Security Analysis and Related Usability of Motion-based CAPTCHAs: Decoding Codewords in Motion

Y. Xu, G. Reynaga, S. Chiasson, J-M. Frahm, F. Monrose, and P. van Oorschot

Abstract—We explore the robustness and usability of moving-image object recognition (video) captchas, designing and implementing automated attacks based on computer vision techniques. Our approach is suitable for broad classes of moving-image captchas involving rigid objects. We first present an attack that defeats instances of such a captcha (NuCaptcha) representing the state-of-the-art, involving dynamic text strings called codewords. We then consider design modifications to mitigate the attacks (e.g., overlapping characters more closely, randomly changing the font of individual characters, or even randomly varying the number of characters in the codeword). We implement the modified captchas and test if designs modified for greater robustness maintain usability. Our lab-based studies show that the modified captchas fail to offer viable usability, even when the captcha strength is reduced below acceptable targets. Worse yet, our GPU-based implementation shows that our automated approach can decode these captchas faster than humans can, and we can do so at a relatively low cost of roughly 50 cents per 1000 captchas solved based on Amazon EC2 rates circa 2012. To further demonstrate the challenges in designing usable captchas, we also implement and test another variant of moving text strings using the known *emerging images* concept. This variant is resilient to our attacks and also offers similar usability to commercially available approaches. We explain why fundamental elements of the emerging images idea resist our current attack where others fail.

Index Terms—CAPTCHAs, security, usability, computer vision

1 Introduction

Humans can recognize a wide variety of objects at a glance, with no apparent effort, despite tremendous variations in the appearance of visual objects; and we can answer a variety of questions regarding shape properties and spatial relationships of what we see. The apparent ease with which we recognize objects belies the magnitude of this feat. We can also do so with astonishing speed (e.g., in a fraction of a second) [37]. Indeed, the Cognitive Science literature abounds with studies on visual perception showing that, for the most part, people do not require noticeably more processing time for object categorization (e.g., deciding whether the object is a bird, a flower, a car) than for more fine grained object classification (e.g., an eagle, a rose) [14]. Grill et al. [20] showed that by the time subjects knew that a picture contained an object at all, they already knew its class. If such easy-for-human tasks are, in contrast, difficult for computers, then they are strong candidates for distinguishing humans from machines.

Since understanding what we see requires *cognitive* ability, it is unsurprising that the decoding of motion-based challenges has been adopted as a security mechanism: various forms of motion-based object recognition tasks have been suggested as reverse Turing tests, or what are called Completely Automated Public Turing tests to tell Computers and Humans Apart (captchas). Among the key properties of captchas are: they must be easily solved by humans; they should be usable; correct solutions should only be attainable by solving the underlying AI problem they are based on; they should be robust (i.e., resist automated attacks); and the cost of answering challenges with automated programs should exceed that of soliciting humans to do the same task [1, 42]. To date, a myriad of text, audio, and video-based captchas have been suggested [22], many of which have succumbed to different attacks [6, 7, 19, 31, 44, 45, 50].

While text-based captchas that prompt users to recognize distorted characters have been the most popular form to date, motion-based or video captchas that provide some form of moving challenge have recently been proposed as the successor to static captchas. One prominent and contemporary example of this new breed of captchas is NuCaptcha [34], which asserts to be "*the most secure and usable captcha*," and serves millions of video captchas per day. The general idea embodied in these approaches is to exploit the remarkable perceptual abilities of humans, which allows us to quickly unravel complex patterns in dynamic scenes. For example, in the case of NuCaptcha, users are shown a video with a series of characters (so-called random *codewords*) moving across a dynamic scene, and solve the captcha by identifying the characters of the codeword. For enhanced security, the codewords are presented among adversarial clutter [31] (e.g., moving backgrounds and other objects with different trajectories), and consecutive characters may even overlap significantly. The underlying assumption is that attacks based on state-of-the-

- *The authors are with the Department of Computer Science, University of North Carolina at Chapel Hill, USA and School of Computer Science, Carleton University, Canada.*
  E-mail contact: yix@cs.unc.edu
- *An earlier version of this paper appeared in the USENIX Security Symposium, August 8-10, 2012 [43].*

art computer vision techniques are likely to fail at uncovering these challenges within video sequences, whereas real users will be able to solve the challenges with little effort.

However, it turns out that in computer vision, object *classification*, is far more challenging than *recognition* of known objects [39]. That is, it is considerably more difficult to capture in a computer recognition system the essence of a dog, a horse, or a tree—i.e., the kind of classification that is natural and immediate for the human visual system [29]. To this day, classification of objects in real-world scenes remains an open and difficult problem. Recognizing known objects, on the other hand, is more tractable, especially where it involves specific shapes undergoing transformations that are easy to compensate for. As we show later, many of these well-defined transformations hold in current motion-based captcha designs, due in part to design choices that increase usability.

In what follows, we present an automated attack to defeat the current state-of-the-art in moving-image object recognition captchas. Through extensive evaluation of several thousand real-world captchas, our attack can completely undermine the security of the most prominent examples of these, namely those generated by NuCaptcha circa 2012. After examining properties that enable our attack, we explore a series of security countermeasures designed to reduce the success of our attacks, including natural extensions to the scheme under examination, as well as an implementation of a recently proposed idea (called Emerging Images [30]) for which attacks do not appear as readily available. Rather than idle conjecture about the efficacy of countermeasures, we evaluate these strengthened variations of moving-image captchas by carrying out and reporting on a usability study where subjects were tasked with solving such captchas.

Our findings highlight the well-known tension between security and usability, which often have subtle influences on each other. In particular, we show that the design of robust and usable moving-image captchas is much harder than it looks.

2 Captcha Taxonomy and Related Work

Most captchas in commercial use today are character-recognition (CR) captchas involving still images of distorted characters; attacks essentially involve building on optical character recognition advances. Audio captchas (AUD) are a distinct second category, though unrelated to our present work. A third major category, image-recognition (IR) captchas, involves classification or recognition, of images or objects other than characters. A well-known example, proposed and then broken, is the Asirra captcha [17, 19] which involves object classification (e.g., distinguishing cats from other animals such as dogs). CR and IR schemes may involve still images (CR-still, IR-still), or various types of dynamic images (CR-dynamic, IR-dynamic). Dynamic text and objects are of main interest in the present paper, and contribute to a cross-class category: moving-image object recognition (MIOR) captchas, involving objects in motion through animations, emergent-image schemes, and video [10–12, 26, 30, 34, 35]. A fourth category, cognitive-based captchas (COG), include puzzles, questions, and other challenges related to the semantics of images or language constructs. We include here content-based video-labeling of YouTube videos [24].

The most comprehensive surveys of captchas to date are those by Hidalgo and Maranon [22] and Basso and Bergadano [2]. We also note other summaries: for defeating classes of AUD captchas, Soupionis [36] and Bursztein et al. [4, 6]; for defeating CR captchas, Yan et al. [44, 47] and Bursztein [7]; for a systematic treatment of IR captchas and attacks, Zhu et al. [50], as well as for robustness guidelines.

Usability has also been a central focus, for example, including a large user study of CR and AUD captchas involving Amazon Mechanical Turk users [5], a user study of video-tagging [24], usability guidelines and frameworks related to CR captchas [46]. Chellapilla et al. [8, 9] also address robustness. Hidalgo et al. [22] and Bursztein et al. [7] also review evaluation guidelines including usability. Research on underground markets for solving captchas [32], and malware-based captcha farms [16], raise interesting questions about the long-term viability of captchas.

Concurrent to our own work, Bursztein [3] presented an approach to break the video captchas used by NuCaptcha. The technique exploits the video by treating it as a series of independent frames, and then applies a frame-based background removal process [7] to discard the video background. Next, frame characteristics (e.g., spatial salient feature density and text aspect ratio of the overlapping letters) are used to detect the codeword, after which a clustering technique is used to help segment the characters of the codeword. As a final step, traditional CR-still based attacks are used to recognize the characters in each of the segmented frames. Some stages of the approach taken by Bursztein have similarities to our baseline method (§3.1) which uses single frame segmentation and recognition. In contrast, our subsequent techniques inherently use temporal information contained in the video to identify the codeword, to improve the segmentation, and to enhance the recognition step during the codeword recovery process.

Lastly, this paper significantly extends our earlier work [43] to include an improved segmentation procedure that does not assume a priori knowledge of the length of the codeword (in §3.2), new countermeasures and an improved classification technique (in §3.3), and new CPU and GPU-based implementations with an accompanying performance evaluation on Amazon's Elastic Cloud service (in §4.2). The extended analysis in this paper show that NuCaptcha's "adaptive captcha authentication" recommendations[1] in response to our attacks and those of Bursztein fail to offer viable alternatives.

3 Our Automated Approach

Our ability to find the different objects in the image that "go together" (i.e., the *segmentation* process [39]), to estimate the trajectory of objects as they move around a scene (i.e., *object tracking*), and to separate the foreground from the background, stems from several design decisions that promote rapid visual identification [15] in today's MIOR captchas. NuCaptcha, for instance, presents a streaming video containing moving text against a dynamic background. The videos have

[1] See the February 2012 press release at http://www.nucaptcha.com/press.

