

A Low-Cost Attack against the hCaptcha System

Md Imran Hossen

Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA, USA
Email: md-imran.hossen1@louisiana.edu

Xiali Hei

Center for Advanced Computer Studies
University of Louisiana at Lafayette
Lafayette, LA, USA
Email: xiali.hei@louisiana.edu

Abstract—CAPTCHAs are a defense mechanism to prevent malicious bot programs from abusing websites on the Internet. hCaptcha is a relatively new but emerging image CAPTCHA service. This paper presents an automated system that can break hCaptcha challenges with a high success rate. We evaluate our system against 270 hCaptcha challenges from live websites and demonstrate that it can solve them with 95.93% accuracy while taking only 18.76 seconds on average to crack a challenge. We run our attack from a docker instance with only 2GB memory (RAM), 3 CPUs, and no GPU devices, demonstrating that it requires minimal resources to launch a successful large-scale attack against the hCaptcha system.

I. INTRODUCTION

CAPTCHAs (*Completely Automated Public Turing Tests to Tell Computers and Humans Apart*) are computer-generated and graded tests that most humans can easily pass, but current computer programs such as Artificial Intelligence (AI) algorithms, cannot pass [42]. CAPTCHAs protect websites from malicious bots and other forms of automated abuse. As a result, the security of CAPTCHAs is critical to defending the Internet against automated attacks.

For years, text CAPTCHAs that ask users to recognize distorted texts from the background of an image have been subjected to automated attacks [9], [11], [14], [16], [17], [27], [28], [44], [44], [45], [47]. Successful attacks against text CAPTCHAs underscore that they are no longer secure against current Machine learning (ML) technologies. As a result, text CAPTCHAs have been primarily replaced by image CAPTCHAs. To some extent, image CAPTCHA schemes are considered more robust to automated attacks than their text counterparts. The rationale behind this is that there are still many hard and open problems in the image recognition domain. However, deep learning (DL) algorithms have recently surpassed humans' cognitive ability in a complex visual recognition task [20], putting the security of image CAPTCHAs in question. Researchers have successfully broken several popular real-world image CAPTCHA schemes exploiting DL technologies [22], [39], [43].

hCaptcha is a relatively new but emerging image CAPTCHA service developed by *Intuition Machines, Inc.* It asks users to select images matching a category/label

provided in the challenge instruction to verify that they are humans and not bots. It is becoming increasingly popular on the Internet as an anti-bot solution. On April 8, 2020, Cloudflare announced that they were ditching Google's reCAPTCHA and adopting hCaptcha on their platforms due to privacy concerns and costs of using reCAPTCHA [32].

Unfortunately, to the best of our knowledge, the security of hCaptcha and its ability to resist automated abuses have not yet been formally evaluated. In this paper, we design and develop an end-to-end system to attack the image hCaptcha system. Our attack is highly effective and efficient: it can break hCaptcha challenges with more than 95% accuracy while taking less than 19 seconds on average to crack a challenge. Most importantly, we show that even a resource-constrained adversary can mount a powerful attack using our system. Our attack reinforces the vulnerability of CAPTCHA designs relying on simple image classification tasks as the underlying AI problem to distinguish between humans and bots.

In summary, we make the following contributions:

- We design and develop a low-cost, end-to-end system to break hCaptcha service.
- We evaluate our system against 270 live hCaptcha challenges and achieve the success rate of attack over 95% with the system taking less than 19 seconds to crack a challenge on average.
- We provide a preliminary security analysis of the hCaptcha system. Our analysis shows that the hCaptcha service employs minimal to no mechanism to resist automated abuses other than asking users to solve a simple image recognition task.

II. HCAPTCHA BACKGROUND

hCaptcha is an image CAPTCHA scheme developed by *Intuition Machines, Inc.* It is intended to be a drop-in replacement for Google's popular CAPTCHA service reCAPTCHA [37]. hCaptcha is mainly designed for using human labor to label machine learning datasets for different companies. Unlike reCAPTCHA, hCaptcha pays the publishers (the website owners hosting hCaptcha service) for every visual challenge successfully solved by website visitors. The hCaptcha marketplace runs on the *HUMAN Protocol*

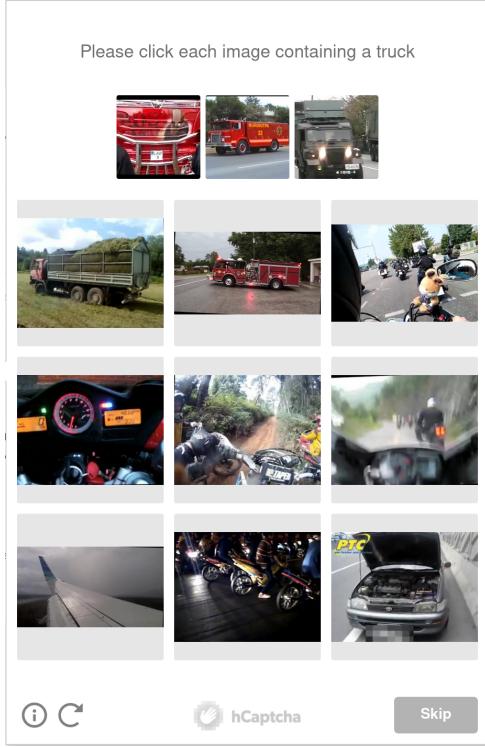


Figure 1: A hCaptcha challenge widget.

[3], which aims to enable a new generation of machine intelligence to apply human labor to AI model advancement to achieve human parity in task performance. Websites using hCaptcha earn *Human Tokens* (HMT) whenever users use the hCaptcha widget on the sites. In recent years, hCaptcha has become increasingly popular among publishers, and according to a 2019 report, 10 million people interact with hCaptcha every month on thousands of websites [36].

