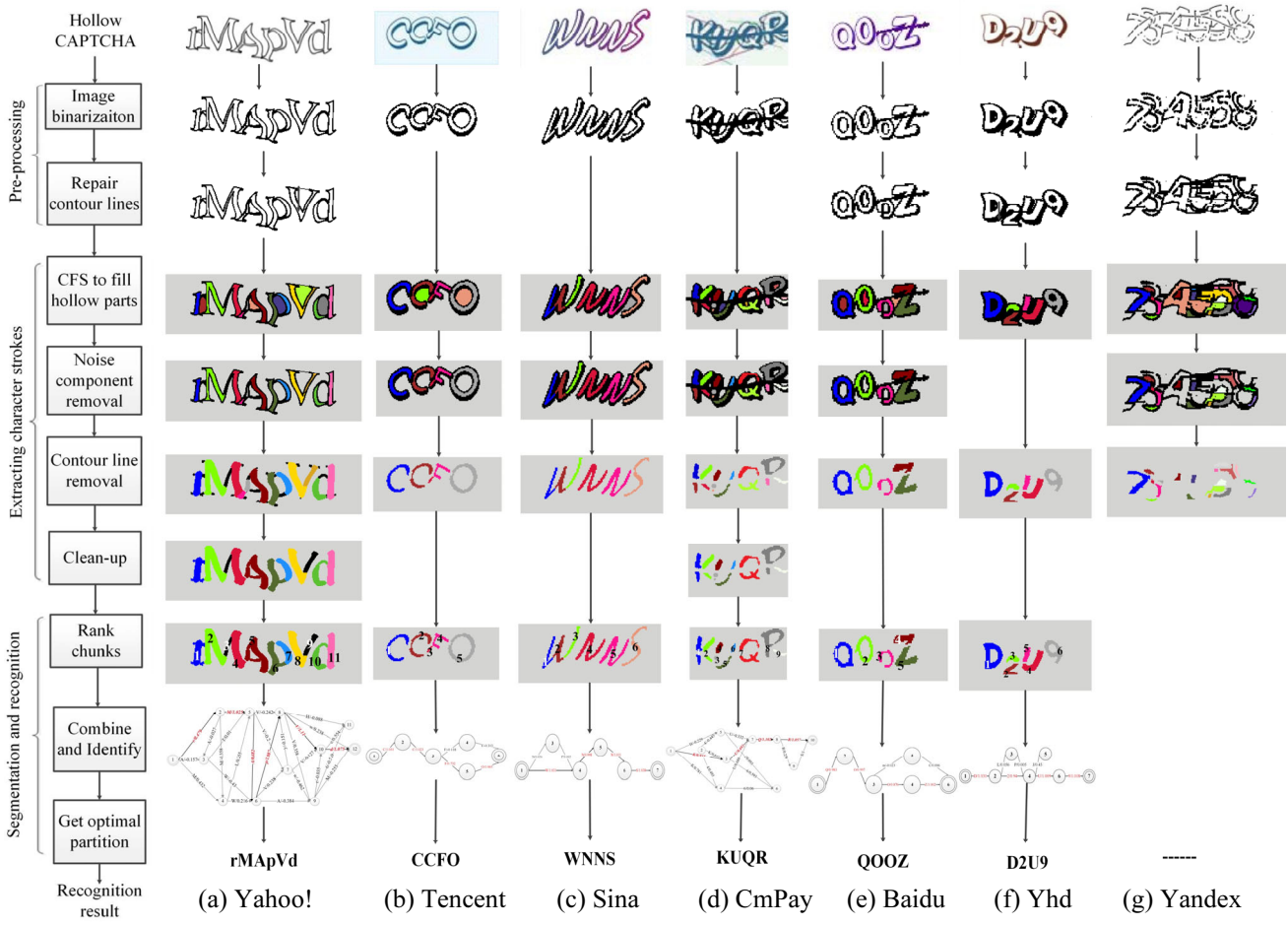


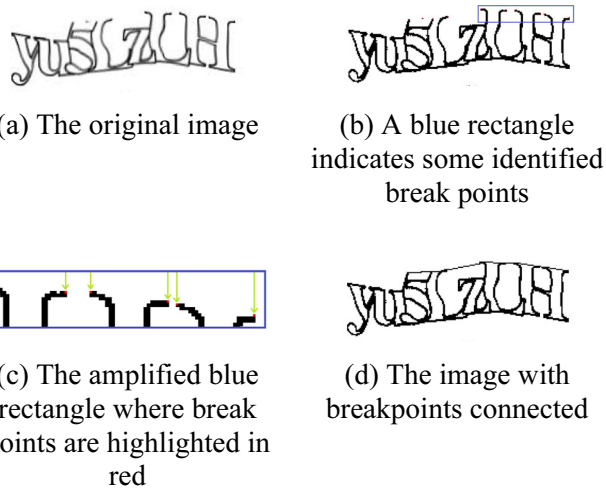
Figure 2. The pipeline of our attack

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, for which CFS will fail filling the hollow parts. In some cases, binarization created broken contours, too. In order to make CFS work, it's necessary to repair these broken contour lines. Lee's algorithm [21], which was initially designed for solving maze routing problems, is used to automatically detect and then repair broken contours (Figure 3).

