JOURNAL OF INFORMATION SCIENCE AND ENGINEERING 35, 611-634 (2019)
DOI: 10.6688/JISE.201905_35(3).0008

Breaking Text-Based CAPTCHAs using Average Vertical Partition

XIYANG LIU, YANG ZHANG, JING HU, MENGYUN TANG AND HAICHANG GAO
Institute of Software Engineering
Xidian University
Xi'an, Shaanxi, 710071 P.R. China
E-mail: {xyliu; hchgao}@xidian.edu.cn

CAPTCHA, which stands for Completely Automated Public Turing Test to Tell Computers and Humans Apart, has been widely used as a security mechanism to defend against automated registration, spam and malicious bot programs. There have been many successful attacks on CAPTCHAs deployed by popular websites, *e.g.*, Google, Yahoo!, and Microsoft. However, most of these methods are ad hoc, and they have lost efficacy with the evolution of CAPTCHA. In this paper, we propose a simple but effective attack on text-based CAPTCHA that uses machine learning to solve the segmentation and recognition problems simultaneously. The method first divides a CAPTCHA image into average blocks and attempts to combine adjacent blocks to form individual characters. A modified K-Nearest Neighbor (KNN) engine is used to recognize these combinations, and using a Dynamic Programming (DP) graph search algorithm, the most likely combinations are selected as the final result. We tested our attack on the popular CAPTCHAs deployed by the top 20 Alexa ranked websites. The success rates range from 5.0% to 74.0%, illustrating the effectiveness and universality of our method. We also tested the applicability of our method on three well-known CAPTCHA schemes. Our attack casts serious doubt on the security of existing text-based CAPTCHAs; therefore, guidelines for designing better text-based CAPTCHAs are discussed at the end of this paper.

Keywords: CAPTCHA, security, text-based, K-nearest neighbor, average vertical partition

1. INTRODUCTION

Since its inception, CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart) has been widely used as a security mechanism to defend against automated registration, spam and malicious bot programs [1, 2].

Text-based CAPTCHA is the most widely deployed CAPCHA scheme [3]. It generally asks users to recognize the English letters or Arabic numerals shown in the CAPTCHA image. Researchers usually attempt to attack text-based CAPTCHA to verify their robustness.

Many successful attacks, such as those reported in [4-6], have emerged. However, the main limitation is that they are neither generalizable nor robust in detecting slight changes in the CAPTCHA. Previous work [7, 8] claimed that their generic methods can solve a large family of text-based CAPTCHAs with distinguished design features. However, existing CAPTCHAs have been improved, and various resistance mechanisms, such as noise arcs, complicated backgrounds and two-layer structures, have been adopted to make them difficult to attack by early generic methods.

Received November 1, 2017; revised May 4, 2018; accepted October 10, 2018.
Communicated by Jing-Ming Guo.

Moreover more websites randomly send CAPTCHAs with different design features to users. Baidu's user register mechanism (see Fig. 1). Therefore, it is clear that the creation of a generic attack method able to attack different text styles is needed.

(a) (b) (c)

Fig. 1. Three styles of Baidu CAPTCHAs using: (a) Character isolation; (b) Picture reversal; (c) Solid characters blended with hollow characters.

In this paper, we propose a simple but effective attack. It divides challenge images into average width blocks along the vertical direction. A modified KNN engine then tests different combinations of adjacent blocks to form individual characters. Finally, an efficient graph search algorithm is used to find the most likely combination as the final recognition result.

Our attack is effective on a wide range of text-based CAPTCHAs. We tested our attack against real CAPTCHAs selected from the top 20 websites identified by Alexa [9] in January 2015. Judged by the criteria commonly used as noted in [10, 11] our attack successfully broke all these schemes. The lowest success rate that we achieved was 5.0% on Google Street View, and the highest, at 74.0%, was on Google Maps. To verify its applicability, we also tested our attack on generally considered difficult versions of CAPTCHAs.

In contrast to the common practice of CAPTCHA analysis, our attack only includes two main steps and handles CAPTCHAs in a unified approach, except for pre-processing. Although appropriate pre-processing techniques were used for some complex CAPTCHAs, our attack is still simple, low-cost and powerful.

We believe that the high performance achieved by our attack has significant value. On The one hand, our work gives suggestions to corresponding websites. On the other hand, our attack does not indicate all the text-based CAPTCHAs are dead. Instead, our work plays a key role in promoting the next generation of text-based CAPTCHAs. Moreover, CAPTCHA designers can use our attack to test their new designs.

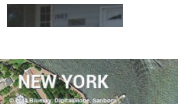
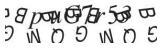
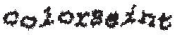
This paper is organized as follows. Section 2 introduces the real-world popular CAPTCHAs we collected from the top 20 websites. We describe our attack in detail in Section 3 and propose an approach to speed up the attack in section 4. Section 5 presents the attack results. In section 6, we verify the applicability of our attack, compare it with previous literature and discuss some defenses for resisting our attack. Section 7 discusses the value of the paper and draws conclusions

2. TARGETED CAPTCHAS

To evaluate the effectiveness of our attack, we selected a wide range of real-world CAPTCHAs with distinct design features. We chose to use the top 20 most popular websites in terms of Alexa ranking. Some of the websites use more Than one scheme simultaneously. Meanwhile, some websites use the same CAPTCHA scheme. In total we collected 13 CAPTCHA schemes with 17 styles, as summarized in Table 1.

Real-world text-based CAPTCHAs exhibit extensive variations in their designs. Based on font styles and positional relationships between adjacent characters, current text-based CAPTCHAs can be classified into three categories: character isolated CAPTCHAs, hollow character CAPTCHAs and CCT CAPTCHAs. Our attack schemes cover all these categories. Additionally, some schemes use noise arcs; some schemes use complex background; and some schemes use dynamic characters. Fonts and the number of characters used also vary across different schemes. Given that these schemes cover all the design styles, it is reasonable to evaluate the effectiveness of our attack using these schemes.

Table 1. Targeted CAPTCHA schemes.

Scheme	Website	Example image	Characteristic
Baidu	baidu.com hao123.com		CCT scheme, rotation used, background confusion, varied font size, hollow and solid combination
Taobao	taobao.com		CCT scheme, rotation used, large alphabet set
Google	google, twitter, youtube, linkedin, twitter, wordpress, blogspot, google.co.in		Background confusion, varied CAPTCHA length, varied font size, background confusion
Sina	sina.com.cn		CCT scheme, hollow scheme, noise interference, varied font size
Weibo	weibo.com		CCT scheme, rotation used
Amazon	amazon.com		CCT scheme, rotation used, constant font
QQ	qq.cn		Background confusion, rotation used, varied fonts
eBay	ebay.com		Only digits used, rotation used, varied fonts
Microsoft	live.com bing.com		Hollow and solid combination, CCT scheme, rotation used
Yahoo!	yahoo.com yahoo.co.jp		Dynamic characters, varied CAPTCHA length, overlap used
Facebook	facebook.com		Rotation used, varied CAPTCHA length, complex lines
Wikipedia	wikipedia.org		Character isolated scheme, varied CAPTCHA length, rotation used

3. ATTACK DETAILS

Our attack uses machine learning to simultaneously solve the segmentation and recognition problems. The key processes behind our attack are as follows.