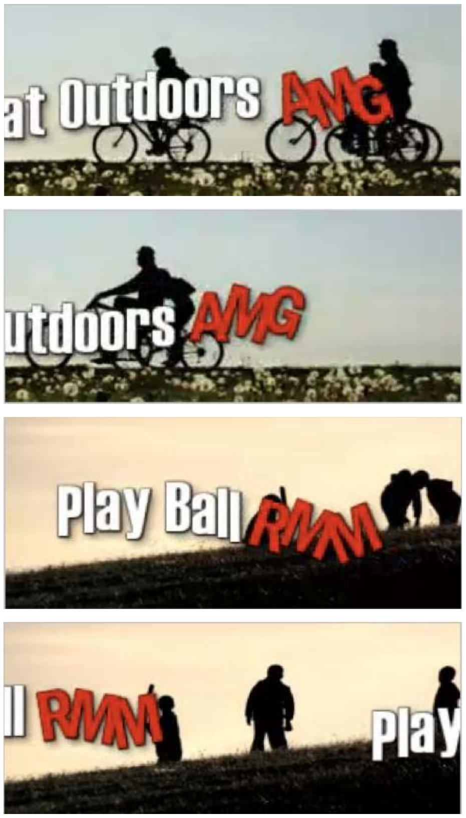


Fig. 1: Example moving-image object recognition (MIOR) captchas from NuCaptcha (see http://nucaptcha.com/demo).

four noticeable characteristics, namely: (1) the letters are presented as rigid objects in order to improve a user's ability to recognize the characters; (2) the background video and the foreground character color are nearly constant in color and always maintain a high contrast—we posit that this is done to ease cognitive burden on users; (3) the random "codewords" each have independent (but overlapping trajectories) which better enable users to distinguish adjacent characters; (4) the codewords are chosen from a reduced alphabet where easily confused characters are omitted. Some examples of a state-of-the-art MIOR captcha are given in Figure 1.

Before delving into the specifics of our most successful attack, we first present a naïve approach for automatically decoding the challenges shown in MIOR captchas. We remind the reader that a video can be thought of as a stream of single pictures that simply provides multiple views of a temporally evolving scene. Moreover, it is well known that human observers perceive a naturally moving scene at a level of about 30 frames per second, and for this reason, video captchas tend to use a comparable frame rate to provide a natural video experience that is not too jerky. Similarly, the challenge shown in the captcha is rendered in multiple frames to allow users to perceive and decode the codewords in an effortless manner. In the NuCaptcha scheme, for example, a single frame may contain the full codeword.

3.1 A Naïve Attack

Given this observation, one way to attack such schemes is to simply apply traditional OCR-based techniques that work well at defeating CR-still captchas (e.g., [31, 44]). More specifically, choose k frames at random, and identify the foreground pixels of the codeword by comparing their color with a given reference color; notice the attacker would likely know this value since the users are asked to, for example, "type the RED moving characters". Next, the length of the codeword can be inferred by finding the leftmost and rightmost pixels on the foreground. This in essence defines a line spanning over the foreground pixels (see Figure 2). The positions of the characters along the line can be determined by dividing the line into n equidistant segments, where n denotes the desired number of characters in the codeword. For each of the segments, compute the center of gravity of the foreground pixels in the vertical area of the image belonging to the segment. Lastly, select an image patch (of the expected size of the characters) around the centers of gravity of the segments, and feed each patch to a classifier. In our work, we use a multi-layer feedforward neural network approach [13] that is known to perform well at this object identification task. The neural network is trained similarly to what we discuss in §3.3.

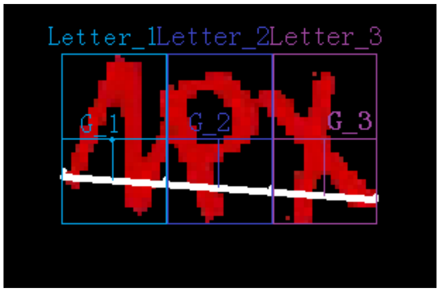


Fig. 2: Using the foreground pixels, find the longest horizontal distance (white line) and the mean value of the vertical area (i.e., the respective bounding boxes shown above).

The above process yields a guess for each of the characters of the codeword in the chosen frames of the video. Let i denote the number of possible answers for each character. By transforming the score from the neural network into the probability p_{ijk} where the j-th character of the codeword corresponds to the i-th character in the k-th frame, we calculate the probability P_{ij} for each character $j = 1, \ldots, n$ of the codeword over all k frames as $P_{ij} = \frac{1}{s_p} \sum_k p_{ijk}$ with $s_p = \sum_{i,j,k} p_{ijk}$. The choice that has the highest probability is selected as the corresponding character. With $k = 10$, this naïve attack succeeded in decoding all 3 characters in the codeword more than a third of the time. While this relatively simple attack already raises doubts about the robustness of this new MIOR captcha, we now present an enhanced attack that not only makes fewer assumptions about pixel invariants [47] in the videos, but also significantly bolsters our success rate.

3.2 Exploiting Temporal Information

A clear limitation of the naïve attack is the fact that it is not easily generalizable and it is not robust to slight changes in the videos. In what follows, we do not assume knowledge of the color of the codewords, nor do we assume that the centers of gravity for each patch are equidistant. To do so, we apply a robust segmentation method that utilizes temporal information to improve our ability to recognize the characters in the video.

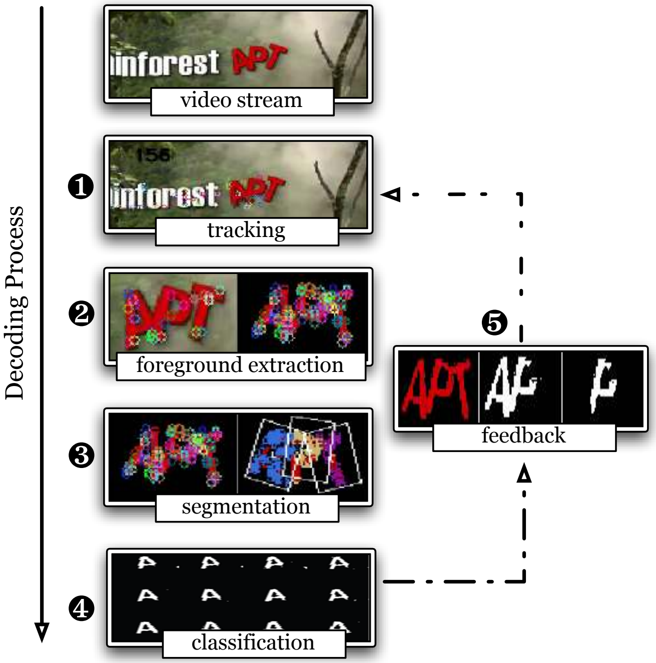


Fig. 3: High-level overview of our attack. (This, and other figures, are best viewed in color.)

A basic overview of our attack is shown in Figure 3. Given a MIOR captcha we extract the motion contained in the video using the concept of salient features. Salient features are characteristic areas of an image that can be reliably detected in several frames. To infer the motion of the salient feature points, we apply object tracking techniques (stage ❶). With a set of salient features at hand, we then use these features to estimate the color statistics of the background. Specifically, we use a Gaussian mixture model [18], which represents the color statistics of the background through a limited set of Gaussian distributions. We use the color model of the background to measure, for all pixels in each frame, their likelihood of belonging to the background. Pixels with low likelihoods are then extracted as foreground pixels (stage ❷). The trajectories of the foreground pixels are then refined using information inferred about the color of these pixels, and a foreground color model is built. Next, to account for the fact that all characters of the codewords move independently, we segment the foreground into n segments as in the naïve attack (stage ❸). We select each image patch containing a candidate character and evaluate the patch using a neural network based classifier [13] (stage ❹). The classifier outputs a likelihood score that the patch contains a character. As a final enhancement, we incorporate a feedback mechanism in which we use high confidence inferences to improve low confidence detections of other patches. The net effect is that we reduce the distractions caused by mutually overlapping characters. Once all segments have been classified, we output our guess for all characters of the codeword. We now discuss the stages of our approach in more detail.

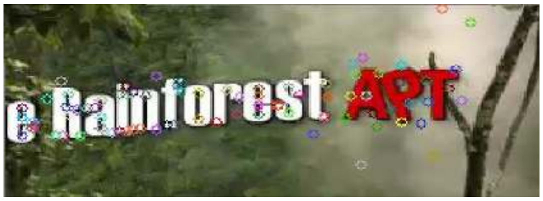


Fig. 4: The circles depict salient features. These salient features are usually corners of an object or texture areas.

Detecting Salient Features and Their Motion (Stage ❶)

A well-known class of salient features in the computer vision community is gray value corners in images. In this paper, we use the Harris corner detector [21] for computing salient features, which uses the image gradient to identify points in the image with two orthogonal gradients of significant magnitude. An example of the detected corners is shown in Figure 4.

After identifying salient features in one frame of the video we now need to identify their respective position in the subsequent frames of the video. To do so, we use an object tracking method that leverages the small motion occurring in between frames. Specifically, we deploy the well known KLT-tracking method [28], which is based on the assumption that the image of a scene object has a constant appearance in the different frames capturing the object (brightness constancy). The positions of a feature in the different frames is called a trajectory. The MIOR captchas by NuCaptcha use constant colors on the characters of the codewords. This implies that the NuCaptcha frames are well suited for our method. Note that no assumption about the specific color is made; only constant appearance of each of the salient features is assumed. We return to this assumption later in Section 4.3.

