Research on the Security of Microsoft's Two-layer Captcha

Haichang Gao, *Member, IEEE*, Mengyun Tang, Yi Liu, Ping Zhang and Xiyang Liu, *Member, IEEE*

***Abstract*—Captcha is a security mechanism designed to differentiate between computers and humans, and is used to defend against malicious bot programs. Text-based Captchas are the most widely deployed differentiation mechanism, and almost all text-based Captchas are single-layered. Numerous successful attacks on the single-layer text-based Captchas deployed by Google, Yahoo! and Amazon have been reported. In 2015, Microsoft deployed a new two-layer Captcha scheme. This appears to be the first application of two-layer Captchas. It is therefore natural to ask a fundamental question: is the two-layer Captcha as secure as its designers expected? Intrigued by this question, we have for the first time systematically analyzed the security of the two-layer Captcha in this paper. We propose a simple but effective method to attack the two-layer Captcha deployed by Microsoft, and achieve a success rate of 44.6% with an average speed of 9.05 seconds on a standard desktop computer (with a 3.3 GHz Intel Core i3 CPU and 2GB RAM), thus demonstrating clear security issues. We also discuss the originality and applicability of our attack, and offer guidelines for designing Captchas with better security and usability.**

***Index Terms*—Captcha, security, text-based, two-layer.**

I. Introduction

CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) is used to prevent automated registration, spam or malicious bot programs [1], [2]. It automatically generates and evaluates a test, difficult for computers to solve, but easy for humans. If the success rate of solving a Captcha for humans reaches 90% or higher, and computer programs only achieve a success rate of less than 1%, the Captcha can be considered secure [3].

Current Captchas can be divided into three categories: text-based, image-based and audio-based. Text-based Captcha is usually based on English letters and Arabic numerals, and uses sophisticated distortion, rotation or noise interference to prevent the recognition of a machine. Compared to the latter two Captcha categories, text-based Captcha is the most widely used scheme [4]. This wide-spread usage is due to its obvious advantages [5], [6]: users' task is a text recognition problem which is intuitive to users worldwide; globally, people can recognize English letters and Arabic numerals, thus text-based Captcha has few localization issues; at last, text-based Captcha was the earliest form of Captchas and people are more willing to undertake this challenge compared to other Captcha forms.

Security is the most significant concern for text-based Captchas. There have been numerous examples of Captcha failure, and many attacks have been proposed, noted in [7]–[9]. Researchers have recently claimed that their simple generic attacks have broken a wide range of text-based Captchas in a single step [10], [11]. Although many text-based Captchas have been proven insecure, the most recent studies [12], [13] suggest that Captcha is still a useful security mechanism, and the security of text-based Captchas is currently a hot topic in the academic field.

With each failed Captcha scheme, Captcha designers accumulated experience. Then they designed better schemes, which aimed to improve both usability and security. These prior attempts also promoted a scientific understanding of Captcha's robustness. As [6] suggested, the robustness of text-based Captchas should rely on the difficulty of finding where each character is (segmentation), rather than what each character is (recognition). Current machine learning algorithms, such as Neural Networks or K-Nearest Neighbors (KNN), can easily recognize distorted or rotated single characters correctly [6]. Therefore, text-based Captchas should be segmentation-resistant, and this principle has become the basis for designing text-based Captchas.

Based on these findings, Microsoft deployed a two-layer Captcha in 2015. It connected the sides of each character, across both top-to-bottom and right-to-left, in order to increase the difficulty of detecting where each character is. This was the first application of two-layer Captchas, and it appeared more secure than previous schemes. All attacks proposed were unsuccessful, including these powerful generic methods proposed in [10], [11]. It is therefore natural to ask a fundamental question: is the two-layer Captcha as secure as its designers expected? This open question precipitated our study.

In this paper, we examine the security of the two-layer Captcha. We attack this scheme by a series of processing steps. First, a novel two-dimensional segmentation approach is proposed to separate a Captcha image along both vertical and horizontal directions, which helps create many single characters and is unlike traditional segmentation techniques. Second, we improved LeNet-5 [14], a radical Convolutional Neural Network (CNN) model, as our recognition engine. Our CNN model, as compared with the traditional template-based KNN, shows improved performance and effectiveness. Third, a Captcha generating imitator was designed which is able to automatically produce Captchas similar to the real-world ones deployed by Microsoft. We adopt it to generate a significant

Manuscript received May 11, 2016; revised October 11, 2016; December 13, 2016 and February 7, 2017; accepted March 1, 2017. Date of publication March 1, 2017; date of current version March 1, 2017. This work was supported by the National Natural Science Foundation of China (61472311) and the Fundamental Research Funds for the Central Universities.

The authors are with the Institute of Software Engineering, Xidian University, Xi'an, Shaanxi, 710071, P.R.China. (e-mail: hchgao@xidian.edu.cn).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TIFS.2017.

amount of data for our training sets to train the CNN. Then, the trained CNN is used to recognize the Captchas deployed in the wild. Our attack has achieved a success rate of 44.6% with an average speed of 9.05 seconds on a standard desktop computer (with a 3.3 GHz Intel Core i3 CPU and 2GB RAM). Judging by both criteria commonly used in the Captcha community [5], [9], we have successfully broken the two-layer Captcha deployed by Microsoft.

This appears to be the first systematic analysis of two-layer Captchas. Our two-dimensional segmentation technique is innovative; it can be applied as a basis for other successful attacks. Additionally, our data preparation method, imitating the Captcha generation process to build training sets, and using it to train the recognition engine to recognize the objects in the real world, is the first attempt in the field of analyzing Captchas' robustness. Our experiments demonstrate that it decreases time spent on data preparation and reduces manual labor. Also, this approach can be adapted for other projects requiring a large amount of data to train machine learning systems, and for people working in Captcha design, our work provides guidelines for designing Captchas with better security and usability.

The remainder of the paper is organized into the following sections. A review of related work is provided in Section 2. Then, Section 3 introduces and analyzes Microsoft's two-layer Captcha. Following that, the whole attack process is presented in Section 4, and Section 5 details the experiment process and presents attack results. In Section 6, we talk about other design choices, the applicability and novelty of this work, and also provide guidelines for designing Captchas with better security and usability. Finally, Section 7 concludes the paper with a discussion of the implications of our work.

II. Related Work

The Automated Turing Tests were first proposed by Moni Naor [15], but they did not provide a formal definition. Alta Vista [16] developed the first practical Automated Turing Test to prevent bots from automatically registering web pages. This system was effective for a while and then was defeated by common Optical Character Recognition (OCR) technology.

Text-based Captcha based on English letters and Arabic numerals, is still the most widely deployed scheme. Many research communities focus on developing attack approaches for existing text-based Captchas, and then explore guidelines for better designs.

In [17], Mori and Malik used sophisticated object recognition algorithms to break Gimpy and EZ-Gimpy with a success rate of 33% and 92% respectively. In [18], Chellapilla broke a number of Captchas with a success rate ranging from 4.89% to 66.2%. Other attack efforts also include the PWNmad [19].

In 2006, Yan and El Ahmad [7] broke most visual schemes provided at Captchaservice.org by simply counting the number of pixels of each segmented character, even though these schemes are resistant to the best OCR software on the market. In 2008, the same authors [8] developed new character segmentation techniques for attacking a number of text-based Captchas, which included the earlier mechanisms designed and deployed by Microsoft, Yahoo! and Google.

Elie Bursztein et al. [9] carried out a systematic study of existing visual Captchas based on distorted characters. They found that 13 of the 15 current Captcha schemes from popular websites were vulnerable to automated attacks, but Google and reCAPTCHA resisted their attack attempts.

In 2013, Gao et al. [20] proposed a generic method to break a family of hollow Captchas, and this is the first application of solving Captchas in a single step. They used the characteristics of hollow fonts to extract character strokes, and then tested different combinations of adjacent character strokes to form individual characters. They tested their method on five hollow Captchas, and the success rates they achieved ranged from 36% to 89%.

Elie Bursztein et al. [10] reported another attack that solved Captchas in a single step by addressing the segmentation and the recognition problems simultaneously. It analyzed all possible ways of segmenting a Captcha, and then used ensemble learning to identify among each sequence of segments the best possible one as the result. This attack had broken a wide range of text-based Captchas with success rates ranging between 5.33% and 55.22%.

In 2015, Karthik et al. [21] proposed two methods to automatically classify Microsoft Captcha samples. One was based on a fine-grain segmentation combined with template matching, and the second was based on CNN and used state-of-the-art recognition techniques. The former achieved a success of 5.56%, and the latter achieved a success of 57.05%.

More recently at NDSS'16, Gao et al. [11] reported a simple generic attack that was the first to use Gabor filters to extract character components along four different directions. It is the most recent research on the analysis of Captcha robustness, and is effective for many text-based Captchas, including character isolated Captchas, hollow character Captchas, and Crowding Characters Together (CCT) Captchas. The success rates they received ranged from 5% to 77%.

There has been a lot of fine research done on text-based Captchas prior to this paper. However, two-layer Captchas have never before been discussed, and they are distinctly different from the other text-based Captchas discussed to date.