We first divide a challenge image into different blocks in the vertical direction. This partition is an average vertical partition (AVP) based on average character width. Then, a modified version of KNN [12] is used as the recognition engine to test different combinations of adjacent blocks to form individual characters based on its reputation as a top performer in text recognition. For character recognition, the modified KNN engine works by computing the similarity between corresponding pixels of the unknown image (the image to be classified) and the known sample image (the image already classified). To decrease the influence of background pixels, it distributes a larger weight for a matched black pixel and a smaller weight for a matched white pixel but a negative weight for a pixel that does not match. The KNN algorithm analyzes the K most similar known sample images and then determines which character the unknown image is. The sum of each pixel's weight is returned as the confidence level of a recognition result. The value of K, as a significant element of the KNN engine, can be found by cross-validation. Finally, a graph search algorithm is used to find the most likely combination as the final result.

Our attack includes two main steps: pre-processing, partition and recognition. Pre-processing uses standard techniques to eliminate the noise of the CAPTCHA images. During partition and recognition, the KNN engine and graph search algorithm are used to test possible combinations and find the most likely one as the recognition result. Fig. 2 shows the pipeline of our attack.

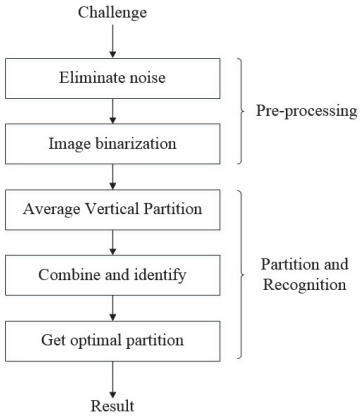


Fig. 2. The pipeline of our attack.

3.1 Pre-processing

As presented in Table 1, most of the targeted schemes are relatively complicated. Therefore, appropriate pre-processing is necessary in attacking CAPTCHAs. First, this pre-processing binarizes the challenge images which convert the challenge image to black and white. Second, it removes background patterns or eliminates other noise in the image that could interfere with further extraction and recognition.

One form of background interference is complicated backgrounds, where, as the QQ scheme shows (see Fig. 3), characters are embedded in street view. Large differences between background and foreground pixel HSV (Hue, Saturation and Value) values are used to remove the background. Fig. 3 (b) illustrates the pre-processing result on a sample image.

As an alternative interference measure, noise arcs appear in many CAPTCHAs, including Facebook and Sina CAPTCHAs. The noise arcs appearing in Facebook CAPTCHA can be easily removed by erosion and dilation (see Fig. 4). For Sina CAPTCHA, analysis shows that the pixels of these noise arcs are the same color, and the pixels at the intersections of character contours and noise arcs are also the same color (see Fig. 5). Based on these characteristics, noise arcs and junction pixels can be easily found. Pixels in noise arcs but not in the junctions are set to white.

(a) Original image. (b) After pre-processing.

Fig. 3. QQ CAPTCHA.

(a) Original image. (b) After pre-processing.

Fig. 4. Facebook CAPTCHA.

(a) Original image. (b) Noise arcs and intersection pixels. (c) After preprocessing.

Fig. 5. Sina CAPTCHA.

For the Microsoft scheme, the two-layer image is separated into two single-layer images. First, we use Color Filling Segmentation (CFS) [4] to convert hollow characters to solid. Then, we detect the top pixels of the CAPTCHA image to generate an envelope line and detect the bottom pixels to generate another envelope line. Finally, the borderline of the upper layer and lower layer can be identified by examining these two envelope lines, as Fig. 6 shows.

(a) Original image. (b) After CFS. (c) Borderline. (d) After preprocessing.

Fig. 6. Microsoft CAPTCHA.

(a) Original image. (b) After pre-processing.

Fig. 7. Yahoo! CAPTCHA.

The Yahoo! scheme uses dynamic CAPTCHA with vertically moving foreground characters and horizontally moving background characters. The background characters are typically much thinner than the foreground characters; thus, the background characters can be easily removed by a double erosion and dilation (see Fig. 7) process.

3.2 Partition and Recognition

After pre-processing, we divide a binarized image into average blocks in the vertical direction and attempt to find the most likely combination of adjacent blocks to form individual characters and recognize the characters. In general, the number of blocks extends far beyond the number of characters to be formed, creating many possible combinations. A modified KNN engine is used to test and recognize these combinations. With an efficient graph search algorithm, we can find the most likely combinations as the final result.

Due to page limits, the Taobao scheme is used to explain key techniques in this process, with details of the Baidu and Sina attack schemes in the Appendix. The technical details are as follows:

Step 1: Average vertical partition (AVP) The AVP is a simple partition method to divide a challenge image into different blocks. The widths of the first and the last block are set equal to the smallest possible character width (see Fig. 8), which can reduce the block numbers and reduce the attack time. The smallest possible character width can be found by a statistical analysis of the widths of all individual characters in the training set. The retained middle part is then divided into average blocks along the vertical direction based on the average character width.

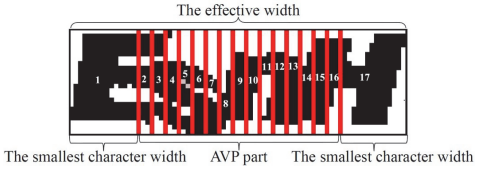


Fig. 8. Average vertically partition.

To discover a suitable width for average blocks, we defined the average character width ω_a, the average CAPTCHA string length l_a and the width of average blocks ω_b for each challenge: $\omega_a = W/l_a$, $l_a = (l_{max} + l_{min})/2$ and $\omega_b = \omega_a/p$, where W denotes the effective width of the challenge, l_{max} and l_{min} denote the maximum and minimum CAPTCHA string length, respectively, and p denotes the number of blocks into which a character will be divided on average. If one scheme has a fixed CAPTCHA string length, the fixed length is the average CAPTCHA length (*e.g.*, Taobao, Baidu, and Sina).

The success rate increases nonlinearly with increasing p value, but the time consumption rapidly increases. To find an appropriate p, we empirically tested different values in experiments to create a balance. Typically, the value of p that we chose varies across different schemes; it always ranges from 6 to 8. For example, 6 is chosen as the p value for the Taobao scheme.

Note, it is difficult to display and explain the following search graph if we use $p = 6$ directly. The search graph and table are too large and complicated. Under the premise of

not affecting the attack results, we set $p = 4$ in the Taobao example. Thus, we can present our tables and figures on one page and illustrate our attack more clearly. In a real attack of the Taobao scheme, $p = 6$ is used. We applied the same process for the examples in Baidu and Sina (see the Appendix).