Motion Trajectory Clustering (Stage ❷)

In a typical video, the detected salient features will be spread throughout the image. In the case of NuCaptcha, the detected features are either on the background, the plain (i.e., non-codeword) characters or the codeword characters. We are foremost interested in obtaining the information of the codeword characters. To identify the codeword characters we use their distinctive motion patterns as their motion is the most irregular motion in the video captcha. In the case of NuCaptcha, we take advantage of the fact that the motion trajectories of the background are significantly less stable (i.e., across consecutive frames) than the trajectories of the features on the characters. Hence we can identify background features by finding motion trajectories covering only a fraction of the sequence. Specifically, we assume presence for less than $l = 20$ frames; we observed little sensitivity with respect to l.

Additionally, given that all characters (plain and codeword) move along a common trajectory, we can further identify this common component by linearly fitting a trajectory to their path. Note that the centers of the rotating codeword characters still move along this trajectory. Accordingly, we use the distinctive rotation of the codeword characters to identify any of their associated patterns by simply searching for the trajectories with the largest deviation from the more common motion trajectory. This identifies the pixels belonging to the codeword characters as well as the plain characters. Additionally, the features on the identified codeword characters allow us to obtain the specific color of the codeword characters without knowing the color a priori (see Figure 5).

Knowing the position of the characters allows us to learn a foreground color model. We use a Gaussian mixture model for foreground learning, which in our case has a single moment (i.e., Gaussian component) corresponding to the foreground color.[2] Given the identified salient features on the background, we also learn a Gaussian mixture for the background, thereby further separating the characters from the background.

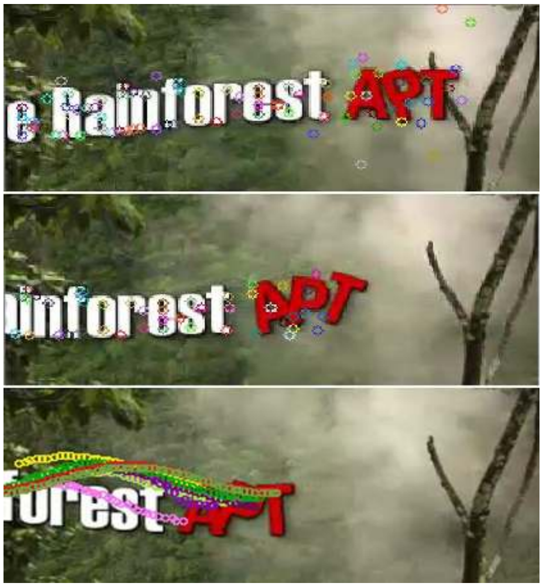


Fig. 5: (Top): salient points in the initial frame. (Middle): salient points with short trajectories in background are discarded. (Lower): trajectories on non-codeword characters are also discarded.

At this point, we have isolated the trajectories of codeword characters, and separated the codewords from the background (see Figure 6). However, to decide which salient features on the codeword characters belong together, additional trajectories are required. To acquire these, we simply relax the constraint on the sharpness of corners we care about (i.e., we lower the threshold for the Harris corner detection algorithm) and rerun the KLT-tracking on the new salient features. This yields significantly more trajectories for use by our segmentation procedure. Notice how dense the salient features are in Figure 7. Note also that since the foreground extraction step provides patches that are not related to the background, we can automatically generate training samples for our classifier, irrespective of the various backgrounds.

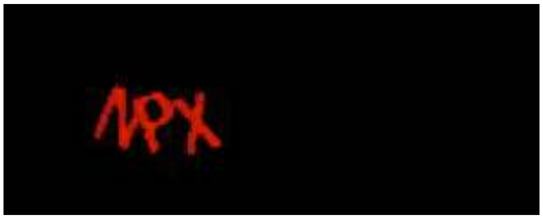


Fig. 6: Example foreground extraction.

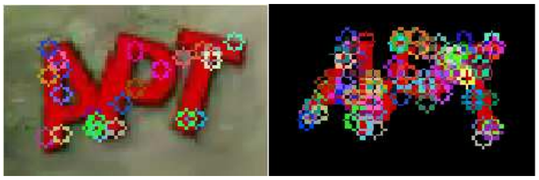


Fig. 7: Rerunning tracking with a lower threshold on corner quality: Left: before modification. Right: after modification.

Segmentation (Stage ❸)

To segment the derived trajectories into groups, we use k-means clustering [23]. We cannot, however, apply the standard k-means approach directly since it relies on Euclidean distances, where each sample is a point. In our case, we need to take the relationship between frames of the video sequence into consideration, and so we must instead use each trajectory as an observation. That is, we cluster the different trajectories. However, this results in a non-Euclidean space because different trajectories have different beginning and ending frames. To measure similarity of trajectories, we leverage the fact that characters are rigid [38] and so the trajectory of the features on the letter should maintain their pairwise distances. Accordingly, we define a distance metric for the trajectories of the feature points that takes into consideration their pairwise spatial distance in every frame, as well as the variation of this pairwise distance, i.e., tracks have a small distance if they are spatially close in all frames and their pairwise distance is close to constant. The output of this stage is shown in Figure 8.

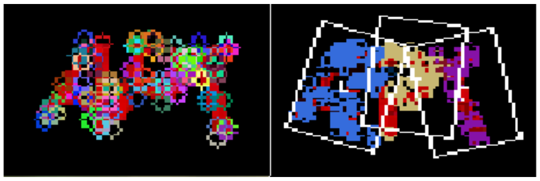


Fig. 8: Left: before segmentation. Right: trajectories are marked with different colors and bounding boxes are calculated based on the center of the trajectories and the orientation of the points. The red areas denote points with no trajectories.

Our initial approach [43] assumed that the codeword length, k, was known in advance. However, there is no reason why the number of symbols in the codeword cannot be randomly chosen at runtime (e.g., NuCaptcha recently suggested using a range of about 3 to 5 random symbols). In this paper, we forego our earlier assumption and instead directly infer the length of the codeword. Our method to do so is simple, yet effective: after extracting the trajectories of salient features, rather than segmenting them into a fixed number of clusters,

[2] In the case where the foreground characters have varying appearance, we use multiple modes.

we simply segment the trajectories into a varying number of clusters (i.e., from 2 to 20 in our implementation).

The intuition is as follows. Recall that in each frame, a segmented cluster consists of a set of salient feature points. Hence, if the codeword is segmented into the correct number of k clusters, the width $w(k)$ of each set of points will be close to the width w_s of a character[3]. Consequently, if the cluster's width $w(k)$ for a specific cluster differs too much from the character width w_s, or if the variation of the widths σ_w^2 (among all k clusters) is too large, we can deduce that the inferred codeword length k is likely incorrect. Additionally, we can leverage the trajectory distance d_t (see §3.2) for points in each of the k clusters to measure correctness. We choose the value for k that minimizes:

$$k = argmin\{\lambda_1|w - w_s| + \lambda_2\sigma_w^2 + \lambda_3 d_t\} \tag{1}$$

where the weights λ_1, λ_2, λ_3 are chosen empirically for the best performance. All our experiments use the same weights.

3.3 Codeword Extraction and Classification (Stage ❹)

Given the center and orientation of each codeword character, the goal is to figure out exactly what that character is. For this task, we extract a fixed-sized area around each character (as in Figure 8), and supply that to our classification stage. Pixels that are close to features of adjacent characters are ambiguous as they could belong to either character. In our implementation, we delete pixels from a cluster if they are closer than 2 pixels to a feature point belonging to an adjacent character.

As mentioned earlier, we use a neural network for classifying the refined patches. A neural network is a mathematical model or computational model that is inspired by the structure of a biological neural network. The training of a neural network is based on the notion of the possibility of learning. Given a specific task to solve, and a class of functions, learning in this context means using a set of observations to find a function which solves the task in some optimal sense. In our application, all twenty letters used by NuCaptcha (namely, '5', '7', 'A', 'C', 'E', 'F', 'G', 'H', 'K', 'M', 'N', 'P', 'R', 'S', 'T', 'V', 'W', 'X', 'Y', 'Z') are considered. A classifier is trained with the refined patches. Each refined patch provides a vector that describes the pattern of the patch. The classifier observes the vectors derived from 50 to 100 patches of each letter and learns a function to distinguish them. After the learning process, the classifier is able to recognize a new patch and assess the probability of correct recognition.

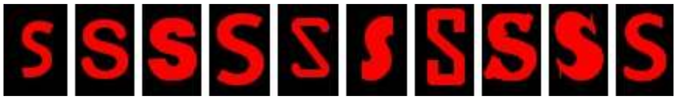


Fig. 9: Different Fonts for Character "S" from left to right: Calibri, Snap ITC, Algerian, Agency FB, Bauhaus 93, OCR A Extended, Euphemia, Arial Black, Arial Rounded MT Bold, Berlin Sans FB.

The use of a Neural network classifier is not only helpful when the codeword characters are of uniform font (e.g., in all of NuCaptcha's MIORs), but also when each character's font is randomly selected from a set of fonts. Indeed, in our implementation, we simply train a classifier on each of 10 different fonts (shown in Figure 9). This yields 10 classifiers, all of which are simultaneously executed on each character and we output the winner as the one with the highest score.

