ACM Computer and Communication security 2011 (CSS'2011)

Text-based CAPTCHA Strengths and Weaknesses

Elie Bursztein, Stanford University *elie@cs.stanford.edu*
Matthieu Martin, Stanford University *mamartin@stanford.edu*
John C. Mitchell *jcm@cs.stanford.edu*

The slides and paper are available from free from http://ly.tl/p22

Follow Elie onTwitter : https://twitter.com/elie and Google+: http://ly.tl/g

Text-based CAPTCHA Strengths and Weaknesses

Elie Bursztein, Matthieu Martin, and John C. Mitchell
Stanford University
elie@cs.stanford.edu, mamartin@stanford.edu, mitchell@cs.stanford.edu

ABSTRACT

We carry out a systematic study of existing visual CAPTCHAs based on distorted characters that are augmented with anti-segmentation techniques. Applying a systematic evaluation methodology to 15 current CAPTCHA schemes from popular web sites , we find that 13 are vulnerable to automated attacks. Based on this evaluation, we identify a series of recommendations for CAPTCHA designers and attackers, and possible future directions for producing more reliable human/computer distinguishers.

Categories and Subject Descriptors

K.6.5 [**Computing Milieux**]: Management of Computing and Information Systems—*Security and Protection*

General Terms

Security, Theory

Keywords

CAPTCHA, reverse Turing test, machine learning, vision algorithm, SVM, KNN classifier.

1. INTRODUCTION

Many websites use CAPTCHAs [25], or *Completely Automated Public Turing tests to tell Computers and Humans Apart,* in an attempt to block automated interactions with their sites. These efforts may be crucial to the success of these sites in various ways. For example, Gmail improves its service by blocking access to automated spammers, eBay improves its marketplace by blocking bots from flooding the site with scams, and Facebook limits creation of fraudulent profiles used to spam honest users or cheat at games. The most widely used CAPTCHA [1] schemes use combinations of distorted characters and obfuscation techniques that humans can recognize but that may be difficult for automated scripts.

[1]For readability purpose, we will write the acronym in lowercase.

CCS'11, October 17–21, 2011, Chicago, Illinois, USA.

Figure 1: Wikipedia captcha example

Captchas are sometimes called "*reverse Turing tests*": because they are intended to allow a computer to determine if a remote client is human or not. In spite of their importance, their extremely widespread use, and a growing number of research studies [7, 8, 31] there is currently no systematic methodology for designing or evaluating captchas. In fact, as we substantiate by thorough study, many popular websites still rely on schemes that are vulnerable to automated attacks. For example, our automated **Decaptcha** tool breaks the Wikipedia scheme, illustrated in figure 1, approximately *25%* of the time. *13* out of *15* of the most widely used current schemes are similarly vulnerable to automated attack by our tool. Therefore, there is a clear need for a comprehensive set of design and testing principles that will lead to more robust captchas.

While fine previous work [7] suggests that captcha security depends on preventing segmentation, we find in our study that relying on segmentation alone does not provide reliable defense against automated attacks. For example, it is possible to exploit the fact that the captcha length is fixed to make an educated guess where to segment the captcha, even if its anti-segmentation technique can't be broken directly. We found that this type of attacks apply to numerous captcha schemes, including eBay and Baidu.

Reflecting on techniques described in the literature [10, 32], available machine-learning techniques [11, 20, 26] available vision algorithms [1, 13, 14], and our own experience with captcha analysis [2, 4, 5], we divide the automated captcha-solving process into five generic steps: *pre-processing, segmentation, post-segmentation, recognition, and post-processing*. While segmentation, the separation of a sequence of characters into individual characters, and recognition, identifying those characters, are intuitive and generally understood, there are good reasons for considering the additional pre-processing and post-processing steps as part of a standard process. For example, preprocessing can remove background patterns or eliminate other additions to the image that could interfere with segmentation, while post-segmentation steps can "clean up" the segmentation output by normalizing the size of each image or otherwise performing steps distinct from segmentation.

After recognition, post-processing can improve accuracy by, for example, applying spell checking to any captcha that is based on actual words (such as Slashdot). Based on this generic captcha-solving architecture, we experimented with various specific algorithms and tried them on various popular website captchas. From these corpus, we identified a set of techniques that make captchas more difficult to solve automatically. By varying these techniques, we created a larger synthetic corpus that allowed us to study the effect of each of these features in detail and refine our automated attack methods. Based on our previous study of how solvable captchas are for humans [3,5], we focused our attention on a range of techniques that are within the grasp of human solvers, although we did consider possible captchas that could be uncomfortably difficult for some humans.

We tested the efficiency of our tool Decaptcha against real captchas from *Authorize, Baidu, Blizzard, Captcha.net, CNN, Digg, eBay, Google, Megaupload, NIH, Recaptcha, Reddit, Skyrock, Slashdot, and Wikipedia*. As far as we know none of these captcha schemes had been reported broken prior to this work. Of these 15 captchas, we had *1%-10%* success rate on two (*Baidu, Skyrock*), *10-24%* on two (*CNN, Digg*), *25-49%* on four (*eBay, Reddit, Slashdot, Wikipedia*), and *50%* or greater on five (*Authorize, Blizzard, Captcha.net, Megaupload, NIH*). To achieve such a high success rate we developed the first successful attacks against captcha schemes that use collapsed characters (*eBay, Megaupload, and Baidu*). Only Google and Recaptcha resisted to our attack attempts, and we reached some informative understanding of why we couldn't break them. Because of Decaptcha genericity we were able to break 7 of these 15 schemes (*Authorize, Baidu, CNN, Megaupload, NIH, Reddit, Wikipedia*) without writing a new algorithm.

Based on our evaluation of real-world and synthetic captchas, we extracted several guidelines and suggestions that we believe will be useful to captcha designers and attackers. For example, randomizing the captcha length and individual relative character size, while relatively painless for humans, are important steps for resisting automated attacks. Similarly, if all characters are the same size, partial segmentation then gives a good estimate of the number of characters, again aiding segmentation. Conversely, creating a wave shape and collapsing or overlayed lines can be effective, relatively speaking. We also find that complex character sets, which can be confusing for humans, are not particularly effective, and we comment on the relative importance of anti-recognition techniques, implementation errors, preparing alternative "backup" schemes in case vulnerabilities are discovered. The main contributions of this work include:

- A generic evaluation tool, **Decaptcha**, designed to evaluate quickly captcha security.
- A state-of-the-art evaluation of anti-recognition techniques and anti-segmentation techniques, and captchas used by the popular websites.
- Successful attacks by a single tool against 13 out of 15 real captcha schemes from popular websites and the first successful attacks on captchas that use collapsed characters (e.g eBay and Baidu).
- A publicly available synthetic corpus designed to replicate security features of real-world captchas, in ranges potentially acceptable to humans, so that designers may test new attack algorithms on them.
- A defense taxonomy and an evaluation of the impact of anti-recognition techniques on the learnability of captchas by automated tools.

2. BACKGROUND

Measuring attack effectiveness. A first step to evaluate attack effectiveness is to measure its **accuracy**, the fraction of captchas that were answered correctly by the captcha solver. However, a particular attacker may choose to respond to some captchas and not others, depending on the confidence in their guess, as web services usually limit the number of attempts per IP [2]. Therefore, a more precise way to evaluate attack effectiveness is through coverage and precision metrics.

Coverage is the fraction of captchas that the solver attempts to answer. **Precision** is the fraction of captchas answered correctly [2]. The captcha design goal is that "*automatic scripts should not be more successful than 1 in 10,000*" attempts (i.e. a precision of 0.01%) [18]. However, we believe that this security goal is too ambitious, random guesses can be sucessful, so we *deem a captcha scheme broken when the attacker is able to reach a precision of at least 1%*.