Once the width of the average blocks ω_b is obtained, the retained middle sections (AVP part) are divided into blocks with width ω_b, as Fig. 8 shows. All blocks in a challenge are numbered from left to right. There are 17 blocks for the example shown in Fig. 8. Note that with differing effective challenge image lengths and the value of p chosen, the total number of blocks for each challenge will be different.

Step 2: Combination and recognition. In this step, we test different combinations of adjacent blocks to form and recognize individual characters.

To record the recognition results, an $n \times n$ table is built for each image, where n is the total number of blocks. If combining blocks i, $i + 1$, ..., j together is feasible to form a single character, KNN is used to determine which character such a combination is likely to be, and the cell (i, j) at the intersection of row i and column j in the table will be filled with the recognition result returned by the KNN engine. Otherwise, the cell (i, j) will be set to NULL if the combination is unfeasible. This occurs in one of the following two scenarios: 1) when its row index i is larger than its column index j and 2) when the width of the combination is greater than the largest possible character width or less than the smallest possible character width. The largest possible character width and the smallest possible character width can be easily received by a simple analysis of the training set, and the value of K, as a significant element of KNN, can be found by cross-validation.

Table 2 shows the $n \times n$ table for the Taobao sample, in which cell (1, 3) indicates that KNN recognizes the combination of blocks 1 to 3 as '*E*' with a confidence level of 0.76. Cells highlighted in the table are the combinations that we aim to detect finally. This process will be illustrated in detail in step 3.

Table 2. *n*×*n* table for Taobao scheme.

	1	2	3	4	5	6	7	8	9	10	11
1	E/0.67	***E/0.77***	E/0.76	E/0.64							
2						f/0.60	6/0.59	Z/0.50	Z/0.49		
3							S/0.67	***S/0.68***	V/0.55	V/0.57	
4								S/0.59	V/0.55	V/0.56	U/0.60
	9	10	11	12	13	14	15	16	17		
5	U/0.55	U/0.57	J/0.61	V/0.49							
6		J/0.74	J/0.74	J/0.60	P/0.61						
7			J/0.65	J/0.60	n/0.61	P/0.64					
8				P/0.57	n/0.59	W/0.59	N/0.60				
9					***n/0.66***	h/0.55	d/0.60	d/0.62			
10						n/0.54	m/0.60	d/0.59			
11							V/0.64	f/0.60			
12								d/0.67			
13											
14									***H/0.74***		
15									H/0.81		
16									H/0.58		
17									Y/0.56		

Step 3: Find optimal combination In this step, we converted the $n \times n$ table into an equivalent graph, and a graph search algorithm is used to find an optimal combination as the final result.

Create equivalent graph If cell (i, j) in the $n \times n$ table is not NULL, it will be represented as an edge from node i to node $j + 1$ in the equivalent graph. According to the principle, we build the equivalent graph of Table 2, as shown in Fig. 9. This means that both cell (1, 3) in Table 2 and the edge from node 1 to node 4 in Fig. 9 indicate that the KNN engine recognizes the combination of blocks 1 to 3 as '*E*' with a confidence level of 0.76.

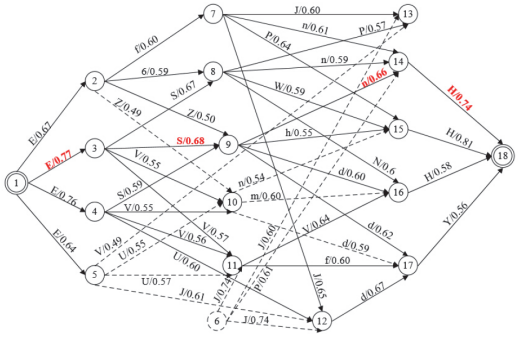


Fig. 9. The equivalent graph of Table 2.

In Section 4, we propose an approach to improve the efficiency of our attack. Those edges and nodes marked with dotted lines in Fig. 9 will be removed because such recognition results are useless for our purpose in most cases; this will be explained in detail later.

Search the equivalent graph Now, the problem of finding an optimal combination is 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 our algorithm manages to find a path with the largest average confidence level but not the largest confidence level sum, which means that it works on not only CAPTCHAs with a fixed string length but also CAPTCHAs with varying string lengths.

We propose a dynamic programming (DP) graph search algorithm. The overlapping sub-problem for DP is for each node j and for each step number (the number of edges) required, the average confidence level along the path ending at node j with this step number should be the largest. The algorithm examines all paths for each node j, and where there may be several possible paths with different step numbers for one node, each case should be recorded. We use a bottom-up approach to find the sub-problem's solution.

The following pseudocode illustrates our process. The traversal starts from node 1 and ends at node $n + 1$, with R being final recognition result. *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, and *step* stores step number of the possible path (the number of edges on the corresponding path or the number of currently recognized characters). *newPath* is a path whose *value* is 0, the *result* is NULL, and the *step* is 0. An array *pathList* stores the paths found by our algorithm for each node. Each element is a list of paths. *Pathj* is a temporary variable; it is also a list of *path*, which is used to store the paths for one node casually. *prej* is a list that stores all the precursors of node *j*. *con*[*i*; *j*] is the confidence

level calculated by KNN 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.

```
Procedure Main()
 Begin
 R←NIL
 for j ← 1 to N + 1
    pathList[j] ← null
    GetValue(j, pathList)
 Select(n + 1, pathList, R)
 End

Procedure GetValue(j, pathList)
 Begin
 path j ← null
 if j = 1
    newPath.step ← 0
    newPath.value ← 0
    newPath.result ← null
    Add(path j, newPath)
 else
    for each i in pre j
       for each path in pathList[i]
          n ← −1
          if ExistStep(path j, path.step + 1, n)
             if path.value + con[i; j] > path j[n].value
                path j[n].value ← path.value + con[i; j]
                path j[n].result ← path.result + char[i; j]
             else
                newPath.step ← path.step + 1
                newPath.value ← path.value + con[i; j]
                newPath.result ← path.result + char[i; j]
                Add(path j, newPath)
 pathList[j] ← path j
 End

Procedure ExistStep(path j, step, num)
 Begin
 for i ← 1 to path j.length
    if path.step in Captcha length
       if path j[i].step = step
          num ← i
          return true
 return false
 End
```

```
Procedure Select(num, pathList, R)
 Begin
 averageValue ← 0
 for each path in pathList[num]
    if path.step in Captcha length
       if path.value/path.step > averageValue
          averageValue ← path:value/path:step
          R ← path:result
 End
```

Table 3 shows the procedure of finding the optimal partition with our DP algorithm for the example in Fig. 8. DP simplifies the search process by recording the largest confidence sum of each node and each reachable step number. The path marked with red in Fig. 9 is the final path that our algorithm detected, and the italic item highlighted in Table 3 indicates the optimal combination that has the largest average confidence level. Specifically, "*ESnH*" is the recognition result in this case.

