Breaking CAPTCHA Security

Puneet Kaur, Vikash Jha , Amandeep Dhir
Helsinki University of Technology
Telecommunications Software and Multimedia Laboratory
`http://www.tml.hut.fi/Opinnot/T-110.5230/`

Abstract

The paper presents a naivete way of attacking CAPTCHAs at the same time we try to analyze our attack against variation of various parameters during CAPTCHA creation . The presented method targets the entire CAPTCHA string using the naive approach of pre-processing the CAPTCHA image files in order to reduce its dimensionality and normalizing its color and pixel distribution. We are targeting the entire CAPTCHA image by considering it as a stream of information. The two similar CAPTCHA image is two stream of information with the noise as the differentiator.If we remove the noise or normalize it then we will be able to match two similar CAPTCHA image from the same system with ease .The paper discusses this method in detail by covering it from various aspects such as: preconditions of the attack, methods used for attack, algorithm used, analysis and protection against the attack.We have also analyzed the CAPTCHA images which employ the variation of parameters against our attack to judge the effectiveness of CAPTCHA images against the attacks that is based on reverse engineering.
KEYWORDS:CAPTCHA,k-NN,Noise Removal

1 Introduction

Internet has provided us various set of services that play significant role in our daily life. In such a scenario, there is a pronounced need to protect these services from abuse like spamming, Denial of Service (DoS), etc, that could be caused by variety of sources like automated programs called bots [13].

The ability of machines to be programmed to perform given tasks at much faster rate as compared to the humans is being exploited by the attackers to launch various attacks. One of the ways to ensure secure access to Internet services from remote servers is to use the method known as CAPTCHA (Completely Automated Public Turing Test to Tell Computers and Humans Apart)[12]. CAPTCHA is a program that gives a challenge that most humans can pass, but current computer programs cannot pass [9]. This particular inability of the computers forms the basis for the CAPTCHA security thus, preventing attacks that are caused by using the processing power of the machine. CAPTCHA involves a challenge response mechanism where a challenge in the form of sequence of visible characters (generally in an image format) is presented to the users. The user is required to recognize and enter those character or images in the same sequence and format as given in the challenge. After all the above said process; which mandates human intervention; is completed, the server does the final check. The server checks all the characters that have been entered by the user against the characters that were presented by the server; if all the characters match then the desired service is provided to the user else it is denied. This whole concept, used for differentiating between humans and computers, is used for preventing various attacks such as: DoS, spamming, unauthorized usage of resources by the computers etc.There are different methods of implementing a CAPTCHA e.g. presenting a picture of an animal, images accompanied by the sound to help in identifying the image or a well known object.In this paper, we are presenting our methodology that focuses on character based image CAPTCHAs which are quite popular.

Various efforts have been made by the attackers to break the CAPTCHA security as well. The current literature and methods for breaking it mostly utilize artificial intelligence [19] and various pattern recognition techniques [20]. Majority of the efforts aimed at breaking CAPTCHA security have been carried out using segmentation and visual techniques by recognizing the individual characters.

The paper proposes a novel way of attacking CAPTCHA that has been devised after thoroughly evaluating their vulnerabilities. The proposed method checks the sample CAPTCHA image against all the subset of characters of length “n” from the character set (comprises of the domain of the CAPTCHAs) after pre-processing the CAPTCHA image files to reduce their dimensionality and normalize color and pixel distribution. This has been achieved by utilizing the basic library functions of Python language. The basic aim is to show that CAPTCHAs can be broken using simple algorithmic techniques and knowledge of how they are created. Various tools like Auto Fill, Form Filter, etc, for filling the web forms automatically while accessing a service can be combined with our method for launching more effective attacks against services using CAPTCHAs.We also analyze our method against the CAPTCHAs which employ significant randomness in pixel orientation as we will see the results in later section.

The rest of the paper is organized as follows,section 2 presents a brief overview of methods used in breaking CAPTCHA.We will present our attack method in section 3 will present some protection mechanism against the attack. In section 4, we will try to analyze our attack method against the system which employ larger degree of randomness in various parameters of CAPTCHA creation and will derive protection mechanism from this analysis. Lastly, we will see test results and efficiency of the attack and the cost incurred in scenarios where our attack can be deployed .