III. Two-layer Captchas

According to font styles and positional relationships between adjacent characters, most of the current text-based Captchas can be classified into three categories: character isolated Captcha, hollow character Captcha and CCT Captcha [11] (see Figures 1(a), (b), and (c) respectively). Some Captchas use a combination of the above-mentioned styles, while others also use some noise arcs or a complex background to enhance their security (Figures 1(d) and (e)). No matter what form of Captcha, there were no two-layer Captchas until Microsoft first deployed one.

The original Captcha used by Microsoft was the character isolated Captcha (See Figure 2(a)). This character isolated scheme had been used for many years before the two-layer Captcha was deployed. When it was broken by Jeff Yan [8], the character-isolated scheme seemed to have become the most vulnerable and other similar schemes were also shown to be insecure.

Fig. 1. Various Captcha categories: (a) Character isolated Captcha, (b) Hollow character Captcha, (c) CCT Captcha, (d) Captcha with noise arcs, and (e) Captcha with complex background.

Fig. 2. Microsoft's Captcha schemes: (a) Early scheme, (b) The first generation of two-layer scheme deployed in 2015.

In 2015, Microsoft deployed a two-layer Captcha (see Figure 2(b)) on its account creation page which can be seen as a vertical combination of two traditional single-layer Captchas. It requires users to recognize the upper-layer characters and then the lower-layer characters to pass the test. The upper layer is the first horizontal line of the Captcha containing three characters, and the lower layer is the second line of the Captcha with another three characters. The characteristics of the two-layer Captcha are summarized as follows:

- An image with six characters in total, three in each layer.
- It uses both hollow and solid characters randomly. Unlike the common solid characters, the hollow characters are those characters formed by contour lines, e.g. characters '**X**', '**P**' and '**R**' in Figure 2(b).
- Each layer utilizes CCT scheme and the whole Captcha is warped and rotated.

This two-layer Captcha is a combination of several styles, in part because the character fonts it utilizes include formal solid characters as well as hollow ones. However, it should be classified essentially as a modified CCT scheme owing to the connection of both top-to-bottom and right-to-left characters.

CCT Captcha, as the most widely used scheme, is also the most difficult of the three main styles to attack. The reasoning is that by connecting characters together, it greatly increases the difficulty of segmenting the Captcha. Judging the position of each character in the single-layer Captcha is relatively hard, the two-layer Captchas are much more difficult. In Microsoft's two-layer scheme, each single character is distorted, as well as the whole Captcha image being rotated; these local and global warping techniques increase the difficulty of segmenting the Captcha. These techniques show that the two-layer Captcha is a good application of the segmentation-resistant principle.

The two-layer Captcha deployed by Microsoft appears more secure than previous Captcha schemes. Clearly, methods such as pixel counting and Color Filling Segmentation (CFS) proposed by Jeff Yan and EI Ahmad [7], [8] will lose their efficiency in a two-layer Captcha. This new scheme is much more sophisticated than earlier schemes, because the number of pixels is not fixed and all of the characters are connected. The powerful attacks which have been claimed to break a wide range of text-based Captchas [10], [11], [20] are also cast into the shade. The method proposed in [20], can only extract strokes of the hollow characters in this two-layer Captcha, but is unable to attack the solid characters. Additionally incomplete character strokes lead to an attack failure. The attack described in [10] is an effective way to detect the boundaries of adjacent characters. However, this method only focuses on vertical segmentation. When applied to two-layer Captchas, the recognition engine has difficulties determining which vertical segmentation set is the best, since each vertical segmentation contains two character parts (the upper character and the lower character). The state-of-the-art study in [11] extracts character components via Gabor filters, and then combines adjacent character components to form individual characters. The major failure of this attack is the mistake in components ranking; that is, a component belonging to the former character is potentially sorted after the components belonging to the latter character. This issue will occur in two-layer Captchas as the bottom character of the prior column is always on the left of the top character of the next column, but the correct character order is the latter prior to the former. Additionally, the hollow characters adopted in Microsoft's Captcha will produce many tiny components, which creates numerous possible combinations, decreasing the success rate.

Even though this type of two-layer Captcha cannot be broken by existing techniques, we hypothesize that a radical approach could challenge its apparent security.

IV. Our Attack

To evaluate the robustness of Microsoft's two-layer Captcha, we conduct a series of experiments which will be presented in this section.

Suppose that the single-layer Captcha attacking problem is a one-dimensional image processing problem, the two-layer Captcha is then a two-dimensional problem. Obviously, the most critical problem of attacking the two-layer Captcha is finding the locations of characters (segmentation). A new segmentation technique is necessary to solve this problem, since existing segmentation methods are no longer useful, as discussed in Section 3.

In order to achieve segmentations that are as exact as possible, we propose a new segmentation method based on the observations presented in Section 3. This approach applies two different segmentations: one is splitting the hollow and solid character's parts, and the other is separating the upper and the lower layers of the Captcha. This two-dimensional segmentation technique is able to isolate as many single characters as possible, leaving only a few connected characters that need to be further segmented via a common method.

The key insight behind our attack is the following: firstly, pre-processing each challenge using standard methods. For

the hollow characters, CFS (introduced in [8]) is used to automatically convert them to solid characters. The filled hollow characters are also separated from solid characters in a Captcha image. Then, an ingenious method, which will be described in detail later, is utilized to convert a two-layer challenge to an upper part and a lower part. Additionally, the characters in Microsoft scheme are obviously rotated, so we rotate the Captcha in pre-processing to make the characters upright.

After pre-processing, each challenge image is separated into four parts: the upper solid characters, the lower solid characters, the upper filled hollow characters and the lower filled hollow characters. Note that the upper filled hollow characters and lower filled hollow characters were originally hollow characters, but we convert them to solid ones during the early pre-processing to ease the burden on the recognition engine. It is possible that no character is contained at all in a certain part, and the most likely case is that the part only contains a single character. Those single characters can be identified by a recognition engine. In the case of a part with connected characters, we improve on the state-of-the-art method described in [11] to separate and recognize them simultaneously, similar to solving a simple single-layer Captcha. Note that, unlike how [11] adopted KNN as their recognition engine, we use CNN [22]–[24] instead, due to the limited effectiveness of KNN and high-speed development of deep learning techniques. The KNN utilized in [11] is essentially based on template matching, whereas CNN extracts features in a hierarchical set of layers, and has partial invariance of translation, rotation, deformation and scale. It has been shown already that CNN can work on text classification [25], [26]. We will discuss the differences of CNN and KNN in detail in Section 6.

Figure 3 shows the high-level flow of our attack, including three key sequential steps: 1) Pre-processing using standard techniques to prepare each challenge image; 2) Extraction of character components to find character strokes; 3) Partition and recognition using the CNN engine assisted graph search. Note that the two latter steps only focus on the parts with connected characters, and if a part only contains one single character it is recognized by our CNN engine.

A. Pre-processing

Image binarization. First, we convert a rich-color challenge to black-and-white using the standard Otsu's threshold method [27], as Figure 4 shows.

CFS [8] to fill hollow characters. This is a flooding algorithm which is used to detect connected non-black pixel blocks. Both hollow characters and noise components are picked out to fill. Different colors are used to distinguish those components. After CFS, the background color is set to light-gray, and different components are filled with distinguishing colors (see Figure 5).

Unfortunately, both the strokes of hollow characters and closed regions are filled by CFS. For our purpose, some measures should be taken to remove those noise components in this step. As illustrated in Figure 5, there are three types

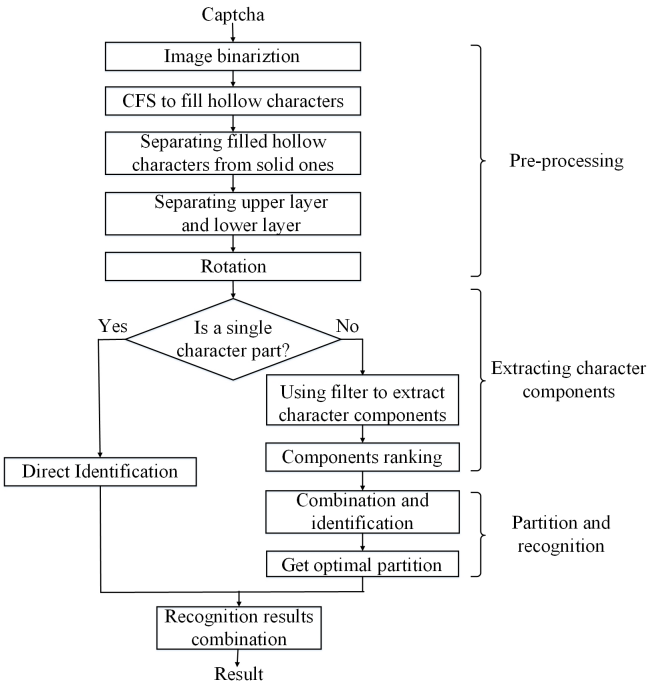


Fig. 3. The pipeline of our attack.