3.4 Feedback Loop Optimization

While the process outlined in stages ❶-❹ works surprisingly well, there are several opportunities for improvement. Perhaps one of the most natural extensions is to utilize a feedback mechanism to boost recognition accuracy. The idea we pursue is based on the observation that an adversary can leverage her confidence about what particular patches represent to improve her overall ability to break the captcha. Specifically, we find and block the character that we are most sure about. The basic idea is that although we may not be able to infer all the characters at once, it is very likely that we can infer some of the characters. By masking the character that we are most confident about, we can simplify the problem into one of decoding a codeword with fewer characters; which is easier to segment and recognize.

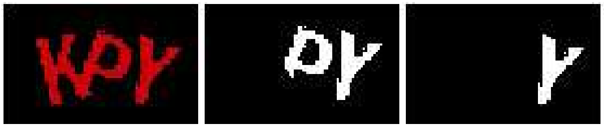


Fig. 10: Iterative decoding of a captcha.

The most confident character can be found using the probability score provided by the classifier, although it is non-trivial to do so without masking out too much of the other characters. We solve this problem as follows. In order to block a character, we try to match it with templates of each character that can be gained by learning. One way to do that is to match scale-invariant feature transforms (SIFT) between the patch and a reference template. While SIFT features can deal with scaling, rotation and translation of characters, there are times when some frames have insufficient SIFT features. Our solution is to find a frame with enough features to apply SIFT, and then warp the template to mask the target character in that frame. Once found, this frame is used as the initial position in an incremental alignment approach based on KLT tracking. Essentially, we combine the benefits of SIFT and KLT to provide a video sequence where the character we are most confident about is omitted. At that point, we rerun our attack, but with one fewer character. This process is repeated until we have no characters left to decode. This process is illustrated in Figure 10.

Note that when the most confident character is removed, part of the remaining letters might be missing, leading to inconsistency with the original training data. To address this, we randomly remove part of each original patch used for training (see Figure 11). The removal simulates the captcha generation process by overlapping randomly chosen and oriented neighboring characters with the trained patch. Any part of the trained patch that is overlapped by the neighboring characters

[3] Parametric values for the width of a character and within-cluster variance can be inferred from training data or the non codeword characters

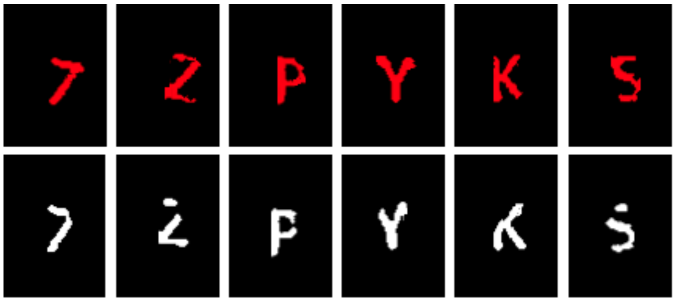


Fig. 11: The training data are modified by randomly removing parts of the patch. The original patches are shown on top with the modified versions below.

is removed from the patch during training. By retraining with the modified training data, we obtain a classifier resistant to missing part of the patches. This new classifier is used for the recognition of the rest of the codeword.

4 Evaluation

We now discuss the results of automated attacks we performed on MIOR captchas. Specifically, the first set of experiments are based on video sequences downloaded off the demo page of NuCaptcha's website. On each visit to the demo page, a captcha with a random 3-character codeword is displayed for 6 seconds before the video loops. The characters of the codeword are chosen from an alphabet of 20 symbols. The displayed captchas were saved locally using a Firefox plugin called `NetVideoHunter`. We downloaded 4500 captchas during November and December of 2011.

The collected videos contain captchas with all 19 backgrounds in use by NuCaptcha as of December 2011. In each of these videos, the backgrounds are of moving scenes (e.g., waves on a beach, kids playing baseball, etc.) and the text in the foreground either moves across the field of view or in-place. We painstakingly labeled each of the videos by hand to obtain the ground truth. We note that while NuCaptcha provides an API for obtaining captchas, we opted not to use it as we did not want to interfere with their service in any way. In addition, our second set of experiments examine several countermeasures and so for ethical reasons we opted to perform such experiments in a controlled manner rather than with any in-the-wild experimentation. These countermeasures are also evaluated in our user study (§5).

4.1 Results

The naïve attack was analyzed on 4000 captchas. The extended attacks (with and without the feedback optimization) were each analyzed on a random sample of 500 captchas. To determine an appropriate training set size, we varied the number of videos as well as the number of extracted frames and examined the recognition rate. The results (not included) show that while accuracy steadily increased with more training videos (e.g., 50 versus 100 videos), we only observed marginal improvement when the number of training patches taken from each video exceeded 1500. In the subsequent analysis, we use 300 video sequences for training (i.e., 900 characters) and for each detected character, we select 2 frames containing that character (yielding 1800 training patches in total). We use dense SIFT descriptors [40] as the features for each patch (i.e., a SIFT descriptor is extracted for each pixel in the patch, and concatenated to form a feature vector). The feature vectors are used to train the neural network. For testing, we choose a *different* set of 200 captchas, almost evenly distributed among the 19 backgrounds. The accuracy of the attacks when we assume in advance that the number of characters in the codeword is set to 3 is shown in Table 1.

Attack Strategy	Single Character Accuracy	3-Character Accuracy
Naïve	68.5% (8216/12000)	36.3% (1450/4000)
Enhanced (*no feedback*)	90.0% (540/600)	75.5% (151/200)
Enhanced (*with feedback*)	90.3% (542/600)	77.0% (154/200)

TABLE 1: Reconstruction accuracy for various attacks. k=3.

We also tested our ability to infer the correct codeword length on videos where the number of characters in the codeword was randomly chosen between 3 and 5. In this case, the feedback loop optimization (§3.4) must be enabled, otherwise the attack performs poorly. Our results show that we are able to correctly estimate the codeword length with 78.4%, 74.8%, and 64.2% for $k = 3, 4, 5$ respectively. This suggest that while randomly adjusting the length of a codeword does offer better resistance against our attack, our success rate in decoding the challenges is still too high to consider this a viable defense strategy.

Overall, our results indicate that the robustness of these MIOR captchas is far weaker than one would hope. In particular, our automated attacks can completely decode the captchas *more than three quarters of the time*. In fact, our success rates are even higher than some of the OCR-based attacks on CR-still captchas [7, 19, 31, 44]. Interestingly, our success rate is close to the lower bound of human success rate (i.e., 75%-90% [32]). We now turn our attention to the runtime performance of the attack.

4.2 Runtime Performance and Economic Analysis

From a practical perspective, the speed of an automated attack on MIOR captchas might be just as important as accuracy. Low success rate and slow speed, for example, may provide clues that an automated solver is being used. Moreover, automated solutions might only be considered valuable if they offer a cheaper alternative than soliciting human labor [16, 32].

	Machine	
Average time (s) per stage	M1	M2
❶: Trajectory Clustering	2.2	2.2
❷: Optical Flow Tracking, @30 fps	11.0	1.2
❸-❹: Segmentation and Classification (3 characters)	0.7	0.7
Feedback Loop Optimization	1.5	1.5
Total Response Time	15.4	5.6
Economic Analysis		
Estimated Retail Cost ($ per 1000 successes)	0.48	0.53
Actual Cost ($ per 1000 successes)	0.42	n/a

TABLE 2: Runtime and solving costs. Machine M1 (from EC2) had 8 CPU cores, and machine M2 (lab) had 1 GPU and 8 CPU cores.

We implemented two versions of our attack. The first is purely CPU-based and uses the OPENCV library for optical flow tracking, the VLFeat library for feature extraction, and the FANN library for the Neural network classification stage. Our second implementation takes advantage of the parallel computing capabilities of graphics processing units (GPUs). In that version, we replace the object tracking (Stage ❷) with a GPU-based KLT tracker [49]. Our CPU-based attack is performed on Amazon's EC2 service where the retail price for CPU computation was $0.66 per hour. Due to incompatibilities on the GPU-supported machines provided by EC2, we only estimate the cost of our attacks if deployed on machines with GPUs (which in 2012 was 3.2 times more expensive, but still at the low cost of only $2.10 per hour with 2 GPUs). The extrapolated processing time is 0.7s for each set of codewords, resulting in approximately 4000 captchas broken per hour.

The results are given in Table 2. Machine M1 (from EC2) had 8 CPU cores, and machine M2 (from our lab) had 1 GPU and 8 CPU cores. In a nutshell, we successfully decoded 1587 samples in one hour on EC2 — i.e., a rate of $0.42 per 1000 captchas solved. Motoyama et al. [32] suggest that in 2007, the cost for a human-assisted attack was roughly $10 per 1000 captchas, and that price has since dropped to around $2 per 1000 captchas. At 42 cents, our solving rate is signficantly cheaper than the going rate for human-assisted captcha solving and more accurate than some OCR-based attacks on traditional CR-still captchas. The response time of our automated attack is also inline with the 10-20s response time for human-assisted solving reported by Motoyama et al. [32].

Lastly, Table 2 clearly shows that the optical flow tracking component is the most time consuming part of the attack. If an adversary is willing to sacrifice accuracy for solving speed, improved response time can be achieved by simply processing fewer frames per second (fps). For example, if we reduce the frame rate from 30 fps to 15 fps, we still achieve a respectable success rate of 60.5%. In this way, the cost can be reduced from 42 cents to 34 cents per 1000 captchas solved.