Websites usually use CAPTCHAs to prevent automated account creation and abuses from malicious bot programs. As such, CAPTCHAs are generally embedded in registration/login forms. When visitors land into a hCaptcha protected webpage, they will need to click on the hCaptcha checkbox “*I am Human*” to initiate the challenge. After that, they will be prompted with the challenge widget where the actual CAPTCHA test is located. The users are required to select all images matching a description (see Figure 1) to pass a hCaptcha challenge. It is worth noting that the system also provides an “*invisible*” mode where users will be automatically prompted with a challenge only when they lack enough trust with the hCaptcha system to prove their humanness.

III. THREAT MODEL

A CAPTCHA scheme is considered broken if a bot can break the CAPTCHA challenges with a success rate higher than 0.01% [13]. Designing a CAPTCHA service with such

constraints is very challenging in practice. Elson *et al.* [15] relaxed the tolerable success rate of attack up to 0.6%. The effectiveness of the attack also depends on the cost of the attack. A powerful adversary who possesses many resources can afford such low success rates and can scale the abuse’s impact by attacking the given CAPTCHA system hundreds of thousands of times.

Diverting from the above threat model and closely following the threat model in [8], our threat model involves an attacker with limited resources. We will assume the attacker is limited to one computer with a small-size RAM and one IP address. Since such an attacker cannot afford to have a lower success rate, we aim for an accuracy benchmark above 50%.

IV. SYSTEM OVERVIEW

Our automated CAPTCHA breaking system solves a hCaptcha image challenge in three main steps: 1) Obtaining the challenge, 2) Solving the challenge, and 3) Submitting and verifying the solution. Step 1 and step 3 are browser-specific tasks and automated by controlling a web browser using browser automation software. For step 2, we use an image classifier to classify candidate images in the challenge to find potential target images for the solution. We now discuss in detail the implementation of each step.

V. IMPLEMENTATION DETAILS

A. Obtaining the hCaptcha Challenge

Websites usually embed the hCaptcha widget on the webpages that need protection from bots, spam, and other forms of automated abuse, such as the login/registration forms. One can generally locate the hCaptcha container by its class name `h-captcha` with a `data-sitekey` attribute set to the public key, a unique key provided by hCaptcha for each registered page. Our system locates `.h-captcha` container inside the HTML form. hCaptcha checkbox widget is rendered as an `iframe` on the webpage. The system switches to the frame and clicks on the “*I am Human*” checkbox identified by `#checkbox`. After that, a new `iframe` element containing the hCaptcha challenge widget pops up. Our system then switches to the challenge widget.

The challenge widget contains the actual image challenge that the users must solve to pass the hCaptcha’s anti-bot test. Our system first locates the challenge instruction `.prompt-text`. The instruction includes the name of the image category/label that users need to select from a set of candidate images/payments. We can locate the payloads by their identifier `.task-image`. The source URLs of the images are generated dynamically and can be accessed only for a few seconds. After that period, the URLs expire. Our bot fetches the payloads and stores them on a predefined location of our computer.

B. Solving the Challenge

We used a deep neural network-based image classifier to classify candidate images in a hCaptcha challenge. Our system provides each image an ID. The images obtained in the previous step and their unique identifiers are sent to the classifier. The classifier returns a label for each image. We process the classifier outputs to filter out the images that do not match the target image category provided in the challenge instruction. For the remaining images, our system keeps the image IDs as a potential solution. We now briefly provide details of our image classifier network.

Network Architecture. Our image classifier network follows the Residual Network (ResNet) [21] architecture. ResNet allows building deeper neural networks by utilizing skip connections or shortcuts to jump over some layers. It solves the vanishing gradient issue with traditional deep neural nets, trains faster, and has been proven to produce state-of-the-art performances in several complex vision tasks. While it is possible to build very deep residual neural networks involving as many as 152 layers, we opted to use only 18 layers residual network (ResNet-18) for our task. We made this choice because we want to run the model on a machine that does not necessarily include a GPU for faster computation. Further, training a deeper network such as ResNet-101 takes more time, and running the inference on CPU takes longer. We found that the ResNet-18 model provides decent performance sufficient for our task.

In our work, we used a ResNet-18 model pretrained on the ImageNet [34] dataset and finetune it for our task because training the entire network from scratch requires a vast amount of training samples, often a time-consuming and computationally expensive task. We found that hCaptcha challenges show images from only nine classes. We reset the final fully connected (FC) layer of the ResNet-18 such that the size of each output sample is set to nine.

Data Collection. With more training samples, deep neural networks learn to extract a better representation of underlying data distribution. At the same time, manually collecting and labeling data is a labor-intensive process. We collected 5000 challenges from 3 hCaptcha protected websites from the period of May 2020 to July 2020. Figure 2 shows the frequencies of different image categories. As shown in Figure 2, we observed only nine image categories (bus and motorbus are considered the same category) frequently appear on hCaptcha challenges. Interesting, data from all of these categories are already available on the OpenImages [25] dataset, a publicly accessible dataset for training machine learning models on various image recognition tasks. Therefore, instead of manually labeling original hCaptcha images, we extracted 45000 images from the nine categories to train our ResNet-18 network. We tried to keep the dataset balanced, but some categories have more training samples than others, making the dataset slightly skewed.

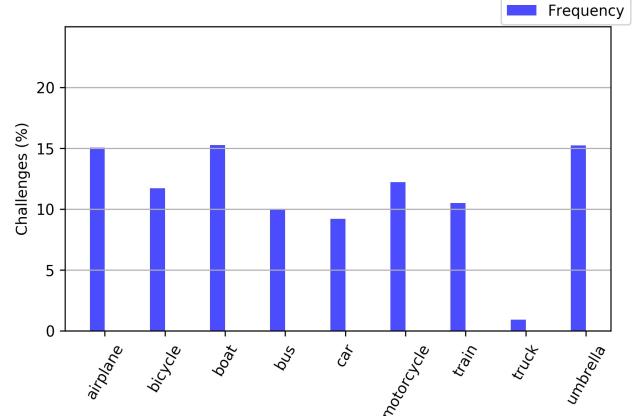


Figure 2: The frequency of each image category appears in collected challenges.