Fig. 4. Image binarization: (a) Original image, (b) Binarized image.

of undesirable components which are considered noise: 1) the closed part within a hollow character that does not belong to character strokes, 2) the closed part within a solid character, 3) the closed part produced by connected characters.

To remove these noise components, we define parameters δ and θ for each color component: $\delta=C/S$ and $\theta=W/Sr$, where C denotes the number of edge pixels that have a black neighbor in this color component, S denotes the total number of pixels in the color component, W denotes the number of pixels within the bounding box of the color component (an external rectangle of the color components area) but belong to the light-gray background or other color components, and Sr denotes the total number of pixels the bounding box of the color component contains, as Figure 6(a) and (b) show.

Due to the various shapes and thin strokes of hollow English

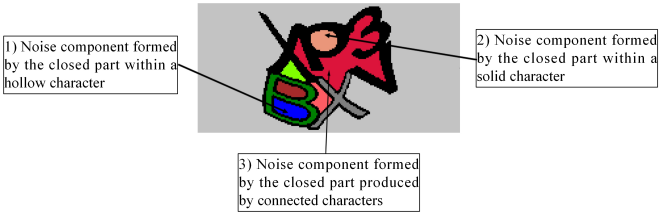


Fig. 5. Image after CFS shows three types of noise components.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

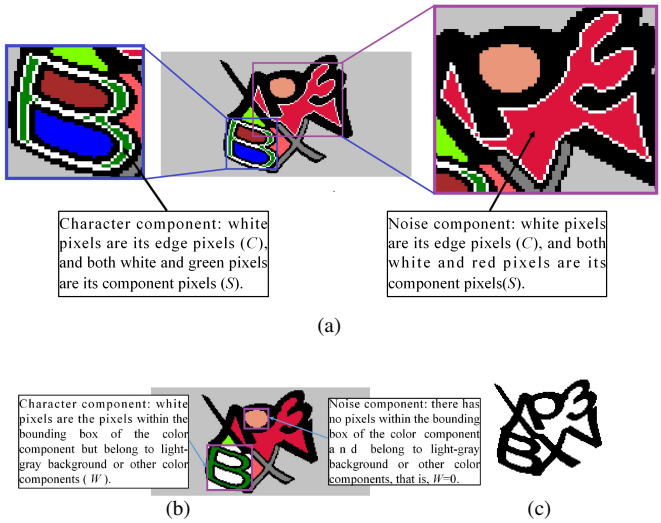


Fig. 6. Noise removal.

letters and Arabic numerals, their color components always have a relatively larger C and a smaller S. Noise components, however, are usually simple with a plump shape. Besides, the surroundings of character components always have a large section of pixels belonging to a light-gray background or other components, whereas the remnant noise components are surrounded by character strokes and with little (or no) such pixels (see Figure 6(a) and (b)). Therefore, with properly chosen threshold values a and b, we have the following: if $\delta < a$ or $\theta < b$ for a component, we consider it as noise and remove it. The values a and b are determined by analyzing a small sample set of data.

After deleting all of the noise components, the color of the character components is changed to black, and Figure 6(c) shows the final result of CFS. The two major contributions of CFS are: first, it reduces the burden on the recognition engine since the diversity of characters significantly decreases the effectiveness of the recognition engine; second, it also avoids failed cases that possibly occur in later extraction of character components, where we adopted the state-of-the-art method in [11] to extract components from connected characters based on Gabor filters. [11] noted that they received the lowest success rate on a hollow scheme deployed by Yahoo!. This extraction method breaks a long hollow text string into a large number of tiny components, and further produces a huge set of possible combinations during partition and recognition, both decreasing the success rate and slowing down the attack speed. Learning from this prior experience, we have converted hollow characters into solid characters in advance.

Separating filled hollow characters from solid characters. This step is considered to achieve the highest number of complete individual characters. It makes use of the characteristics of hollow characters formed by thin contour lines that can be easily dislodged. First, we erode the binarized image (Figure 6(c)) to remove the hollow characters part. The solid characters part is also damaged during this step, but we simply recover it by dilation. Finally, by removing the solid characters part from the image received by CFS (Figure 6(c)), the filled

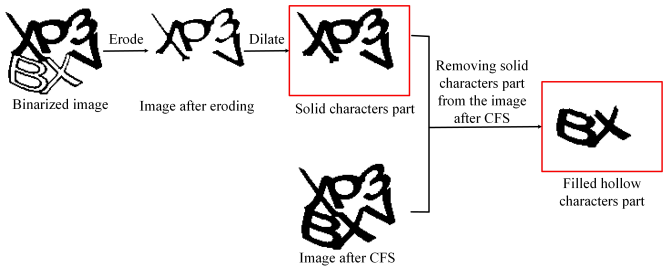


Fig. 7. Process of separating the solid characters and filled hollow characters.

hollow characters part is found, as shown in Figure 7. The outputs, the filled hollow characters part and solid characters part, marked with red rectangles in Figure 7, are the inputs of next process.

It may seem that this step can be skipped and that we could consider these filled hollow characters as solid ones equally. However, it is necessary in our attack. This process, as the first step of our two-dimensional segmentation algorithm, helps to receive as many individual characters as possible. [6] suggested that the challenge of breaking a Captcha depends on the difficulty of finding where each character is. Thus we add this step to our attack in order to increase segmentation accuracy.

Separating upper and lower layers. The two-layer structure is used as a defense measure to disguise where a single character is, and it is necessary to separate a two-layer challenge image into two one-layer images. We have found that the borderline of upper layer and lower layer ranges across challenges, but it waves along the variation trend of the envelope lines of a challenge. Based on this characteristic of Microsoft's two-layer Captcha, we start by finding the top and bottom envelope lines for each challenge image, and then generate the middle line of these two envelope lines as the borderline of a challenge image's upper and lower layers. Then, we apply the borderline to the filled hollow characters and the solid characters parts, and finally separate the upper and lower layers.

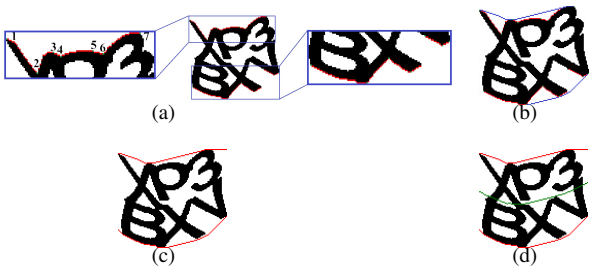


Fig. 8. The generation of envelope lines and borderline: (a) Two curves and inflexions, (b) Connected inflexions, (c) Envelope lines, (d) Borderline.

Generating these two envelope lines and the borderline is the key step of this process and works as follows:

i) *Generate two curves by following the top pixels and bottom pixels of the challenge image* (as the red and green pixels shown in Figure 8(a)).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

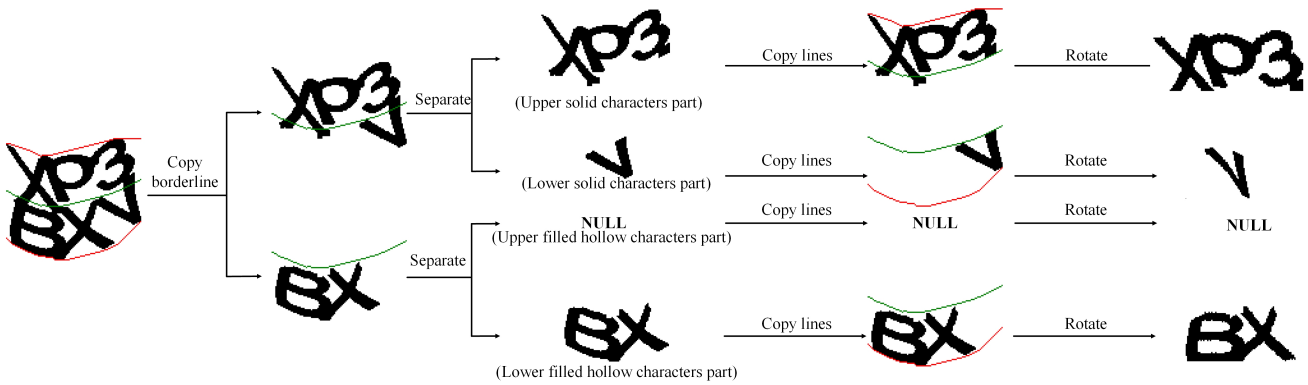


Fig. 9. The process of separation and rotation.

ii) *Find all inflexion points of the curves* (those pixels marked with green in Figure 8(a)). Note that inflexion points include both crest and trough pixels. We define crest pixels as those pixels whose left curve pixel's y-ordinate is equal to or larger, while the right curve pixel's y-ordinate is larger. Similarly, trough pixels are those pixels whose left curve pixel's y-ordinate is equal to or smaller, while the right curve pixel's y-ordinate is smaller.

iii) *Smooth curves and generate envelope lines*. We first simply define the slope α of two inflexion points as follows:

$$\alpha = \frac{|y_1 - y_2|}{|x_1 - x_2|} \tag{1}$$

where (x_1, y_1) and (x_2, y_2) are the coordinates of these two inflexion points. Then we examine the α value of each pair of inflexion points. With a properly chosen threshold t, this works as follows: if $\alpha > t$ and these two inflexions both are crest pixels or trough pixels, connect them with a smooth curve directly (as the case of inflexion points 3 and 7 in Figure 8(a)); if $\alpha > t$ but one of these two inflexion points is crest pixel, and another is trough pixel, connect the current inflexion point and the latter inflexion point's next inflexion point with a smooth curve (as the case of inflexion points 1 and 2 in Figure 8(a)). The whole sub-step process is iterated twice - one for the top curve and another for the bottom curve. Those connected curves are marked with blue in Figure 8(b). Figure 8(c) depicts the final envelope lines which are marked with red. Those connected curves are converted to red as a part of the envelope lines. If an envelope line's end doesn't connect with the edge of the image, we extended it to the edge.

iv) *Create the borderline*. We find the middle line between two envelop lines and use it as the borderline of the upper layer and lower layer, as shown by the green line in Figure 8(d).