Another important consideration is how to choose the test set on which the solver is evaluated. We argue that cross-validation is useful for initial experimentation but is not sufficient to deem a captcha scheme insecure as it does not reflect real-world conditions where the solver attacking a website is presented with previously unknown captchas. Instead we adopt the machine learning community's best practices. We use a test set that is entirely different from the training set to evaluate the solver's effectiveness. We must avoid skewing the precision evaluation due to a single easy captcha in the test set. This is especially important when the solver's precision is close to 1% mark. Therefore, we advocate to *use a large test set, of at least 1,000 captchas*. Now, the solver must solve at least 10 unseen captchas before reaching the 1% precision mark required to deem a scheme insecure. Every evaluation performed in this work follows these best practices.

Attacking captchas. Prior to this work, state of the art automated solvers used a three-stage approach consisting of **preprocessing**, **segmentation** and **classification** stages [9]. Previous experiments have established that systems combining custom segmentation with machine learning greatly outperform off-the-shelf OCR system at breaking captchas. For example, [2] showed that on the eBay audio captcha, the accuracy of a state of the art speech recognizer does not exceed 1%, whereas a custom classifier can exceed 75%. This three-stage approach works as follow: first, the solver pre-processes the captcha to make it easier to analyze, for instance by removing colors or by applying noise reduction techniques. Next, the solver attempts to segment the captcha into chunks that contain exactly one character, for example by using a clustering algorithm on the image. Finally, a classifier, such as a support vector machine (SVM) or a neural network, is used to recognize which character is contained in each chunk. Accordingly, we will refer to **anti-recognition techniques** to describe the image/text manipulations that aim at preventing the recognition of individual characters and to **anti-segmentation techniques** to describe image/text manipulations that aim at preventing the solver from splitting the captcha into individual characters. We will refer to the **core-features** to describe the captcha's basic design features, including its charset, font, length, whether this length is random, and so forth.

Many experiments [7] and attacks [32] have demonstrated that most captcha schemes are broken if they can be reliably segmented. Accordingly robust text-based schemes must make it difficult for the solver to determine *where* each character is. However, even if anti-segmentation techniques are essential to captcha security, they are only effective when the captcha core features and anti-recognition techniques are properly designed and implemented. Instead of solely focusing on preventing segmentation, we will show in this evaluation section that secure design principles need to be applied at all layers to create a secure scheme to avoid "side-channel attacks". Finally we introduced in [4] a new metric called **Learnability** which evaluates captcha strength based on the number of labeled examples required to train a classifier to a given precision level. Our learnability metric provides insight into how to properly choose anti-recognition techniques and core-features.

3. CORPUS

In this section we present the captcha corpus we used to establish our design principles and breaking techniques. As a starting point we collected and annotated 15 real-world schemes used by popular websites to evaluate Decaptcha performances against top-of-the-line captchas schemes. Decaptcha was able to break 13 of these 15 schemes. We analyzed these captchas to come up with a set of relevant security features that we used to create our synthetic corpus designed to study the effect of each of these features in detail and refine attacking techniques.

3.1 Popular Real World Captchas

To collect a representative sample of captchas, we consulted the Alexa list of most used websites[2] and identified the top sites which presented captchas as part of their account registration process. Additionally, we collected captchas from sites which provide captchas to other sites, e.g. Recaptcha.net and captchas.net. For each website or captcha scheme presented in figure 2, we collected directly from the website, *11,000* captchas that we had labeled by humans via Amazon crowd-sourcing service Mechanical Turk [5]. Decaptcha is able to break all of them except Recaptcha and Google.

3.1.1 Real-world Captcha Security Features

As visible in figure 2, real-world captchas exhibit a lot of variation in their design. By analyzing how each scheme is constructed we grouped the security defenses used in these schemes into the following ten techniques. Following the taxonomy presented in section 2, these techniques were assigned into the anti-recognition or the anti-segmentation category. We assigned to the anti-recognition category every feature that didn't directly prevent segmentation.

The anti-recognition techniques considered are: 1. **Multi-fonts** Using multiple fonts or font-faces. 2. **Charset** Which charset the scheme uses. 3. **Font size** Using variable font size. 4. **Distortion** Distorting the captcha globally using attractor fields. 5. **Blurring** Blurring letters. 6. **Tilting** Rotating characters with various angles. 7. **Waving** Rotating the characters in a wave fashion.

The anti-segmentation techniques considered are: 1. **Complex background** Try to hide the text in a complex background to "confuse" the solver. 2. **Lines** Add extra lines to prevent the solver from knowing what are the real character segments. 3. **Collapsing** Remove the space between characters to prevent segmentation.

[2]http://www.alexa.com/topsites

Scheme	Range from [5]	Generated
Authorize	95 – 98	92
Baidu	90 – 93	90
Blizzard	89 – 95	91
Ebay	93 – 93	94
Recaptcha	72 – 75	93

Table 1: Optimistic solving accuracy across schemes, comparing real world captchas to generated versions

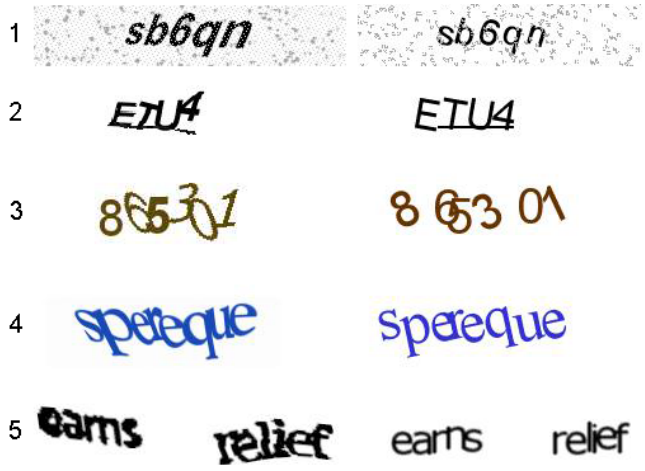


Figure 3: Real world captchas and our generated versions (generated on the left, real on the right) Captcha schemes depicted 1:Authorize, 2:Baidu, 3:eBay, 4:Google, 5:Recaptcha

3.2 Synthetic corpus

To generate our synthetic corpus we created a captcha generator. Using Mechnical turk we experimentally validated that our captcha generator is able to replicate real-world captchas. Synthetic captchas created by our generator have a similar accuracy to real world-captcha. Each generated captcha, (Figure 3), was annotated *1000* time by human using Mechanical Turk. We then measured the overall accuracy of each scheme, and compared these results to the scheme-level accuracies reported in [5]. While solving accuracy can be measured exactly for our fake captchas, the real ones used in our previous work were scraped from the web, and accordingly their true solutions were not known. So we can only compare our result to the *optimistic solving accuracy* metric we used previously. The table 1 shows for each scheme the optimistic solving accuracies reported in [5] (one for Mechanical Turk and one for an underground service) and the solving accuracy we measured on our generated captchas. As shown in table 1, the solving accuracy for our fake captchas are similar to the one observed on real world captchas except for Recaptcha. This experiment support the hypothesis that our taxonomy and its implementation are able to accurately replicate real world designs.

4. ATTACKING RECOGNITION

In this section we discuss how to represent a captcha so it is easy to process by machine learning algorithms. We motivate our algorithmic choices and evaluate their effectiveness on various anti-recognitions features. Based on the performance of the different machine learning algorithms, we can compare and recommend anti-recognition techniques.