We take Yahoo! scheme as an example. First, we identify break points in contour lines. Just like solving the maze 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 3 (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 3 (d)).

Incorrect connections may be created, e.g. Figures 3(d), but our attack can cope with them, as explained later. The second step of Figure 2 shows the images after repairing contour lines. Note that this step is only necessary for Yahoo!, Baidu, Yhd and Yandex.



(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

Figure 3. Repair contour lines

5.2 Extracting Character Strokes

Fill hollow parts with CFS. CFS 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 so that readers can easily distinguish them from each other. After this step, the background color in an image is set to light gray, but contour lines and other solid parts such as thick interference arcs remain in black. The third step of Figure 2 shows the images after CFS.

Noise component removal. In Figure 4, 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.

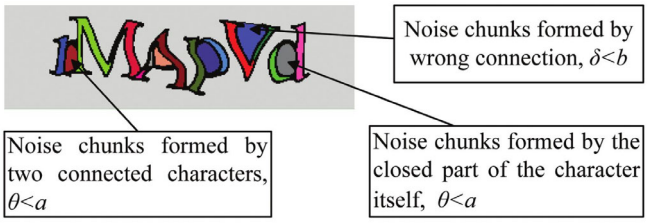


Figure 4. Three types of noise components

We first define parameters θ and δ for each color component: θ = *Cg/C* and δ= *C/S*, where *Cg* denotes the number of edge pixels that have a light gray 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 θ < *a* for a component, it is a noise component of the first two types; if δ< *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 hollow schemes, and it works on most of them (except for Yandex). Components surviving our noise removal are considered to belong to character bodies, and are thus preserved.

Contour line removal. In most schemes except for Yahoo!, black pixels are the interference arcs and contour lines, and all character strokes are in non-black colors after removing the noise components. So it is straightforward to remove both contour lines and the interference arcs by switching all black pixels to the background color.

However, it requires more subtle techniques for the Yahoo! scheme. We have also implemented the more subtle approaches. Which approach is needed for handling a particular Captcha scheme can be automatically determined. 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 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, as illustrated in fifth step of Figure 2.

Clean-up. Tiny pixel blocks might be created by contour line removal (Figure 2(a)). In clean-up, they are either removed directly or merged with adjacent larger strokes via an automated algorithm. The sixth step of Figure 2 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. 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 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. This step is the same for all the schemes. Here we use Yahoo! and CmPay as examples.

First, all strokes in an image are numbered in an incremental order from the upper left to lower right (Figure 5).

(a) Yahoo! (b) CmPay

Figure 5. What remains are all character strokes, rank ordered

Then we attempt to combine strokes or components following the incremental rank order. An $n \times n$ table is built for each image to record whether a combination is legitimate to form an individual character, where n is the total number of strokes in the image.

If it is infeasible to combine strokes $i, i+1, ..., k$ altogether to form a single character, the cell at the intersection of row i and column k in the table (cell $(i; k)$) will be set to NULL. This occurs when its row index i is larger than its column index k (we try combinations only in a monotonic order), or the width of the combination is greater than the largest possible character width, or less than the smallest possible character width, which is also empirically established with a simple analysis of the sample set.

If such a combination is feasible, 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.

In our implementation, an image input to the CNN is normalized to the size of 28×28 pixels; 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 [17], 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.

Table 1 and Table 2 show the $n \times n$ table built for the Yahoo! sample and the CmPay sample, respectively. For example, in Table 1, the cell (1, 1) indicates that the CNN recognize this single stroke as '*r*' with a confidence level of 0.479; the cell (1, 3) indicates that the combination of strokes 1, 2 and 3 is recognized as '*M*' with a confidence level of 0.432. Each empty cell $(i; k)$ indicates that a combination of strokes $i, i + 1, ..., k$ is infeasible for one reason or another.

Table 1. The $n \times n$ table generated by CNN for the Yahoo! sample in Figure 5

	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											

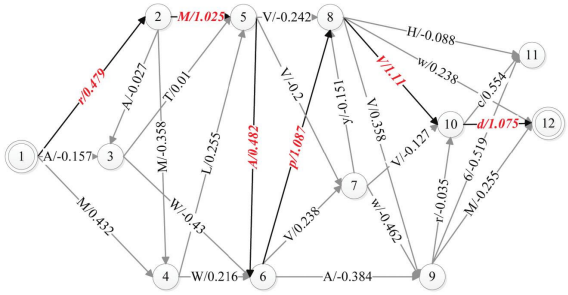
Table 2. The $n \times n$ table generated by CNN for the CmPay sample in Figure 5

	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					6/0.006	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

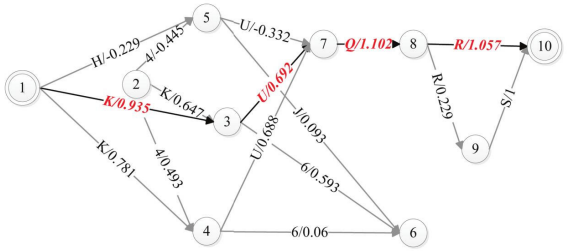
A $n \times n$ table gives all plausible stroke combinations for an image. Our task now is to use information in the table to find the most likely way of forming characters, i.e., finding the best segmentation or partition.