With this early version of our attack method, we achieved a success rate of 24.2% on the Taobao scheme, and its average attack speed was 4.89 seconds. The attack speed increases with the effective width of a CAPTCHA image since AVP will produce numerous blocks. Thus, there will be a large number of possible combinations, and then, the increased number of times that we call the KNN engine will decrease the attack speed. The lowest average attack speed was on the Facebook scheme, approximately 28 seconds. However, this is acceptable because the requirement of usability in CAPTCHA design is to demand that each human user solve a CAPTCHA within 30 seconds. However, we still propose an approach in Section 4 to speed up our attack and improve the performance of our method.

Table 3. The search process for Taobao CAPTCHA.

j	*step*	*Path*	*value*	*result*
2	1	1→2	0.67	E
3	1	1→3	0.77	E
4	1	1→4	0.76	E
5	1	1→5	0.64	E
7	2	1→2→7	1.27	Ef
8	2	1→3→8	1.44	ES
9	2	1→3→9	1.45	ES
10	2	1→3→10	1.32	EV
11	2	1→3→11	1.34	EV
12	2	1→4→12	1.36	EU
12	3	1→2→7→12	1.87	EfJ
13	2	1→5→13	1.13	EV
13	3	1→3→8→13	2.01	ESP
14	3	1→3→9→14	2.11	ESn
15	3	1→3→8→15	2.03	ESW
16	3	1→3→9→16	2.05	ESd
17	3	1→3→9→17	2.07	ESd
17	4	1→2→7→12→17	2.54	EfJd
18	*4*	*1→3→9→14→18*	*2.85*	*ESnH*

4. SPEED UP

In this section, we propose an approach that makes our attack faster at the expense of success rate. Note that the approach proposed in this section applies equally well to all CAPTCHA schemes. Therefore, it has no influence on the generality of our attack.

A simple time cost analysis shows that a significant percentage of the attack time is spent on calling the KNN engine in our attack because each scaled combination image must be compared with known sample images in the training set. Thus, we propose some methods to reduce the use of the KNN engine.

During cross-validation, we test our KNN engine with different values of K and then find a K with the largest recognition rate. We choose the smallest confidence level of the correct recognition results as a threshold from cross-validation. If the confidence level of one combination is smaller than this threshold, we consider the corresponding recognition result to be incorrect or that such a combination is unreasonable. We can remove those combinations from the graph to reduce the search space; however, a far better approach would be to not use KNN to recognize them any more if we can predict that their confidence levels are very small. To our surprise, we indeed found some approaches to achieve this.

After an exhaustive analysis of the data presented in Table 2, we find that within the width range, combining with block i, the confidence level varied depending on the number of combined blocks. If the combinations with a higher confidence level than the threshold that we detected have appeared, then a combination with a smaller confidence level than this threshold appears, and almost all later possible combinations have a confidence level that is smaller than the threshold. Thus, the confidence level that was smaller than the threshold first appeared to be chosen as a sign of stopping the later combination process. Once it appeared, we no longer continue to use KNN to attempt later combinations, even though these combinations' widths seem to be feasible for forng individual characters. In addition, for each start block i, the three components with the three largest confidence levels are always concentrated, and this centrostigma is most likely the break point between characters. The final optimal combination chosen by our algorithm also proves this. Therefore, we do not need to use each block as the start block to combine with later blocks to form a possible component. Instead, for the current start block, we discover the three components with the three largest confidence levels and mark their next block as the start block for the next combination process.

Based on the above analysis, we propose a dynamic combination method. This method examines the process of combination and recognition and generates a new $n \times n$ table. The details are as follows:

1. Initialize an empty queue to store blocks that can be a start block and initially add block 1 to the queue.
2. Take out a start block i, use the KNN engine to attempt combinations that start from it until the stopping condition is satisfied. The stopping condition is when a combination with a higher confidence level than the threshold is found and when a combination with a smaller confidence level than this threshold first appears.
3. For the current start block i, find three components with the three highest confidence levels and examine the next block for each of them to determine whether it has been added to the queue of the start blocks. If not, add it to the queue.

4. Repeat steps 2 and 3 until the queue is empty.
5. Remove all the components whose confidence level is smaller than the threshold.

The threshold for the Taobao scheme is 0.50. Table 4 shows the final $n \times n$ table obtained by our new dynamic combination method, where only the components useful for finding the optimal partition are retained. Fig. 10 shows the equivalent graph of Table 4. The edges and nodes marked with dotted lines in Fig. 9 are removed because the dynamic combination method did not test these useless combinations further or because such a combination is incorrect; the usage of the KNN engine decreases noticeably. For the Taobao scheme, when the dynamic combination method was used, we achieved a final success rate of 24.6%, with an average attack speed of 3.43 seconds. The success rate is slightly higher than previously found in this case, and the improvement in speed is obvious.

The approach speeds up the attack which forecasts all the components that seem to have relatively small confidence levels. It also removes the components that have been recognized but seem to be incorrect or unreasonable. Compared with the original attack, our speed-up method will lead to varying success rates (see Section 5), and it is difficult to confirm that the success rate of a scheme will increase or decrease. However, this variability is trivial. What is more important is that the improvement in our speed is huge.

Table 4. The final *n*×*n* table for Taobao scheme received by using dynamic combination.

	1	2	3	4	5	6	7	8	9	10	11
1	E/0.67	***E/0.77***	E/0.76	E/0.64							
2						f/0.60	6/0.59	Z/0.50			
3							S/0.67	***S/0.68***	V/0.55	V/0.57	
4								S/0.59	V/0.55	V/0.56	U/0.60
	9	10	11	12	13	14	15	16	17		
5	U/0.55	U/0.57	J/0.61	V/0.49							
6											
7			J/0.65	J/0.60	n/0.61	P/0.64					
8				P/0.57	n/0.59	W/0.59	N/0.60				
9					***n/0.66***	h/0.55	d/0.60	d/0.62			
10											
11							V/0.64	f/0.60			
12								d/0.67			
13											
14									***H/0.74***		
15									H/0.81		
16									H/0.58		
17									Y/0.56		

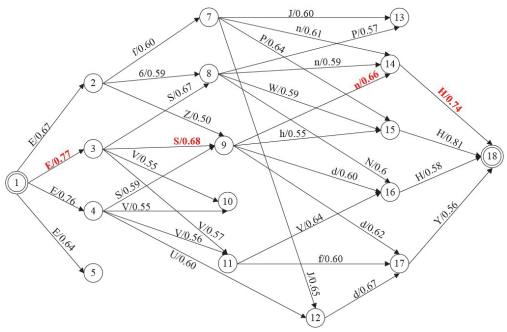


Fig. 10. The equivalent graph of Table 4.

5. RESULTS

We tested our attack on all schemes in Table 1. For each scheme, we collected 500 random CAPTCHAs as the sample sets (used for extracting character samples) and another 500 as test sets from the corresponding websites. All these challenge images were collected in January 2015.