We create the borderline of the upper and lower layers, and apply it to the filled hollow characters and the solid characters parts, and then separate the upper and lower layers of them(see Figure 9). Note that if solid characters part or hollow characters part only appears in either upper layer or lower layer in a challenge, it is possible to generate a blank image without characters, as the case of our example image.

Rotation. For the Microsoft's Captcha whose characters are obviously rotated and the rotation angle of each character is different, we rotate the Captcha in pre-processing and make the characters upright. The three lines, two envelope lines and the borderline that we found in previous step, also plays an important role in this step. First, we copy the two upper lines to the two upper layer images found from the prior step, and copy the two lower lines to the two lower layer images. Then use the guide line projection technique to find the most likely rotation angle [28]. Finally, rotate the image using this rotation angle (see Figure 9).

Thus a challenge image has been divided into four images with all characters rotated: upper solid characters part, lower solid characters part, upper filled hollow characters part, and lower filled hollow characters part.

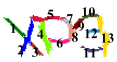
B. Extracting Character Components

After pre-processing, we have many separated individual characters, e.g. '**V**' in the example image. For images like Figure 2(b), where hollow characters and solid characters are alternatively located, we can successfully separate each individual character only via our two-dimensional segmentation approach. We then utilize CNN engine to recognize the separated single characters, and for other images containing connected characters, we adopt the method presented in [11] to break them, similar to solving a simple single-layer Captcha with three connected characters at most. Note that, we have improved the approach in [11] with regard to many details, especially the graph search algorithm and utilizing CNN to replace the template-based KNN.

First, the approach uses Log-Gabor filters [29] to extract connected characters information along four directions by convolving a Captcha image with each of the filters. Then, it binarizes the resulting images to create black and white character components. Table I shows the extraction results along each of the four directions for each part of the example Captcha. This step only applies to those with connected characters, as in the example image: the upper solid characters and the lower filled hollow characters. Extracted components are stored in four separate images of the same dimension.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

TABLE I
EXTRACTION RESULTS

	Upper solid characters part	Lower filled hollow characters part
Angle		
0		
$\pi/4$		
$\pi/2$		
$3\pi/4$		
+		

We leverage CFS [8] to collect these components from each image. Then sort them according to the coordinates (x, y), which record the average x-coordinate and y-coordinate of each component. These sorted components are numerically ordered, as the last row of Table I shows. We have marked the extracted character components with different colors so that readers can easily distinguish them from each other.

Unlike the ranking method introduced in [11], we sort by average coordinate rather than each component's top-left pixel, creating a more accurate location of each component. The extraction result of the upper solid characters presented in the last row of Table I is a good example. If it is ranked by each component's top-left pixel, component 5 would be ordered after component 3. This leads to the mistake of following a sequential combination.

C. Partition and Recognition

Graph building. We try different combinations of adjacent components to form individual characters, and build a directed graph with n+1 nodes for each image whose edges record whether a combination is feasible, where n is the total number of components. For the extraction result of the upper solid characters part, n=13 (as the left figure in the last row of Table I shows); and for the extraction result of the lower filled hollow characters part, n=10 (see the right figure in the last row of Table I).

The edge from node i to node k+1 on the directed graph indicates the combination of components i, i+1, . . ., k is feasible to form a larger single component (its width is larger than the smallest character width and smaller than the largest character width). We also adopted the method introduced in [11] to remove the redundant nodes with no feasible path passing through them. All redundant nodes and their connecting edges are pruned, and the final graphs built for our examples, as Figure 10 shows. For instance, the edge from node 1 to node 4 means the combination of combining components 1 to 3 is possible to form a legitimate individual character. The data on each edge in Figure 10 will be marked in the next step.

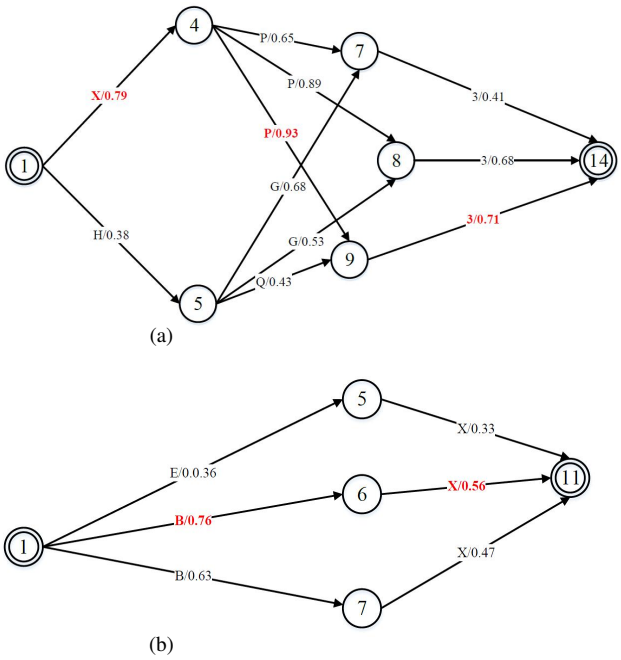


Fig. 10. The updated equivalent graphs of our example: (a)The final equivalent graph of the upper solid characters part; (b)The final equivalent graph of the lower filled hollow characters part.

Recognizing component combinations. We use CNN as our recognition engine to determine which character each of the remaining edges in the graph is likely to be. In terms of text recognition, CNN has a much better performance than the template-based KNN introduced in [11], and they will be compared in Section 6 in detail. We update the edges with the recognition results of corresponding combinations returned by the CNN engine, as Figure 10 shows. For example, the edge from node 1 to node 4 in Figure 10(a) indicates that CNN recognizes the combination of components 1 to 3 as '***X***' with a confidence level of 0.79.

Graph searching. The problem of finding an optimal combination has been converted into a problem of finding a path ending at node n+1 with the largest average confidence level, and its step (the number of edges on the corresponding path) is equal to the Captcha string length (the number of characters in the challenge). Note that we adopted a Dynamic Programming (DP) graph search algorithm to achieve this, similar to the algorithm introduced in [11], but our algorithm is more detailed and has been improved. Instead of finding a target path with the largest confidence level sum, our algorithm manages to find a path with largest average confidence level, which makes it not only effective for fixed-string length Captchas, but also for Captchas with a varied string length.

Procedure Main()
Begin
$R \leftarrow NIL$
for $j \leftarrow 1$ *to* $n+1$
  $pathList[j] \leftarrow null$
  $GetValue(j, pahList)$
$Select(n+1, pathList, R)$
End

Procedure GetValue$(j, pahList)$
Begin
$pathj \leftarrow null$
if $j = 1$
  $newPath.step \leftarrow 0$
  $newPath.value \leftarrow 0$
  $newPath.result \leftarrow null$
  Add$(pathj, newPath)$
else
  foreach i *in* $prej$
    foreach $path$ *in* $pathList[i]$
      $n \leftarrow -1$
      if $ExistStep(pathj, path.step + 1, n)$
        if $path.value + con[i; j] > pathj[n].value$
          $pathj[n].value \leftarrow path.value + con[i; j]$
          $pathj[n].result \leftarrow path.result + char[i; j]$
        else
          $newPath.step \leftarrow path.step + 1$
          $newPath.value \leftarrow path.value + con[i; j]$
          $newPath.result \leftarrow path.result + char[i; j]$
          Add$(pathj, newPath)$
$pathList[j] \leftarrow pathj$
End

Procedure ExistStep$(pathj, step, num)$
Begin
for $i \leftarrow 1$ *to* $pathj.length$
  if $path.step$ *in* $Captcha$ $length$
    if $pathj[i].step = step$
      $num \leftarrow i$
      return $true$
return $false$
End

Procedure Select$(num, pathList, R)$
Begin
$averageValue \leftarrow 0$
foreach $path$ *in* $pathList[num]$
  if $path.step$ *in* $Captcha$ $length$
    if $path.value/path.step > averageValue$
      $averageValue \leftarrow path.value/path.step$
      $R \leftarrow path.result$
End

The pseudo code illustrates our DP process. The traversal starts from node 1 and ends at node *n*+1, with *R* being the final recognition result that we aim for. *path* stores a possible path in the equivalent graph, and we define three parameters for it: *value*, *result*, and *step*. *value* stores the confidence level sum of the possible path, *result* stores the corresponding result string of the possible path, and *step* stores step number of the possible path (the number of edges on the corresponding path, or the number of current recognized characters), respectively. *newPath* is a *path* whose *value* is 0, *result* is NULL and *step* is 0. An array *pathList* stores the paths found by our algorithm for each node, e.g. *pathList*[*j*] stores the paths from node 1 to node *j* which has the largest average confidence level with corresponding different step numbers. *Pathj* is a temporary list of *path*s. *prej* is a list that stores all the precursors of node *j*. *con*[*i*; *j*] is the confidence level calculated by CNN for the combination formed by combining blocks from *i* to *j*, and *char*[*i*; *j*] is its corresponding recognition result. *averageValue* represents the average confidence level.