We convert each $n \times n$ table to an equivalent directed and weighted graph. Figure 6 gives such graphs that are equivalent to Tables 1 and 2, respectively. Each non-empty cell $(i; j)$ in the table is represented as an arc $< i; j+1 >$ (i.e. a directed edge linking vertexes i and $j+1$) in the graph, and the associated weight on the edge gives both the recognition result of this combination and the confidence level calculated by the neural network for this result. In each graph, the nodes are numbered from 1 to $n + 1$, and nodes $1,2,...n$ represent the corresponding strokes, respectively. Clearly, the

number of arcs in a graph equals to the number of non-empty cells in its corresponding table.



(a)



(b)

Figure 6. The equivalent graphs of Table 1 and Table 2

Now, we search the graph to find an optimal partition. We define the target problem is to select a path as following:

- 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.

In our previous work [22], we used Depth-First-Search (DFS) algorithm to get the final result. It starts from node 1 in the graph and explores along each branch until the path length reaches the Captcha string length before backtracking. All paths of a length equaling to the Captcha string length in a graph are traversed using DFS, and then the path ending at the *n+1* node with the largest confidence level sum is selected.

DepthFirstSearch()
1 $i \leftarrow 1$
2 $step \leftarrow 0$
3 $p \leftarrow 0$
4 $sum \leftarrow 0$
5 $R \leftarrow NIL$
6 $S \leftarrow NIL$
7 *Traverse*(i, $step$, sum, S)
8 *print* R

Traverse(*i*, *step*, *sum*, *S*)
1 **for** $j \leftarrow i$ *to* n
2 **do if** $a_{i,j} \neq NULL$ **then**
3 $sum \leftarrow sum + a_{i,j}$
4 $S \leftarrow strcat(S, S_{i,j})$
5 $step \leftarrow step + 1$
6 **else**
7 **if** $step \in Captcha\ \ length$ **and** $p < sum$ **then**
8 $p \leftarrow sum$
9 $R \leftarrow S$
10 **if** $step \leq max\ Captcha\ length$ **then**
11 *Traverse*($j + 1$, $step$, sum, S)

This DFS algorithm is not optimal, since it will both explore paths that cannot reach the last node of the graph and will re-explore previously visited nodes after their best following partition has been discovered.

Here we introduce a new algorithm, which uses an Integer Partition (IP) algorithm to select the optimal partition which has the highest confidence sum. It is also a graph search algorithm, but better than DFS [22]. This algorithm reduces the search space by skipping paths that do not end at the n+1 node.

The rationale is the following. Assume that m is the Captcha length, our task is to find the most likely way of forming m characters using n components, i.e., finding the best partition. This task is similar to the classical integer partition problem: in number theory and combinatorics, a partition of a positive integer n, is a way of representing n as a sum of m positive integers. We first work out all partitions that divide integer n into m parts, then select the partition with the largest sum of confidence levels.

The length of Captcha strings is 6 to 8 in Yahoo! scheme, 6 in Yandex scheme and 4 in Tencent, Sina, CmPay, Baidu and Yhd schemes.

We still 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. In the partition, each part denotes the number of components to be combined. For example, for the sample CmPay challenge in Figure 5, a candidate partition of the integer is 9=2+4+1+2 (n=9, m=4), which indicates the partition of components is 1 ~ 2('*K*'/ 0.935), 3 ~ 6('*U*'/ 0.692), 7 ~ 7('*Q*'/ 1.102), 8 ~ 9('*R*'/ 1.057). That is to say, each part in the partition is corresponding to a cell in the $n \times n$ table generated by CNN. While in the procedure of figuring out all the partitions, if a part in the partition is invalid (the corresponding cell in the table is null), this partition is ignored directly.

The pseudo code in the below sketches the

recognition process, with key variables defined as follows.

$S_{i,j}$: the character recognized by the CNN as the combination of components *i, i+1,· · ·, j*
$a_{i,j}$: the confidence level of $S_{i,j}$
p : The parts that has been partitioned
index : The length of *p*
s : The sum of all the elements in *p*
sum: the confident value sum of a partition
S: the recognition result of this partition
R: the final result string
V : the confident value sum of R

Recognition()
1 $R \leftarrow NIL$, $v \leftarrow -99$, $num \leftarrow n$
2 **foreach** $m \in$ *Captcha length*
3 **do** *length*(*p*) ← *m*
4 *Partition*(*num*, *m*, 0)
5 *Print R*

Partition(*num*, *m*, *index*)
1 **if** $m \neq 1$ **then**
2 **for** *j* ← 1 to *num* – 1
3 **do** *s* ← 0
4 **for** *jj* ← 0 to *index* − 1
5 **do** *s* ← *s* + *p*[*jj*]
6 **if** $a_{s+1,\, s+j} \neq NULL$ **then**
7 *p*[*index*] ← *j*
8 *Partition*(*num*−*j*, *m*−1, *index*+1)
9 **else**
10 *s* ← 0
11 **for** *jj* ← 0 to *index* − 1
12 **do** *s* ← *s* + *p*[*jj*]
13 **if** $a_{s+1,\, n} \neq NULL$ **then**
14 *p*[*index*] ← *n* − *s*
15 *Select*(*length*(*p*))

Select(*m*)
1 $S \leftarrow NIL$, $sum \leftarrow 0$, $s \leftarrow 0$
2 **for** *j* ← 0 to *m* − 1
3 **do** *s* ← *s* + *p*[*j*]
4 $S \leftarrow strcat(S, S_{s+1,\, s+p[j]})$
5 $sum \leftarrow sum + a_{s+1,\, s+p[j]}$
6 **if** *sum* > *v* **then**
7 *v* ← *sum*
8 *R* ← *S*

As generated by our attack program, Table 3 and Table 4 show all likely partitions and for each partition, its results and 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.