2 Related Work

CAPTCHAs has been attacked earlier with various techniques such as image orientation [13], artificial intelligence(AI) [19, 11] and segmentation techniques. In the case of segmentation attack, CAPTCHA breaking is reduced to the problem of individual character recognition.There has been a detailed description of character segmentation attacks against Microsoft and Yahoo! CAPTCHAs [18, 21]. But of late CAPTCHA design has evolved to have the resistance against the segmentation attack [5]. In contrast to segmentation attack our method is designed to target the entire CAPTCHA string so segmentation resistant techniques may not be a protection measure against our attack. Sometimes segmentation attacks also requires the use of external hardware such as OCRs etc which may not be a viable option everywhere. CAPTCHAs are also attacked using AI techniques where the CAPTCHA string is guessed by measuring the color intensity of the image. The image is divided into horizontal strips and vertical strips to judge the intensity of each square formed as result of intersection of horizontal and vertical strips.We could not find any literature where CAPTCHAs have been attacked with the angle of reverse engineering. In this paper,we are exploring the scenario in which we will attack CAPTCHAs, treating them as a stream of information and we will compare two similar CAPTCHA image by removing the noise from these information streams . Due to cheap workforce in some part of the world, the manual breaking of CAPTCHAs using relay traffic has also become an easy and viable method to break CAPTCHA security [7]. As we know that cost in terms of resource and money also governs the viability of the attack so, we will present a view on the cost aspect of our attack with a comparison to the manual breaking of CAPTCHAs in later section of this paper.

3 Attack Method

This section presents various details concerning attacks. We have discussed various preconditions of the attack along with the methodology of the attack that is primarily based on reverse engineering of creation of CAPTCHA image.

Precondition

As the attack is mainly based on the algorithmic techniques, so no specific hardware like OCR(Optical character recognition) or external reader is required. The basic precondition for the attack is to collect several (at least 60 to 70) CAPTCHAs from the site that is intended to be attacked. The collected CAPTCHA images can help to analyze the image structure, distribution of pixels, fonts of the characters and differences between the color intensity of the background color and the text color.

As the character set for creating CAPTCHA images is quite large so huge disk space is required for storing the CAPTCHA files that consists of all the possible string combinations of required length. For example in our case, we have considered a set of 28 characters and a string length of 6. Thus, the total numbers of files that need to be stored are 28^6, which will require a space of 1370 GB(3Kb per CAPTCHA image). But the reduction of disk space can be achieved by: reducing the dimensionality of the file, removing the redundant parts and only storing the relevant parts of the image. In our experiments, we were able to achieve the space reduction of about 73% (950 bytes from 3431 bytes),so we required 375 GB only . It could be still compressed further to lesser bytes i.e. approximately 400 to 500 bytes but, that will require more processing and time to further study the image processing and storage techniques. The proposed attack method uses 700GB of space to successfully attack a CAPTCHA of length 6 made from character set having 28 characters. The current implementation takes the liberty of space as hard disks are getting cheaper day by day [10, 2] so space is not a major constraint now a days. This attack can be carried out under normal conditions with the use of a regular desktop or laptop and probably a 500GB of external memory disk, considering the frequent use and low cost of such hard disks, it should not be considered as the special hardware.

Method of Attack

This sub-section discusses about the method of attack by presenting the generic algorithm working behind the attack. This has been discussed using an example CAPTCHA file that is sufficiently complex and covers all the aspects of CAPTCHA creation [3].The attack consists of two stages:

(i) **offline processing stage**

(ii) **online processing stage**

The offline processing stage consists of all the steps that need to be followed in order to make preparations for the attack. During this stage all the images generated from the string of length "n" (six in our case) are processed and stored in a repository. This stage can be explained using the sample CAPTCHA image, shown in Figure-1.