We ran both our early attack (without using the dynamic combination method) and the improved attack (using the dynamic combination method) on the test set, where our program had no prior knowledge about any particular sample within. The success rate and average attack speed for each scheme are shown in Table 5.

Table 5. Attack results.

Scheme	Without using dynamic combination method		Using dynamic combination method	
	Success rate	Speed(s)	Success rate	Speed(s)
Baidu	54.2%	10.47	52.0%	6.28
Taobao	24.2%	4.89	24.6%	3.43
Google Street View	5.0%	3.77	5.0%	2.10
Google Map	74.0%	5.04	74.0%	3.67
Sina	37.6%	15.00	35.6%	8.53
Weibo	31.8%	8.25	32.6%	4.02
Amazon	51.6%	20.72	49.2%	7.47
QQ	46.4%	10.62	43.0%	5.57
eBay	47.2%	11.62	45.6%	5.48
Microsoft	15.2%	19.66	14.2%	6.12
Yahoo!	20.4%	21.16	20.0%	12.58
Facebook	12.4%	28.06	13.0%	16.56
Wikipedia	37.4%	10.99	36.6%	5.37

Success rate The 1500 selected character samples that we extracted from sample sets were normalized to 28×28 pixels as the training sets for the KNN engine. This meant that not all the CAPTCHA images in the sample sets are used to extract character samples. The number of CAPTCHA images in the sample sets that we used varies between CAPTCHA schemes. However, we ensured that we used 1500 character samples for each scheme.

The success rates of our final attack range from 5.0% to 74.0%. Our attack can break most schemes with a success rate of greater than 13% except for Google Street View (5.0%). A commonly accepted goal for CAPTCHA robustness is to prevent automated attacks from achieving greater than a 0.01% success rate [10]. However, this goal was considered too ambitious by some researchers. For example, [11] suggested that a CAPTCHA scheme is broken if an automated attack achieves a success rate of 1%. According to either criterion, our attack has broken all the CAPTCHAs deployed by the top 20 websites.

Our success rates for breaking the Google Street View scheme, Microsoft scheme and Facebook scheme are relatively low, and we attempted to determine possible reasons for this.

The Google Street View scheme uses street view pictures with digits and characters, and our attack achieved the lowest success rate on it. Although the QQ scheme also uses

street views as background, they are essentially different. Obvious differences between the foreground and background in the QQ scheme can be used to easily find and erase the background. However, the foreground characters of the Google Street View scheme are part of Google Street View (see Table 1). It is difficult to precisely detect where the characters are, and thus, it is difficult to completely remove the background for each challenge. All these factors affected the success rate.

Microsoft's CAPTCHA, a two-layer structure that uses both solid and hollow characters, is a relatively complicated scheme. For this scheme, pre-processing played an important role in our attack. It is clear that the failure of filling the hollow characters by CFS and separating the upper and lower layers will influence our success rate. Moreover, the characters in the Microsoft challenge images are rotated by different angles. This is a defense against our attack (see Section 6).

For the Facebook CAPTCHA, erosion and dilation were used to remove its intensive noise arcs. However, the foreground characters are also seriously damaged during this process and will further affect the recognition accuracy of individual characters and the success rate.

Average attack speed We implemented our attack in C# and tested all the target schemes on a desktop computer with a 3.3GHz Intel Core i3 CPU and 2 GB of RAM.

The time required by our final attack ranged from 2.10 seconds to 16.56 seconds. The Yahoo! and Facebook schemes took longer (12.58 seconds and 16.56 seconds, respectively). The following explains why it takes more time to attack these two schemes. First, both Yahoo! and Facebook have a wider range of individual character widths, leading to the production of a larger number of possible combinations. Additionally, they use a relatively larger value of p (7 is chosen for the Yahoo! Scheme, and 8 is chosen for the Facebook scheme), which increases block production. The increased KNN engine use and the enlargement of the search space of the equivalent graph result in increased attack time. Additionally, the longer CAPTCHA string lengths of Yahoo! and Facebook also have a similar influence on the speed.

Comparison The success rates of the schemes using the dynamic combination method decreased slightly (no more than 3.4%) than before (*e.g.*, Baidu, Sina, Amazon, QQ, eBay, Microsoft, Yahoo! and Wikipedia), which is trivial. However, the time consumption is greatly reduced (49% decrease on average). More importantly, the success rates of some other schemes increased (*e.g.*, Taobao, Weibo and Facebook).

As illustrated in Section 4, the variations in success rates cannot be avoided. Relative to the negligible decrease in success rates under some schemes, the significant improvement in speed is quite important. Therefore, our speed-up method still has a positive effect on the attack performance.

6. DISCUSSION

6.1 Redundant Nodes Removal

Obviously, some nodes in the equivalent graph are redundant (*e.g.*, nodes 5, 10 and 13) and will never be on the targeted path (see Fig. 10). To reduce the complexity of the

graph search and save time, we can remove them with a simple algorithm.

For each node *i*, we compute the shortest path from node 1 to node *i* and the shortest path from node *i* to node *n*+1. If the sum of the steps of these two shortest paths is larger than the largest possible CAPTCHA string length, node *i* is considered redundant and will be removed, as well as the edges connected to it. Fig. 11 shows the equivalent graph after removing redundant nodes (*e.g.*, nodes 5, 10 and 13) and their connected edges.

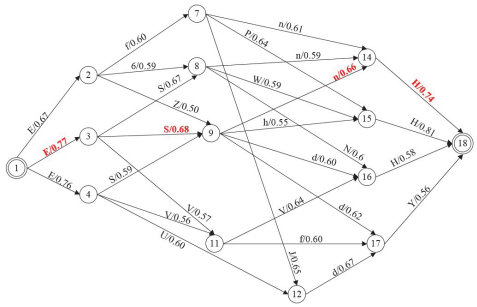


Fig. 11. The equivalent graph after removing redundant nodes.

However, the time required for graph searching only represents a small proportion on the total attack time, whereas a greater proportion of time is spent using the KNN engine. Although removing these redundant nodes can save some time, this reduction is less than 0.1 seconds on average. It demonstrates that this step does not add value. We did not implement it in our final attack.

6.2 Effects of Pre-processing

Pre-processing techniques were used to remove background, noise arcs and other aspects in certain complicated schemes. To investigate how pre-processing effects our attack results, we take the QQ and Sina schemes as representatives of CAPTCHAs. Instead of removing their background or noise arcs, we simply binarize them during pre-processing. The later segmentation and recognition process is similar to that in Section 3. The difference is a noise class added to cover segmented blocks that only contain background or noise arcs.

Without special pre-processing techniques, our attack achieves success rates of 5.0% and 12.5% on the QQ and Sina schemes, respectively, which are much lower than those achieved by previous attacks (46.4% on QQ and 37.6% on Sina). This indicates preprocessing contributes significantly to our performance. Existing CAPTCHAs are significantly more sophisticated than previous schemes; therefore, we believe appropriate preprocessing techniques are necessary. However, our attack still successfully broke these two CAPTCHA schemes according to the commonly used criteria [10, 11] even without removing background or noise arcs.