Table 3. Each partition and the corresponding result for the Yahoo! sample in Figure 5

Partitions	Result	Confidence	Partitions	Result	Confidence	Partitions	Result	Confidence	Partitions	Result	Confidence
1+1+2+1+2+4	rATApw	2.15	1+3+2+1+1+3	rMVJVM	1.26	1+1+2+1+2+3	rATAVwM	.45	2+2+1+1+1+3	ATAVJVM	0.44
1+1+2+1+3+3	rATAAM	0.32	1+3+2+1+2+2	rMVJVd	3.33	1+1+2+1+3+2	rATAVVd	2.06	2+2+1+1+2+2	ATAVJVd	2.51
1+1+2+2+1+4	rATVJw	0.19	1+3+2+2+1+2	rMVwrd	1.97	1+1+2+2+1+3	rATApVM	2.10	2+2+1+2+1+2	ATAVwrd	1.16
1+1+2+2+2+3	rATVwM	-0.35	1+3+3+1+1+2	rMVVrd	2.69	1+1+2+2+2+2	rATApVd	4.17	2+2+1+1+1+2	ATApVrd	2.81
1+1+2+2+3+2	rATVVd	1.26	2+2+1+1+1+4	ATAWJw	0.49	1+1+2+3+1+2	rATAArd	1.52	2+2+2+1+1+2	ATVJVrd	0.85
1+1+2+3+1+3	rATVVM	0.37	2+2+1+1+2+3	ATAVwM	-0.05	1+1+2+1+1+3	Ratvjvm	0.14	2+3+1+1+1+2	AVVJVrd	0.81
1+1+2+3+2+2	rATVVd	2.44	2+2+1+1+3+2	ATAVVd	1.56	1+1+2+1+2+2	rATVJVd	2.21	3+1+1+1+1+3	MLAVJVM	1.18
1+1+3+1+1+4	rAVVJw	0.15	2+2+1+2+1+3	ATApVM	1.61	1+1+2+2+1+2	rATVwrd	0.85	3+1+1+1+2+	MLAVJVd	3.25
1+1+3+1+2+3	rAVVwM	-0.39	2+2+1+2+2+2	ATApVd	3.67	1+1+2+1+1+2	rATVVrd	1.57	3+1+1+2+1+2	MLAVwrd	1.89
1+1+3+1+3+2	rAVVVd	1.22	2+2+1+3+1+2	ATAArd	1.03	1+1+2+1+1+3	rAVVJVM	0.10	3+1+1+1+1+2	MLApVrd	3.54
1+1+3+2+1+3	rAVpVM	1.26	2+2+2+1+1+3	ATVJVM	-0.36	1+1+3+1+2+2	rAVVJVd	2.17	3+1+2+1+1+2	MLVJVrd	1.58
1+1+3+2+2+2	rAVpVd	3.33	2+2+2+1+2+2	ATVJVd	1.71	1+1+3+2+1+2	rAVVwrd	0.81	3+2+1+1+1+2	MwVJVrd	1.52
1+1+3+3+1+2	rAVArd	0.68	2+2+2+2+1+2	ATVwrd	0.35	1+1+3+1+1+2	rAVpVrd	2.47	1+1+2+1+1+1+3	rATAVJVM	0.94
1+2+1+1+2+4	rMLApw	2.15	2+2+3+1+1+2	ATVVrd	1.074	1+1+3+1+1+4	rMLAVJw	0.99	1+1+2+1+1+2+2	rATAVJVd	3.01
1+2+1+1+3+3	rMLAAM	0.32	2+3+1+1+1+3	AVVJVM	-0.39	1+2+1+1+2+3	rMLAVwM	0.45	1+1+2+1+2+1+2	rATAVwrd	1.65
1+2+1+2+1+4	rMLVJw	0.19	2+3+1+1+2+2	AVVJVd	1.67	1+2+1+1+3+2	rMLAVVd	2.06	1+1+2+1+1+1+2	rATApVrd	3.31
1+2+1+2+2+3	rMLVwM	-0.35	2+3+1+2+1+2	AVVwrd	0.32	1+2+1+2+1+3	rMLApVM	2.11	1+1+2+2+1+1+2	rATVJVrd	1.34
1+2+1+2+3+2	rMLVVd	1.26	2+3+2+1+1+2	AVpVrd	1.97	1+2+1+2+2+2	rMLApVd	4.18	1+1+3+1+1+1+2	rAVVJVrd	1.31
1+2+1+3+1+3	rMLVVM	0.37	3+1+1+1+1+4	MLAVJw	1.22	1+2+1+3+1+2	rMLAArd	1.53	1+2+1+1+1+1+3	Rmlavjvm	0.95
1+2+1+3+2+2	rMVVd	2.44	3+1+1+1+2+3	MLAVwM	0.68	1+2+1+1+1+3	rMLVJVM	0.145	1+2+1+1+1+2+2	rMLAVJVd	3.02
1+2+2+1+1+4	rMwVJw	0.13	3+1+1+1+3+2	MLAVVd	2.29	1+2+1+1+2+2	rMLVJVd	2.21	1+2+1+1+2+1+2	rMLAVwrd	1.66
1+2+2+1+2+3	rMwVwM	-0.41	3+1+1+2+1+3	MLApVM	2.34	1+2+1+2+1+2	rMLVwrd	0.86	1+2+1+1+1+1+2	rMLApVrd	3.31
1+2+2+1+3+2	rMwVVd	1.20	3+1+1+2+2+2	MLApVd	4.41	1+2+1+1+1+2	rMLVVrd	1.58	1+2+1+2+1+1+2	rMLVJVrd	1.35
1+2+2+2+1+3	rMwpVM	1.25	3+1+1+3+1+2	MLAArd	1.76	1+2+2+1+1+3	rMwVJVM	0.088	1+3+2+1+1+1+2	rMwVJVrd	1.29
1+2+2+2+2+2	rMwpVd	3.32	3+1+2+1+1+3	MLVJVM	0.37	1+2+2+1+2+2	rMwVJVd	2.16	1+3+1+1+1+1+2	rMAVJVrd	3.27
1+2+2+3+1+2	rMwArd	0.67	3+1+2+1+2+2	MLVJVd	2.44	1+2+2+2+1+2	rMwVwrd	0.80	2+2+1+1+1+1+2	ATAVJVrd	1.65
1+3+1+1+1+4	rMAVJw	2.11	3+1+2+2+1+2	MLVwrd	1.09	1+2+2+1+1+2	rMwpVrd	2.45	3+1+1+1+1+1+2	MLAVJVrd	2.38
1+3+1+1+2+3	rMAVwM	1.57	3+1+3+1+1+2	MLVVrd	1.81	1+3+1+1+1+3	rMAVJVM	2.06			
1+3+1+1+3+2	rMAVVD	3.18	3+2+1+1+1+3	MwVJVM	0.317	1+3+1+1+2+2	rMAVJVd	4.13			
1+3+1+2+1+3	rMApVM	3.23	3+2+1+1+2+2	MwVJVd	2.39	1+3+1+2+1+2	rMAVwrd	2.78			
1+3+1+2+2+2	***rMApVd***	***5.29***	3+2+2+1+1+2	MwpVrd	2.68	1+3+1+1+1+2	rMApVrd	4.43			
1+3+1+3+1+2	rMAArd	2.65	3+2+1+2+1+2	MwVwrd	1.03	1+3+2+1+1+2	rMVJVrd	2.47			