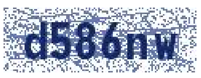


Figure 1: Sample CAPTCHA Image

This image file is normally 120x40 pixel file but the real information is confined between the pixels (14,6) and (106,32). It can also be noticed that the current image has a RGB mode of representation for each pixel, which can be converted into a single representation for each pixel. This is achieved by using default converter of the Python Image Library (PIL) [4]. This converter performs the pixel transformation according to ITU-R 601-2 luma transform expressed by the formula:

L = R*299/1000+G*587/1000+B*114/1000

After the conversion of RGB pixel values to a single pixel value, reduce the dimensionality of image by half i.e. 60x20 and remove the portions that does not contain any useful information. After all these steps the resulting image looks like the image shown below (Figure-2)

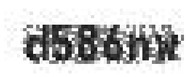
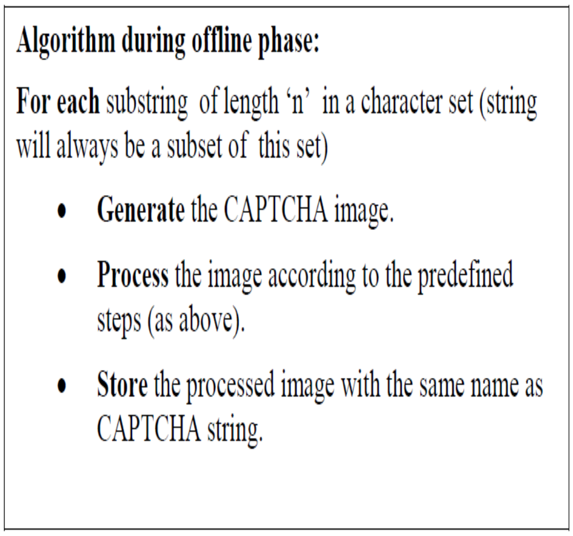


Figure 2: CAPTCHA Image After Processing Figure 1

The steps followed in the offline processing stage can be summarized as follows:

1. Convert the RGB pixel value for each pixel to a single pixel value.
2. Remove (whiten/blacken, depends on the image) the portion which does not contain any useful information.
3. Reduce the dimensionality by certain amount (half or three times depending on the image size).

It should be noted that steps 2 and 3 are system dependent as the decision regarding which parts of the image should be removed and by how much is completely dependent on the CAPTCHA image files produced by the system. The algorithm working behind the offline processing stage is presented below (Table-1).



Algorithm during offline phase:

For each substring of length 'n' in a character set (string will always be a subset of this set)

- **Generate** the CAPTCHA image.
- **Process** the image according to the predefined steps (as above).
- **Store** the processed image with the same name as CAPTCHA string.

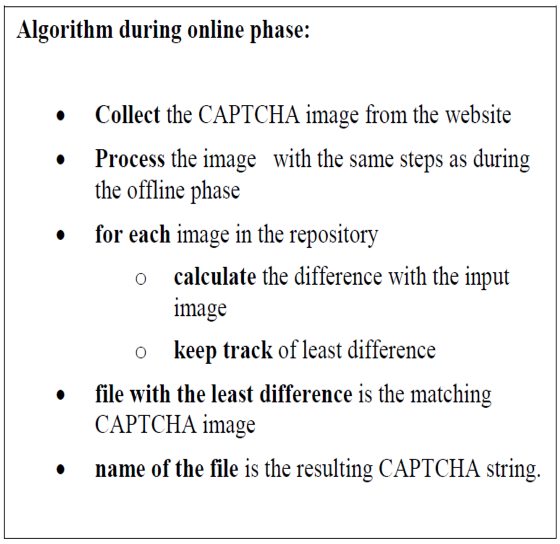
Table 1 Algorithm during offline processing stage

In case of our attack, the image is generated for the all the subset strings having length of 6 (can vary from system to system but, is generally constant for the system) from a given character set and each image is processed using the steps as mentioned above. The CAPTCHA image is stored with the same name as the string itself as this will help the string to be easily located during the online processing phase.

During the online processing stage, the CAPTCHA challenge is presented to the machine. The machine first collects the CAPTCHA image (a python script has been written for collecting the image) and then subjects it to the same steps that were followed during the offline processing phase. The machine then compares the resulting image with the similarly processed images (during the offline stage) placed in the repository. The file comparison that gives the least pixel difference can be considered to be similar to the image presented to the machine. The required CAPTCHA string can be obtained using the name of the file from the repository.