TABLE II
SEARCH PROCESS FOR THE UPPER SOLID CHARACTERS PART

j	*step*[*j*]	*value*[*j*]	*result*[*j*]
4	1	0.79	X
5	1	0.38	H
7	2	1.44	XP
8	2	1.68	XP
9	2	1.72	XP
14	*3*	*2.43*	*XP3*

TABLE III
SEARCH PROCESS FOR THE LOWER FILLED HOLLOW CHARACTERS PART

j	*step*[*j*]	*value*[*j*]	*result*[*j*]
5	1	0.36	E
6	1	0.76	B
7	1	0.63	B
11	*2*	*1.32*	*BX*

Generated by our attack program, Tables II and III show the graph search processes of the graphs depicted in Figure 10 with our DP algorithm. DP simplifies the search process by recording the largest confidence sum of each node. The italic items highlighted in the table indicates the optimal partition that has the highest confidence-level sum. That is, "***XP3***" is the recognition result of the upper solid characters part, and "***BX***" is the recognition result of the lower filled hollow characters.

Thus we have the recognition results of the upper solid characters part and the lower filled hollow characters part of the example challenge image. The lower solid characters part, which only contains a single character, is sent to the CNN engine to be recognized, and '***V***' is returned as the result. For the upper filled hollow characters part, with no black pixels, nothing further is required. The recognition results are organized in order as the final result. That is, "***XP3BXV***" is the final result in this case. Figure 11 shows the whole process of our attack.

V. EXPERIMENTS AND RESULTS

A. Data Preparation

A key characteristic of CNN is that it usually requires enough data to train, especially for deep learning systems.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

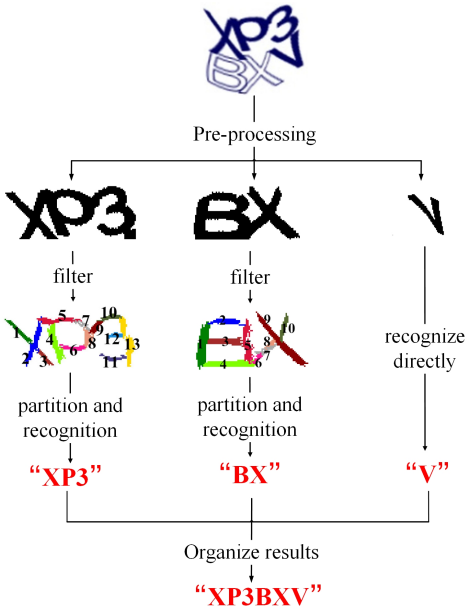


Fig. 11. Whole process of our attack.

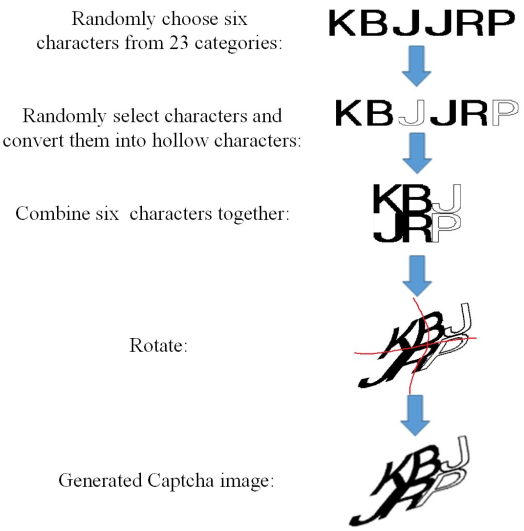


Fig. 12. Our Captcha generation imitator.

However, in our field, there has never been such a data set which contains a large number of Captchas that match the requirement for training the network. Thus, the first step is to build a data set. The main approach in data collection is to mine the Captchas from corresponding websites and then add labels (recognizing the content of each Captcha artificially) to them. Obviously, it is time-consuming and has a high potential error rate, particularly during the process of adding labels.

To achieve a sufficient data to train our CNN, we propose an innovative approach to build the training set. Instead of collecting Captchas by mining the Microsoft's website, we design an imitator to imitate Microsoft's Captcha generating process, and automatically produce Captchas that are very similar to those deployed by Microsoft in its real-world website.

Figure 12 shows our automatic generation process:

i) *Randomly choose six characters from 23 categories.* This step randomly generates six upright solid character images with a size of 50×50 pixels. The original Microsoft's two-layer Captchas contains 23 different characters (it discards characters that are easily confused, e.g. '**1**' and '**I**'), and each image has six characters. Thus, we generate the same number of characters from the sample alphabet.

ii) *Randomly select characters and convert them into hollow characters.* The number of chosen characters ranges from 0 to 6.

iii) *Combine six characters together.* The process simply combines these six characters by locating three character images in each layer; this is the primary form of the Captcha.

iv) *Rotate.* We rotate the combined image along random Sine curves on both x-direction and y-direction. Finally, we resize the image and put it in the center of a white background which has the same size (200×100 pixels) as the original Captcha image.

We consider only the shapes of characters not the color of the Captchas, since we often binarize Captchas in the first step of pre-processing. The final generated Captchas show that they contain the main characteristics of the original Microsoft's two-layer Captchas. By doing this, we can create as many Captcha images as we want, without the financial and time costs of adding labels.

Note that, we only utilize this Captcha generation imitator to produce the Captchas used for training set, the test set is still built by the traditional data preparation method of mining corresponding websites. We automatically generated 10000 random Captchas for the training set via the imitator. As for the test set, we collected 500 random Captchas by mining Microsoft's website in 2015.

The effectiveness of this data preparation method will be illustrated later. It is time saving and high performance, and can be adapted to other projects which require large amounts of data to train machine learning systems.

B. CNN Model

We choose the most rudimentary CNN model, LeNet-5 [14], as our recognition engine, initially designed for handwritten and machine-printed character recognition.The original LeNet-5 architecture consisted of three convolutional layers, two subsampling layers and followed by two fully connected layers. The convolution layers convolve the maps from the previous layer with a number of filters to extract features, and these filters are trained during the training process. The subsampling layers take samples of features extracted from the previous convolutional layer to reduce the amount of calculation while ensuring useful information. The full connected layers classify the features extracted from previous layers into target categories. Figure 13 depicts the architecture of our modified LeNet-5. To achieve higher grade features,we improve LeNet-5 by adding another convolutional layer and another subsampling layer (marked in red in Figure 13), but the main layout of the network is the same.

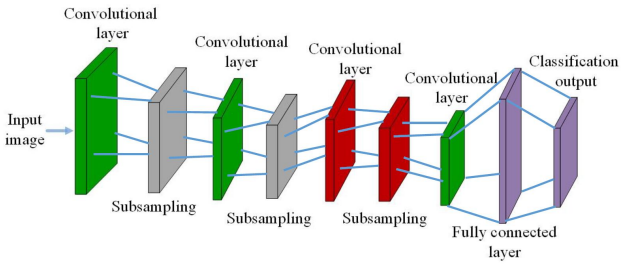


Fig. 13. A modified LeNet-5.

Character samples extracted from the training set are normalized to 28×28 pixels as the input images. Each convolutional layer and each subsampling layer contains 32 feature maps. Every convolutional layer has a convolution filter of 4×4, and the slide step of each subsampling layer is 2.

C. Results

We have implemented all the algorithms, attacks and the Captcha generating imitator in C# on Microsoft Visual Studio. The modified LeNet-5 is developed on Caffe [30], a deep learning framework created by a UC Berkeley team. We train LeNet-5 network with a GPU (NVIDIA TITAN X GPU, 32GB RAM) which significantly reduced the training time, and tested our attack on a desktop computer with a 3.3 GHz Intel Core i3 CPU and 2GB RAM.

Success rate. Our attack has achieved a success rate of 44.6% on the test set. [5] proposed a commonly accepted standard for Captchas robustness in preventing automated attacks from achieving higher than 0.01% success, but it was considered too ambitious by some researchers. [9] suggested that a Captchas scheme is broken if an automated attack achieves a success rate higher than 1%. According to either criterion, the high success rate we have received clearly indicates that our attack has broken the two-layer Captcha deployed by Microsoft. Therefore, we can definitely announce that the two-layer Captcha is not as robust as its designers expected. Our results also prove that the novel data preparation approach we proposed, with mass data automatically generated by our Captcha generation imitator to train the deep learning system then use it to recognize real-word objects, is feasible and effective.