6.3 Applicability

To verify the simplicity and effectiveness of our attack method, we tested it on otherCAPTCHAs.

The first case is the Yandex scheme (see Fig. 12 (a)). It is used by the Russian search engine Yandex in its user password recovery mechanism. This is a hollow CAPTCHA, blending thick intersecting interference arcs and broken contours. Previous work [13] proposed a general method to break hollow CAPTCHAs, but both broken contours and thick interference are actually the main defense methods recommended in [13] to defeat their attack.

Another scheme is the difficult Yahoo! scheme (see Fig. 12 (b)). The Yahoo! version that we broke in the previous section is a new, dynamic version that was rolled out by Yahoo! in January 2015. [7] considered the difficult Yahoo! scheme as the most difficult, and they achieved the lowest success rate on it using their attack method.

We also tested our method's applicability to an old version of reCAPTCHA (see Fig. 12 (c)). Characters in this version of reCAPTCHA are heavily rotated and closely connect. This strong defense method defeats attacks and will be illustrated in detail in a later section. In CCS'11, [11] reported that the Stanford team achieved a zero success rate on this version.

(a) Yandex CAPTCHA. (b) Hard Yahoo! CAPTCHA. (c) Early reCAPTCHA.

Fig. 12. Three generally considered difficult CAPTCHAs.

In contrast, our attack achieves a success rate of 9.0% on the Yandex scheme, 8.0% on the difficult Yahoo! scheme and 2.6% on reCAPTCHA. The average attack speed is 8.45, 11.56 and 14.4 seconds, respectively.

According to the criteria proposed in [11], the results indicated that we have successfully broken all three difficult schemes. We broke the Yandex scheme without removing its interference arcs or repairing the broken contours. For the difficult Yahoo! scheme, our attack achieved a success rate of 8%, which is significantly better than the result reported in [7] (5.33%). Our success rate in attacking the old version of reCAPTCHA is significantly lower than that of a recent work by Google [14]. However, for our attack, we only used 300 training samples (including 1500 extracted character images) compared with the millions of training samples used by Google. Additionally, Google's approach requires sophisticated deep-learning algorithms, tens of millions of training images, an advanced distributed computing infrastructure, and computers with powerful CPUs and huge memory resources. Our attack is simple and does not have special hardware requirements. It is designed to work on standard computers.

6.4 Comparing with Prior Work

Since Moni Naor [15] first proposed the concept of automated Turing tests, many scholars have focused on proposing valid automated Turing tests and methods for breaking them.

Many innovations and notable attack methods have been proposed. Jeff Yan and El Ahmad [4-6] proposed methods, including histogram analysis, vertical segmentation algorithms, and CFS, to extract individual characters. These methods often serve as a part of a successful attack, but when used alone, only occasionally do they constitute a

successful attack. Other *ad hoc* attacks also include [16-20], among which [16-18] reported various attacks on early versions of reCAPTCHA, and [16, 19] presented attacks against Microsoft's CAPTCHA.

Gao *et al.* [14] reported an attack on a family of hollow CAPTCHAs at CCS'13. This is the first research on solving CAPTCHAs in a single step. However, this method only works on hollow CAPTCHAs because it utilizes the characteristics of hollow fonts to extract character strokes, whereas our attack works on almost all CAPTCHAs. Because increasingly more websites use CAPTCHAs with different styles and design features, there is an opportunity to create a simple and generic attack method to solve different text styles.

Bursztein *et al.* [11] proposed a CAPTCHA breaker, Decaptcha, which is claimed to be a generic attack. It uses a five-stage pipeline: preprocessing, segmentation, post-segmentation, recognition, and post-preprocessing. For each stage, various techniques are used for different CAPTCHAs. However, our attack only requires two main steps. Aside from special techniques used for some complicated CAPTCHA schemes to remove background and noise arcs or obtain foreground characters in pre-processing, all these schemes are processed in the same manner during the partition and recognition procedure. From Table 1, we find that multiple pre-processing techniques for different CAPTCHAs are necessary with increasing CAPTCHA complexity. Undoubtedly, our attack is simpler and more general than Decaptcha. Additionally, we tested our attacks on early reCAPTCHA, eBay and Wikipedia schemes, which they reported as well. Decaptcha failed to break the early reCAPTCHA, whereas our attack was able to break it. Our attack also achieves a success rate of 45.6% on eBay CAPTCHA and 36.6% on Wikipedia CAPTCHA, being better than the results (43% on eBay CAPTCHA and 25% on Wikipedia CAPTCHA) reported in [11]. We used fewer than 300 CAPTCHAs (1500 character samples) as training sets for the KNN engine, whereas they used 500 CAPTCHAs. The line chart illustrated in [11] indicated that enlarging the training set would increase the success rate. Previous work [7] proposed another generic attack that includes four components: cut-point detector, slicer, scorer and arbiter. The cut-point detector finds all possible cuts by examining the second derivative of the curve generated by following the top pixels of the CAPTCHA and the curve generated by following the bottom pixels of the CAPTCHA.

Each cut is constructed by connecting the inflection points one from the top and one from the bottom. Next, the slicer applies heuristics to extract the meaningful potential segments based on the cut points. A large number of segments are generated in this step; thus, their computation time in their early version of the algorithm is huge (a 12-letter-long CAPTCHA took up to 9 hours to compute). Optimizations were proposed to make their algorithm run faster but at the expense of accuracy. These optimizations included reducing the number of cuts by removing all the cuts that have an angle of greater than 30 degrees as well as the cut lines that pass through too many black pixels, therein performing local decision recognition rather than looking at the entire CAPTCHA. Then, the scorer performs OCR on segments and assigns a recognition confidence score to each of them, and the arbiter processes the scores and determines which letters have the highest probability.

It is obvious that their attack is significantly more complex than our attack. Although they have taken measures to reduce the number of segments, their segment number is still significantly higher than our segment number. Although we also propose an

approach to increase the attack speed, our method is still simpler than their method. More importantly, we do not rely on any human intervention, as in their method.

At NDSS'16, Gao *et al.* [8] proposed a simple generic attack based on Gabor filters. The novelty of their method was utilizing Gabor filters to first extract character components along four different directions. Their method seems to be effective on the schemes presented in their paper, but when we tested it on CAPTCHAs with complicated background, it always introduced extra noise components. In addition, currently deployed text-based CAPTCHAs have various defense mechanisms, some of which being extremely new, *e.g.*, two-layer structures, backgrounds of natural scenes, and dynamic characters, as shown in Table 1. These CAPTCHAs are significantly more complicated than prior CAPTCHAs; thus, the applicability of their emphasized method is suspect. In contrast, our attack overcame not only some of the most promising defenses but also these new defense mechanisms; certainly, our attack has greater applicability.