4.3 Mitigation

To highlight some of the tensions that exists between the security and usability of MIOR captchas, we explore a series of possible mitigations to our attacks. In order to do so, we generate video captchas that closely mimic those from NuCaptcha. In particular, we built a framework for generating videos with characters that move across a background scene with constant velocity in the horizontal direction, and move up and down harmonically. Similar to NuCaptcha, the characters of the codeword also rotates. Our framework is tunable, and all the parameters are set to the defaults calculated from the original videos from NuCaptcha (denoted *Standard*). We refer the interested reader to Appendix A for more details on how we set the parameters. Given this framework, we explore the following defenses:

- *Extended*: the codeword consists of $m > 3$ random characters moving across a dynamic scene.
- *Overlapping*: same as the *Standard* case (i.e., $m = 3$), except characters are more closely overlapped.
- *Semi-Transparent*: identical to the *Standard* case, except that the characters are semi-transparent.
- *Emerging* objects: a different MIOR captcha where the codewords are 3 characters but created using an "Emerging Images" [30] concept (see below).

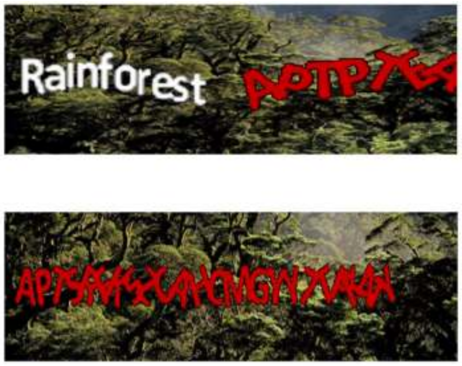


Fig. 12: Extended case. Top: scrolling; bottom: in-place.

Increasing the number of random characters shown in the captcha would be a natural way to mitigate our attack. Hence, the *Extended* characters case is meant to investigate the point at which the success rate of our attacks falls below a predefined threshold. An example is shown in Figure 12. Similarly, we initially thought that increasing the overlap between consecutive characters (i.e., the *Overlapping* defense, Fig. 13) might be a viable alternative. We estimate the degree that two characters overlap by the ratio of the horizontal distance of their centers and their average width. That is, suppose that one character is 20 pixels wide, and the other is 30 pixels wide. If the horizontal distance of their centers is 20 pixels, then their overlap ratio is computed as $20/\frac{20+30}{2} = 0.8$. The smaller this overlap ratio, the more the characters overlap. In both the original captchas from NuCaptcha and our *Standard* case, the overlap ratio is 0.95 for any two adjacent characters.

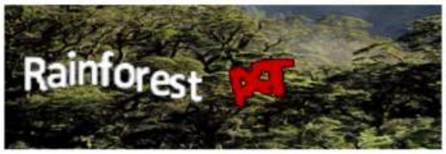


Fig. 13: Overlapping characters (with ratio = 0.49).

The *Semi-Transparent* defense is an attempt to break the assumption that the foreground is of constant color. In this case, foreground extraction (stage ❷) will be difficult. To mimic this defense strategy, we adjust the background-to-foreground pixel ratio. An example is shown in Figure 14.

The final countermeasure is based on the notion of *Emerging Images* proposed by Mitra et al. [30]. Emergence refers to "the unique human ability to aggregate information from seemingly meaningless pieces, and to perceive a whole that is meaningful" [30].[4] The concept has been exploited in Computer Graphics to prevent automated tracking by computers,

[4] See examples at http://graphics.stanford.edu/~niloy/research/emergence/emergence_image_siga_09.html.

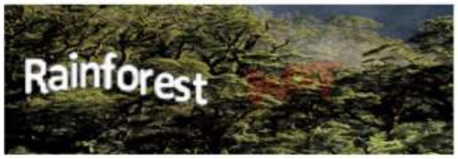


Fig. 14: Semi-transparent: 80% background to 20% foreground pixel ratio. (Best viewed in color.)

while simultaneously allowing for high recognition rates in humans because of our remarkable visual system. We apply the concepts outlined by Mitra et al. [30] to generate captchas that are resilient to our attacks. The key differences between our implementation and the original paper is that our input is 2D characters instead of 3D objects, and we do not have the luxury of incorporating shadow information. Our Emerging captchas are constructed as follows:

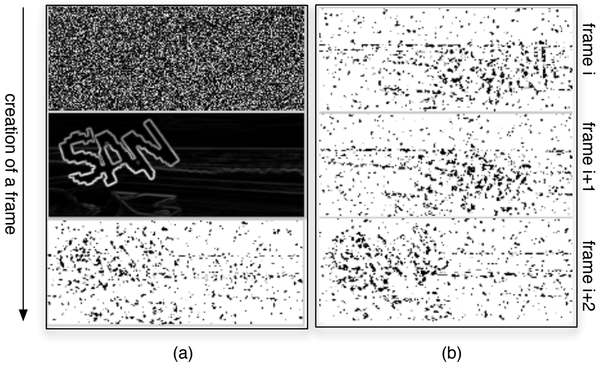


Fig. 15: Emerging captcha. (a) Top: noisy background frame. Middle: derivative of foreground image. Bottom: single frame for an Emerging captcha. (b) Successive frames.

1) We build a noisy frame I_{bg} by creating an image with each pixel following a Gaussian distribution. We blur the image such that the value of each pixel is related to nearby pixels. We also include time correspondence by filtering in the time domain. That is, each frame is a mixture of a new noisy image and the last frame.
2) We generate an image I_{fg} similar to that in NuCaptcha. We then find the edges in the image by calculating the norm of derivatives of the image.
3) We combine I_{bg} and I_{fg} by creating a new image I where each pixel in I is defined as $I(x, y) = I_{bg}(x, y)e^{\frac{I_{fg}}{0.6}}$. In this way, the pixels near the boundary of characters in I are made more noisy than other pixels.
4) We define a constant threshold $t < 0$. All pixel values in I that are larger than t are made white. All the other pixels in I are made black.

The above procedure results in a series of frames where no single frame contains the codeword in a way that is easy to segment. The pixels near the boundaries of the characters are also more likely to be blacker than other pixels, which the human visual system somehow uses to identify the structure from motion. This feat remains challenging for computers since the points near the boundaries change color randomly, making it difficult, if not impossible, to track using existing techniques. An illustration is shown in Figure 15. To the best of our knowledge, we provide the first concrete instantiation of the notion of Emerging Images applied to captchas, as well as a corresponding lab-based usability study (§5).

We refer interested readers to http://www.cs.unc.edu/videocaptcha/ for examples of the mitigation strategies.

4.3.1 Results

We now report on the results of running attacks on captchas employing the aforementioned defenses. Figure 16 depicts the results for the *Extended* defense strategy. In these experiments, we generated 100 random captchas for each $m \in [3, 23]$. Our results clearly show that simply increasing the codeword length is not necessarily a viable defense. In fact, even at 23 characters, our success rate is still 5%, on average.

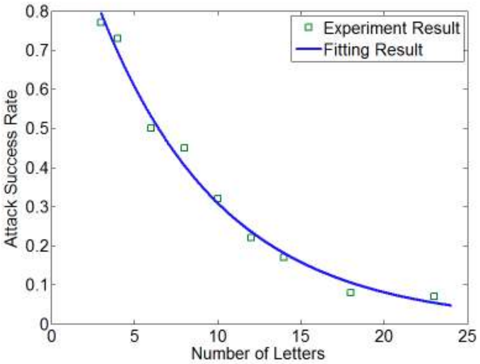


Fig. 16: Attack success as a function of codeword length.

Figure 17 shows the results for the *Overlapping* defense strategy. As before, the results are averaged over 100 sequences per point. The graph shows that the success rate drops steadily as the overlap ratio decreases (denoted as "sensitivity" level in that plot). Interestingly, NuCaptcha mentions that this defense strategy is in fact one of the security features enabled by its behavioral analysis engine. The images provided on their website for the "*very secure*" mode, however, have an overlap ratio of 0.78, which our attacks would still be able to break more than 50% of the time.[5] Our success rate is still relatively high (at 5%) even when the overlap ratio is as low as 0.49. At that point, the second character in the 3-character codeward is 100% overlapped and the first and third are 51% overlapped.

Figure 17 also shows the results for the *Semi-Transparent* experiment. In that case, we varied the transparency of the foreground pixel from 100% down to 20%. Even when the codewords are barely visible (to the human eye), we are still able to break the captchas 5% of the time. An example of one such captcha (with a background to foreground ratio of 80 to 20 percent) was shown earlier in Figure 14.

Lastly, we generated 100 captchas based on our implementation of the Emerging Images concept. It comes as no surprise that the attacks in this paper were not able to decode a single one of these challenges — precisely because these captchas were specifically designed to make optical flow tracking and

[5] See the security features discussed at http://www.nucaptcha.com/features/security-features, 2012.

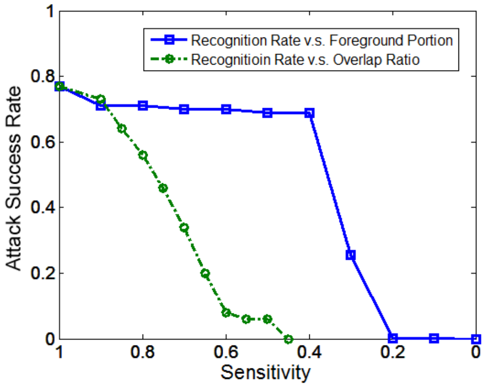


Fig. 17: Attack success rate against *Overlapping* and *Semi-Transparent* defenses. Sensitivity refers to the overlap ratio (circles) or the background-to-foreground ratio (squares).

object segmentation difficult. From a security perspective, these MIOR captchas are more robust than the other defenses we examined. We return to that discussion in §6.