The task of finding the least difference files can be done using two different ways. The first way is to perform pixel by pixel comparison, considering two pixels(at same (x,y) in two images to be compared) to be different only when they differ by a certain amount (150 in our case, i.e normalizing the noise which is explained later). At any point of time during the execution, the least minimum difference and the name of the encountered image file for least difference till that time can be stored. Once all the files in the repository have been compared then the resultant CAPTCHA image (having least difference) is given by the algorithm.

The second way is to flatten the image for the entire pixel set (60x20 in our case) into a vector having the values of 0 or 1. The value of 1 (contributing pixel) can be assigned for a pixel if it is less than a certain value (200 in our case) else value of 0 (non-contributing pixel) is assigned(again ,normalizing the noise). Then, we can use k-NN (k nearest neighbor) algorithm [15, 17]for finding the files of least difference. The advantage in this case is that vector can stored in ascending order and binary search [8] can be performed to find out a certain key. We will see the efficiency analysis of both the cases in later section. The difference in both the cases comes out to be less than 15 pixels for the same CAPTCHA string. The algorithm followed during the online processing stage has been shown below (Table-2).



Algorithm during online phase:

- **Collect** the CAPTCHA image from the website
- **Process** the image with the same steps as during the offline phase
- **for each** image in the repository
  - o **calculate** the difference with the input image
  - o **keep track** of least difference
- **file with the least difference** is the matching CAPTCHA image
- **name of the file** is the resulting CAPTCHA string.

Table 2 Algorithm during online processing stage

Analysis of Attack

This section elaborates more about the method of attack from the technical perspective to understand the underlying principle behind the working of the attack. The discussion will be done using the same sample CAPTCHA image file that was used in the previous section. We will first understand how a character based CAPTCHA image is generated and how two same CAPTCHA image having same string is made different.
For analyzing the proposed method, let us start by the very first step i.e. examining how the CAPTCHA image was generated. This will help us to understand what processing needs to be done on the image file and how offline and online phase of the algorithm help to match the two similar CAPTCHA images . To generate a CAPTCHA image as shown in Figure-3, the first step is to generate the string in a textbox of size 120x40

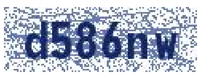


Figure 3: Sample CAPTCHA Image

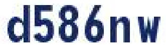


Figure 4: Step 1 of CAPTCHA Image Generation

After the creation of the textbox of desired size, randomness is added to the image. This can be done by adding some noise in the background, which is generally an iterative process of drawing a filled circle and lines from randomly selected points across the image (that appear as random dots in the background). The points are selected by a random function based on seed (commonly local time). These randomly selected points itself acts as the differentiator between the images of same string. So, adding noise to the above plain image (Figure-4) the will generate the CAPTCHA as in Figure-5. When this Figure-5 is processed using Table-1, we get the processed version of Figure-5, shown as Figure-6.

If we suppose that the remote server presents string “d586nw” as a CAPTCHA string (CAPTCHA code) to the machine(Figure-7). The presented CAPTCHA image will not be exactly same as it was generated above Figure-5 because of the differences in noise (random background dots) of both the image.To find whether two CAPTCHA images are same or not, the most natural way will be to convert Figure-5 and Figure-7 to Figure-4 and then make comparison