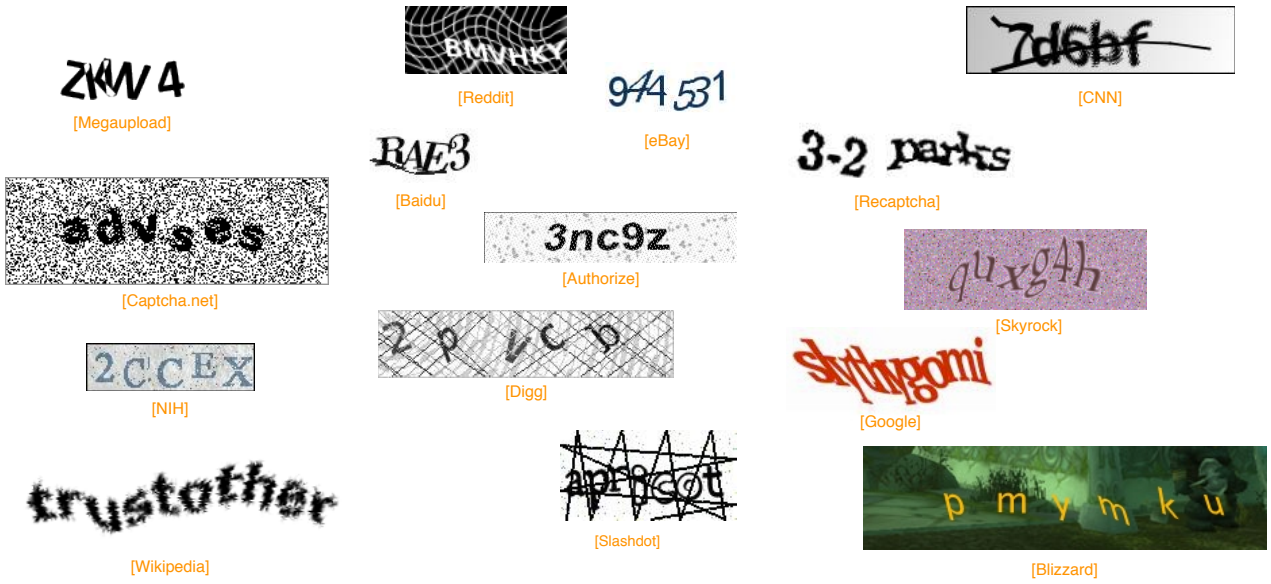


Figure 2: Samples of the 15 popular real world captcha schemes analyzed during our evaluation

4.1 Captcha representation recommendations

Character recognition is a core problem in machine learning. In the context of captchas prehaps the most relevant work produced by the machine learning community is on the MNIST database of handwritten digits challenge [21] which aims to recognize (distorted) handwritten digits. From this body of work, the most useful article for captcha security research is [20] which provide a deep analysis on how to efficiently recognize digits.

Based on this work and confirmed by our experimentations with Decaptcha on multiples schemes, we recommend:

- **Binarize letters**: While keeping letters in gray scale is useful for certain image algorithms, classifiers work better and faster on binary features so binarizing the letters in black and white is recommended. For example our custom distance algorithm is *35%* faster on binary vectors than integer (gray scale/color) vectors.

- **Work at the pixel level**: The most efficient way to represent letter is to use a matrix that encodes their pixel representation. Using "receptors" as, sometime recommended while doing standard OCR, is not efficient in our case because of the distortion, rotations and other captcha deformations.

When the captchas can't be segmented and we have to recognize the letters without segmentation, an alternative promising approach. would be to use very high level and complex image descriptors, such as SURF [1] and SIFT [23], that are invariant to rotation and very stable against distortion. In theory describing letters with robust "interest points" will make the approach faster and more stable. A huge hurdle for using this kind of descriptors to break captchas is the fact that the number of points that describe each letter can't be normalized, which prevents the use of the classifiers that are so efficient at recognizing characters when combined with the standard approach.

4.2 Recommended Classifiers

In terms of accuracy, the choice of classifier does not matter greatly because many modern classifiers perform strikingly well (i.e. *97%* - *99.5%*) on the MNIST dataset. Recall that to deem a scheme insecure our system only needs to reach 1% precision. In practice, small differences in classifier accuracy never substantially changed system performance. Accordingly instead of using the classifier that have the best accuracy we choose to evaluate the classifiers that are the easiest to use. Specifically, we focused on classifiers that are fast to train and require minimal parameter tuning.

We argue that having a classifier that is easy to parameterize and which is fast is the best choice for captcha security evaluation because most of the work is done before the recognition phase, so this phase should be as stable and as fast as possible. Waiting a couple of hours or even 15 minutes to see if a modification in the pipeline had an impact on the breaker performance would make the evaluation a very tedious process. We choose to **use SVM** [11] (Support Vector Machines) because this class of classifiers has become the de-facto classifier over the last few years and is known to almost always yield very good performance regardless of the problem. We choose to use a linear kernel rather than a polynomial kernel which would have achieved better performance because a linear kernel is an order of magnitude faster to train and does not require any parameter tweaking. We also recommend **using KNN** [12] (K Nearest Neighbors) classifier because it is the fastest classifier and it has nice stability properties that make it very reliable. The relative simplicity of the KNN allowed us to write our own version which was optimized to work on our binary vectors and with our sliding windows algorithm.

KNN requires more configuration than SVM as the number of neighbors (K) needs to be selected. To remove the burden of setting it by hand, we rely on a heuristic that computes the optimal K value, which is often 1, by performing a cross validation on the training set to find the optimal maximal K value. Because this heuristic requires quadratic time in the number of vectors in the dataset, we use the random sampling method when the vector set is too big (> 300). On our desktop computers, our KNN algorithm takes 20 seconds to learn a data set of 500 captchas and 2 minutes to classify 1,000 captchas. Because of its speed, KNN is our algorithm of choice when evaluating real world captcha schemes.

4.3 Anti-recognition features evaluation

Before evaluating real-world captchas, we wanted to compare the effectiveness of the anti-segmentation features in isolation to understand their impact on the classifier performance. Effectiveness here is quantified by the scheme learnability and the classifier success rate. To compute these numbers we repeatedly trained our classifier varying the size of the training set from 10 to 500 examples. Each testing phase was done on 1,000 captchas coming from a different set. The SVM results are summarized in the chart 4(a) and the KNN results are summarized in the chart 4(b).

The first observation we can make about these results is the fact that they support our claims that any reasonable classifier is "good enough" to build a captcha breaker. Overall, the SVM and the KNN classifiers both achieve very good results and exhibit a very similar learning rate. The only two major differences is that SVM does better on distortion (*61%* vs *50%*) and KNN performs better with the mix of five complex fonts (*62%* vs *59%*). As predicted by the theory, the KNN results are also more stable than the SVM ones, but as visible in the charts, the SVM accuracy jittering is minimal (at most *5%*) and is unlikely to affect the outcome of a security evaluation.

Recommendation. The results of our evaluation lead us to the following recommendations regarding anti-segmentation features.

- **Use a small non-confusable charset**: While using a larger charset slightly impacts the classifier accuracy and decreases the scheme's learnability, the security gain is too small to be useful: forcing the attacker to learn on 40 captchas instead of 10 reduces the accuracy from *100%* to *92%* which negligible compared to the loss in human accuracy (*98%* for 0-9 down to *82%* for azAZ09 [3]). Accordingly, since increasing the charset does not offer a significant security gain, a captcha charset should be small, with no caps at the very least, and should not contain confusing letters (e.g. i-j) to make it easy for humans to solve.

- **Don't use distortion**: Applying a distortion is the most effective way for reducing classifier accuracy and decreasing scheme learnability. However, this is not sufficient to prevent a classifier from being effective - this should be avoided and replaced with a proper anti-segmentation technique as distortion also harms user accuracy significantly [3].

- **Use rotation only in conjunction with anti-segmentation**: Rotating characters by itself doesn't significantly impede classifier accuracy and learnability; accordingly, their sole use is in conjunction with anti-segmentation techniques to make the size of each character unpredictable (See section 5.3).

  It is interesting to note here that we ran an additional experiment, in which we tried to learn on straight characters and tried to classify examples from this dataset. As predicted by the theory, SVM and KNN can't recognize rotated characters if they don't learn on them. Having classifiers insensitive to rotation is one of the main rationales behind the creation and use of more complex classifiers such as CNN (Convolutional Neural Networks) [20].