Training ResNet-18. We split the dataset into three sets: training, validation, and testing sets. The training set is the data that the network will learn from. The validation set is used for fine-tuning the model’s hyperparameters. The testing set is used for assessing the model’s generality on unseen data. We used the following hyper-parameters for training our network: the batch size of 32, categorical cross-entropy as the loss function, Adam optimizer with a learning rate of 0.0001. We trained the model for 40 epochs. The model achieved an accuracy of over 88% on the testing set. The training was performed on a machine running Arch Linux OS with an NVIDIA GeForce RTX 2070 GPU and took about 143 minutes to complete. Note that training the network is mainly a one-time task. However, we may need to retrain the model once in a while if new image categories appear in the challenges.

C. Submitting and Verifying the Solution

Once the image classifier determines the potential images matching the target label, our system locates corresponding payloads `task-image` in the challenge widget and performs a mouse click on each of them. Next, it submits the solution by clicking on the submit button, which is identified by `.button-submit`. Once our bot submits the solution, we need to verify whether the challenge is passed or not. The status of challenge can be monitored via `aria-label` attribute of `#checkbox` in hCaptcha widget. The `aria-label` attribute contains the text string — “You are verified” — when a challenge is successfully solved; otherwise, hCaptcha triggers an error message to indicate a failure. The system identifies error messages via their class name `display-error`.

Further, hCaptcha provides a mechanism for verifying a challenge from the server-side of web applications using the hCaptcha widget. When the user successfully solves

a CAPTCHA, the hCaptcha script inserts a unique token, `h-captcha-response`, into the HTML form data. The server-side needs to check whether the token is valid at the API endpoint URL provided by hCaptcha. The request to the endpoint expects two parameters: hCaptcha secret API key associated with the website and the `h-captcha-response` token POSTed from the HTML page. Upon receiving the request, the endpoint returns a JSON response. If the token is valid, the “Success” field in the response is set to “True”; otherwise “False”.

VI. IMPLEMENTATION AND EVALUATION PLATFORM

For performing browser-specific tasks, such as visiting and interacting with hCaptcha protected websites, initiating and submitting the hCaptcha challenge, our bot utilizes the puppeteer [5] web automation software. Google develops the puppeteer web automation framework to control the Chrome web browser programmatically. However, we used puppeteer-firefox [6] with the Firefox web browser because it is easier to customize Firefox. Specifically, we used puppeteer-firefox 0.5.1, with Firefox 65.0. Our image classifier network, ResNet-18, was built on top of PyTorch 1.7.0.

We ran all of our experiments inside a docker container running the Ubuntu 20.04 image. To simulate a low-resource attack, we configured the container such that the maximum amount of memory (RAM) it could access from the host machine is 2GB. We also set the number of CPU cores the container could use to 3. The host machine on which we ran the container has 8 Intel core i7-8550U (1.8GHz) processors, 16 GB of RAM running Arch Linux OS.

Experimental Setting. We used three websites for our experiments: www.hcaptcha.com, 2captcha.com, and one of our own websites, respectively. We did not affect the security/availability of the tested websites by limiting our bot’s interactions only to the hCaptcha related components on the hCaptcha protected webpages. Further, we did not send excessive requests within a short time window to prevent DoSing the sites.

We accessed the websites from a regular and non-academic IP address unless otherwise specified. We launched our system with a fresh browser profile during each visit, *i.e.*, no caches or cookies were retained from prior requests. We also did not attempt to change our browser environment’s configuration, *i.e.*, using custom User-Agent header, changing screen resolution, *etc.*

VII. ATTACK EVALUATION

Accuracy and Speed of Attack. We submitted 270 challenges using our automated system, and it successfully solved 259 of them, resulting in an accuracy or success rate of attack of 95.93%. It takes 18.76 seconds on average to crack a challenge. Figure 3 depicts the breakdown of our system’s attack speed by individual modules. Automating

browsing-related activities (*e.g.*, initiating the challenge, interacting with checkbox and challenge widget, and submitting and verifying challenge) takes more time. Our deep learning classifier (the Solver) takes 3.79 seconds to classify the images (usually 9) in a challenge, on average. Note that one can further speed up this process by running the inference on a GPU-enabled machine; however, our attack focuses on a low-cost attack, and we show that one can mount a highly accurate attack under minimal resource constraints using our system.

We came across only 9 image categories in 270 submitted challenges. Figure 4 shows the frequency and success rate of attack for these image categories. Figure 5 shows the probability distribution of the number of image selections per challenge in all submitted challenges. The majority of the challenges have 2 to 5 images as correct solutions. We also noticed some challenges having as many as 14 images as part of the correct image selections.

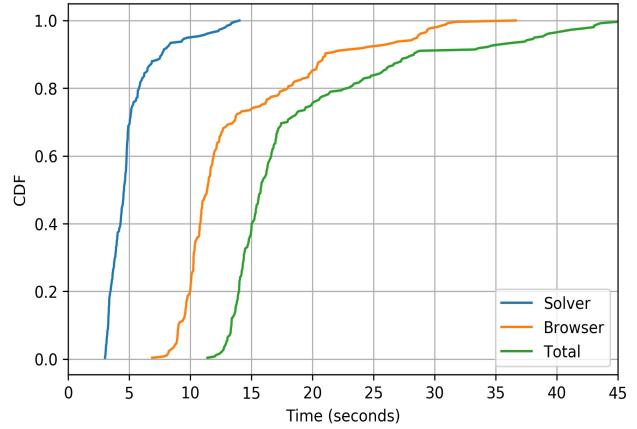


Figure 3: Cumulative distribution of time required by each module.