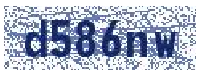


Figure 5: Initial Version

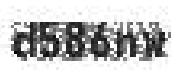


Figure 6: Processed Version of fig.5

between the both. This can be done by eliminating the noise, but eliminating the noise is not an easy task because the noise mixes with the text color that leads to distributed value of pixels across the image. It should also be noted that previously allocated values of pixels for text also change because the pixel values of the noise intermixes with the text itself. So, these pixel values should be converted to a single pixel value based on some transformation (we have used ITU-R 601-2 luma transform). Then we should make observations for the intensity of the text and the noise pixels.We will observe that intensity of text pixel is higher than the noise, this will help us to filter the noise based on pixel value.The pixel value greater than a threshold value(differs from system to system in our case it was 200, have to find out by carefully observing the gray scale image from the system) is noise. After figuring out the threshold pixel value for the noise, the redundant parts of the information are removed. The resultant image looks like the image in Figure-8 and Figure-6, which differ from one another only with respect to the dots in the background. If we eliminate these noises(normalize the pixels greater than threshold pixel value) or keep a track on the limit of difference due to these noises while comparing, then Figure-8 and Figure-7 can result to be same. We proceeded with this reasoning and found that end result validates our logic. The comparison of two CAPTCHA image for same string, even if they are not same initially, can be made equalized by processing them in manner that normalizes their variations.These reasoning sufficiently explains the purpose of processing steps during offline and online phase of attack, these processing steps brings the two CAPTCHA images to the same level so that these can be compared.

The main reason this logic works is the fact that the space for adding randomness is very less i.e. (equivalent to image size i.e. 120x40 in our case). The possibility of adding randomness further reduces if we consider just the actual information part i.e. (94x32). This randomness can be easily removed if all the values below a certain limit are mapped to be same. This will result into Figure-8 being more or less like Figure-6. This was validated with nearly 200 to 400 CAPTCHA image files as samples and every time the

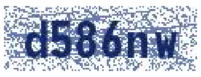


Figure 7: Image Presented from Remote Server

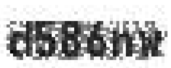
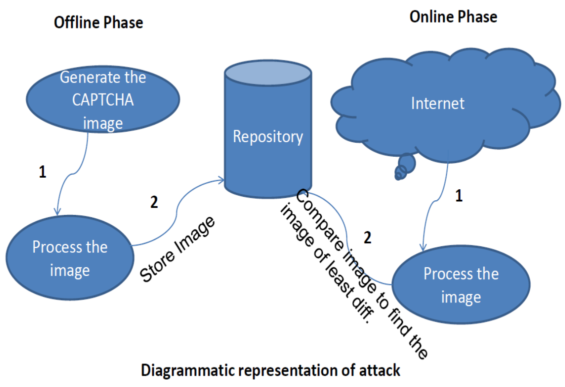


Figure 8: Processed Version of fig.7

least difference came out to be less than 15 for two same CAPTCHA string.


Offline Phase
Online Phase
Generate the CAPTCHA image
1
Process the image
2
Store Image
Repository
Internet
1
Process the image
2
Compare image to find the image of least diff.
Diagrammatic representation of attack

4 Effectiveness of the attack and protection

In this section we will first state the results of our algorithm in scenarios where the parameters such as rotation of pixels, starting position of the CAPTCHA string, height and width of individual character etc, this will help us better to visualize the protection mechanism clearly. We have distorted the above CAPTCHA image by variation of various parameters(by rotating the individual pixels and randomizing the starting pixels) as is shown in figure below(Figure 9).

When we applied our algorithm to the system implementing such variation then we found that our attack may not be effective against such protection mechanism.With a very wide variation in the parameters our attack was successful at about 50% of trials which cannot be considered as effective in real world scenario. We even tried to find the nearest neighbours of the target CAPTCHA images, we were successful in getting the target CAPTCHA from the search space of 10^6 when we increased the nearest neighbour space to 10 15 which may not be always feasible to analyze .We tried with complete reversal of CAPTCHA creation(undo the rotation of pixels , guess the range of location of maximum number of pixels etc) then our algorithm was effective and yielded result every time. But we think it is very difficult to reverse engineer every CAPTCHA creation system unless we do not posses the algorithm for the CAPTCHA creation.

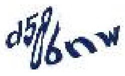


Figure 9: Distorted image(bilinear transformation of the image)

As the attack is mainly based on exploitation of certain fixed characteristics of the CAPTCHA so, the protection mechanism also emanates directly from the idea of variation of every parameter of CAPTCHA.We analyzed our attack for the reason why it worked for the CAPTCHAs where the parameters are relatively static. There are certain constraints in such CAPTCHAs for example, the dimensionality of the CAPTCHA cannot be increased to a large extent so while adding noise in the background greater randomness can not be achieved. Some of the mechanisms to protect against our attack, which we analyzed in detail to check the effectiveness of the attack in case where the CAPTCHA creation is randomized, are listed below.