4.3.2 Discussion

The question remains, however, whether for any of the defenses, parameters could be tuned to increase the robustness and still retain *usablility*. We explore precisely that question next. That said, the forthcoming analysis raises interesting questions, especially as it relates to the robustness of captchas. In particular, there is no consensus on the required adversarial effort a captcha should present, or the security threshold in terms of success rate that adversaries should be held below. For example, Chellapilla et al. [8] state: “automated attacks should not be more than 0.01% successful but the human success rate should be at least 90%”. Others argue that “if it is at least as expensive for an attacker to break the challenge by machine than it would be to pay a human to take the captcha, the test can be considered secure” [22]. Zhu et al. [50] use the metric that the bot success rate should not exceed 0.6%.

In the course of our pilot studies, it became clear that if the parameters for the *Extended*, *Overlapping*, and *Semi-Transparent* countermeasures are set too stringently (e.g., to defeat automated attacks 99% of the time), then the resulting MIOR captchas would be exceedingly difficult for humans to solve. Therefore, to better measure the tension between usability and security, we set the parameters for the videos (in §5) to values where our attacks have a 5% success rate, despite that being intolerably high for practical security. Any captcha at this parametrization, which is found to be unusable, is thus entirely unviable.

5 User Study

We now report on an IRB-approved user study with 25 participants that we conducted to assess the usability of the aforementioned countermeasures. If the challenges produced by the countermeasures prove too difficult for both computers and humans to solve, then they are not viable as captcha challenges. We chose a controlled lab study because besides collecting quantitative performance data, it gave us the opportunity to collect participants’ impromptu reactions and comments, and allowed us to interview participants about their experience. This type of information is invaluable in learning *why* certain mitigation strategies are unacceptable or difficult for users and learning which strategies are deemed most acceptable. Additionally, while web-based or Mechanical Turk studies may have allowed us to collect data from more participants, such approaches lack the richness of data available when the experimenter has the opportunity to interact with the participants one-on-one. Mechanical Turk studies have previously been used in captcha research [5] when the goal of the studies are entirely performance-based. However, since we are studying new mitigation strategies, we felt that it was important to gather both qualitative and quantitative data for a more holistic perspective.

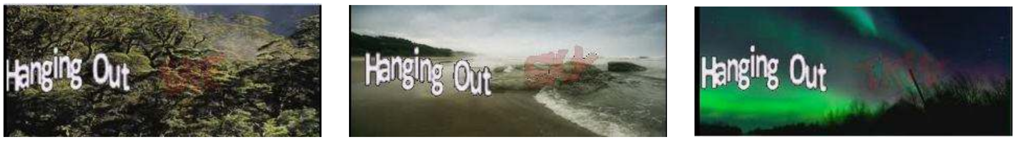
5.1 Methodology

We compared the defenses in §4.3 to a *Standard* approach which mimics NuCaptcha’s design. In these captchas the video contains scrolling text with 2-3 words in white font, followed by 3 random red characters that move along the same trajectory as the white words. Similar to NuCaptcha, the red characters (i.e., the codewords) also independently rotate as they move. For the *Extended* strategy, we set $m = 23$. All 23 characters are continuously visible on the screen. During pilot testing, we also tried a scrolling 23-character variation of the *Extended* scheme. However, this proved extremely difficult for users to solve and they voiced strong dislike (and outrage) for the variation. For the *Overlapping* strategy, we set the ratio to be 0.49. Recall that at this ratio, the middle character is overlapped 100% of the time, and the others are 51% overlapped. For the *Semi-Transparent* strategy, we set the ratio to be 80% background and 20% foreground. For all experiments, we use the same alphabet (of 20 characters) in NuCaptcha’s original videos.

A *challenge* refers to a single captcha puzzle to be solved by the user. Each challenge was displayed on a 6-second video clip that used a canvas of size 300×126 and looped continuously. This is the same specification used in NuCaptcha’s videos. Three different HD video backgrounds (of a forest, a beach, and a sky) were used. Some examples are shown in Figure 18. Sixty challenges were generated for each variation (20 for each background, as applicable).

We also tested the *Emerging* strategy. The three-character codeword was represented by black and white pixel-based noise as described in §4.3. Sixty challenges were generated using the same video parameters as the other conditions.

The twenty-five participants were undergraduate, graduate students, staff and faculty (15 males, 10 females, mean age 26) from a variety of disciplines. A within-subjects experimental design was used, where each participant had a chance to complete a set of 10 captchas for each strategy. The order of presentation for the variations was counterbalanced according to a 5×5 Latin Square to eliminate biases from learning effects; Latin Squares are preferred over random ordering of conditions because randomization could lead to a situation where one condition is favored (e.g., appearing in the last position more frequently than other conditions, giving participants



(a) Forest background (b) Beach background (c) Sky background

Fig. 18: Three backgrounds used for the challenges, shown for the *Semi-Transparent* variant.

more chance to practice). Within each variation, challenges were randomly selected.

A simple web-based user interface was designed where users could enter their response in the textbox and press submit, could request a new challenge, or could access the help file. Indication of correctness was provided when users submitted their responses, and users were randomly shown the next challenge in the set. Immediately after completing the 10 challenges for a variation, users were asked to complete a paper-based questionnaire collecting their perception and opinion of that variation. At the end of the session, a brief interview was conducted to gather any overall comments. Each participant completed their session one-on-one with the experimenter. A session lasted at most 45 minutes and users were compensated $15 for their time.

5.2 Data Collection

The user interface was instrumented to log each user's interactions with the system. For each challenge, the user's textual response, the timing information, and the outcome was recorded. A challenge could result in three possible outcomes: success, error, or skipped. Questionnaire and interview data was also collected.

5.3 Analysis

Our analysis focused on the effects of five different captcha variants on outcomes and solving times. We also analyzed and reviewed questionnaire data representing participant perceptions of the five variants. We used several statistical tests and the within-subjects design of our study impacted our choice of statistical tests; in each case the chosen test accounted for the fact that we had multiple data points from each participant. In all of our tests, we chose $p < 0.05$ as the threshold for determining statistical significance.

One-way repeated-measures ANOVAs [25] were used to evaluate aggregate differences between the means for success rates and times. When the ANOVA revealed a significant difference, we used post-hoc Tukey HSD tests [27] to determine between which pairs the differences occurred. Here, we were interested only in whether the four proposed mitigation strategies differed from the *Standard* variant, so we report only on these four cases.

Our questionnaires used Likert-scale responses to assess agreement with particular statements (1 - Strongly Disagree, 10 - Strongly Agree). To compare this ordinal data, we used the non-parametric Friedman's Test [27]. When overall significant differences were found, we used post-hoc Pairwise Wilcoxon tests with Bonferroni correction to see which of the four proposed variants differed from the *Standard* variant.

Outcomes: Participants were presented with 10 challenges of each variant. Figure 19 shows a stacked bar graph representing the mean number of success, error, and skipped outcomes. To be identified as a *Success*, the user's response had to be entirely correct. An *Error* occurred when the user's response did not match the challenge's solution. A *Skipped* outcome occurred when the participant pressed the "Get A New Challenge" button and was presented with a different challenge. We observe differences in the outcomes, with the *Standard* variant being most successful and the *Semi-Transparent* variant resulting in the most skipped outcomes.

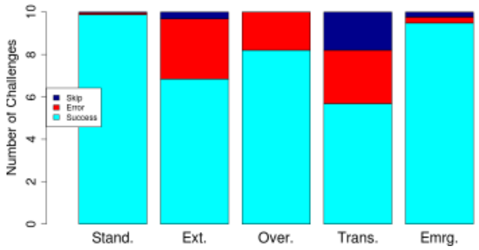


Fig. 19: Mean number of success, error, and skipped outcomes for *Standard, Extended, Overlapping, Semi-Transparent and Emerging* variants, respectively.

For the purposes of our statistical tests, errors and skipped outcomes were grouped since in both cases the user was unable to solve the challenge. Each participant was given a score comprising the number of successful outcomes for each variant (out of 10 challenges).[6]

A one-way repeated-measure ANOVA showed significant differences between the five variants ($F(4, 120) = 29.12, p < 0.001$). We used post-hoc Tukey HSD tests to see whether any of the differences occurred between the *Standard* variant and any of the other four variants. The tests showed a statistically significant difference between all pairs except for the *Standard*⇔*Emerging* pair. This means that the *Extended*, *Overlapping*, and *Semi-Transparent* variants had a significantly lower number of successes than the *Standard* variant, while *Emerging* variant showed no difference.

Time to Solve: The time to solve was measured as the time between when the challenge was displayed to when the response was received. This included the time to type the answer (correctly or incorrectly), as well as the time it took the system to receive the reply (since the challenges were served from our local server, transmission time was negligible). Times for skipped challenges were not included since users made "skip" decisions very quickly and this may unfairly skew the

[6] One participant opted to view only six challenges in each of the *Extended* and *Emerging* variants. We count the remaining four as skips.