Solution flexibility. We observed hCaptcha often accepts one or two incorrect image selection(s) while solving a hCaptcha challenge. We manually solved some hCaptcha challenges by choosing different combinations of correct and wrong image selections to test whether hCaptcha provides any solution flexibility. Table I lists the results of our experiment. For single-prompt CAPTCHAs, users must solve only a single image challenge to pass a test. Double-prompt CAPTCHAs require that two image challenges must be solved subsequently to pass a test. From Table I, one can see that, to some extent, hCaptcha is flexible while determining a correct solution to a challenge.

Influence of IP address. To see whether clients’ IP address type affects the attacker’s success rate, we submitted 200 challenges separately from three IP addresses. The three IP addresses are an academic IP, a VPN IP, and a Tor

Table I: Results of solution flexibility: Combinations for passing the hCaptcha. Here, n = number of correct image selections, and k = number of wrong image selections.

Image selection	Constraint	Pass (%)
n Correct + k Wrong (Single-prompt)	$k \leq 1$	73.50
n Correct + k Wrong (Double-prompt)	$k \leq 1$	24.50
($n-1$) Correct (Single-prompt)	$n \geq 3$	71.50
($n-1$) Correct (Double-prompt)	$n \geq 3$	61.50
($n-1$) Correct + k Wrong (Single-prompt)	$k > 0$	20.00
($n-1$) Correct + k Wrong (Double-prompt)	$k > 0$	30.50

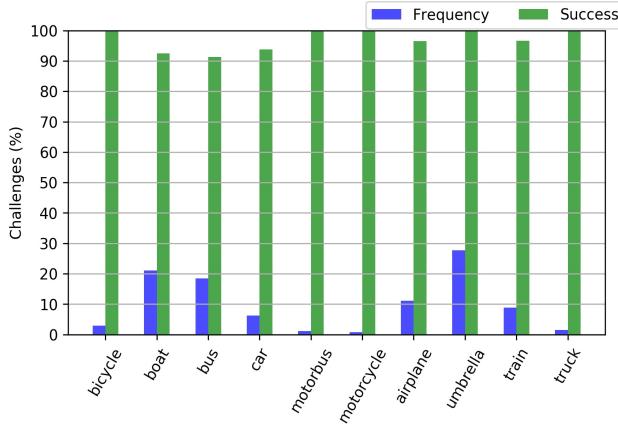


Figure 4: The accuracy and frequency of each image category in the solved challenges.

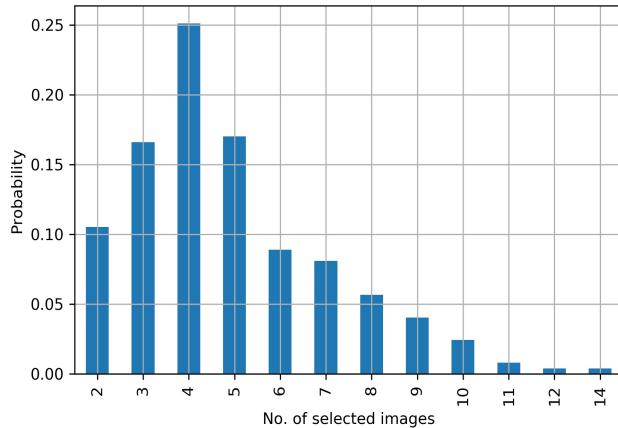


Figure 5: The probability distribution of no. of images selected per challenge.

network IP. One might expect an IP address belonging to an academic network might seem less malicious than an IP address that belongs to a Tor exit node. While accessing the hCaptcha-protected webpage, we sent all 200 requests from a single IP address in a row with a 30-second gap between two subsequent requests. Interestingly, our system achieved a success rate of over 90% for all three IP addresses.

This suggests that hCaptcha generally does not discriminate against users' IP address types.

Adaptability. We also tested whether hCaptcha adapts the challenge difficulty according to the suspiciousness level of the users. For instance, a client who accesses a hCaptcha enabled webpage using web automation software is more likely to be a bot than someone accessing the page from a regular browser. In such a scenario, an adaptive CAPTCHA service would escalate the threat level for malicious clients by asking them to solve more complex challenges or solve multiple challenges before accessing the online service. Hossen *et al.* [22] showed that Google's image reCAPTCHA system adopts such policies to limit the malicious bot program's abuse. We tested if hCaptcha has such measures in place by running several experiments.

Our system tried mimicking a regular user browser by overriding several Navigator JavaScript properties in our main experiment. For instance, we set the `navigator.webdriver` to "False", which is usually set to "True" while using a web automation software, `navigator.plugins` to a random number, and screen resolution to the size of a regular desktop computer, etc. To test hCaptcha's adaptability, we set up an experiment to control the web browser from the automation software with all default settings. We also ran the browser in headless mode while solving the challenges. The rationale for doing this is that setting the browser to headless mode sends a clear signal that the client is using web automation software. This also signals that the request is likely to be generated from an automated bot program. We solved 100 challenges with this setting. Furthermore, we submitted the same number of challenges using Selenium [7] WebDriver for Firefox as well. Selenium is the most popular web automation software. We analyzed the results for each experimental setting to identify any discrepancies among these different settings. However, we did not notice any distinct pattern that can distinguish the settings. For example, we came across the same nine image categories, achieved similar accuracy (over 90%) in all experimental settings. Further, none of the requests were blocked in any of the experimental settings. Our analysis indicates that hCaptcha solely relies on correct image selections to verify a solution without adapting challenges based on users' threat levels.