Attack speed. The average attack speed of our attack on the test set is 9.05 seconds, demonstrating its efficiency. The usability requirement demands each human user to solve a Captcha in less than 30 seconds. Some Captchas deployed in the wild reported an average solving speed of more than 46 seconds [31].

VI. Discussion

A. Other Design Choices

1) Recognition engines: Before we chose CNN as our recognition engine, we tested the KNN reported in [11] as a candidate. The KNN used in [11] works by measuring the similarity between corresponding pixels of two images. The confidence level of a recognition result is also derived from this similarity value. In order to decrease the importance of matching background pixels in decision making, it assigns a larger weight to similar black pixels, but a smaller weight to similar white pixels. If two corresponding pixels do not match, a negative value will be added to the similarity calculation. In essence, the KNN used in [11] is based on template matching.

TABLE IV
Success Rates vs. Recognition Engines

Recognition engine	CNN	KNN
Success rate	44.6%	13.4%
Average attack speed (s)	9.05	8.39

For the attempt using KNN, we mine 300 random Captchas from Microsoft's website as the training set to use as templates, and use the same test for CNN as mentioned before. The results achieved by KNN are compared with CNN in Table IV. It shows that KNN only receives a success rate of 13.4%, much lower than the success rate finally achieved by our CNN engine. But the average attack speed of CNN is a little slower than KNN (9.05 seconds vs. 8.39 seconds). Note that, we also consider using a larger training set for KNN, however, this causes the attack speed to drop off sharply.

To compare the effectiveness of CNN and KNN in more detail, we conduct another group of experiments to evaluate the effects of the size of training sets on recognition engines' speed and recognition results. We utilize five training sets including 100, 200, 300, 400 and 500 random Captcha images generated by our Captcha generating imitator respectively, and 500 random Captchas mined from Microsoft's website are used as the test set. The sample images extracted from the training sets are used for making templates for KNN and for training the CNN model. We not only test the size of training sets' effect on full Captcha images' success rate and average attack speed, but also verify their influence on recognizing well separated individual character images extracted from the test set manually.

Table V lists the recognition rates and recognition speeds received by recognizing individual character images and the success rates and average attack speeds of full Captcha images with different sized training sets. It shows that both the recognition rates of CNN and KNN improve with the increase size of training sets. Their corresponding success rates on full Captcha images are also increased. However, as noted previously, the average recognition speed of KNN drops off sharply with the enlargement of training sets, whereas CNN's speed remains relatively stable. Using the same training set, the recognition rates and success rates of CNN is much higher than KNN, which may be the major factor in understanding the gap between the final success rates received by CNN and KNN. It seems that KNN can utilize a larger training set to improve the recognition rate, but the decrease in speed is also an issue. Therefore, the effectiveness of KNN is limited.

Even though both experiments use a training set of 300, the success rate achieved by KNN in Table V is a little lower than listed in Table IV (9.8% vs. 13.4%). This is because the

TABLE V
THE SIZE OF TRAINING SETS EFFECT ON RECOGNITION ENGINES

Size of training set	CNN Recognition rate	CNN Recognition speed(ms)	CNN Success rate	CNN Attack speed(s)	KNN Recognition rate	KNN Recognition speed (ms)	KNN Success rate	KNN Attack speed(s)
100	77.79%	3.10	13.0%	9.24	58.38%	0.78	5.4%	3.66
200	87.59%	3.23	16.6%	9.19	60.85%	1.58	7.6%	5.92
300	90.44%	3.41	19.8%	8.99	64.07%	2.39	9.8%	8.21
400	91.60%	3.04	23.4%	9.45	66.95%	3.21	10.8%	10.95
500	92.87%	3.58	26.2%	9.01	68.59%	4.43	11.6%	14.47

former uses Captcha images generated by our imitator as the training set, whereas the latter utilizes real-world Captchas. According to our experience, the more similar the training and test images, the better the recognition results will be. This comparison also indicates that our final attack success rate can be improved if real-world Captchas are used for training.

Of course, these results do not mean that CNN is better than KNN in all aspects. CNN requires a large amount of data to train, and this training process generally relies on advanced hardware support (e.g. GPU), whereas KNN has a lesser requirement.

2) Graph search algorithms: Several other graph search algorithms are tried in order to find the most likely results, and are compared with the algorithm proposed in Section 4.

Depth-First Search (DFS) was used by Gao at al. in [20]. It starts from node 1 in the graph and explores each branch until the path length reaches the Captcha string length before backtracking. DFS finds all paths whose steps are within the range of the Captcha string length, and then a path ending at node n+1 with the largest confidence level sum is chosen. DFS is not optimal, since it often explores paths that can't reach the node n+1, and re-explores previously visited nodes even though the path that it is aiming to find has already been discovered.

Integer Partition algorithm (IP) is another graph search algorithm we considered. It works by finding the most likely way of forming m characters using n character components, where m is the Captcha string length. All combinations were recorded, and the one with the largest confidence level sum was selected. This algorithm reduces the search space compared with DFS, since it skipped all paths that do not reach node n+1. However it was still time-consuming, as it has to find all combinations.

Dynamic Programming graph search algorithm proposed in [11] (DP [11]) prunes out redundant nodes on the graph which not only saves time spent on character recognition but also decreases the search space of the graph. Thus, it is much more effective than the above two algorithms. DP [11] is similar to ours, but instead of searching for a target path with the largest confidence level sum, we search for the path with the average confidence level as our final result. From our experience, the confidence level sum of a path with more steps (characters) is often larger than the confidence level sum of the path with fewer steps (characters).When a Captcha scheme has a varied string length, it seems to be more likely for DP [11] to find the path with the most steps as the final result, even though it is not the case where the Captcha image has the largest Captcha string length. Our algorithm has improved on [11] to avoid this problem. However, the speed of DP [11] is close to the speed we achieved, as their search space is same.

TABLE VI
ATTACK SPEEDS VS. GRAPH SEARCH ALGORITHMS

Graph search algorithm	DFS	IP	DP [11]	DP
Average attack speed (s)	10.37	10.15	9.23	9.05

We compare the average attack speed of our DP algorithm, DFS, IP and DP [11] in Table VI and find that our DP algorithm (marked with red in Table VI) is optimal.

B. The Latest Generation of Microsoft's Two-layer Captcha

The two-layer Captcha that we analyzed previously is the first generation deployed by Microsoft in January, 2015. There are now several newer generations and here we have chosen the latest one to analyze its robustness.

We collected the latest generation of Microsoft's two-layer Captcha from its account creation page in November, 2016. As the first image in Figure 14 shows, the Captcha has been improved: all characters are hollow; each layer contains a random number of characters (3 to 6 characters in each layer and 6 to 12 characters in total); and it uses 45 character categories, far more than the 23 categories used in the first generation.

To evaluate the security of the latest two-layer Captcha, we follow the main steps presented in Section 4. Since all characters are hollow in the new scheme, we skip the process of separating filled characters from solid ones after CFS, and separate the upper and lower layers directly (see Figure 14).

Our attack has achieved a success rate of 28.0% on the latest generation of Microsoft's two-layer Captchas with an average attack speed of 12.56 seconds. The result shows that, even though the new scheme has applied several new resistance mechanisms to enhance its security, it is still weak in the face of our attack. What is also important to mention is that the new generation uses many similar characters, e.g. '**6**' and '**b**', '**a**' and '**9**'. It indeed increases the burden of the recognition engine. Unfortunately, it is clear that it also decreases the recognition accuracy by humans. From the standpoints of security and usability, we do not consider the improved two-layer Captcha to be a good design.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

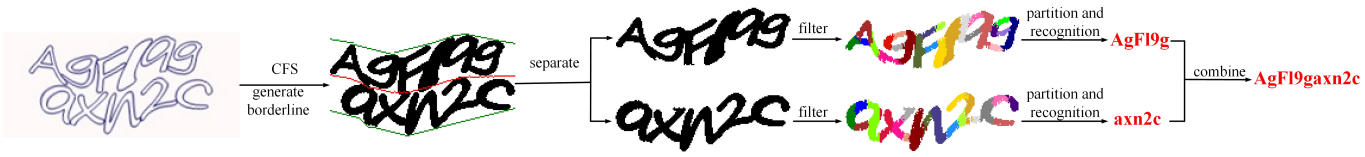


Fig. 14. The attack process of the latest two-layer Captcha deployed by Microsoft.

C. Applicability

Our attack can also be applied to other Captchas. Some cases are discussed as follows:

reCAPTCHA. We test our attack on an old version of reCAPTCHA (see Figure 15(a)), deployed by Google in 2013. Its characters are strongly rotated, which will lead to failure in components ranking. In order to avoid this kind of failure, we use techniques in pre-processing to make the characters upright. It is found that the rotation angle of each character is almost the same. We calculate the rotate angles of the challenge images, and then rotate and straighten the characters (see Figure 15(b) and (c)). Later processes are the same as described in section 4.2.2 and 4.2.3 (see Figure 15(d)).