- **Use multiple fonts**: Using multiple fonts is an effective principle as it decreases significantly the classifier accuracy and will render the segmentation more difficult by making the size of characters unpredictable.

5. SEGMENTATION

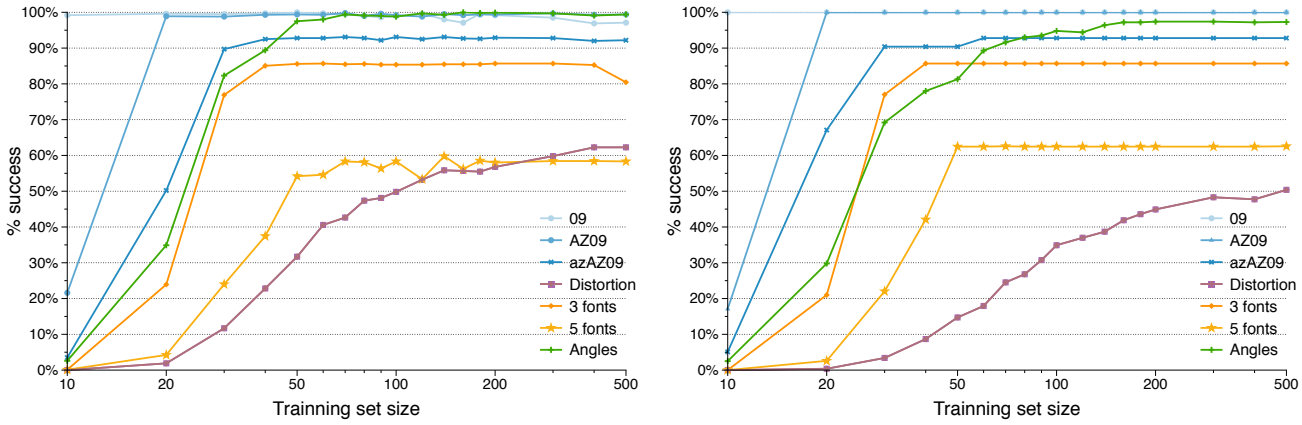
As seen in the previous section 4, while carefully chosen anti-recognition techniques help slow down the learning process and reduce classifier accuracy, they are not sufficient by themselves. In this section we analyze the effectiveness of the *7* anti-segmentation techniques we found in the wild on real captchas schemes and show their limitations. Note that we made the choice to focus on attacking techniques that are as generic as possible rather that technique optimized to break a specific captcha scheme. This choice make the techniques described below applicable to other schemes (we were able to break 13 schemes with the 7 techniques described below) at the expense of a couple of accuracy points. Based on the following analysis, we provide recommendations on which technique to use and how to implement them.

5.1 Background Confusion

Under the term **background confusion** we regroup all the techniques that try to prevent segmentation by "blending" the captcha text with the background. There are three main ways to achieve this: using a complex image background (figure 5), having a background that has "very" similar colors to the text (figure 6) and adding noise to the captcha (figure 7).

Some captchas schemes combine multiples background confusion techniques. However instead of increasing the security, combining background confusion techniques often lead to decrease it as it makes the scheme susceptible to more attacks. This is for example the case for Authorize (figure 8) which combines color similarity and noise: using gray noises make it susceptible to de-noising and anti-color attacks.

Complex background. The idea behind using a complex background is that the lines/shapes "inside it" will be confused with the real text and thus will prevent the breaker from isolating and segmenting the captcha. Eventhough previous works [31] have demonstrated that usually this type of defense is insecure, many captchas still rely on it. One of the most prominent examples of captcha using this type of defense is the one (figure 5) that Blizzard uses for all their websites (World of Warcraft, Starcraft II and Battle.net) . While they are using random backgrounds generated from game screenshots to prevent breakers from learning its shape they still have to make letters "stand out" from the background so that humans can decipher the captcha. We found out that that the easiest way to deal with captcha schemes that use random backgrounds but a finite number of colors is to use a technique that we call *anti-pattern*: for all the possible font colors remove everything from the captcha that is not close to this color and test if you get a reasonable number of clusters (letters) with the right amount of pixels. As visible in figure 5, this is very effective against Blizzard captchas and Decaptcha solves *70%* on them.



(a) SVM classifier

(b) KNN classifier

Figure 4: Effectiveness of classifiers on various anti-recognition features. These graphs depict how fast each classifier precision improves as more examples are added to the training set.

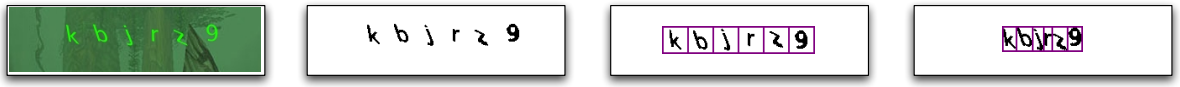


Figure 5: Example of the Blizzard pipeline

Color similarity. A related approach to the complex background techniques is to use colors that are perceived as very different by humans but are in reality very close in the RGB spectrum. The best and most sophisticated example of captcha scheme that uses this kind of technique is the Skyrock scheme visible in figure 6. While the letters appear very distinct to the human eye, when represented on the RGB spectrum they are so close that it is almost impossible to use the CFS [32] or the Anti-pattern techniques on it. However, as visible in figure 6, an effective way to counter this defense is to have the breaker work on a different color representation that is closer to the human perception, namely the HSV or HSL [29] ones, and *binarize* the captcha by using a threshold based on the hue or the saturation. For Skyrock we use a threshold based on the hue value. Changing the color space representation allows Decaptcha to get *2%* precision on Skyrock.

Noise. The last and "most efficient" technique used to confuse the segmentation is to add random noise to the image. For example, this technique is used in Captcha.net as visible in figure 7. Note that the noise must have the same color as the text because otherwise the anti-pattern technique can be applied to remove it. To de-noise captchas many techniques have been proposed over the years, including using the standard image filter *erode* [30]. However it turns out that using a MRF (Markov Random Field) aka Gibbs algorithm [14] is far more effective. A Gibbs de-noising algorithm is an iterative algorithm that works by computing the energy of each pixel based on its surroundings and removing pixels that have an energy below a certain threshold. The algorithm completes when there are no more pixels to remove. The energy of a given pixel is computed by summing the values on a gray scale of its 8 surrounding pixels and dividing by 8. As visible in figure 7 this algorithm completely negates the Captcha.net anti-segmentation defense and, accordingly, decaptcha is able to achieve *73%* precision on Captcha.net.

For Authorize, which also use noise, Decaptcha also achieves *66%* precision. As we will see in the next section using the Gibbs algorithm is also the best approach when the lines are smaller that the characters.

Recommendation. Overall, we believe that using any background confusion technique as a security mechanism is **not secure** and we recommend not relying on these kinds of techniques. While it is true that certain complex backgrounds are harder than others to remove, with sufficient effort and custom pre-processing, it is likely than any of these backgrounds can be processed. Accordingly, we recommend using background **only for cosmetic purposes**.

5.2 Using lines

A second approach to prevent segmentation is to use line(s) that cross multiple characters. This approach is used by Digg (figure 9) and Slashdot (figure 10) for instance. While it is possible to use lines that do not cross multiples characters , like the old Microsoft captcha, it has been proven to be a totally insecure approach [32] and is, therefore, not discussed here. In the wild we saw two types of lines used to prevent segmentation: *small lines* that cross the captcha's letters (e.g. Digg) and *large lines* of the width as the characters' lines that cross entire captchas (e.g. Slashdot and CNN).

Small lines. The first approach is to use small lines that will prevent the captcha from being segmented. This is the strategy used by Digg (figure 9). The standard approach to deal with small lines is to use a histogram-based segmentation [17, 31] that projects the captcha pixels to the X or Y coordinates.