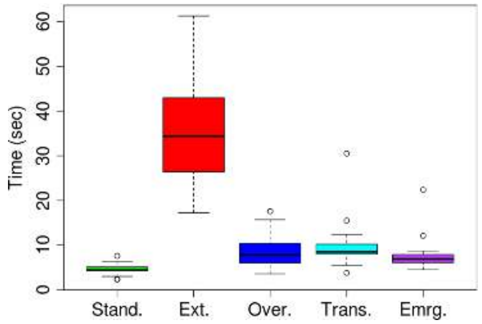


Fig. 20: Time taken to solve the MBOR captchas.

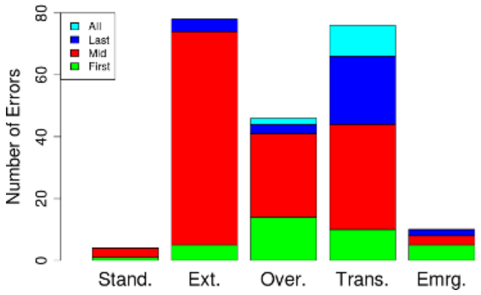


Fig. 21: Location of errors within the codewords.

results towards shorter mean times. We include challenges that resulted in errors because in these cases participants actively tried to solve the challenge. The time distributions are depicted in Figure 20 using boxplots. Notice that the *Extended* variant took considerably longer to solve than the others.

We examined the differences in mean times using a one-way repeated-measure ANOVA. The ANOVA showed overall significant differences between the five variants ($F(4, 120) = 112.95, p < 0.001$). Once again, we compared the *Standard* variant to the others in our post-hoc tests. Tukey HSD tests showed no significant differences between the *Standard*⇔*Emerging* or *Standard*⇔*Overlapping* pairs. However, significant differences were found for the *Standard*⇔*Semi-Transparent* and *Standard*⇔*Extended* pairs. This means that the *Semi-Transparent* and *Extended* variants took significantly longer to solve than the *Standard* variant, but the others showed no differences.

Skipped outcomes: The choice of background appears to have especially impacted the usability of the *Semi-Transparent* variant. Participants most frequently skipped challenges for the *Semi-Transparent* variant and found the Forest background especially difficult to use. Many users would immediately skip any challenge that appeared with the Forest background because the transparent letters were simply too difficult to see. For the *Semi-Transparent* variant, 35% of challenges presented on the Forest background were skipped, compared to 17-18% of challenges using the other two backgrounds. Participants' verbal and written comments confirm that they found the Forest background very difficult, with some users mentioning that they could not even find the letters as they scrolled over some parts of the image.

Errors: Figure 21 shows the distribution of errors. It shows that the majority of errors were made on the middle characters of the challenge. We also examined the types of errors, and found that most were mistakes between characters that have similar appearances. The most commonly confused pairs were: S/5, P/R, E/F, V/N, C/G, and 7/T. About half of the errors for the *Extended* variant were due to confusing pairs of characters, while the other half involved either missing letters or including extra ones. For the other variants, nearly all errors were due to confusing pairs of characters.

User perception: Immediately after completing the set of challenges for each variant, participants completed a Likert-scale questionnaire to collect their opinion and perception of that variant. For each variant, participants were asked to rate their agreement with the following statements:

1) It was easy to accurately solve the challenge
2) The challenges were easy to understand
3) This captcha mechanism was pleasant to use
4) This captcha mechanism is more prone to mistakes than traditional text-based captchas

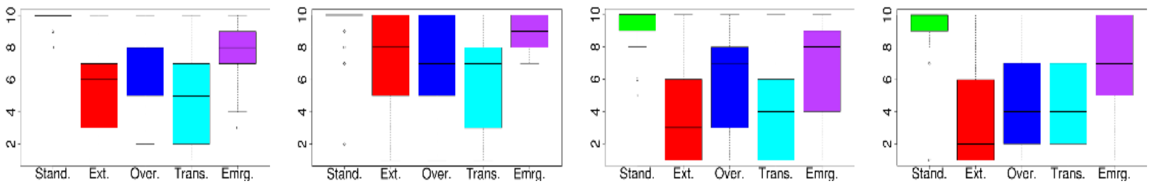
Figure 22 shows boxplots representing users' responses. Since Q.4 was negatively worded, responses were inverted for easier comparisons. In all cases, higher values on the y-axis indicate a more favorable response.

The results show that users clearly preferred the *Standard* variant and rated the others considerably lower on all subjective measures. Friedman's Tests showed overall significant differences for each question ($p < 0.001$). Pairwise Wilcoxon Tests with Bonferroni correction were used to assess differences between the *Standard* variant and each of the other variants. Significant differences were found between each pair compared. The only exceptions are that users felt that the *Extended* and *Emerging* variants were no more difficult to understand (Question 2) than the *Standard* variant. This result appears to contradict the results observed in Figure 22 and we believe that this is because the Wilcoxon test compares ranks rather than means or medians.

Comments: Participants had the opportunity to provide written comments about each variant and offer verbal comments to the experimenter (see examples in Appendix B). Participants clearly preferred the *Standard* variant, and most disliked the *Extended* variant. Of the remaining schemes, the *Emerging* variant seemed most acceptable although it also had its share of negative reactions (e.g., one person found it "hideous").

6 Summary and Concluding Remarks

Our attack inherently leverages the temporal information in the moving-image object recognition (MIOR) captchas, and also exploits the fact that only object recognition of known objects is needed. Our methods also rely on a reasonably consistent appearance or slowly varying appearance over time. That said, they can be applied to any set of known objects or narrowly defined objects under affine transformations that are known to work well with detection methods in computer vision [41]. For the specific case of NuCaptcha, we showed that not only are there inherent weaknesses in the current MIOR captcha design, but that several obvious countermeasures are not viable attack countermeasures. More importantly,



(a) Accuracy (b) Easy to understand (c) Pleasant to use (d) More error-prone (responses inverted)

Fig. 22: Likert-scale responses: 1 is most negative, 10 is most positive.

our work highlights the fact that the choice of underlying hard problem by NuCaptcha's designers was misguided; its particular implementation falls into a solvable subclass of computer vision problems.

In the case of emergent captchas, our attacks fail for two main reasons. First, in each frame there are not enough visual cues that help distinguish the characters from the background. Second, the codewords have no temporally consistent appearance. Combined, these two facts pose significant challenges to existing computer vision methods, which typically assume reasonably consistent appearance and visually distinctive foregrounds [48]. Nevertheless, our user study showed that people had little trouble solving these captchas. This bodes well for emergent captchas—per today's attacks.

Looking towards the future, greater robustness would result if MIOR captchas required automated attacks to perform classification, categorization of classes with large inner class variance, or to identify higher level semantics to understand the presented challenge. Consider, for example, the case where the user is presented with two objects (a person and a truck) at the same scale, and asked to identify which one is larger. To succeed, the automated attack would need to determine the objects (without prior knowledge of what the objects are) and then understand the relationship. Humans can perform this task because of the inherent priors learned in daily life, but this feat remains a daunting problem in computer vision. Therefore, this combination seems to offer the right balance and underscores the ideas put forth by Naor [33] and von Ahn et al. [1]—i.e., it is prudent to employ hard (and useful) underlying AI problems in captchas since it leads to a win-win situation: either the captcha is not broken and there is a way to distinguish between humans and computers, or it is broken and a useful problem is solved.

Acknowledgments

This work is supported in part by funding from the Natural Sciences and Engineering Research Council of Canada and the National Science Foundation (award 1148895).

References

[1] L. V. Ahn, M. Blum, N. J. Hopper, and J. Langford. CAPTCHA: using hard AI problems for security. In *Eurocrypt*, pages 294–311, 2003.

[2] A. Basso and F. Bergadano. Anti-bot strategies based on human interactive proofs. In P. Stavroulakis and M. Stamp, editors, *Handbook of Information and Communication Security*, pages 273–291. Springer, 2010.

[3] E. Bursztein. How we broke the NuCaptcha video scheme and what we proposed to fix it. See http://elie.im/blog/security/how-we-broke-the-nucaptcha\-video-scheme-and-what-we-propose-to-fix-it/, Accessed March, 2012.

[4] E. Bursztein and S. Bethard. DeCAPTCHA: breaking 75% of eBay audio CAPTCHAs. In *Proceedings of the 3rd USENIX Workshop on Offensive Technologies*, 2009.

[5] E. Bursztein, S. Bethard, C. Fabry, J. C. Mitchell, and D. Jurafsky. How good are humans at solving CAPTCHAs? a large scale evaluation. In *IEEE Symposium on Security and Privacy*, pages 399–413, 2010.

[6] E. Bursztein, R. Beauxis, H. Paskov, D. Perito, C. Fabry, and J. C. Mitchell. The failure of noise-based non-continuous audio CAPTCHAs. In *IEEE Symposium on Security and Privacy*, pages 19–31, 2011.

[7] E. Bursztein, M. Martin, and J. Mitchell. Text-based CAPTCHA strengths and weaknesses. In *ACM Conference on Computer and Communications Security*, pages 125–138, 2011.

[8] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski. Designing human friendly human interaction proofs (HIPs). In *ACM Conference on Human Factors in Computing Systems*, pages 711–720, 2005.

