Bots Work Better than Human Beings: An Online System to Break Google's Image-based reCaptcha v2

Abstract

The image-based captcha is currently the most widely used captcha scheme to protect web services from automated attacks. Among the main image-based captchas, Google's reCaptcha v2 is adopted by millions of websites. Its motto (principle) is to create a task that is "Easy for humans, hard for bots." As technology advances, we want to investigate whether this main principle is well conformed to or not. Previous works show that most existing image-based captchas could be vulnerable to deep learning-based attacks. However, all of the work is offline attacks/analysis, could not provide practical measurements of to what extent automated programs can pass these captchas. In this paper, we design and implement the first fully online system (ImageBreaker) to break Google's most recent image-based reCaptcha v2. We leverage advanced object recognition technologies and browser automation techniques to build our system, and the system achieves the highest success rate at breaking reCaptcha v2 compared to other recent works. Notably, our online attack tests can make a success rate of 92.40% while taking 14.86 seconds per captcha on average; the time includes the delay we added in the system, online image downloading time, and the log-generation time. Our offline attack tests achieve a success rate of 95.00% while taking 5.27 seconds per captcha, much higher than 79% success rate in related work. Our online and offline success rates are both higher than 92%, comparable with the success rates of six captcha-solving services with thousands of human labors. If considering time, our offline attack achieves the best performance. It shows that the implementation of reCaptcha v2 does not conform to its motto well because we find that our bot can also solve the captchas with dynamic images **even better** than human labors. We also point out the design flaws and present possible future direction and countermeasures to shed light on the security of captchas.

1 Introduction

CAPTCHAs or *Completely Automated Public Turing Test to Tell Computers and Humans Apart* are automated tests designed to tell humans and computers apart. They rely on the intuition that most humans can pass these tests, while current computer programs cannot [51]. Since captchas have the potential to distinguish human beings from bots with a high probability, web administrators are relying more and more upon captcha based services to protect critical web services from bots and automated scripts. For example, captchas are used to stop bots from creating fake accounts, posting automatically submitted votes on online pools, etc. Once such a critical security mechanism is broken, bots can gain access to services they are not allowed. For this reason, it is crucial to keep captchas secure and reliable.

The captchas have evolved over the years. Initially proven to be robust against malicious attacks, text, and audio-based captchas are now considered vulnerable to different machine learning-based attacks [12,48,54]. Researchers have proposed different techniques and alternative designs to strengthen the security of the existing captchas [7, 29, 50]. However, text, audio, and image-based captchas are still predominant and widely used. Especially, image-based captchas have gained widespread popularity in recent years.

Being deployed by more than 4.5 million websites, Google's image-based captcha service, reCaptcha v2, is the most popular right now [4]. As most people find solving a captcha challenge annoying, reCaptcha uses an *advanced and adaptive* risk analysis engine to make an initial judgment on whether a request is from a legitimate user or an automated program. If the system determines the user making the request is likely to be an automated program, then reCaptcha will not allow the user to continue until he/she solves challenges and pass the verification.

Recently the security of the image-based captcha systems, including reCaptcha v2, has come under scrutiny. Sivakorn et al. used different deep learning-based image classifiers technologies to break image-based reCaptcha v2 [46]. Likewise, Weng et al. developed several deep convolutional neural network (CNN) based offline image classifiers to show vulnerabilities of ten popular image-based captcha systems, including Google reCaptcha 2015 and reCaptcha 2018, to machine

learning technologies [52]. However, we notice that the most recent reCaptcha v2 showing good immunity to image classifiers after we experiment with four state-of-art pre-trained image classifiers: Xception [16], ResNet [28], VGG16 [45], and InceptionV3 [47]. We have tested these models on 100 randomly selected images from five frequently used categories from original reCaptcha v2 challenges, and the best average accuracy we obtain is less than 40%. The poor performance of image classifiers on reCaptcha images is primarily due to the wide variation of background in the recent images. For example, relative simple images like an image of a stop sign with grass in the background were used at the beginning of reCaptcha v2. Right now, the images used by reCaptcha v2 usually have rich information and also frequently contain multiple objects from a common scene, intelligent enough to confuse even sophisticated image classification models.