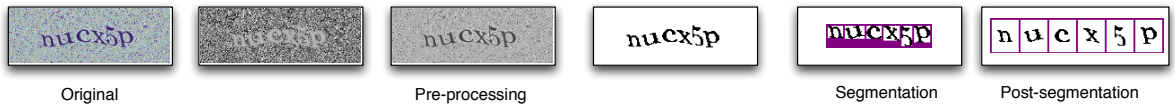


Figure 6: Example of the Skyrock pipeline

Figure 7: Example of the Captcha.net pipeline

This approach "works" because the region where the characters are is denser and therefore will create peaks in the histogram. The problem with this approach is how to determine the threshold and the size of the windows around it. It turns out that binarizing the captcha and then using a Gibbs de-noising algorithm with character reconstruction (see figure 9) is actually more efficient as it does not require such a brittle and complex tuning. Using Gibbs Decaptcha is able to achieve *86%* recall and *20%* precision on Digg captchas.

Big lines. The second approach is to use lines that have the same "width" as the character segments. The main advantage of this approach is that it is not susceptible to de-noising algorithms. However, it is susceptible to line-finding algorithms, such as the Canny edge detection [6] and the Hough Transform [13], because the lines cross the entire captcha. An illustration of our own implementation of the Hough Transform that preserves letters is visible in figure 10. As one can see, our implementation is able to find all the lines very accurately. The difficulty lies in the removal process that must preserve the letters. To do this, before removing a pixel we look at its surroundings to decide whether or not to remove it. The main reason behind Decaptcha's relatively low precision (*35%* precision) on Slashdot is the fact that Slashdot fonts have hollow characters that end up oftentimes damaged beyond repair when the lines are removed.

Recommendation. Based on our evaluation of captcha schemes we believe that using lines is a secure anti-segmentation defense when properly implemented. Overall, the goal of these principles is to prevent the attacker from finding a discriminator that will allow him to to tell apart character segments and lines. We recommend that in addition to the general security principles discussed in section 6, designers follow the following design principles when implementing this defense:

- **Use large lines:** Using lines that are not as wide as the character segments gives an attacker a robust discriminator and makes the line anti-segmentation technique vulnerable to many attack techniques including de-noising, projection-based segmentation and, in some rare cases, even the simple erode filter.

- **Keep the line within the captchas:** Line finding algorithms, such as the Hough transform, are very efficient at finding lines so for a defense mechanism to be effective, lines must cross only some of the captcha letters, so that it is impossible to tell whether it is a line or a character segment.

- **Don't use a strange slope:** Keep the angle of the line on par with the character segments otherwise the line slope will be used as a discriminator by the attacker. When using lines as anti-segmentation waving the captcha and tilting the characters will help ensure that it is hard for the attacker to distinguish between the lines and the character segments.

- **Match slopes:** The slope of the anti-segmentation lines must be roughly equivalent to the slope of a subset of the character segments. Otherwise when projecting in a Hough space the anti-segmentation lines will appear as outliners that are easily spotted.

- **Match color:** Anti-segmentation lines must be in the same color as the characters.

- **Randomize the length:** Make sure that the length of the line is variable to prevent the attacker from using its size as a discriminator.

5.3 Collapsing

Collapsing is considered by far to be the most secure anti-segmentation technique. While this is generally true, in practice the security of collapsing is often impeded by design flaws at the core feature level or at the anti-recognition level. That is why we distinguish two cases: one where the attacker can exploit a design flaw to predict the characters' segmentation despite the collapsing and the case where there is no flaw and the attacker is forced to "brute force" the captcha.

Predictable collapsing. Having the characters collapsed either by removing the space between characters ala Recaptcha or tilting them sufficiently ala eBay (figure 11) is insufficient to prevent the segmentation because the attacker can still guess where the cuts are likely to occur if the width of the letters is too regular and/or the number of letters is known in advance. As visible in figure 11 this is the case for eBay - we can't figure out where to cut but we know that there are 6 digits in their captchas and because the letter width is roughly always the same, we can make an educated guess and segment with reasonable success. We call this technique the **opportunistic segmentation** because it relies on"side channel information" to work. Overall, this segmentation works, as visible in figure 11, by first applying the standard CFS segmentation and then, based on the size of each segmented block, deciding how many characters each block contains using the fact that we either know the length of the captcha or the average size of the letters. Using this technique Decaptcha is able to achieve *43%* precision on eBay captchas.

Figure 8: Example of the Authorize pipeline

Figure 9: Example of the Digg pipeline using Gibbs

Figure 10: Example of the Slashdot pipeline

Figure 11: Example of the eBay pipeline

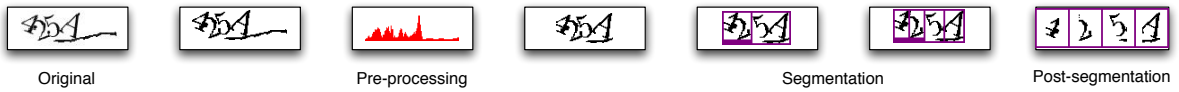


Figure 12: Example of the Baidu pipeline

Figure 13: Example of the CNN pipeline

Even if it seems at first sight that randomizing either the size of the letters or the length would be sufficient to prevent this kind of attack, this is not the case. Take Baidu (figure 12) for example. Even if Baidu performs heavy tilting and uses lines to prevent the attacker from guessing where to cut, knowing that the captcha has a length of 4 and using a projection based segmentation to get rid of the trailing lines allows Decaptcha to have a *5%* precision on Baidu captchas. It works better on CNN (figure 13) where we get *50%* recall and *16%* accuracy.

Unpredictable collapsing. When the number of characters is unknown and the average size of each character is unpredictable as in the Google captcha case, then the only option is to try to recognize each letter of the captcha directly without segmenting it. This kind of approach is fairly common and one solution might be to train on character templates segmented by hand and then use a space displacement neural network [24] to recognize the characters without segmenting first.

Recommendation. We recommend to **use collapsing as the main anti-segmentation technique**. Provided that all the other aspects of the captcha are properly designed, this anti-segmentation technique provides an efficient defense against segmentation. It is also advised to not use too aggressive collapsing, as after a certain threshold (-5px) the human accuracy drops drastically [3].

6. DESIGN PRINCIPLES FOR CREATING A SECURE CAPTCHA

In this section we briefly summarize our results, on both our synthetic corpus and on real-world captchas, to provide a comprehensive assessment of the state of the art. We then provide general principles for how to design secure captchas based on the lessons learned while doing this massive evaluation. We finish by discussing future research directions that are likely to change the current state of the art sooner or later.

6.1 Real World Captchas Evaluation Summary

Table 2 summarizes Decaptcha recall and precision on the *15* real-world schemes that we use as the basis of evaluation during the course of this work. For all the results provided in this table, we followed our recommended best practices and tested Decaptcha on 1,000 testing examples that were never used during the exploration or training phase. We only report in this table the precision achieved by KNN on 500 examples as SVM achieved very similar results. Our results also support our proposal to use the 1% precision mark to deem a scheme broken as we either clearly break a scheme or we don't, but we are never in the range of the 0.5% success rate. This evaluation also supports our claim (Section 4) that the best classifiers to evaluate image captcha security are those which are the simplest to configure and fastest to run, as recognition was never the bottleneck. Another thing we learned from this evaluation is that the design flaws introduced at the core feature and anti-recognition levels make a huge difference in the captcha scheme's overall security, regardless of the anti-segmentation technique(s) used. For example, because Slashdot used words we were able to bump Decaptcha accuracy from *24%* to *35%* by loosening the segmentation process and relying heavily on the spellchecking process. Similarly, we wouldn't have been able to achieve *43%* precision on eBay captchas without exploiting the fact that they are using a fixed number of digits and a very regular font width.

Overall, while we were able to break every scheme except Google and Recaptcha to a certain extent, it is clear that some schemes were more broken than others. When compared to the anti-segmentation technique used it is clear that relying on lines or collapsing is more secure than relying on a confusion background.