Blocking. One of the design goals of the hCaptcha system is not to leak detections in real-time. To ensure we did not miss any server-side blocking during our main experiment, we deployed hCaptcha on our website. We created a demo web application that allows a client to register for an account with a user name and password. The registration form is protected by hCaptcha. The web application backend processes the form data only when the client solves a valid CAPTCHA test. Once the user submits the form, the server-side code sends the user response token `h-captcha-response` POSTed with the form to the hCaptcha backend for verification. If the authentication succeeds, the form is processed. Otherwise, we show a warning to the clients that they failed the robot verification test and their inputs could not be processed.

The hCaptcha deployment console allows website owners to adjust the *difficulty level*¹ of served CAPTCHA tests for the clients accessing the site. It has four difficulty levels: *easy*, *moderate*, *difficult*, and *always on*. By default, the CAPTCHA difficulty level is set to *moderate*. We ran our system against our web application and attempted to register for 400 fake accounts by automatically filling out the form and solving the challenges. We were able to create 369 such accounts. That means our bot could crack 369 hCaptcha challenges automatically, resulting in an attack success rate of 92.25%. Next, we increase the difficulty level to *difficult* and tried to create the same number of fake accounts as before. This time, we were able to register for 354 accounts.

Note that all the requests to our web application were sent in a row with only a 1-second delay between two subsequent requests. Further, we followed the same experimental setting mentioned in Section VI during this experiment. During our experiment, only 17 of our attempts (out of the total 800 combined) were blocked by hCaptcha with the message — “Rate limited or network error. Please retry.” Besides rate-limiting the bot to a particular session, we did not observe any strict blocking policy by hCaptcha to prevent a malicious client from accessing the service for a specific period.

Next, we attempted to trigger blocking deliberately by sending too many requests simultaneously. We launched 50 instances of our bot program concurrently ten times with a 2-second delay between two subsequent iterations against our hCaptcha-enabled webpage. We noticed the hCaptcha system blocked many of our requests with the warning message — “Your computer or network has sent too many requests.” Specifically, the number of blockages for the ten iterations are 24, 40, 48, 29, 28, 26, 26, 29, 30, and 28, respectively.

¹<https://medium.com/@hCaptcha/how-hcaptcha-difficulty-settings-work-13d84279d378>

Image Repetition. We found that hCaptcha often repeats images across different challenges. We computed the MD5 hashes of 48330 images collected from the hCaptcha challenges during our analysis and identified 9854 redundant images belonging to 1985 sets of identical images. Cryptographic hash functions such as MD5 may not provide an accurate number of repeated images since the slightest modification in the input will produce a drastic change in the output. As such, we used the *perceptual image hash* (pHash) [23] algorithm to find similar or completely identical images in the submitted challenges. Interestingly, we found the same 1985 set of images in our pHash analysis as well. That means while repeating the same image across multiple challenges, hCaptcha makes no attempt to modify the image and gives exact copies of it.

User Data Collection. Besides asking users to prove their humanness by passing a CAPTCHA challenge, hCaptcha collects information about users’ browsers and their devices to assess their susceptibility to being bots. We analyzed the hCaptcha client-side JavaScript library responsible for rendering the challenge on the users’ browsers. We found that it checks the following data: browser family (*e.g.*, Chrome, Firefox, Internet Explorer) along with version numbers, Operating System family (*e.g.*, Windows, Linux, Mac OS), OS type (*e.g.*, desktop, mobile). From the web browser, hCaptcha probes the device’s screen resolution, the number of plugins installed, mime types, whether it supports canvas and Web Assembly, and whether the device supports touch. In addition to these, hCaptcha also uses dynamic information such as touch events, keypress events, scroll positions, etc.

A. Online Attacks

We also performed an online attack using three state-of-the-art online vision API services for image recognition. These services are Google Cloud Vision API [2], Amazon Rekognition [1], and Microsoft Azure Cognitive Vision API [4]. First, we submitted several hundred hCaptcha challenge images from different categories to these services separately and analyzed the classification results. Note that the vision APIs can recognize multiple objects in an image, thus returning multiple labels along with the confidence scores (see Table II). We found that the labels’ names are mostly compatible with hCaptcha image classes by manually analyzing the label sets. As a result, we could simply map a label set for an image returned by a vision API directly to the original hCaptcha challenge image class.

We developed a proof-of-concept system by replacing our solver module with a particular vision API service. The system works as follows. First, it visits a predefined webpage using hCaptcha. Second, the system initiates the challenge and downloads the images. Third, we send the images to the API service for recognition. Fourth, the system analyzes the label set for each image. If one of the tags in a set matches the hCaptcha target image category, it marks it as

Table II: List of labels returned by three image recognition APIs for a sample image from hCaptcha challenge.

Image	Google Cloud Vision	Microsoft Computer Vision	Amazon Rekognition
	Land vehicle, Vehicle, Transport Truck, Car, Mode of transport, Motor vehicle, Trailer truck, Trailer, Asphalt	outdoor, truck, road, transport, street, parked, trailer, car, large, lot, parking, front, sitting, driving, side, bed, city, bus, fire, man	Truck, Transportation, Vehicle, Tow Truck, Person, Human, Trailer Truck

Table III: Attack performance of off-the-shelf vision APIs.

Vision API	Accuracy (%)	Speed (s)
Amazon Rekognition	92	16.85
Microsoft Computer Vision	98	14.93
Google Cloud Vision	96	15.28

a potential solution and saves the image ID. Finally, our system clicks on potential target images in the hCaptcha challenge widget and clicks on the “Submit” button. We then verify whether the challenge has been passed or not using the method described in Section V.