Recent attacks on text-based CAPTCHAs are typically based on deep learning techniques. Previous work [21] introduced a Captcha-breaking network combining convolutional neural networks (CNNs) and recurrent neural networks (RNNs). However, they only tested two real-world CAPTCHA schemes (Facebook and Wikipedia), whereas our work has utilized a more comprehensive analysis that includes 13 CAPTCHA schemes. Tang *et al.* [22] proposed a CNN-based method that successfully broke 14 text-based CAPTCHAs and three Chinese CAPTCHAs. However, an important characteristic of these deep-learning-based attacks is the large amount of training data, and advanced hardware (*e.g.*, GPUs) is necessary. Our attacks need less training data and can be run on a standard desktop computer.

Reference [23] proposed a network named Recursive Cortical Network (RCN) to break text CAPTCHAs and successfully broke four schemes, with success rates varying from 57.1% to 66.6%. This is an important work that is based on deep learning that only requires a few clean individual characters to train the network. However, some parameters must be manually tuned to prepare these characters. Most importantly, their attack time is 94 seconds, which is extremely slow.

6.5 Defense

Clearly, some countermeasures may circumvent our attack to an extent by mitigating segmentation and recognition. Considering these two perspectives, we derived the following core set of design principles that CAPTCHA designers need to follow to create schemes resilient to state-of-the-art attackers or mitigate our attack.

Rotation Our attack aims to find the separation lines between characters, and these separation lines are vertical. By rotating the characters, the vertical separation lines will no longer be able to be used to separate adjacent characters completely, as the old reCAPTCHA example shows in Fig. 12 (c). To evaluate the effectiveness of rotation as a defense, the old reCAPTCHA is chosen for an experiment. We calculated the rotation angles of the challenge images, and then rotated and straightened the characters (see Fig. 13). The success rate achieved by our attack on the rotated test set is 20.8% vs. 2.6% on the original set. This indicates that rotating does have a positive effect in enhancing security.

(a) (b) (c)

Fig. 13. Rotating defense on old reCAPTCHA: (a) Separation lines of an old reCAPTCHA example; (b) Find rotation angel; (c) Rotated image.

Overlapping Overlapping removes space between characters and makes them more complex; it is considered to be by far the most secure anti-segmentation technique [11]. Because of the strong overlap by the characters in eBay CAPTCHAs (Fig. 14), we chose these CAPTCHAs as a case study to evaluate the effectiveness of overlapping. A total of 200 CAPTCHAs with overlapped characters are chosen manually as the test set. The new success rate is 23.0%, which is significantly lower than the total success rate (45.6%). Undoubtedly, overlapping can reduce the attack success rate.

(a) Example with overlapped characters. (b) Example without overlapped characters.

Fig. 14. Examples of eBay CAPTCHA.

Increasing the alphabet size To verify the influence of alphabet size on the success rate, we chose the eBay and Taobao schemes for comparison. The two schemes have the following in common: no noise arcs or background confusion, and they use rotation and CCT to defend against attacks. Only digits are used in the eBay scheme, and the alphabet size is 10, whereas Taobao uses both digits and letters and uses an alphabet size of 42. The attack success rate of the eBay scheme is 45.6%, whereas Taobao's success rate is only 24.6%, significantly lower than that of eBay's scheme. This indicates that increasing the alphabet size has a positive effect on resisting our attack.

Increasing the effective length of the CAPTCHA image Using a relatively large effective image increases the resistance to attack. The larger the effective length of the utilized image, the more blocks that are retained after AVP, and the larger the solution space will be. This will decrease the attack success rate and speed.

Enlarging the width range of individual characters This process contributes to security because the larger the width difference between the thinnest and widest characters, the more possible combinations that will be produced and the more time that the KNN engine requires to recognize them.

Using varying CAPTCHA string lengths When varying CAPTCHA lengths are used, attackers must test different string lengths increases the search space of our graph search algorithm, resulting in a longer selection time for the final partition.

The introduction of complex interference arcs and using two-layer structures and dynamic characters are also defense alternatives that are effective in resisting our attack because they increase the difficulty of pre-processing.

7. CONCLUSION

This paper introduces a simple and generic method to attack CAPTCHAs that uses machine learning to solve the segmentation and recognition problems simultaneously. Our attack includes two key components. The pre-processing component uses image processing techniques to remove additions. The partition and recognition component divides challenge images into average blocks along the vertical; then, it uses a modified KNN engine and an efficient graph search algorithm to test different combinations of adjacent blocks to form individual characters and find the most likely combinations as our recognition result.

To speed up the attack process, we also proposed an approach to make our algorithm run faster for a trivial decrease in success rate. Our attack has successfully broken all of the utilized CAPTCHAs, and most attacks showed a high success rate and a high attack speed. In contrast to typical CAPTCHA analysis methods, the attack presented in this paper is simple, low cost and powerful. A full defense against our attack is discussed in the last section of this paper. We expect our work to inspire more novel attacks and defenses as well as innovative designs in this interesting interdisciplinary area.

In summary, this work contributes to a comprehensive understanding of existing text-based CAPTCHAs. What is the next step for text-based CAPTCHAs? Are there CAPTCHA alternatives with better security? These are interesting but challenging problems, and we share them with all research communities.

ACKNOWLEDGMENT

The authors thank the reviewers for their careful reading of this paper and for their helpful and constructive comments. This project is supported by the National Natural Science Foundation of China (61472311).

REFERENCES

1. L. von Ahn, M. Blum, and J. Langford, "Telling humans and computers apart automatically," *Communications of the ACM*, Vol. 47, 2004, pp. 56-60.
2. L. von Ahn, M. Blum, N. J. Hopper, and J. Langford, "Captcha: Using hard AI problems for security," in *Proceedings of International Conference on Theory and Applications of Cryptographic Techniques*, 2003, pp. 294-311.
3. J. Yan and A. S. El Ahmad, "Usability of captchas or usability issues in captcha design," in *Proceedings of the 4th ACM Symposium on Usable Privacy and Security*, 2008, pp. 44-52.
4. 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*, 2008, pp. 543-554.
5. A. S. El Ahmad, J. Yan, and M. Tayara, *The Robustness of Google CAPTCHA's*, Computing Science, Newcastle University, 2011.
6. J. Yan and A. S. El Ahmad, "Breaking visual captchas with naive pattern recognition algorithms," in *Proceedings of the 23rd Annual IEEE Computer Security Ap-*