The figure 14 depicts the learning rate of Decaptcha against the various real-world schemes. The first observation we can make is that with 100 captchas we are already able to know if the scheme is broken or not. The second thing that we can notice is that the anti-segmentation techniques affect the learning rate: when these curves are compared to the earlier ones that focused on anti-recognition techniques only (Figures 4(a), 4(b)) it is apparent that the learning rate is slower when anti-recognition techniques are solely used. The shape of the real world scheme learning curves are very similar to the shape of the distortion technique curve which also tampers with letters integrity.

Scheme	Recall	Precision	Anti-segmentation
Authorize	84%	66%	background confusion
Baidu	98%	5%	collapsing
Blizzard	75%	70%	background confusion
Captcha.net	96%	73%	background confusion
CNN	50%	16%	line
Digg	86%	20%	line
eBay	95%	43%	collapsing
Google	0%	0%	collapsing
Megaupload	n/a	93%	collapsing
NIH	87%	72%	background confusion
Recaptcha	0%	0%	collapsing
Reddit	71%	42%	background confusion
Skyrock	30%	2%	background confusion
Slashdot	52%	35%	lines
Wikipedia	57%	25%	n/a

Table 2: Real world captchas summary

6.2 Design principles

Based on our evaluation results and experimentation with Decaptcha , 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. Overall, captcha scheme security comes from having a sound and coherent design at the core design, anti-recognition and anti-segmentation levels. Anti-segmentation techniques are only effective if the anti-recognition techniques and core design are sound. For example, using collapsing is only effective if the size and the number of characters are random. Failling to randomize either of these leaves the scheme vulnerable to an opportunistic segmentation such as in the eBay scheme. The Google scheme that implements all the design principles proposed in this section remains unbroken even-though it is in use for more than 4 years.

Core feature principles. The following principles apply to the design of the captcha core features:

1. **Randomize the captcha length**: Don't use a fixed length, it gives too much information to the attacker.
2. **Randomize the character size**: Make sure the attacker can't make educated guesses by using several font sizes / several fonts. As reported in section 4, using several fonts reduces the classifier accuracy and the scheme's learnability.

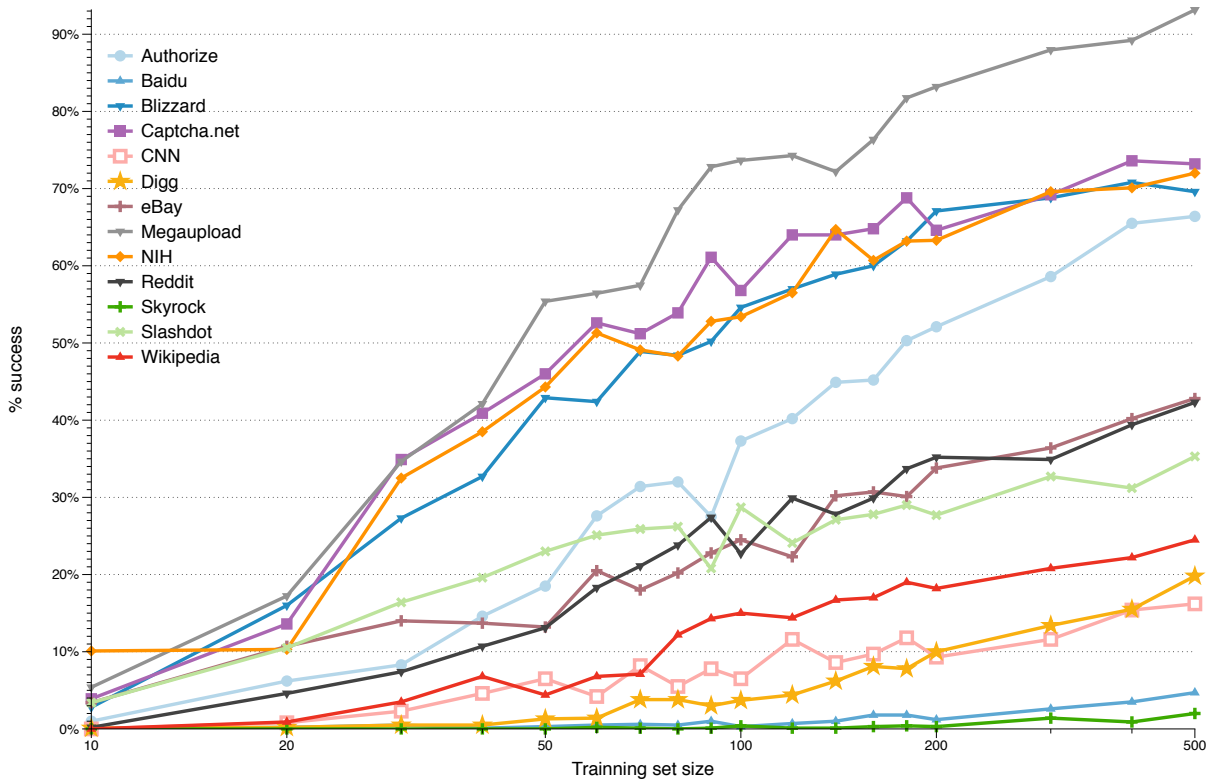


Figure 14: Real schemes learnability: Accuracy of Decaptcha using KNN vs the size of the training set. Logarithmic scale

3. **Wave the captcha**: Waving the captcha increases the difficulty of finding cut points in case of collapsing and helps mitigate the risk of the attacker finding the added line based on its slope when using lines.

Anti-recognition.

1. **Use anti-recognition techniques as a means of strengthening captcha security**: Don't rely on anti-recognition techniques to protect your scheme, use them to strengthen the overall captcha scheme security. Because most classifier efficiency is sensitive to rotation, scaling and rotating some characters and using various font sizes will reduce the recognition efficiency and increase the anti-segmentation security by making character width less predictable.

2. **Don't use a complex charset**: Using a large charset does not improve significantly the captcha scheme's security and really hurts human accuracy, thus using a non-confusable charset is the best option.

Anti-Segmentation.

1. **Use collapsing or lines**: Given the current state of the art, using any sort of complex background as an anti-segmentation technique is considered to be insecure. Using lines or collapsing correctly are the only two secure options currently. Complex background can be used as a second line of defense (e.g. the ellipses used in some Recaptcha's captchas).

2. **Be careful while implementing**: To be effective, anti-segmentation techniques must be implemented very carefully. When using lines, follow all the recommendations provided in section 5.2 and when implementing collapsing, make sure to follow the recommendations provided in section 5.3.

3. **Create alternative schemes**: As with cryptography algorithms, it is good practice to have alternative captcha schemes that can be rolled out in case of a break. Variations of the same battle-hardened schemes with additional security features is likely the easiest way to prepare alternative schemes. This seems to be the strategy of Recaptcha, which has alternative schemes that surface from time to time.

7. DECAPTCHA

In this section we present our captcha breaker, *Decaptcha*, which is able to break many popular captchas including eBay, Wikipedia and Digg. Then we discuss the rationale behind its five stage pipeline, its benefits, and its drawbacks, and conclude by deriving principles on how do build a successful solver.

Decaptcha implements a refined version of the three stage approach in 15,000 lines of code in C#. We chose C# because it offers a good tradeoff between speed, safety, robustness and the availability of AI/Vision libraries. We also chose C# because of the visual studio interface builder quality, as evaluating captcha security efficiently requires designing a fairly complex UI for debugging and tweaking purposes. Decaptcha uses the aForge framework [19] and the Accord framework that provide easy access to image manipulation filters, and standard machine learning algorithms such as SVM [11].