We submitted 100 live challenges using each API service. Table III depicts the accuracy and speed of our online attack. All these services achieved an accuracy or success rate of attack over 90%, with Microsoft Azure Cognitive Vision API having the highest success rate (98%). In summary, our original system provides comparable performance to the online vision API-based attack. It is also more cost-effective since those vision services usually incur charges for each API request.

VIII. COUNTERMEASURES

We discuss potential measures to counter our attack, their limitations, and potential impacts in this section.

Use Broader Image Categories. hCaptcha uses only a small number of image categories, making it trivial for an attacker to collect a sufficient amount of data to train a highly accurate image classifier. Expanding the image categories will make this process relatively challenging. Note that doing so does not necessarily prevent the attack, hence provides a temporary solution only.

Adversarial Examples. Machine learning models, including deep neural networks, have been shown to be vulnerable to adversarial examples [19], [30], [40], specifically crafted inputs that can trick the models into making wrong predictions. Recent work [29], [38] has already demonstrated the efficacy of using adversarial examples in image CAPTCHA designs. Designers can take advantage of this vulnerability by injecting adversarial perturbations in the CAPTCHA challenge images to dupe deep learning-based classifiers, thus lowering the attack accuracy.

Resist Web Automation Software. Since most bots rely on the web automation software to launch automated attacks, fingerprinting and resisting requests originating from widely used web automation frameworks will likely lower attackers’ success rates.

Adaptability. Adapting the challenge based on users’ suspiciousness levels and presenting complex challenges to highly suspicious clients and easy ones to users most likely to be humans will discourage malicious bots while providing easy passes to legitimate humans. However, determining the suspiciousness and scoring the requests based on that might require extensive experiments.

Commonsense Knowledge. When facing a task that involves higher-order reasoning, machines do not usually perform well. Designers can exploit this weakness by forming the instruction that requires some common sense knowledge to decode what image category needs to be selected to pass a challenge, making the underlying AI problem harder for computers. This, however, may negatively impact the overall usability of the CAPTCHA scheme for humans; therefore, it requires further research to determine whether such a design is practical in the real world.

IX. RELATED WORK

Image CAPTCHAs. The Asirra CAPTCHA [15], proposed in 2007, relied on the presumed difficulty of automatically distinguishing images of dogs and cats. However, in 2008, Golle *et al.* [18] developed a machine learning classifier trained on color and texture features, automatically solving Asirra CAPTCHA challenges with a probability of 10.3%. The ARTiFACIAL CAPTCHA scheme proposed by Rui *et al.* [33] requires users to identify faces and facial features within a heavily distorted image. Zhu *et al.* [48] demonstrated successful attacks against a series of earlier image CAPTCHA schemes, including ARTiFACIAL. The authors also recommended several guidelines for designing robust image CAPTCHA schemes based on the insights gathered from their attacks. Yardi *et al.* [46] proposed photo-based authentication for social networks where a user is required to identify subjects who are uniquely known to him/her to pass the CAPTCHA test. However, Polakis *et al.* [31] showed that the photo-based authentication system could

be automatically solved by leveraging publicly available data and face recognition algorithms. Sivakorn *et al.* [39] demonstrated an attack against the earlier implementation of image reCAPTCHA v2 service by leveraging online image annotation services. While that version of reCAPTCHA v2 is no longer in use and the current version is likely to be immune to such attacks, their attack revealed some interesting insights into reCAPTCHA’s advanced risk analysis engine, which determines users’ likelihood of being bots using several signals, including users’ browser environment. In 2019, Weng *et al.* [43] demonstrated a series of deep learning-based attacks against different real-world image CAPTCHA services and found them highly vulnerable automated attacks. More recently, Hossen *et al.* [22] proposed an object detection-based solver that was able to break the latest version of reCAPTCHA v2 challenges with a success rate of over 83%. Their attack also demonstrated that anti-recognition techniques such as noise and distortion to render the images unrecognizable to deep learning technologies could be bypassed to a great extent by an advanced attacker.

Text CAPTCHAs. The security of text CAPTCHAs has been extensively studied in the literature. Most text CAPTCHAs deployed on the Internet are highly vulnerable to machine learning-based attacks. Mori *et al.* [27] developed object recognition techniques for breaking Gimpy and EZ-Gimpy CAPTCHAs that are based on recognizing the word in the presence of clutter, obtaining a success rate of 33% and 92%, respectively. Yan *et al.* [45] presented novel character segmentation techniques to attack the Microsoft CAPTCHA, which was designed to be segmentation-resistant at that time. Li *et al.* [26] conducted a comprehensive study on e-banking CAPTCHA schemes and developed a set of image processing and pattern recognition techniques to break the schemes. Their attacks achieved an almost 100% success rate in most cases. Bursztein *et al.* [12] evaluated the strengths and weaknesses of text CAPTCHAs and showed that automated attacks could break most of them. In 2014, Bursztein *et al.* [9] presented a novel approach to solving text CAPTCHAs in a single step using machine learning to attack the segmentation and the recognition problems concurrently. Their approach was generically applicable to all evaluated schemes, achieving a success rate significant enough to consider them broken. Gao *et al.* [17] proposed a simple, low-cost attack to break a wide range of real-world text CAPTCHAs with a success rate ranging from 5% to 77%. Recently, Ye *et al.* [47] presented a GAN-based approach requiring only a small amount of training samples to break the most widely used text CAPTCHAs.