[9] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski. Building segmentation based human-friendly human interaction proofs (hips). In *Human Interactive Proofs, Second International Workshop*, pages 1–26, 2005.

[10] J. Cui, W. Zhang, Y. Peng, Y. Liang, B. Xiao, J. Mei, D. Zhang, and X. Wang. A 3-layer Dynamic CAPTCHA Implementation. In *Workshop on Education Technology and Computer Science*, volume 1, pages 23–26, march 2010.

[11] J.-S. Cui, J.-T. Mei, X. Wang, D. Zhang, and W.-Z. Zhang. A CAPTCHA Implementation Based on 3D Animation. In *International Conference on Multimedia Information Networking and Security*, volume 2, pages 179 –182, nov. 2009.

[12] J.-S. Cui, J.-T. Mei, W.-Z. Zhang, X. Wang, and D. Zhang. A CAPTCHA Implementation Based on Moving Objects Recognition Problem. In *International Conference on E-Business and E-Government*, pages 1277–1280, may 2010.

[13] H. Demuth and M. Beale. Neural network toolbox for use with matlab. 1993.

[14] J. J. DiCarlo and D. D. Cox. Untangling invariant object recognition. *Trends in Cognitive Sciences*, 11:333–341, 2007.

[15] J. Driver and G. Baylis. Edge-assignment and figure-ground segmentation in short-term visual matching. *Cognitive Psychology*, 31:248–306, 1996.

[16] M. Egele, L. Bilge, E. Kirda, and C. Kruegel. Captcha smuggling: hijacking web browsing sessions to create captcha farms. In *ACM Symposium on Applied Computing*, pages 1865–1870, 2010.

[17] J. Elson, J. R. Douceur, J. Howell, and J. Saul. Asirra: a CAPTCHA that exploits interest-aligned manual image categorization. In *Proceedings of the ACM Conference on Computer and Communications Security*, pages 366–374, 2007.

[18] N. Friedman and S. J. Russell. Image segmentation in video sequences: A probabilistic approach. *CoRR (technical report)*,

abs/1302.1539, 2013. URL http://arxiv.org/abs/1302.1539.

[19] P. Golle. Machine learning attacks against the Asirra CAPTCHA. In *ACM Conference on Computer and Communications Security*, pages 535–542, 2008.

[20] K. Grill-Spector and N. Kanwisher. Visual recognition: as soon as you know it is there, you know what it is. *Psychological Science*, 16(2):152–160, 2005.

[21] C. Harris and M. Stephens. A combined corner and edge detection. In *Proceedings of The Fourth Alvey Vision Conference*, volume 15, pages 147–151, 1988.

[22] J. M. G. Hidalgo and G. Alvarez. CAPTCHAs: An Artificial Intelligence Application to Web Security. *Advances in Computers*, 83:109–181, 2011.

[23] A. Jain, M. Murty, and P. Flynn. Data clustering: a review. *ACM computing Surveys*, 31(3):264–323, 1999.

[24] K. A. Kluever and R. Zanibbi. Balancing usability and security in a video CAPTCHA. In *Symposium on Usable Privacy and Security*, pages 1–14, 2009.

[25] J. Lazar, J. H. Feng, and H. Hochheiser. *Research Methods in Human-Computer Interaction*. John Wiley and Sons, 2010.

[26] W.-H. Liao and C.-C. Chang. Embedding information within dynamic visual patterns. In *IEEE Multimedia and Expo*, volume 2, pages 895–898, June 2004.

[27] R. Lowry. *Concepts and Applications of Inferential Statistics*. Vassar College, http://www.vassarstats.net/textbook/, 1998.

[28] B. Lucas and T. Kanade. An iterative image registration technique with an application to stereo vision. In *Joint Conf. on Artificial Intelligence*, pages 674–679, 1981.

[29] D. Marr. *Vision: a computational investigation into the human representation and processing of visual information*. W. H. Freeman, San Francisco, 1982.

[30] N. J. Mitra, H.-K. Chu, T.-Y. Lee, L. Wolf, H. Yeshurun, and D. Cohen-Or. Emerging images. *ACM Transactions on Graphics*, 28(5), 2009.

[31] G. Mori and J. Malik. Recognizing objects in adversarial clutter: breaking a visual CAPTCHA. In *Computer Vision and Pattern Recognition*, volume 1, pages 134 –141, June 2003.

[32] M. Motoyama, K. Levchenko, C. Kanich, D. McCoy, G. M. Voelker, and S. Savage. Re: CAPTCHAs-understanding CAPTCHA-solving services in an economic context. In *USENIX Security Symposium*, pages 435–462, 2010.

[33] M. Naor. Verification of a human in the loop or identification via the Turing test, 1996.

[34] NuCaptcha. Whitepaper: NuCaptcha & Traditional Captcha, 2011. http://nucaptcha.com.

[35] M. Shirali-Shahreza and S. Shirali-Shahreza. Motion CAPTCHA. In *Conference on Human System Interactions*, pages 1042–1044, May 2008.

[36] Y. Soupionis and D. Gritzalis. Audio CAPTCHA: Existing solutions assessment and a new implementation for VoIP telephony. *Computers & Security*, 29(5):603–618, 2010.

[37] S. Thorpe, D. Fize, and C. Marlot. Speed of processing in the human visual system. *Nature*, 381(6582):520–522, 1996.

[38] S. Ullman. Computational studies in the interpretation of structure and motion: Summary and extension. In *Human and Machine Vision*. Academic Press, 1983.

[39] S. Ullman. *High-Level Vision: Object Recognition and Visual Cognition*. The MIT Press, July 2000.

[40] A. Vedaldi and B. Fulkerson. VLFeat: An open and portable library of computer vision algorithms. In *Intl. conference on Multimedia*, pages 1469–1472, 2010.

[41] P. A. Viola and M. J. Jones. Rapid object detection using a boosted cascade of simple features. In *Computer Vision and Pattern Recognition*, 2001.

[42] L. von Ahn, M. Blum, and J. Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47:56–60, February 2004.

[43] Y. Xu, G. Reynaga, S. Chiasson, J.-M. Frahm, F. Monrose, and P. Van Oorschot. Security and usability challenges of moving-object captchas: decoding codewords in motion. In *Proceedings of the USENIX Security Symposium*, Aug. 2012.

[44] J. Yan and A. S. E. Ahmad. Breaking visual CAPTCHAs with naive pattern recognition algorithms. In *ACSAC*, pages 279–291, 2007.

[45] J. Yan and A. S. E. Ahmad. A low-cost attack on a Microsoft CAPTCHA. In *ACM Conference on Computer and Communications Security*, pages 543–554, 2008.

[46] J. Yan and A. S. E. Ahmad. Usability of CAPTCHAs or usability issues in CAPTCHA design. In *SOUPS*, pages 44–52, 2008.

[47] J. Yan and A. El Ahmad. CAPTCHA robustness: A security engineering perspective. *Computer*, 44(2):54 –60, feb. 2011.

[48] A. Yilmaz, O. Javed, and M. Shah. Object tracking: A survey. *ACM Comput. Surv.*, 38, December 2006.

[49] C. Zach, D. Gallup, and J. Frahm. Fast gain-adaptive KLT tracking on the GPU. In *Computer Vision and Pattern Recognition Workshops*, pages 1–7, 2008.

[50] B. B. Zhu, J. Yan, Q. Li, C. Yang, J. Liu, N. Xu, M. Yi, and K. Cai. Attacks and design of image recognition CAPTCHAs. In *ACM Conference on Computer and Communications Security*, pages 187–200, 2010.

Appendix A
Parameters for video generation

Similar to NuCaptcha's videos, our videos contain letters that move across a background scene with constant velocity in the horizontal direction, and move up and down harmonically (i.e., $y(t) = A * sin(\omega t + \psi)$, y is the vertical position of the letter, t is the frame id, and A, ω, ψ are adjustable parameters). The horizontal distance between two letters is a function of their average width. If their widths are $width_1, width_2$, the distance between their centers are set to be $\alpha * \frac{width_1 + width_2}{2}$, where α is an adjustable parameter that indicates how much two letters overlap. Our letters also rotate and loop around. The angle θ to which a letter rotates is also decided by a sin function $\theta = \theta_0 * sin(\omega_\theta t + \psi_\theta)$, where $\theta_0, \omega_\theta, \psi_\theta$ are adjustable parameters. For the standard case, we set the parameters the same as in NuCaptcha's videos. We adjust these parameters based on the type of defenses we explore (in Section 4.3).

Appendix B
Comments from User Study

Table 3 highlights some of the free-form responses written on the questionnaire used in our study.

Variant	Comments
Standard	- User friendly - It was too easy - Much easier than traditional captchas
Extended	- My mother would not be able to solve these - Giant Pain in the Butt! Sheer mass of text was overwhelming and I got lost many times
Overlapping	- Letters too bunched – several loops needed to decipher - Takes longer because I had to wait for the letter to move a bit so I can see more of it - Still had a dizzying affect. Not pleasant
Semi-Transparent	- With some backgrounds I almost didn't realize there were red letters - It was almost faded and very time consuming. I think I made more mistakes in this mechanism
Emerging	- I'd feel dizzy after staring at it for more than 1 min - It was hideous! Like an early 2000s website. But it did do the job. It made my eyes feel 'fuzzy' after a while - It was good, better than the challenges with line through letters

TABLE 3: Sample participant comments for each variant