plications Conference, 2007, pp. 279-291.
7. E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell, "The end is nigh: Generic solving of text-based captchas." in *Proceedings of Usenix Workshop on Offensive Technology*, 2014, pp. 1-15.
8. 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 *Proceedings of Network and Distributed System Security Symposium*, 2016.
9. "Alexa top 500 global sites," Alexa Internet, 2015.
10. K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski, "Building segmentation based human-friendly human interaction proofs (hips)," *Human Interactive Proofs*, Springer, 2005, pp. 1-26.
11. 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*, 2011, pp. 125-138.
12. B. V. Dasarathy, *Nearest Neighbor* (*NN*) *Norms: NN Pattern Classification Techniques*, IEEE Computer Society Press, Los Alamitos, CA, 1991.
13. H. Gao, W. Wang, J. Qi, X. Wang, X. Liu, and J. Yan, "The robustness of hollow captchas," in *Proceedings of ACM SIGSAC Conference on Computer and Communications Security*, 2013, pp. 1075-1086.
14. 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.
15. 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, 1996.
16. C. Cruz-Perez, O. Starostenko, F. Uceda-Ponga, V. Alarcon-Aquino, and L. Reyes-Cabrera, "Breaking recaptchas with unpredictable collapse: heuristic character segmentation and recognition," in *Proceedings of Mexican Conference on Pattern Recognition*, 2012, pp. 155-165.
17. P. Baecher, N. Büscher, M. Fischlin, and B. Milde, "Breaking recaptcha: A holistic approach via shape recognition," in *Future Challenges in Security and Privacy for Academia and Industry*, Springer, Berlin, Heidelberg, Vol. 354, 2011, pp. 56-67.
18. O. Starostenko, C. Cruz-Perez, F. Uceda-Ponga, and V. Alarcon-Aquino, "Breaking text-based captchas with variable word and character orientation," *Pattern Recognition*, Vol. 48, 2015, pp. 1101-1112.
19. C. H. B. L.-P. Karthik and R. A. Recasens, "Breaking microsoft's captcha," Semantic Scholar, 2015.
20. H. Gao, M. Tang, Y. Liu, P. Zhang, and X. Liu, "Research on the security of microsoft's two-layer captcha," *IEEE Transactions on Information Forensics and Security*, Vol. 12, 2017, pp. 1671-1685.
21. T. A. Le, A. G. Baydin, R. Zinkov, and F. Wood, "Using synthetic data to train neural networks is model-based reasoning," in *Proceedings of IEEE International Joint Conference on Neural Networks*, 2017, pp. 3514-3521.
22. M. Tang, H. Gao, Y. Zhang, Y. Liu, P. Zhang, and P. Wang, "Research on deep learning techniques in breaking text-based captchas and designing image-based captcha," *IEEE Transactions on Information Forensics and Security*, Vol. 13, 2018, pp. 2522-2537.

23. D. George, W. Lehrach, K. Kansky, M. La´zaro-Gredilla, C. Laan, B. Marthi, X. Lou, Z. Meng, Y. Liu, H. Wang, *et al.*, "A generative vision model that trains with high data efficiency and breaks text-based captchas," *Science*, Vol. 358, 2017, p. eaag2612.

APPENDICES

Here, we take the Baidu scheme as another example to show our attack details. Fig. 15 shows how the Baidu challenge images were divided into different blocks by AVP and all blocks rank ordered. The sample image is divided into 17 blocks. Table 6 shows the final $n \times n$ table for the Baidu scheme, and Fig. 16 is its search graph. Table 7 shows the graph search process for the Baidu scheme using our graph search algorithm, and "*Xbf8*" is the final recognition result.

Fig. 15. All blocks rank ordered.

Table 6. The final $n \times n$ table for Baidu sample.

	1	2	3	4	5	6	7	8	9	10	11
1	J/0.52	7/0.53	X/0.63	***X/0.65***	X/0.40	L/0.30	A/0.35	A/0.38			
2											
3						J/0.41	R/0.31				
4						J/0.40	Y/0.43	V/0.36	V/0.37	V/0.35	
	7	8	9	10	11	12	13	14	15	16	17
5	K/0.57	b/0.73	***b/0.75***	L/0.38	M/0.49	U/0.48	H/0.49	H/0.49	H/0.49		
6											
7			5/0.37								
8				L/0.42	H/0.51	d/0.39	d/0.38	d/0.38	d/0.38		
9					J/0.43	Y/0.47	Y/0.49	Y/0.49	Y/0.49	L/0.31	H/0.38
10						f/0.71	***f/0.73***	f/0.73	f/0.73	L/0.39	U/0.45
11							t/0.65	t/0.65	t/0.65	L/0.42	U/0.48
12								t/0.44	t/0.44		
13									U/0.43		
14										J/0.49	***8/0.77***
15											8/0.77
16											8/0.77
17											8/0.60

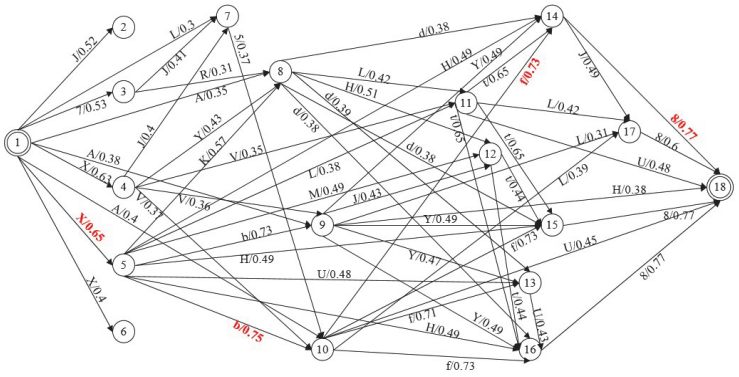


Fig. 16. The equivalent graph for Baidu sample.

Table 7. The search process for Baidu sample.

j	*step*	*value*	*result*	*j*	*step*	*value*	*result*	*j*	*step*	*value*	*result*
2	1	0.52	J	10	3	1.41	XJ5	15	4	2.30	XKLt
3	1	0.53	7	11	2	1.03	XL	16	2	1.14	XH
4	1	0.63	X	11	3	1.64	XKL	16	3	2.13	Xbf
5	1	0.65	X	12	2	1.14	XM	16	4	2.55	XbfU
6	1	0.40	X	12	3	1.82	XbJ	17	2	0.79	AL
7	1	0.30	L	13	2	1.14	XU	17	3	1.79	XbL
7	2	1.04	XJ	13	3	2.12	Xbf	17	4	2.62	XbfJ
8	1	0.35	A	13	4	2.13	XJ5f	18	2	0.86	AU
8	2	1.22	KX	14	2	1.14	XH	18	3	1.91	XH8
9	1	0.38	A	14	3	2.13	Xbf	*18*	*4*	*2.90*	*Xbf8*
9	2	1.38	Xb	14	4	2.30	XKLt				
10	1	0.40	A	15	2	1.14	XH				
10	2	1.40	Xb	15	3	2.13	Xbf				

Xiyang Liu is a Professor in Xidian University and a member of the ACM. 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.

Yang Zhang is a master degree candidate in Computer Science at Xidian University. Her current research interest is CAPTCHA.

Jing Hu is a master degree candidate in Computer Science at Xidian University. Her current research interest is CAPTCHA.

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

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.