Table 4. Each partition and the corresponding result for the CmPay sample in Figure 5

Partitions	Result	Confidence
4+2+1+2	HUQR	1.598
3+3+1+2	KUQR	3.628
2+4+1+2	***KUQR***	***3.786***

Note that the techniques we used are generically applicable to all the hollow schemes. Figure 2 includes the pipeline of our attack on seven schemes.

6 Evaluations

6.1 Analysis of Our Attack

We have implemented our attack and tested it on all the 7 hollow schemes. We present our evaluations as follows.

Data collection. For each of the schemes, we collected 1000 random Captchas as a sample set by mining from the corresponding websites, and another 500 as a test set.

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, 3333 characters from Yhd samples, and 4115 characters from Yandex samples to train the CNN.

Success rate. We test our attack both on the sample set and the test set of the all schemes. The success rates are listed in Table 5. For the Yahoo! scheme, we achieved a success rate of 56% on the sample set. Then we ran our attack on the test set, about which our program had no prior knowledge about any particular sample within; we achieved a success rate of 36%. Similarly, 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. For Yhd, our success was 36% on the sample set and 35% on the test set. But for Yandex, our approach does not work (we will discuss it later).

Table 5. The success rate and speed of our attack

Scheme	Success on sample set	Success on test set(T)	Avg time per challenge		Avg time per success(T/R)	
			DFS	IP	DFS	IP
Yahoo!	56%	36%	5.30s	5.17s	14.72s	14.36s
Tencent	93%	89%	1.23s	1.14s	1.38s	1.28s
Sina	63%	59%	1.77s	1.36s	3.00s	2.31s
CmPay	73%	66%	4.25s	3.70s	6.43s	5.61s
Baidu	57%	51%	3.87s	3.52s	7.58s	6.91s
Yhd	36%	35%	0.56s	0.49s	0.78s	0.69s
Yandex	0	0	-	-	-	-

A commonly accepted goal for Captcha robustness is to prevent automated attacks from achieving a success rate of higher than 0.01% [4]. But this goal is considered too ambitious by some researchers. [15] suggested that a Captcha scheme is broken when the attacker achieves an accuracy rate of at least 1%. According to either criterion, six 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 5 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 1 to 15 seconds. Clearly, our attack is efficient and poses a realistic threat to all the hollow schemes.

Table 5 also compares the speed of precious DFS algorithm and our IP search algorithm. It is clearly that our IP Search algorithm is better than the previous DFS algorithm in attack speed, because the IP algorithm reduces the search space. With the same success rates, our attack speeds on Yahoo!, Tencent, Sina, CmPay and Baidu in table 5 are all faster than before. Specifically, the time consumption of Yahoo!, Tencent, Sina, CmPay, Baidu and Yhd was decreased by 2.5%, 5.7%, 23.2%, 12.9%, 9.0% and 12.5%.