Audio CAPTCHAs. In 2002, Kochanski *et al.* [24] proposed using the speech recognition problem for the reverse Turing test. They developed a synthetic benchmark for evaluating the efficacy of automated solvers against audio CAPTCHAs. The paper concluded that humans significantly

outperform automatic speech recognition (ASR) systems when noise/distortion is injected into spoken digits. Tam *et al.* [41] tested the security of audio CAPTCHAs from popular websites against several machine learning algorithms and achieved correct solutions for test samples with an accuracy of up to 71%. Bursztein *et al.* [10] developed an automated solver called Decaptcha that was able to break 75% of eBay audio CAPTCHAs. In 2015, Sano *et al.* [35] developed an audio reCAPTCHA solver based on speech recognition techniques using hidden Markov models (HMMs). Their attack successfully broke the earlier version of audio reCAPTCHA challenges with 52% accuracy. In 2017, Bock *et al.* [8] developed the unCaptcha, a low-resource and powerful audio CAPTCHA solver that leverages off-the-shelf speech-to-text services with a novel phonetic mapping technique to break audio reCAPTCHA challenges with over 85% accuracy.

X. CONCLUSION

We present a low-resource, high success rate attack on the hCaptcha service. Our automated CAPTCHA breaker solves hCaptcha challenges with 95.93% accuracy, making its reverse Turing tests broken. Our security analysis demonstrates that hCaptcha lacks stringent security measures to prevent automated abuses, which will have a severe consequence on the security of online services that rely on hCaptcha to defend against malicious bots. In the future, we plan to investigate the effectiveness of our attack methodology on other image CAPTCHA services relying on image recognition as their underlying AI problem.

RESPONSIBLE DISCLOSURE

We reported our attack and countermeasures to the hCaptcha security team to help them make the system more robust to automated attacks. They responded that their system would have been pretty confident that our traffic was automated based on the techniques we used, and we would never have observed additional countermeasures. However, we did not notice any measures preventing our bot from passing the image CAPTCHA tests during our experiment. The hCaptcha security team stated that they could not disclose system internals and behavior details since it is proprietary software but mentioned that the website owners would not earn any *Human Tokens* (HMT) for the traffics flagged as automated by the system even when the bot bypasses the CAPTCHA tests. But the hCaptcha deployment dashboard shows we earned 0.0717 HMT for the challenges that our automated program solved.

ACKNOWLEDGMENTS

The authors thank the anonymous reviewers for their valuable comments that helped improve this paper. This work is supported in part by US NSF under grant OIA-1946231.

REFERENCES

- [1] Amazon rekognition. <https://aws.amazon.com/rekognition/>. Last accessed 22 December 2020.
- [2] Google cloud vision api. <https://cloud.google.com/vision>. Last accessed 22 December 2020.
- [3] Human protocol. <https://www.humanprotocol.org/>. Last accessed 29 March 2021.
- [4] Microsoft azure computer vision. <https://azure.microsoft.com/en-us/services/cognitive-services/computer-vision/>. Last accessed 22 December 2020.
- [5] Puppeteer — Tools for Web Developers — Google Developers. <https://developers.google.com/web/tools/puppeteer/>. Last accessed 22 December 2020.
- [6] puppeteer-firefox. <https://github.com/puppeteer/puppeteer/tree/master/experimental/puppeteer-firefox>. Last accessed 22 December 2020.
- [7] Selenium - Web Browser Automation. <https://selenium.dev/>. Last accessed 22 December 2020.
- [8] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. uncaptcha: A low-resource defeat of recaptcha's audio challenge. In *Proceedings of the 11th USENIX Conference on Offensive Technologies*, WOOT'17, Berkeley, CA, USA, 2017. USENIX Association.
- [9] Elie Bursztein, Jonathan Aigrain, Angelika Moscicki, and John C. Mitchell. The end is nigh: Generic solving of text-based captchas. In *Proceedings of the 8th USENIX Conference on Offensive Technologies*, WOOT'14, Berkeley, CA, USA, 2014. USENIX Association.
- [10] Elie Bursztein and Steven Bethard. Decaptcha: Breaking 75% of ebay audio captchas. In *Proceedings of the 3rd USENIX Conference on Offensive Technologies*, WOOT'09, Berkeley, CA, USA, 2009. USENIX Association.
- [11] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, page 125–138, New York, NY, USA, 2011. Association for Computing Machinery.
- [12] Elie Bursztein, Matthieu Martin, and John Mitchell. Text-based captcha strengths and weaknesses. *Proceedings of the 18th ACM conference on Computer and communications security - CCS '11*, 2011.
- [13] Kumar Chellapilla, Kevin Larson, Patrice Simard, Mary Czerwinski, and Mary Czerwinski. Designing human friendly human interaction proofs (hips). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '05, pages 711–720, New York, NY, USA, 2005. ACM.
- [14] Kumar Chellapilla and Patrice Y. Simard. Using machine learning to break visual human interaction proofs (hips). In *Proceedings of the 17th International Conference on Neural Information Processing Systems*, NIPS'04, pages 265–272, Cambridge, MA, USA, 2004. MIT Press.
- [15] Jeremy Elson, John Douceur, Jon Howell, and Jared Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. pages 366–374, 01 2007.
- [16] Haichang Gao, Wei Wang, Jiao Qi, Xuqin Wang, Xiyang Liu, and Jeff Yan. The robustness of hollow captchas. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, page 1075–1086, New York, NY, USA, 2013. Association for Computing Machinery.
- [17] Haichang Gao, Jeff Yan, Fang Cao, Zhengya Zhang, Lei Lei, Mengyun Tang, Ping Zhang, Xin Zhou, Xuqin Wang, and Jiawei Li. A simple generic attack on text captchas. In *NDSS*, 2016.
- [18] Philippe Golle. Machine learning attacks against the asirra captcha. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 535–542, New York, NY, USA, 2008. ACM.
- [19] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.
- [20] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. *2015 IEEE International Conference on Computer Vision (ICCV)*, Dec 2015.
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.
- [22] Md Imran Hossen, Yazhou Tu, Md Fazle Rabby, Md Nazmul Islam, Hui Cao, and Xiali Hei. An object detection based solver for google's image recaptcha v2. In *23rd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2020)*, pages 269–284, San Sebastian, October 2020. USENIX Association.
- [23] Evan Klinger and David Starkweather. phash.org: Home of phash, the open source perceptual hash library. <https://www.phash.org/>. Last accessed 15 March 2021.
- [24] G. Kochanski, D. Lopresti, and C. Shih. A reverse turing test using speech. In *INTERSPEECH*, 2002.
- [25] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Mallochi, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, 2018.
- [26] Shujun Li, S. Amier Haider Shah, M. Asad Usman Khan, Syed Ali Khayam, Ahmad-Reza Sadeghi, and Roland Schmitz. Breaking e-banking captchas. In *Proceedings of the 26th Annual Computer Security Applications Conference*, ACSAC '10, page 171–180, New York, NY, USA, 2010. Association for Computing Machinery.
- [27] Greg Mori and Jitendra Malik. Recognizing objects in adversarial clutter: Breaking a visual captcha. In *Proceedings of the 2003 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, CVPR'03, pages 134–141, Washington, DC, USA, 2003. IEEE Computer Society.