1. The main reason behind the success of our attack is the fact that all the characters of the CAPTCHA string are of same font, for the example presented in this paper all characters are in monofont [3] . This phenomenon is true for most of websites that use the CAPTCHA security mechanism i.e they have same font for all the characters of CAPTCHA string. If the font of the characters is varied(chosen randomly), then this attack can be made difficult. For example: in a string “abc345” , “a” can be presented in impact, “b” in times similarly “5” in Georgia, etc. These variation can be easily achieved using pixel rotation and randomizing the starting position.

2. Increasing the range of strings can also make attack more difficult. It can be easily done by modifying the program to produce CAPTCHA string of all the lengths less than equal to “n”. Most of the websites present string of fixed length, but this can be changed to produce the strings of all lengths from a minimum to maximum length. For example: if in our case fixing the maximum length to be 6 then currently we just produce 28^6, but covering all the length till 6 from a minimum of 4 will give us an increase of $28^4 + 28^5$ which will make the attack really difficult.

5 Test Results

This section will show the test result of our experiments that covers how many pixels a sample image differed from the image stored in the repository. We are mainly covering the scenario where our attack is successful with high probability. It will also look into the efficiency of the algorithm with increase in the size of the repository.
To start with, let us first consider a repository size of 10,000,00 files. The CAPTCHA string "myhm49" was searched which gets processed to Figure-11 (after removing noise and reduction of size)
Sample Input file:

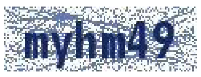


Figure 10: Input Image

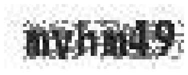


Figure 11: Processed Version of Above Image.

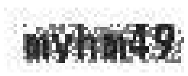


Figure 12: Repository Version of Figure 10.

The amount of time taken to search is 2 min 40 sec and the difference of pixel is 9. Similarly, for a repository size of 10,000,000 files, time taken to search "8tt9qt" (Figure-13) is 17 min 39 sec.
The amount of time take to search is 17 min 39 sec and the difference of pixel is 11.
Now, let us see the boundary cases, in which only one of the character differs between the sample input and the file to which it is compared and see how our logic behaves. Let us compare "8tt9at"(Figure-14) to "8tt9qt"(Figure-15) and measure the pixel difference.

The pixel difference between the files came out to be 16, 15 and 20 (we ran the test three times with three different set of files) but every time it was less than the image file

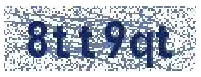


Figure 13: Input Image

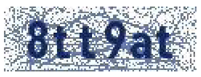


Figure 14: Input Image

having string "8tt9qt" 9,10 and 6 respectively. The same test holds good for entire range of chosen characters. Every time the image having the same string turns out to be the winner (having least pixel difference).
Similarly, let us go through one more example, in which the one of the character of input file gets changed after processing. CAPTCHA string is "9yg6cv"(Figure-16) which after processing becomes very similar to "9vg6cv". But, then also the pixel difference of "9yg6cv" with "9vg6cv" is more than the pixel difference with the image of the string "9yg6cv".
Although,both the images are quite similar but, still the image with the correct string "9yg6cv" itself is the winner.
Evaluation of other CAPTCHAs from various websites was also done. All of them can be attacked using our algorithm of course, but the processing steps needs to be readjusted for its parameter for each system. As for example the CAPTCHA image (taken from htttp://www.blogger.com) (Figure-17) shown below can be converted easily to a processed version of same color and hence can be traced back to original string.
At the same time we have some really weak CAPTCHAs that are prevalent in many websites and should be done away with, as for example the below CAPTCHA (taken from https://loanservice.uasecho.com/home.aspx)(Figure-18)
Breaking this type of CAPTCHA security can be easily achieved with our approach because it has very less range of strings 9^6 and it is not at all difficult to pre-process it (as it does not even mix the colors and has same font throughout).