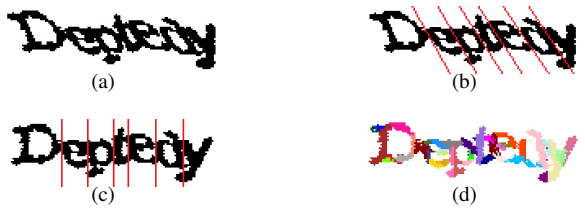


Fig. 15. An example image of reCAPTCHA: (a)Binarized image; (b)Find rotation angel; (c) Rotated image; (d)After extracting character components.

Our attack has received a success rate of 9.2% on reCAPTCHA, and its average attack speed is 8.46 seconds. It indicates that the key techniques of our attack are also applicable to reCAPTCHA.

Sina. To verify the applicability of our attack, we also test it on a version of the Sina scheme. A random thick noise arc that passes through each challenge (see Figure 16) is an obvious feature. We tried two attacks on the Sina scheme: one using the Gabor filter to extract character components without any pre-processing, and another removing the thick noise arcs in pre-processing. As shown by Table VII, our attacks on Captchas without removing the noise arcs only achieves a success rate of 11.4%, and on Captchas with the noise arcs removed has achieved a success rate of 24%, much higher than the former. This demonstrates the negative effect of the noise arcs on character recognition. But the average attack speed of the latter is slower than the former (7.21 seconds vs. 5.24 seconds), as the noise removal has a time cost. However, with or without removing the noise arcs, our attacks have successfully broken the Sina scheme.

Fig. 16. Original challenge image of Sina scheme.

TABLE VII
ATTACK RESULTS OF SINA SCHEME

	Without removing noise arcs	Removing noise arcs
pre-processing		
extracting character components		
Success rate	11.4%	24%
Average attack speed (s)	5.24	7.21

The experiments on both reCAPTCHA and Sina use a training set with 300 random Captchas and a test set with another 500 random Captchas, and all Captcha images are mined from corresponding websites. It is worthwhile to note that both the Google and Sina schemes we discussed above indicate that the key components of our attack are also applicable to single-layer Captchas.

D. Novelty

From our review of the literature, this is the first security analysis on the state-of-the-art two-layer Captchas. Among all the related works we presented earlier, no research had been conducted on the robustness of two-layer Captchas.

A recent attack reported in [21] has a slight similarity to our attack, which is also designed for the Captcha deployed by Microsoft, but has a focus on a single-layer version of Microsoft's Captchas. As a test, we separate the upper and lower layers of the two-layer scheme in advance, and then implement the attack of [21] on them. However, it only achieved a success rate of 17.4%, much lower than the success rate we achieved (44.6%) and the success rate they achieved on the single-layer scheme (57.05%).This is mainly because their equally distributed segmentation makes the attack be strongly affected by the huge gap between the smallest and largest width of individual characters in a Captcha image, even though the attack is assisted with a prediction cropping strategy. The variation of character width in the single-layer scheme that they analyzed is much smaller than in the two-layer Captcha, as Figure 17 shows. Moreover, their method only considered the Captchas with a fixed string length, whereas ours is also available to schemes with a varied string length. Therefore, our method has a greater applicability than theirs.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TIFS.2017.2682704, IEEE Transactions on Information Forensics and Security

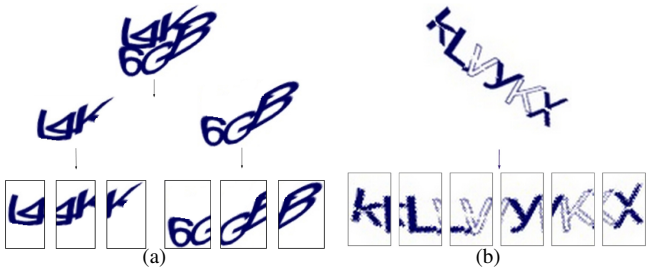


Fig. 17. The attack introduced in [21] works on Microsoft's two-layer Captcha and single-layer Captcha respectively: (a) Microsoft's two-layer Captcha; (b) Microsoft's single-layer Captcha.

Additionally, we mimicked the Captchas generation process to produce mass data for the purpose of training the recognition engine, then used the training results to recognize the Captchas deployed in the wild. This is time-saving and highly effective. Our data preparation method is the first practice in the field of analyzing Captchas robustness.

Moreover, our two-dimensional segmentation technique proposed in pre-processing is original. Captcha designers utilized both hollow characters and solid characters to increase the difficulty of recognition. However, our two-dimensional segmentation approach makes use of this resistance mechanism to find as many single characters as possible. The method used in separating the upper and lower layers, which utilizes envelope lines, is also innovative. As a unique pre-processing technique, it can be broadly applied and serves as a building block for other successful attacks.

E. Improving Security

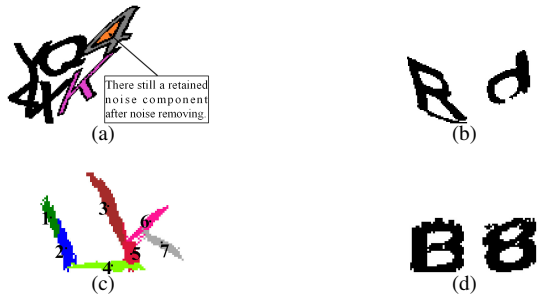


Fig. 18. (a) Failure of removing noise components after CFS; (b) Failure of separating the upper layer and lower layer; (c)Failure of components ranking; (d) Failure of character recognizing: 'B' and '8'.

Thus far, we have shown a way to attack the two-layer Captchas deployed by Microsoft. Our attack results have indicated that two-layer Captchas are not as secure as the designers expected. To identify which design features contribute to improving a two-layer Captcha's security, we analyze all cases of failure of our attack for reasons that may affects the success rate in test set, and there are four types of failure as follows:

- *Failure in removing noise components after CFS.* We have used two methods to remove noise components, however, these methods cannot completely remove all noise components. As Figure 18(a) shows, there is still a small noise component within character '**4**' after CFS and noise removal. If a character has a noise component that was not removed, it will have an impact on character recognition.
- *Failure in separating the upper and lower layers.* We used a borderline generated by two envelope lines to separate the upper and lower layers. It is not surprising that there is a certain amount of error. Figure 18(b) shows one of the four parts of a Captcha image with an inexact separation of the upper and lower layers where the character '**R**' retains a noise stroke from the lower layer, resulting in '**R**' recognized as '**B**'.
- *Failure in components ranking.* We ranked all components according to the coordinates (x, y), which record the average x-coordinate and y-coordinate of each component. A typical case was that a component belonging to the former character probably sorted after the components belonging to a later character. As shown in Figure 18(c), component 5 belongs to character '**L**', but it ranks after the first component (component 3) of character '**K**'. Pixels marked with black are the average coordinates (x, y) of each component. Our combination algorithm combined components in sequence to form individual characters.There is no doubt that the failure of a component's ranking creates the combination error. We also found that this failure is often caused by rotating and overlapping.
- *Failure in character recognition.* We found that our CNN engine does not always work efficiently on recognizing some similar characters. Some typical cases are: '**B**' and '**8**' (as Figure 18(d) shows), '**Y**' and '**V**', '**S**' and '**5**'.

Based on the above discussion, we analyze the design features that do improve security from anti-segmentation and anti-recognition aspects.

1) Anti-segmentation: It was proposed in [8] that a Captcha should be designed to be segmentation resistant. In terms of two-layer Captchas, segmentation includes the segmentation between layers and the segmentation between adjacent characters.

Design features that increase the difficulty of segmentation between layers include the following:

Making the two layers skewed or stretched. In our attack, we generated the borderline of the two layers according to two envelope lines. This method has achieved a good result, showing that most challenge images can be separated properly with a minor negative effect on later character recognition, since the characters' wave trend of two layers in the Microsoft scheme are similar. If the upper layer and lower layer had a different random wave trend respectively, our method will lose its efficiency, as shown in Figure 19.

Using a slight randomization of the characters height. Our separation method that acted on two layers works by roughly generating a middle line between two envelope lines. It significantly depends on the relatively fixed height of the characters in the layers and the same wave trend of each layer. If the gap between the upper characters height and the lower characters height is large, the generated middle line cannot separate the upper and lower layers properly.

(a) (b)

Fig. 19. Resisting segmentation between layers by making the two layers skewed or stretched: (a) Original image; (b) Stretched image.

Using a multi-layer structure is also an alternative approach to increase the difficulty of separation between layers. The more layers a Captcha uses, the more difficult it will be to detect each layer. However, when using this resistance mechanism, changes to the usability should also be carefully considered.

Design features that resist the segmentation between adjacent characters include the following:

Rotating and overlapping help to resist the segmentation between adjacent characters. Our attack used the Gabor filter to extract character components from connected characters and tried different combinations to form possible individual characters. As stated earlier, incorrect component ranking is a significant failure that leads to incorrect component combinations and further has significant negative influence on character segmentation. Rotating and overlapping are the major reasons that generate this failure. Indeed, they significantly contribute to resisting the segmentation between adjacent characters.