[28] Gabriel Moy, Nathan Jones, Curt Harkless, and Randall Potter. Distortion estimation techniques in solving visual captchas. In *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, CVPR'04*, pages 23–28, Washington, DC, USA, 2004. IEEE Computer Society.

[29] M. Osadchy, J. Hernandez-Castro, S. Gibson, O. Dunkelman, and D. Pérez-Cabo. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. *IEEE Transactions on Information Forensics and Security*, 12(11):2640–2653, Nov 2017.

[30] Nicolas Papernot, Patrick D. McDaniel, Somesh Jha, Matt Fredrikson, Z. Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, pages 372–387, 2015.

[31] Iasonas Polakis, Marco Lancini, Georgios Kontaxis, Federico Maggi, Sotiris Ioannidis, Angelos D. Keromytis, and Stefano Zanero. All your face are belong to us: Breaking facebook’s social authentication. In *Proceedings of the 28th Annual Computer Security Applications Conference*, ACSAC ’12, page 399–408, New York, NY, USA, 2012. Association for Computing Machinery.

[32] Matthew Prince and Sergi Isasi. Moving from recaptcha to haptcha. <https://blog.cloudflare.com/moving-from-recaptcha-to-haptcha/>, 2020. Last accessed 18 August 2020.

[33] Yong Rui and Zicheng Liu. Artifacial: automated reverse turing test using facial features. *Multimedia Systems*, 9:493–502, 06 2004.

[34] Olga Russakovska, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. Imagenet large scale visual recognition challenge. *Int. J. Comput. Vision*, 115(3):211–252, December 2015.

[35] Shotaro Sano, Takuma Otsuka, Katsutoshi Itoyama, and Hiroshi Okuno. Hmm-based attacks on google’s recaptcha with continuous visual and audio symbols. *Journal of Information Processing*, 23:814–826, 11 2015.

[36] Katharine Schwab. Suspicious of google’s recaptcha? here’s a popular alternative. <https://www.fastcompany.com/90377406/suspicious-of-googles-recaptcha-heres-a-popular-alternative>, 2019.

[37] Vinay Shet. Google online security blog: Are you a robot? introducing “no captcha recaptcha”. <https://security.googleblog.com/2014/12/are-you-robot-introducing-no-captcha.html>, 2014.

[38] Chenghui Shi, Xiaogang Xu, Shouling Ji, Kai Bu, Jianhai Chen, Raheem Beyah, and Ting Wang. Adversarial captchas, 2019.

[39] Suphanee Sivakorn, Iasonas Polakis, and Angelos D. Keromytis. I am robot: (deep) learning to break semantic image captchas. *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*, Mar 2016.

[40] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks, 2013.

[41] Jennifer Tam, Jiri Simska, Sean Hyde, and Luis V. Ahn. Breaking audio captchas. In D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, editors, *Advances in Neural Information Processing Systems 21*, pages 1625–1632. Curran Associates, Inc., 2009.

[42] Luis von Ahn, Manuel Blum, and John Langford. Telling humans and computers apart automatically. *Commun. ACM*, 47(2):56–60, February 2004.

[43] Haijin Weng, Binbin Zhao, Shouling Ji, Jianhai Chen, Ting Wang, Qinming He, and Raheem Beyah. Towards understanding the security of modern image captchas and underground captcha-solving services. *Big Data Mining and Analytics*, 2:118–144, 06 2019.

[44] Jeff Yan and Ahmad Salah El Ahmad. Breaking visual captchas with naive pattern recognition algorithms. *Twenty-Third Annual Computer Security Applications Conference (ACSAC 2007)*, pages 279–291, 2007.

[45] Jeff Yan and Ahmad Salah El Ahmad. A low-cost attack on a microsoft captcha. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS ’08, pages 543–554, New York, NY, USA, 2008. ACM.

[46] Sarita Yardi, Nick Feamster, and Amy Bruckman. Photo-based authentication using social networks. New York, NY, USA, 2008. Association for Computing Machinery.

[47] Guixin Ye, Zhanyong Tang, Dingyi Fang, Zhanxing Zhu, Yansong Feng, Pengfei Xu, Xiaojiang Chen, and Zheng Wang. Yet another text captcha solver: A generative adversarial network based approach. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS ’18, pages 332–348, New York, NY, USA, 2018. ACM.

[48] Bin B. Zhu, Jeff Yan, Qiuje Li, Chao Yang, Jia Liu, Ning Xu, Meng Yi, and Kaiwei Cai. Attacks and design of image recognition captchas. In *Proceedings of the 17th ACM Conference on Computer and Communications Security*, CCS’10, pages 187–200, New York, NY, USA, 2010. ACM.