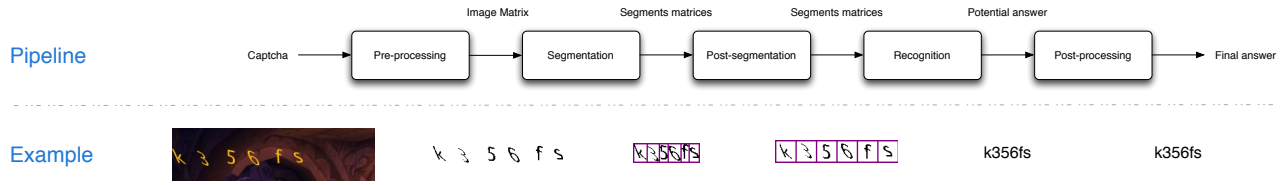


Figure 15: Decaptcha pipeline

Overall, although we tried to use existing libraries as much as possible we ended up writing roughly 80% of Decaptcha code, which took us at least a year of development. For example, we rewrote a KNN algorithm [12] because we needed a confidence metric and we rewrote various distance algorithms to maximize the speed on binary vectors. Note that Decaptcha in its current version is able to work on audio and image captchas.

7.1 Decaptcha pipeline

Decaptcha uses the five stage pipeline illustrated in figure 15. These stages are:

1. **Preprocessing**: In this first stage, the captcha's background is removed using several algorithms and the captcha is binarized (represented in black and white) and stored in a matrix of binary values. Transforming the captcha into a binary matrix makes the rest of the pipeline easier to implement, as the remaining algorithm works on a well-defined abstract object. The downside of using a binary representation is that we lose the pixel intensity. However in practice this was never an issue.

2. **Segmentation**: In this stage Decaptcha attempts to segment the captchas using various segmentation techniques, the most common being **CFS [32] (Color Filling Segmentation)** which uses a paint bucket flood filling algorithm [28]. This is the default segmentation technique because it allows us to segment the captcha letters even if they are tilted, as long as they are not contiguous.

3. **Post-Segmentation**: At this stage the segments are processed individually to make the recognition easier. During this phase the segments' sizes are always normalized.

4. **Recognition**: In training mode, this stage is used to teach the classifier what each letter looks like after the captcha has been segmented. In testing mode, the classifier is used in predictive mode to recognize each character.

5. **Post-processing**: During this stage the classifier's output is improved when possible. For example, spell checking is performed on the classifier's output for Slashdot because we know that this captcha scheme uses dictionary words. Using spellchecking allows us to increase our precision on Slashdot from *24%* to *35%* .

7.2 Design principles to write a captcha solver

Before writing the full blown version of Decaptcha in C#, we wrote a prototype in Ruby two years ago. Building this prototype allowed us to learn a couple of key principles that need to be applied to create a sucessful evaluation framework.

Here is the list of the four main design principles specific to captcha breaking that made the current Decaptcha implementation (Figure 16) an effective attack framework:

1. **Aiming for generality**: Decaptcha development was focused on algorithm generality and simplicity rather than accuracy optimization. We made this choice very early as we believed that on the long run it will yield better results. The fact that we were able to break the last three schemes evaluated in this paper, namely CNN, Megaupload and Reddit, in less than 3 hours without writing a new algorithm support this hypothesis. Overall we believe that this focus on generality and simplicity is what makes Decaptcha truly different from the previous add-hoc designed to break a single scheme.

2. **Immediate visual feedback**: When trying to break a captcha scheme most of the time is spend on trying and tweaking various algorithms, so it is essential to have quick feedback on how the change affected the attack's performance. We discovered that it is far more effective to provide this feedback in a pie chart form with a defined color code than use a table with raw numbers. As visible in figure 16 in Decaptcha the pie chart is in the center of the interface, which allows us to immediately see how efficient the current pipeline is. For example, in the screenshot it is very easy to see that in this tryout we have an overall *success of* 66% *(green)* on Blizzard captchas, that *5%* of the failures occur at the *recognition stage (yellow)*, *18%* - at the *segmentation stage (orange)* and *11%* - at the *pre-processing stage (red)*.

3. **Visual debugging**: Similarly, we discovered that the only way to understand quickly how an algorithm is behaving is to look at how it affects and interacts with the captchas. That is why the ability to view the visual pipeline for a given captcha sample with a simple click is essential. In Decaptcha we implemented this principle by allowing the user to display a given captcha pipeline stage on the right side of the interface by clicking on a captcha from the list located in the middle of the interface. For example, in the example of figure 16, we selected a captcha that failed the segmentation stage and the fact that the failure occurs at the segmentation is clear by looking at the pipeline states. It also makes it very easy to understand that this segmentation failure is due to an error of our anti-pattern algorithm, which removed most of the background pattern except a few pixels at the bottom right, due to the similarity of their color to the text color.

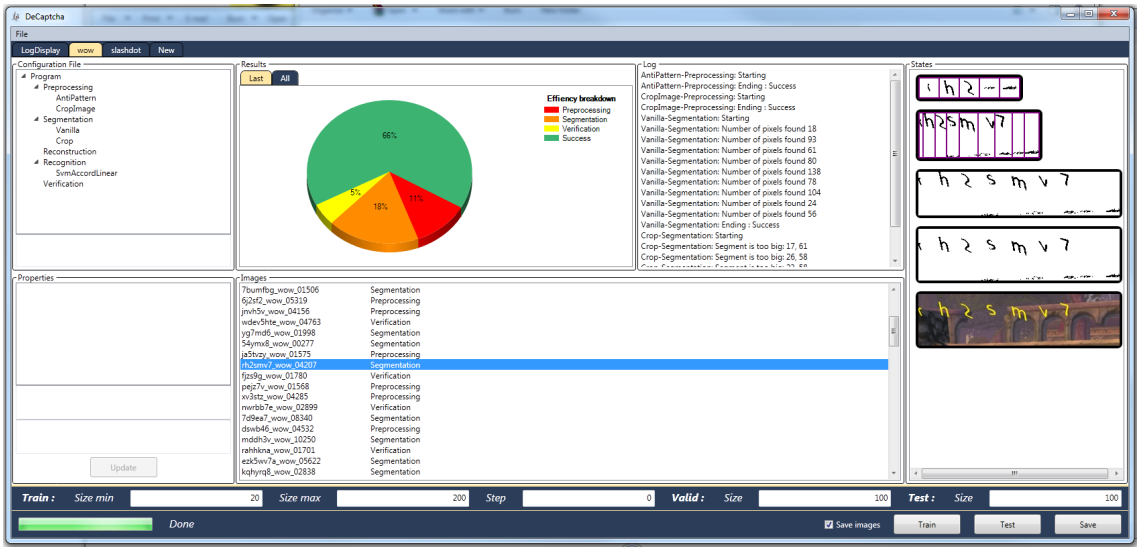


Figure 16: Decaptcha interface

4. **Algorithm independence**: Finding the optimal set of algorithms to break a given scheme is not trivial and often we ended up swapping one algorithm for another either because we found a better-performing algorithm or because we changed the approach. For example, for de-noising a captcha we moved from using an anti-pattern algorithm to a Markov Random Field algorithm [14]. Being able to combine algorithms as "lego bricks" without worrying about side effects is one of the keys to Decaptcha 's success. Having a flexible pipeline is achieved by abstracting the image representation as a matrix and ensuring that every algorithm has no side effects. This design also allows us to parallelize pipeline executions which is important because image processing and machine learning algorithms are usually slow. The algorithm independence principle is also what allows Decaptcha to work on image and audio captchas indistinctly.

5. **Exposing algorithm attributes**: Being able to change algorithm parameters such as a threshold without editing and recompiling the code makes a huge difference. Oftentimes, by tweaking parameters we were able to gain up to *40%* in accuracy or segmentation efficiency. We tried to find a way to automatically optimize parameters but it turned out that modifying the parameters in one algorithm in isolation is not effective, as changing the behavior of one algorithm often requires re-adjusting parameters of algorithms used later in the pipeline. For example, being more aggressive when de-noising will force us to be more aggressive when reconstructing the captcha's characters afterward.