Increasing Captcha string length also helps. The more characters a Captcha image contains, the larger the number of extracted characters components there will be. This will produce a huge possible set of combinations, and further lower the success rate and speed of the attack.

Enlarging the gap between the smallest and largest width of individual characters also will produce more character components. Its overall effect is similar to increasing Captcha string length.

Adopting a varied Captcha string length enlarges the solution set that our graph search algorithm is required to go through. This might slow down our attack significantly.

2) Anti-recognition: Some plausible options that are applicable to almost all text-based Captchas to help to resist character recognition include the following:

Using similar characters. Our recognition engine is less effective in recognition of similar characters, as described before. Using similar characters (like '**B**' and '**8**', '**Y**' and '**V**', '**S**' and '**5**' and so on) will decrease the recognition accuracy of recognition engine, but has little negative impact on usability.

Using both solid and hollow characters. The Microsoft scheme used both solid characters and hollow characters, and we converted the hollow characters to solid by using CFS. Failing to remove noise components after CFS has a significant effect on character recognition. Even if we did not fill the hollow characters, a larger training set must be used to ensure recognition accuracy, because of the larger range of characters. This decreases the attack speed. Therefore, using both solid characters and hollow characters will help to enhance security.

VII. Summary and Conclusion

We have for the first time systematically analyzed the robustness of a two-layer Captcha deployed by Microsoft, and answered a primary question: is the two-layer Captcha as secure as expected? Our simple and effective attack achieved a success rate of 44.6% with an average speed of 9.05 seconds on a standard desktop computer. The results clearly indicate that the two-layer Captcha is not secure, in contrast to early claims.

We improved LeNet-5, a radical CNN model, as our recognition engine. Our CNN model, compared with the traditional template-based KNN, shows much better performance and effectiveness. To trace the source of their differences, we also offered a detailed and systematic analysis of CNN and KNN.

Additionally, we implemented a Captcha generating imitator to recreate the generation of Microsoft's two-layer Captcha as a training system. Compared with the common approach to access training sets, it is simple but highly effective. This concept, training a recognition engine with automatically generated data and then utilizing it to recognize real-world objects, is novel in the field of analyzing the robustness of Captchas.

Another contribution of this work is our new two-dimensional segmentation technique. It breaks the resistance mechanism adopted by Microsoft's two-layer Captcha, a mixture of hollow characters and solid characters, by separating a Captcha image into hollow character and solid character parts. In contrast to traditional segmentation techniques which only consider a vertical direction, this two-dimensional approach is applied to both vertical and horizontal directions. As a novel segmentation approach, it can serve as a building block for other successful attacks. Our work also provides a warning to Captcha designers that when adding a resistance mechanism to a new design, it is essential to carefully evaluate whether the resistance mechanism is possible to be taken on by attackers as a whole.

Our attack has indicated the weakness of the two-layer Captcha, however, rather than discarding it completely, it is worthwhile to improve it and enhance its robustness. We have identified good design features for improving the security of two-layer Captchas, and provided guidelines for designing generic single-layer text-based Captchas.

As illustrated by our attack, the accepted practice in two-layer Captcha designs is not as secure as proposed. Designing better two-layer or multi-layer Captchas with higher security levels than their predecessors is an open problem we share with the whole research community. We hope our work can inspire novel attacks and defenses, and innovative designs in this interesting area.

References

[1] L. Von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard ai problems for security," in *Advances in CryptologyłEUROCRYPT 2003*. Springer, 2003, pp. 294–311.

[2] L. Von Ahn, M. Blum, and J. Langford, "Telling humans and computers apart automatically," *Communications of the ACM*, vol. 47, no. 2, pp. 56–60, 2004.

[3] E. Bursztein, M. Martin, and J. Mitchell, "Text-based captcha strengths and weaknesses," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 125–138.

[4] J. Yan and A. S. El Ahmad, "Usability of captchas or usability issues in captcha design," in *Proceedings of the 4th symposium on Usable privacy and security*. ACM, 2008, pp. 44–52.

[5] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski, "Building segmentation based human-friendly human interaction proofs (hips)," in *Human Interactive Proofs*. Springer, 2005, pp. 1–26.

[6] C. Kumar, L. Kevin, S. Patrice, Y., and C. Mary, "Computers beat humans at single character recognition in reading based human interaction proofs (hips)," in *CEAS 2005 - Second Conference on Email and Anti-Spam, July 21-22, 2005, Stanford University, California, USA*, 2005.

[7] J. Yan and A. S. El Ahmad, "Breaking visual captchas with naive pattern recognition algorithms," in *Computer Security Applications Conference, 2007. ACSAC 2007. Twenty-Third Annual*. IEEE, 2007, pp. 279–291.

[8] J. Yan and A. S. El, Ahmad, "A low-cost attack on a microsoft captcha," in *Proceedings of the 15th ACM conference on Computer and communications security*. ACM, 2008, pp. 543–554.

[9] E. Bursztein, M. Martin, and J. Mitchell, "Text-based captcha strengths and weaknesses," in *Proceedings of the 18th ACM conference on Computer and communications security*. ACM, 2011, pp. 125–138.

[10] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell, "The end is nigh: Generic solving of text-based captchas," in *8th USENIX Workshop on Offensive Technologies (WOOT 14)*, 2014.

[11] H. Gao, J. Yan, F. Cao, Z. Zhang, L. Lei, M. Tang, P. Zhang, X. Zhou, X. Wang, and J. Li, "A simple generic attack on text captchas," in *Proc. Network and Distributed System Security Symposium (NDSS). San Diego, USA*, 2016.

[12] K. Thomas, D. Mccoy, C. Grier, A. Kolcz, and V. Paxson, "Trafficking fraudulent accounts: the role of the underground market in twitter spam and abuse," *In USENIX Security Symposium*, pp. 195–210, 2013.

[13] E. Bursztein, A. Moscicki, C. Fabry, S. Bethard, J. C. Mitchell, and D. Jurafsky, "Easy does it: more usable captchas," in *Sigchi Conference on Human Factors in Computing Systems*, 2014, pp. 2637–2646.

[14] Y. Lecun, "Lenet-5, convolutional neural networks."

[15] M. Naor, "Verification of a human in the loop or identification via the turing test," 1996.

[16] D. Lillibridge, Mark, M. Abadi, K. Bharat, and Z. Broder, Andrei, "Method for selectively restricting access to computer systems," Feb. 27 2001, uS Patent 6,195,698.

[17] G. Mori and J. Malik, "Recognizing objects in adversarial clutter: Breaking a visual captcha," in *Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on*, vol. 1. IEEE, 2003, pp. I–134.

[18] P. Simard, "Using machine learning to break visual human interaction proofs (hips," *Advances in neural information processing systems*, vol. 17, pp. 265–272, 2005.

[19] S. Hocevar, "Pwntcha-captcha decoder web site," 2007.

[20] H. Gao, W. Wang, J. Qi, X. Wang, X. Liu, and J. Yan, "The robustness of hollow captchas," in *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. ACM, 2013, pp. 1075–1086.

[21] C. Karthik and A. Recasens, Rajendran, "Breaking microsofts captcha," 2015.

[22] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[23] B. M. ONeil, "Neural network for recognition of handwritten digits," 2013.

[24] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *null*. IEEE, 2003, p. 958.

[25] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Handwritten digit recognition with a back-propagation network," in *Advances in neural information processing systems*. Citeseer, 1990.

[26] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, "Multi-digit number recognition from street view imagery using deep convolutional neural networks," *arXiv preprint arXiv:1312.6082*, 2013.

[27] N. Otsu, "A threshold selection method from gray-level histograms," *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[28] H. Gao, W. Wang, Y. Fan, J. Qi, and X. Liu, "The robustness of" connecting characters together" captchas." *J. Inf. Sci. Eng.*, vol. 30, no. 2, pp. 347–369, 2014.

[29] D. J. Field, "Relations between the statistics of natural images and the response properties of cortical cells," *JOSA A*, vol. 4, no. 12, pp. 2379–2394, 1987.

[30] V. Turchenko and A. Luczak, "Caffe: Convolutional architecture for fast feature embedding," *Eprint Arxiv*, pp. 675–678, 2014.

[31] M. Mohamed, N. Sachdeva, M. Georgescu, S. Gao, N. Saxena, C. Zhang, P. Kumaraguru, P. C. Van Oorschot, and W. B. Chen, "A three-way investigation of a game-captcha: automated attacks, relay attacks and usability," in *ACM Symposium on Information, Computer and Communications Security*, 2014, pp. 195–206.

Haichang Gao is an associate 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.

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

Yi Liu is a master degree candidate in computer science at Xidian University. His current research interest is CAPTCHA.

Ping Zhang is a master degree candidate in computer science at Xidian University. His current research interest is CAPTCHA.

Xiyang Liu is a professor in Xidian University and a member of the IEEE. He has published more than twenty papers. Now he is in charge of a project of the National Science and Technology Major Project. Currently, he leads the Software Engineering Institute at Xidian University. His research interests include computer security and trustworthy computing.