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.

6.2 Faliures of Our Attack

The experiments show our attack is general for attacking most hollow schemes, except for Yandex. For Yandex scheme, there are two main reasons for the failure results. See Figure 2(g). First, the contour lines are seriously broken, and there are many break points marked after using Lee's algorithm [21]. Second, the interference arcs are similar to the contour line. Badly broken and partially overlap with the character strokes, make it especially difficult to segment.

Besides, we also test our approach on some non-hollow schemes and find it not suitable for these schemes without change. We test our approach on Amazon and Yahoo! non-hollow Captchas, the success rate are all 0%, as the key component of our attack is using CFS to extract character strokes from hollow

fonts. The solid stroke means we can't use CFS to extract the stroke components. The only few color chunks we extracted are the noise chunks (Figure 7).

(a) Amazon (b) Yahoo!

Figure 7. Non-hollow captchas using CFS

7 Discussions

7.1 Novelty

To our best knowledge, this is the most detailed 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 [4]. However, the key component of our attack, the graph search algorithm based segmentation and recognition, is 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 [15], [3] and [27], do not work on the hollow Captchas studied in this paper. On the other hand, hollow Captchas are converted into CCT schemes after filling hollow part with CFS. Admittedly, it is possible to use those anti-CCT techniques to break the filled hollow Captchas. However, it also introduces the issue of segmentation which is the major challenge of breaking Captchas, as [28] suggested that the robustness of text-based Captchas should rely on the difficulty of finding where each character is (segmentation). After extracting character strokes, our method has segment adjacent characters totally, but each character is also separated into a few components. It is easy for our IP algorithm to find the best combinations of the extracted character components. In a word, our method is more simple and effective than using anti-CCT algorithms after CFS.

Among all the related work we have discussed earlier, two attacks have some similarity with ours. One is the attack reported in [14] on a MegaUpload Captcha. 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. The other is [6], their idea of combining segmentation and recognition in a single step is somewhat like ours. But different from ours, they use a brute-force similar approach with five steps included to look for possible segmenting points.

The most recent attack reported in [26] also analyzed two hollow Captcha schemes, Yahoo! and QQ (Tencent), which are same with the Yahoo! and Tencent schemes presented in this paper. It utilized Gabor filters to extract character components. However, this method doesn't work well on hollow Captchas formed by thin contour lines. The Yahoo! scheme received the lowest success rate (5.0%) in [26] since its extraction method breaks a thin text string into a large number of tiny components, which produces a huge possible set of combination, whereas ours achieved a much higher success rate of 36%. Even for Tencent which is formed by thick contour lines, our approach is also much effective than theirs (89% vs. 56%).

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 and Yhd); 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 6 summarizes the main features of these schemes. As they represent different designs, each with distinctive features, our attack is of some generic value.

Table 6. Main features of seven hollow Captchas

Scheme	Interference Arcs	Broken Contour Line	Contour Line Thickness	Hollow Styles	String Length	Character Overlap	Aplhabet 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
Yhd	No	Yes	Varied	Uniform	Fixed (4)	Yes	32
Yandex	Yes	Yes	Uniform	Uniform	Fixed (6)	Yes	10

C. Lessons

We also analyze 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 7 schemes.

String length matters. This was first observed in [4] and then confirmed in [15]. 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 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. Our failure on Yandex, which formed by seriously broken contour lines, is a good explanation. 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.

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" (Figure 8). 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.

(a) thick strokes vs (b) thin strokes (both after CFS)

Figure 8. Yahoo!'s hollow styles

Having multiple designs and deploying them alternately in a random order is good for security, as first suggested in [23] and then confirmed in [15]. 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 [24] contour lines to a uniform thickness is useful for our attack, but not essential. Therefore, we do not consider variations in contour thickness contribute much to security.

Thin interference arcs, which are easily removed by binarization. Note that 'thin' here means the arcs are much thinner than the character contour lines.

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 "Do" and "Don't" 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.

Apparently, Yandex is the best among all the 7 schemes. Overlapped, seriously broken contour lines, interference arcs and variations in the width of individual characters width, all make great contributions to its unique strength.

The Yahoo! scheme is the most time consuming among all the 6 broken schemes, and the following features contribute to it: (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. To the contrary, Tencent is the worst design, as it made the worst choices for almost all security features.

7.3 Towards Better Designs

From our attack process and the above lessons, we can see that Yandex scheme has the most security mechanism, and Yahoo! scheme, although broken, have some good security features. Yhd scheme, though not recommended, provides a new thought of security mechanism. We will follow their success design and explore how to evolve these schemes into better designs. Plausible options include the following.

Using broken contours more often. In Yandex, Yahoo! and Yhd, the broken contours are used. Among them, the Yandex scheme used this design in all of its challenges. But not all of the Yahoo! and Yhd

challenges contain broken contours. Using broken contours is meaningful, 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 interference arcs which similar to character strokes to cut through characters so that when adjacent strokes are combined for tests, combined characters contain extra strokes 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 alphabet size. This is a simple but effective solution, and with little negative impact on usability (if confusing characters are excluded from the alphabet).