Moreover, the most recent works on reCaptcha have only demonstrated offline attacks to evaluate vulnerability; however, due to the adaptive capability of the latest reCaptcha risk analysis engine, a successful offline attack does not necessarily reflect a practical online attack. In an offline attack, the attacker first collects some images from reCaptcha challenges. Then the performance of the solver is assessed by its ability to discern similar images that match with the target object in a challenge. However, the advanced risk analysis engine does not expose many of its adaptive properties unless the program engages in captcha solving process. For instance, reCaptcha often repeats identical or similar images to such a program. Further, we observe reCaptcha randomly distorts images by adding noise and blurriness in some cases when our system is submitting and solving captchas. In some instances, it asks us to solve captchas for some object when the target object is not present in those images at all. Furthermore, in more than two months of observation, we only encounter 13 object categories (see Table 1) while collecting information from reCaptcha challenges using an automated script. However, once our online system starts to break captchas, we begin to notice categories that were not exposed to us before. Additionally, handing dynamic captchas (see section 2) effectively requires special strategies. Many of these aspects are overlooked in an offline attack. We also find that simply collecting reCaptcha images for offline analysis does not necessarily provide the images one would have got while solving real captcha challenges. Besides, offline attacks are evaluated based on assumptions that are not often correct. For these reasons, offline attacks or analysis cannot provide practical measurements of to what extent reCaptcha v2 can be passed by automated programs. For the same reasons, an effective online attack model is different and more challenging than an offline attack, and it will have a higher impact on the captcha security. Also, the success rates of offline bots and online captcha-solving services using human labors are not comparable due to real factors like IP blocking, captcha cascading, captcha's difficulty-level escalating, etc. To investigate whether the bots can solve the captcha better than human beings, an online captcha solving program to compare the success rates of human labors and bots is desired.

In this paper, we conduct a comprehensive study of reCaptcha v2. Through our security analysis, we find that the essential flaw is the design of reCaptcha v2 change the typical object recognition problem to an object category verification problem. It reduces the problem to be easier; thus, it reduces the difficulty level of the challenge for a bot to solve. Through extensive experimentation, we evaluate the adaptive property of the imaged-based challenges provided by reCaptcha v2. Our rigorous analysis reveals that the most recent version of reCaptcha v2 does include wide variations in challenge images rendering prior classifier based attacks ineffective. We design and implement a novel and sophisticated system that leverages advanced object detection and browser automation techniques to break Google's image-based reCaptcha v2, with high efficacy. Taking the state-of-art object recognition technique, YOLOv3, as a basis, we develop a sophisticated customized object detection and localization pipeline that is very precise at recognizing the target objects in reCaptcha challenges. Our online attack tests achieve a weighted success rate of 92.40% while taking 14.86 secs per captcha on average; the time includes the delay we added in the system, online image downloading time, and the log-generation time. Our offline attack tests achieve a weighted success rate around 95.00% taking 5.27 seconds for each captcha on average, much higher than 79% success rate in [52]. If using captcha-solving services with thousands of human labors, within 5.27 seconds, the success rate range for Google ReCaptcha 2018 is [85%-92%] in [52]. Our online and offline success rates are both higher than 92%. It shows that the motto of reCaptcha v2 "easy for human, hard for bots" [10] is violated because we find our bot can also solving their image-based captchas **even better** than human labors [1]. We call for secure captcha designs that comply with the motto.

We list the main contributions of this paper as follows:

- We conduct extensive security analysis on Google's image-based captcha service, reCaptha v2. We investigate and document the internal workflow of different types of captchas. Our findings also reveal adaptive properties of the advanced risk analysis engine that have not examined prior research.
- We develop a novel and fully online system called ImageBreaker by combining advanced computer vision and browser automation techniques. Specifically, we develop a highly effective customized object recognition system based on YOLOv3 that combines with browser automation module to solve the captchas. To the best of our knowledge, we are the first to develop a real-time Im-

[1] An anonymized demo video of the attack is available at https://youtu.be/l3sS_fTlK_8.

ageBreaker that can break Google's image-based captcha service, reCaptcha v2 with high accuracy.

- We test our ImageBreaker against websites that use this service. Our results show that for reCaptcha v2, the bots could do better than human beings when solving the captcha challenges.
- We also point out the design flaws and present the future design direction and countermeasures to defend against our bots according to our experiments.