Cost of the attack

As a part of the precondition we have said that our method uses considerable amount of disk space for storing the repository of the processed CAPTCHA image. But if we analyze the present CAPTCHA string that is presented by most of the websites, we can use some of the information from the structure of the CAPTCHA string to reduce the space. For example, the maximum character set that is available for making the CAPTCHA string is 51.There can be 51^6 CAPTCHA string possible from character set , requiring huge amount

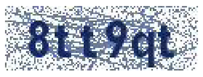


Figure 15: Input Image

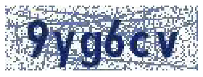


Figure 16: Input Image

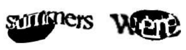


Figure 17: Input Image

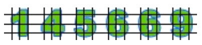


Figure 18: Input Image

of space. But if we observe carefully then we may find that uppercase letters are present only at the beginning in CAPTCHA string presented by most of the systems. This will result in reduction of total CAPTCHA string to 21*28^5 and thus the space will reduce by a huge amount. The point here that we want to emphasize is that disk space or any other resource utilization can be reduced or controlled by carefully analyzing the information collected from the system during the precondition phase.

Every CAPTCHA system is susceptible to manual breaking of the CAPTCHA.Due to cheap availability of the labor in some countries manual attack has become quite viable. We will present the cost analysis of our attack as against manual breaking of CAPTCHAs highlighting the important observations.If we take the $2 as the standard charge for breaking 1000 CAPTCHAs [7]then for braking 50000 CAPTCHAs, $100 needs to be paid. This much of the cost is sufficient to break a CAPTCHA space size of 35^6 with our method and we can reuse the same resource to attack next system.But for the manual method if we intend to go for the other system then we need to pay another $100 which adds unto the cost.We should also look into the fact that doing a monotonous job results into degrading of productivity in case of manual method which is quite unlikely in case of machines [6].

If we extend our concept a bit further and run our algorithm from a distributed set of machines for targeting a system [14]. This will result in the reduction of problem space and will also bring a substantial set of resource to attack the system and thus making the entire attack very cheap. For example if we consider CAPTCHA space size of 21*28^5 and 20 machines(of 320 GB of each which is quite normal these days) to attack,then we do not need any additional resource to launch the attack.This will also result in reduction of problem space for each machine and thus making the search of CAPTCHA string faster.We should also be aware of the fact that for manual attack also we need a set of distributed machines for allowing the set of user to break CAPTCHAs with considerable speed.

Efficiency

In case of brute force method the efficiency of the proposed method of attack decreases as the size of the repository increases, but then also the method is sufficiently fast as we can see from the graph (Figure-19) below. When we consider searching in an ordered list of vectors then the search space can be traversed efficiently with a complexity of O(log$_2$n) using binary search as is evident(Figure-20)

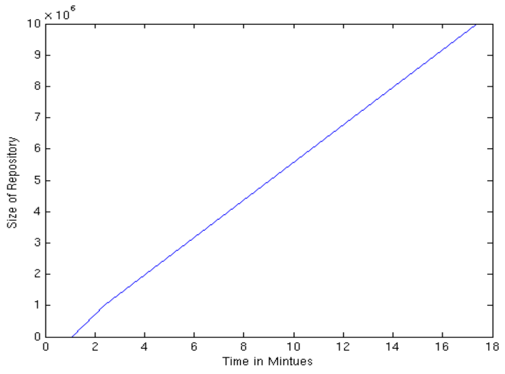


Figure 19: Repository Size Vs Time(Brute Force)

6 Conclusions

We have approached the problem of breaking the CAPTCHA with a completely new perspective and have been largely successful with all the systems where the CAPTCHA creation is based on relatively static parameters[22][23]. The approach does consume a lot of space but it can be made space efficient with a little more study as has been explained in the above section. Moreover, ever declining price of harddisk makes the question of more space really redundant.The efficient process of analyzing the CAPTCHA images from a system can really make the attack very easy. The fact that this attack emanates from reverse engineering the pro-