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 more possible combinations it will produce, and the larger a solution set our graph search algorithm is required to go through. This might significantly slow down our attack and decrease the attack success rate.

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.

Careful studies are still 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

8.1 About Hollow Schemes

We proposed a simple attack on hollow Captchas and have shown that this new type of text scheme has serious security problems. With an effective attack, we have broken the hollow schemes deployed by Yahoo!, Tencent, Sina, CmPay, Baidu and Yhd with a success rate of 36%, 89%, 59%, 66%, 51% and 35% respectively, and it only takes just seconds for our attack to break each of the scheme on a standard desktop computer. 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 is not as secure as expected. 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.

Our attack helped to identify good design features for better security by comparing representative designs of popular hollow Captchas, 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.

8.2 Application Extension

Our attack inherently leverages the components information to segment the Captchas. As our segmentation algorithms are designed for hollow schemes, our attack is only suitable for hollow schemes specifically. The attack cannot be directly applied to attack non-hollow schemes, but the attack procedure reveals useful insights, find a generic method to get the character components' information, whether it is hollow or non-hollow, and then get the best partition.

Specifically, the segmentation is not limited to segmenting between characters. If we can find a method that can divide a Captcha challenge into many components, then we can use this information to reconstruct the character, and then find the most likely string.

After the traditional segmentation algorithms failed in separating the hollow characters apart, we use the CFS algorithms to get the strokes of each character successfully. By analyzing all possible combinations, we get the most likely to be the correct one. So, for the non-hollow characters, if we find a method to get the information of each character, then our approach will likely be extended to non-hollow schemes.

What's more important, the attack approaches are no longer confined to the segment then recognize approach. The boundaries between segmenting and recognizing become fuzzy.

8.3 Future Work

Given the practical relevance and intellectual interest of the hollow Captcha technology, we have proved the insecurity of hollow Captchas. However, there have emerged many other Captchas with sophisticated design features, e.g. complex background,

using both hollow and solid characters and two-layer Captchas. It is therefore natural to ask an essential question: Are these Captchas as secure as their designers expected? This is our ongoing work.

On the other hand, a lot of text-based Captchas had have been broken, especially some state-of-art works claimed that they can attack a variety of text-based Captchas deployed in the wild via a single step method [26]. It is clearly the common practice in text-based Captcha designs is dubious and shaking. The increasing insecurity of text-based Captchas creates a radical question: Can text Captchas still take the responsibility of Internet security? This is an open problem we share with whole research communities.

Acknowledgements

We thank anonymous reviewers for helpful comments. This project is supported by the National Natural Science Foundation of China (61472311) and the Fundamental Research Funds for the Central Universities.

References

[1] L. Von Ahn, M. Blum, J. Langford, Telling Humans and Computers Apart Automatically, *Communications of the ACM,* Vol. 47, No. 2, pp. 56-60, February, 2004.

[2] J. Yan, A. S. E. Ahmad, Usability of Captchas or Usability Issues in Captcha Design, *Proceedings of the 4th Symposium on Usable Privacy and Security*, Pittsburgh, PA, 2008, pp. 44-52.

[3] A. S. El Ahmad, J. Yan, M. Tayara, The Robustness of Google CAPTCHAs, *Computing Science Technical Report CS-TR-1278*, September, 2011.

[4] J. Yan, A. S. El Ahmad, A Low-cost Attack on a Microsoft Captcha, *Proceedings of the 15th ACM Conference on Computer and Communications Security*, Alexandria, VA, 2008, pp. 543-554.

[5] Vicarious Inc, *Vicaricous- Turing Test 1: Captcha*, http://vimeo.com/77431982.

[6] E. Bursztein, J. Aigrain, A. Moscicki, J. C. Mitchell, The End is Nigh: Generic Solving of Text-based Captchas, *8th USENIX Workshop on Offensive Technologies(WOOT 14)*, San Diego, CA, 2014, pp. 1-15.

[7] M. Naor, *Verification of A Human in the Loop or Identification via the Turing Test*, http://www.wisdom.weizmann.ac.il/ ˜naor/PAPERS/humanabs.html.

[8] M. D. Lillibridge, M. Abadi, K. Bharat, A. Z. Broder, Method for Selectively Restricting Access to Computer Systems, Feb. 27 2001, *US Patent* 6,195,698B1.

[9] G. Mori, J. Malik, Recognizing Objects in Adversarial Clutter: Breaking A Visual Captcha, *Proceedings of 2003 IEEE Computer Society Conference* on Computer Vision and Pattern Recognition, Vol. 1, Madison, WI, 2003, pp. I-134-I-141.

[10] G. Moy, N. Jones, C. Harkless, R. Potter, Distortion Estimation Techniques in Solving Visual Captchas, *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition,* Vol. 2, Washington, DC, 2004, pp. II-23-II-28.

[11] K. Chellapilla, P. Y. Simard, Using Machine Learning to Break Visual Human Interaction Proofs (Hips), *Advances in Neural Information Processing Systems* 17 (NIPS 2004), Vancouver, Canada, 2005, pp. 265-272.

[12] J.-S. Lee, M.-H. Hsieh, Preserving User-participation for Insecure Network Communications with CAPTCHA and Visual Secret Sharing Technique, *IET Networks,* Vol. 2, No. 2, pp. 81-91, June, 2013.