2 Extensive Analysis

2.1 Background

The first version of reCaptcha used to display distorted texts in an image, and users were asked to type the text in a text box to solve the captcha. This type of captcha was predominant for quite some years, and it was made segmentation-resistant to counter attacks that previous text-based captchas suffered from [33, 34]. However, the unprecedented advancement of artificial intelligence has made it trivial to solve text-based captchas even when the segmentation-resistant property is in place [8, 17]. To address the aforementioned issues, Google in 2014 introduces reCaptcha v2 [44] also known as "no captcha reCaptcha". This version aims at high usability and user-friendliness while remaining secure against spam and bot. Indeed, Google claims this version to "easy for human" and "hard on bots". They accomplished this goal by building an advanced and adaptive risk analysis engine that scores users' requests from 0.0 to 1.0 based on different aspects of the users' environment. Google has been updating the reCaptcha v2 periodically to keep it robust against emerging security threats. We analyze and test our attack on the most recent version (as of August 2019) of reCaptcha v2 in this paper.

2.2 Anatomy of a reCaptcha challenge

In this subsection, we analyze the security of image-based reCaptcha V2 in a more realistic online setting. We observe several key features as follows: 1) We find the dynamic captcha and adaptive challenge images not discussed in previous works. 2) Five objects types including "traffic light", "bus", and "car" are more than 73% of the recurred objects for static captchas. 3) We only encounter five object categories in dynamic captcha challenges. Further, four of these objects are already present in the top five frequent objects in the static captchas.

Challenge widget. If a click in a checkbox widget returns a low score and the advanced risk analysis engine suspect that the request is from a bot, a new `iframe` element pops up on the page where the actual challenge is being displayed (Fig. 1). To pass the verification, the users must solve a challenge per instruction provided in the challenge widget (Fig. 1).

The challenge widget can be divided into three sections: top, middle, and bottom (Fig. 1). The top section contains the text description (hint text) and instruction about how to solve the challenge. The element that holds the instruction text can

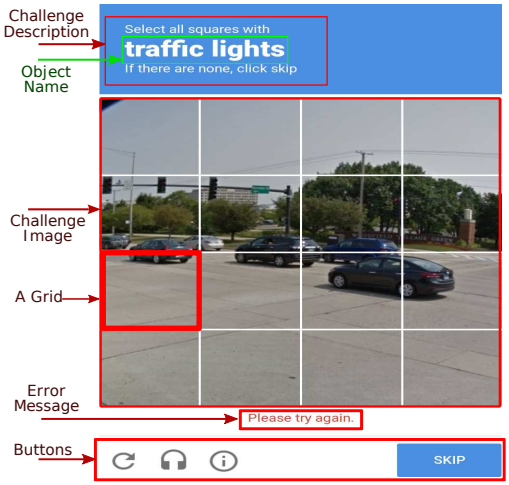

Figure 1: Challenge widget

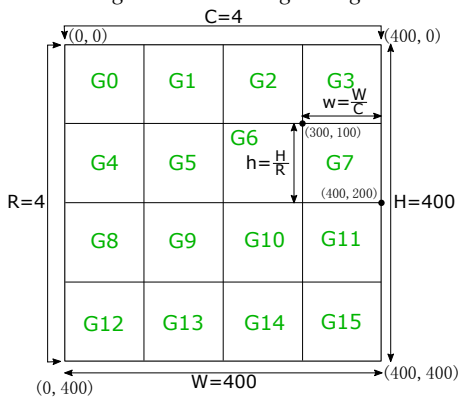

Figure 2: Representing the challenge image as a $R \times C$ rectangular grids, where R is the number of rows in HTML table holding the challenge image and C is the number of cells per row.

be located by its identifier `rc-imageselect-instructions`. The text description also contains the target object name. The section in the middle holds the candidate images. The users have to select images that contain the target object mentioned the hint text. The last and bottom section holds multiple buttons. The first three buttons on the left have the following functions: the reload button is used for getting a new captcha challenge; the middle button with the microphone icon is used for getting an audio challenge, and the third button or help button contains instruction about solving the captcha. The fourth button or the verify button on the right is used for submitting the challenge.