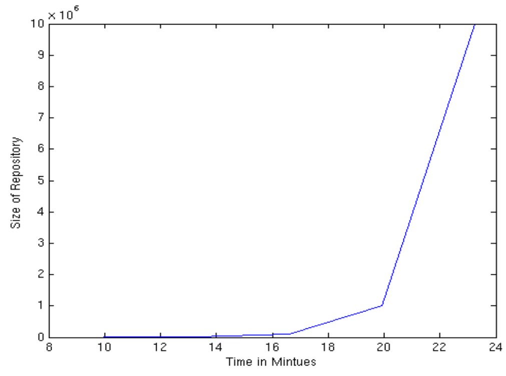


Figure 20: Repository Size Vs Time(Binary search)

cess of creation of CAPTCHA this may have higher success rate for an arbitrary system of considerable complexity. This approach enables breaking of a system based on CAPTCHA security without any external specialized hardware and it can be done with a machine of normal capability.

Acknowledgements

We are thankful to Billy Brumley (Researcher Aalto-School of Science and Technology) for suggestion of the attack methods and improvement in attack. We are also thankful to the course personnel and our student colleagues for useful review comments.

References

[1] Adavanced captchas :http://recaptcha.net.

[2] Hard disk trends http://www.mattscomputertrends.com /harddrives.html.

[3] PHP CAPTCHA Security :http://www.white-hat-web-design.co.uk /articles/php-captcha.php.

[4] Python image library :http://www.pythonware.com/products/pil/

[5] S. A. and S. S. Advanced collage acm international conference on multimedia. *Fifth International Conference on Information*, 2008.

[6] E. Amoroso. *Fundamentals of Computer Security Technology*. Prentice Hall, USA, 1994.

[7] V. Bajaj. Spammers pay others to answer security tests :http://www.nytimes.com/2010/04/26/technology/ 26captcha.html?src=me&ref=technology, 2010.

[8] J. L. Bentely. Multidimaensinal binary search tree used for associative searching. *Jounral of ACM*, 1975.

[9] D. E. Denning. *Information Warfare and Security*. ACM Press Books, USA, 1999.

[10] G. Edward and R. F. Hoyt. Future trends in hard disk drives. *IEEE Transactions on Magnetics*, 32(3), May 1996.

[11] P. Golle. Machine learning attacks against the asirra captcha,. In *15th ACM conference on Computer and communications security*, 2008.

[12] C. Kaur. Is it human or computer? defending e-commerce with captcha. *IEEE IT professional*, 45:219–223, 1 2005.

[13] Rich Gossweiler , Maryam Kamvar and Shumeet Baluja. *What's Up CAPTCHA? A CAPTCHA Based On Image Orientation*, USA, 2009. Google.

[14] A. Ogijenko. Captchas: Humans vs. bots. *IEEE Transactions on Security and Privacy*, 2008.

[15] M. P. Ciaccia. M-tree an efficient access method for similarity search in metric spaces. In *VLDB International Conference*, USA, Sept 1997. ACM.

[16] Vikash Jha, Puneet Kaur, Amandeep Dhir. Vulnerability report on breaking captcha security. Technical report, Aalto School of sceience and Technology, Finland,2010.

[17] D. S. Arya. An algorithm for ap- proximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6):891–923, 1998.

[18] Y. S.Y Huang. Segmentation algorithm for breaking msn and yahoo captchas. *IEEE Transactions on Information Theory*, 2008.

[19] L. von Ahn, M. Blum, N. J. Hopper, and J. Langford. Captcha:using hard ai problems for security, 2008.

[20] J. Yan and A. S. E. Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *23rd Annual Computer Security Applications Conference*, pages 279–291, 2007.

[21] J. Yan and A. S. E. Ahmad. Low-cost attack on a microsoft captcha, 2008.

[22] https://loanservice.uasecho.com/home.aspx.

[23] htttp://www.blogger.com.

Appendix

This section shows the processed and the repository images of the input images presented in the Test Results section.

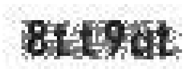


Figure 21: Processed Image of Figure-13

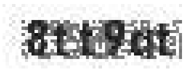


Figure 22: Repository Image of the Figure-13

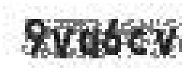


Figure 23: Processed Image of Figure 16