[13] J. Yan, A. S. El Ahmad, Breaking Visual Captchas with Naive Pattern Recognition Algorithms, *Twenty-Third Annual Computer Security Applications Conference,* Miami Beach, FL, 2007, pp. 279-291.

[14] A. S. El Ahmad, J. Yan, L. Marshall, The Robustness of A New Captcha, *Proceedings of the Third European Workshop on System Security*, Paris, France, 2010, pp. 36-41.

[15] E. Bursztein, M. Martin, J. Mitchell, Text-based Captcha Strengths and Weaknesses, *Proceedings of the 18th ACM Conference on Computer and Communications Security*, Chicago, IL, 2011, pp. 125-138.

[16] Y. Xu, G. Reynaga, S. Chiasson, J.-M. Frahm, F. Monrose, P. Van Oorschot, Security and Usability Challenges of Moving-object Captchas: Decoding Codewords in Motion, in *21st USENIX Security Symposium*, Bellevue, WA, 2012, pp. 1-16.

[17] Y. LeCun, L. Bottou, Y. Bengio, P. Haffner, Gradient-based Learning Applied to Document Recognition, *Proceedings of the IEEE*, Vol. 86, No. 11, pp. 2278-2324, November, 1998.

[18] S. D. Boutas, L. E. Anagnostopoulos, V. Loumos, E. Kayafas, An Intelligent Web Recommendation System for Ubiquitous Geolocation Awareness, *International Journal of Ad Hoc and Ubiquitous Computing*, Vol. 14, No. 1, pp. 1-15, September, 2013.

[19] P. Y. Simard, D. Steinkraus, J. C. Platt, Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis, *Seventh International Conference on Document Analysis and Recognition*, Edinburgh, UK, 2003, pp. 958-963.

[20] N. Otsu, A Threshold Selection Method from Gray-level Histograms, *IEEE Transactions on Systems, Man, and Cybernetics*, Vol. 9, No. 1, pp. 62-66, January, 1979.

[21] J. H. Hoel, Some Variations of Lee's Algorithm, *IEEE Transactions on Computers*, Vol. C-25, No. 1, pp. 19-24, January, 1976.

[22] Hai-Chang Gao, W. Wang, J. Qi, X. Wang, X. Liu, J. Yan, The Robustness of Hollow Captchas, *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security,* Berlin, Germany, 2013, pp. 1075-1086.

[23] B. B. Zhu, J. Yan, Q Li, C. Yang, J. Liu, N. Xu, M. Yi, K. Cai, Attacks and Design of Image Recognition Captchas, *Proceedings of the 17th ACM Conference on Computer and Communications Security*, Chicago, IL, 2010, pp. 187-200.

[24] T. Y. Zhang, C. Y. Suen, A Fast Parallel Algorithm for Thinning Digital Patterns, *Communications of the ACM*, Vol. 27, No. 3, pp. 236-239, March, 1984.

[25] C. Hong, B. Lopez-Pineda, K. Rajendran, A. Recasens, *Breaking Microsoft's CAPTCHA*, https://courses.csail.mit.edu/6.857/2016/files/hong-lopezpineda-rajendram-recansens.pdf

[26] H. Gao, J. Yan, F. Cao, Z. Zhang, L. Lei, M. Tang, A Simple Generic Attack on Text Captchas, *Proceedings of Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, 2016, pp.1-14.

[27] H. Gao, W. Wang, Y. Fan, J. Qi, X. Liu, The Robustness of "Connecting Characters Together" Captchas, *Journal of Information Science and Engineering,* Vol. 30, No. 2, pp. 347-369, March, 2014.

[28] K. Chellapilla, K. Larson, P. Y. Simard, M. Czerwinski, Computers Beat Humans at Single Character Recognition in Reading based Human Interaction Proofs (Hips), *CEAS 2005 - Second Conference on Email and Anti-Spam*, Stanford CA, 2005, pp.1-8.

[29] Y. L. Cun, B. Boser, J. S. Denker, R. E. Howard, W. Habbard, L. D. Jackel, D. Henderson, Handwritten Digit Recognition with a Back-propagation Network, *Advances in Neural Information Processing Systems 2*, Denver, CO, 1989, pp. 396-404.

[30] T. A. Rashid, A. L. Jabar, Improvement on Predicting Employee Behaviour through Intelligent Techniques, *IET Networks*, Vol. 5, No. 5, pp. 136-142, September, 2016.

Biographies

Haichang Gao is a professor in Xidian University and a member of the IEEE. He has published more than thirty papers. Now he is in charge of a project of the National Natural Science Foundation of China. His current research interests include CAPTCHA, computer security and machine learning.

Ping Wang is a doctoral degree candidate in School of Computer Science and Technology at Xidian University. Her current research interests are CAPTCHA and authentication.

Jeff Yan is a professor in Department of Computer and Information Science, Linköping University. He interested in most aspects of computer and network security, both theoretical and practical, and his recent work focuses on systems security, including human aspects of security (e.g. usable security). His previous contributions illustrate both his view of security and research methodology.

Mengyun Tang is a master degree candidate in computer science at Xidian University. Her current research interest is CAPTCHA.

Fang Cao is a master in computer science at Xidian University. His current research interest is CAPTCHA.