8. FURTHER RELEVANT WORK

In this section we summarize the related work cited in the paper and discuss further relevant work.

Captcha. In [10] the authors propose using machine learning classifiers to attacks captchas. In [7] the same authors study how efficient statistical classifier are at recognizing captcha letters. In [5] the authors study how good humans are at solving well-known captchas using Mechanical Turk.

In [15] the authors were able to break the Microsoft ASIRRA captcha using SVM. In [32] the authors were able to break the old Microsoft captcha using the two phase approach. In [30] the author proposes using the erode and dilate filter to segment captchas. [31] is one of the first papers to propose the use of histogram-based segmentation against captchas.

Recognition algorithm. The perceptron, the simplest neural network, has been used as a linear classifier since 1957 [27]. The convolutive neural networks which are considered to be the most efficient neural network to recognize letters were introduced in [20]. The space displacement neural network that attempts to recognize digits without segmentation was introduced in [24]. The support vector machines were introduced in [11]. The KNN algorithm is described in detail in [12]. The use of a bag of features to recognize objects in images is a very active field. The closest work to ours in this area is by [22], where the authors try to segment and categorize objects using this approach.

Machine vision algorithms. Detecting and removing lines is a well studied field in computer vision since the '70s. Two well-known and efficient algorithms that can be used against captchas with lines are the Canny detection [6] and the Hough Transform [13]. Removing noise using a Markov Random Field (Gibbs) was introduced in [14]. Many image descriptors have been proposed over the last decades: one of the first and most used descriptors is the the Harris Corner detector [16] introduced in 1988. However, recently it has been replaced by more complex descriptors that are insensitive to scale and rotation (to a certain extent). Of these, the two most notable and promising for dealing with captchas are SIFT [23] and SURF [1].

9. CONCLUSION

As a contribution toward improving the systematic evaluation and design of visual captchas, we evaluated various automated methods on real world captchas and synthetic one generated by varying significant features in ranges potentially acceptable to human users. We evaluated state-of-the-art anti-segmentation techniques, state-of-the-art anti-recognition techniques, and captchas used by the most popular websites.

We tested the efficiency of our tool Decaptcha against real captchas from *Authorize, Baidu, Blizzard, Captcha.net, CNN, Digg, eBay, Google, Megaupload, NIH, Recaptcha, Reddit, Skyrock, Slashdot, and Wikipedia.* On these 15 captchas, we had *1%-10%* success rate on two (*Baidu, Skyrock*), *10-24%* on two (*CNN, Digg*), *25-49%* on four (*eBay, Reddit, Slashdot, Wikipedia*), and *50%* or greater on five (*Authorize, Blizzard, Captcha.net, Megaupload, NIH*). To achieve such a high success rate we developed the first successful attacks on captchas that use collapsed characters (*eBay and Baidu*). Only Google and Recaptcha resisted to our attack attempts, and we reached some informative understanding of why we couldn't break them. Because of Decaptcha genericity we were able to break 7 of these 15 schemes without writing a new algorithm. Overall, our analysis led to a series of recommendations for captcha designers, including recommendations to use some anti-segmentation techniques, and recommendations not to use features that are ineffective against automated attacks but counterproductive for humans.

Acknowledgment

We thank Markus Jakobsson, Dave Jackson, Aleksandra Korolova and our anonymous reviewers for their comments and suggestions. This work was partially supported by the National Science Foundation, the Air Force Office of Scientific Research, and the Office of Naval Research.

10. REFERENCES

[1] H. Bay, T. Tuytelaars, and L. Van Gool. Surf: Speeded up robust features. *Computer Vision–ECCV 2006*, pages 404–417, 2006.

[2] E. Bursztein and S. Bethard. Decaptcha: breaking 75% of eBay audio CAPTCHAs. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, page 8. USENIX Association, 2009.

[3] E. Bursztein, S. Bethard, Fabry C., Dan Jurafsky, and John C. Mitchell. Design parameters and human-solvability of text-based captchas. To appears.

[4] Elie Bursztein, Romain Bauxis, Hristo Paskov, Daniele Perito, Celine Fabry, and John C. Mitchell. The failure of noise-based non-continuous audio captchas. In *Security and Privacy*, 2011.

[5] Elie Bursztein, Steven Bethard, John C. Mitchell, Dan Jurafsky, and Celine Fabry. How good are humans at solving captchas? a large scale evaluation. In *Security and Privacy*, 2010.

[6] J. Canny. A computational approach to edge detection. *Readings in computer vision: issues, problems, principles, and paradigms*, 184:87–116, 1987.

[7] K. Chellapilla, K. Larson, P.Y. Simard, and M. Czerwinski. Computers beat humans at single character recognition in reading based human interaction proofs (hips). In *CEAS*, 2005.

[8] K Chellapilla and P Simard. Using machine learning to break visual human interaction proofs. In MIT Press, editor, *Neural Information Processing Systems (NIPS)*, 2004.

[9] K. Chellapilla and P. Simard. Using machine learning to break visual human interaction proofs (HIPs). *Advances in Neural Information Processing Systems*, 17, 2004.

[10] K. Chellapilla and P.Y. Simard. Using machine learning to break visual hips. In *Conf. on Neural Information Processing Systems, NIPS 2004*, 2004.

[11] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[12] B.V. Dasarathy. Nearest Neighbor ({NN}) Norms:{NN} Pattern Classification Techniques. 1991.

[13] R.O. Duda and P.E. Hart. Use of the Hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1):11–15, 1972.

[14] S. Geman and D. Geman. Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images*. *Journal of Applied Statistics*, 20(5):25–62, 1993.

[15] P. Golle. Machine learning attacks against the asirra captcha. In *ACM CCS 2008*, 2008.

[16] C. Harris and M. Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK, 1988.

[17] S.Y. Huang, Y.K. Lee, G. Bell, and Z. Ou. A projection-based segmentation algorithm for breaking MSN and YAHOO CAPTCHAs. In *Proceedings of the World Congress on Engineering*, volume 1. Citeseer, 2008.

[18] P Simard K Chellapilla, K Larson and M Czerwinski. Building segmentation based human- friendly human interaction proofs. In Springer-Verlag, editor, *2nd Int'l Workshop on Human Interaction Proofs*, 2005.

[19] Andrew Kirillov. aforge framework. http://www.aforgenet.com/framework/.

[20] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[21] Yann Lecun. The mnist database of handwritten digits algorithm results. http://yann.lecun.com/exdb/mnist/.

[22] B. Leibe, A. Leonardis, and B. Schiele. Robust object detection with interleaved categorization and segmentation. *International Journal of Computer Vision*, 77(1):259–289, 2008.

[23] D.G. Lowe. Object recognition from local scale-invariant features. In *iccv*, page 1150. Published by the IEEE Computer Society, 1999.

[24] O. Matan, C.J.C. Burges, and J.S. Denker. Multi-digit recognition using a space displacement neural network. *Advances in Neural Information Processing Systems*, pages 488–488, 1993.

[25] Moni Naor. Verification of a human in the loop or identification via the turing test. Available electronically: http://www.wisdom.weizmann.ac.il/~naor/PAPERS/human.ps, 1997.

[26] R. Quinlan. *Machine Learning*. Morgan Kaufmann Pub.

[27] F. Rosenblatt. The perceptron: a perceiving and recognizing automation (projet PARA), Cornell Aeronautical Laboratory Report. 1957.

[28] Wikipedia. Flood fill algorithm. http://en.wikipedia.org/wiki/Flood_fill.

[29] Wikipedia. Hsl and hsv color representaiton. http://en.wikipedia.org/wiki/HSL_and_HSV.

[30] J. Wilkins. Strong captcha guidelines v1. 2. *Retrieved Nov*, 10:2010, 2009.

[31] J. Yan and A.S.E. Ahmad. Breaking visual captchas with naive pattern recognition algorithms. In *ACSAC 2007*, 2007.

[32] 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*, pages 543–554. ACM, 2008.