To better understand the challenge, we focus our attention to the middle section of the challenge widget that holds actual challenge images. An HTML table inside a `div` element with an ID `rc-imageselect-target` contains the actual candidate images in the challenge where each cell (`tr` element) of the table acts as a clickable button. Specifically, these candidate images could be extracted by selecting `img` tags inside `div` elements having a class name `rc-imageselect-tile`. For the sake of simplicity, we call each cell as a grid (See Fig. 1).

Although it appears that each grid holds a unique image, the `src` attribute of each image element refers to the same URL in the initial challenge widget. reCaptcha uses CSS style to show different parts of the same image on individual grids in order. We could represent the challenge image as a set of grids. If the HTML table has R rows and C cells per row and the original image is of $W \times H$ dimensions, then we can consider the challenge image as $R \times C$ rectangular shape grids (Fig. 2) each having width $w = \frac{W}{C}$ and height $h = \frac{H}{R}$. We can localize the grids precisely using the top-left and bottom-right coordinates of them. Fig. 2 illustrates the whole process.

We encounter two variants of image-based captcha in our experiment: static captcha and dynamic captcha. We discuss them briefly now.

Static captcha. It is a simple image selection based captcha challenge. Figure 3 shows an example of a static captcha. As we can see, it requires the user to select the potential grids holding the target object(s) and click `verify` (or `submit`) button. If the reCaptcha backend considers the selections to be correct, the user gets verified. Otherwise, it triggers a new challenge.

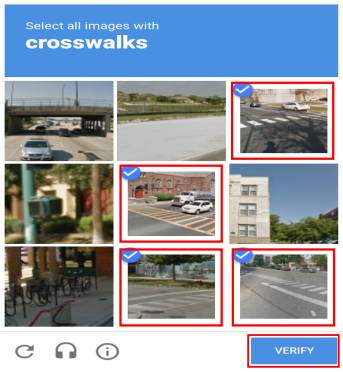

Figure 3: Static captcha. The user simply selects the potential grids holding the target object and clicks `verify` button.

Dynamic captcha. Initially, the challenge image is presented the same way as a static captcha. However, once users click on potential grids holding the target objects, new images get loaded on the selected grids (Figure 4, second image). It requires users to continue to click on all the potential grids until there are no images with target object left in the challenge widget. Finally, the user clicks `verify` to submit the challenge.

Dynamic captcha handling in an automated manner is a relatively complex process compared to static captchas. We need to keep track of the selected grids and download new images when they get loaded. It requires careful programming of the dynamic captcha handler script because we are primarily interacting with asynchronous Javascript codes.

2.3 Adaptive challenges

reCaptcha uses adaptive challenges to restrict automated software or bots from solving captchas. In our experiment, we notice the vast diversity in challenge images. Once our system engages in breaking the captchas, we start getting difficult challenges. reCaptcha usually increases the difficulty level for suspicious requests. Figure 5 shows several images with distorted backgrounds. Nearly 10% of images we face belong to these types. The target objects in these challenges often have unusual shapes. For instance, the captcha may ask to solve a challenge for a motorcycle; however, the challenge image may only include part of a tire in it. The purpose is to make the challenge ambiguous to confuse the automated programs intentionally. Surprisingly, our system can identify all of the objects showing in figure 5.

2.4 Image Repeatedness

We collect 6165 challenge images from reCaptcha protected websites from 05-15-2019 to 07-22-2019. We find the fingerprint of each image using *difference hash* (dhash). Our analysis shows that only 6080 images have unique dhash values. It means there are 85 completely identical images. We also find 228 similar images. We consider two images to be similar if the bit difference (hamming distance) between their hashes is less than 10.

We also attempt to solve 565 reCaptcha challenges using our system from 07-25-2019 to 07-28-2019. Surprisingly, we do not find any completely identical or nearly similar images in the submitted challenges. It signals that once the bot engages in captcha solving, reCaptcha tries refraining from displaying the same image across multiple challenges. This finding also reveals that simply collecting reCaptcha images for offline analysis does not necessarily provide the images one would have got while solving real captcha challenges.

3 Threat Model

We assume that the adversary wants to access some web services protected by reCaptcha using an automated script. We also assume the attacker has the computational power to train and deploy an object recognition system to build a captcha breaker. A recent low to mid-range Graphical Processing Unit (GPU) with 3-4 GB memory should provide sufficient computational power to launch the attack. The adversary may need to change the Internet Protocol (IP) address of the device by tunneling the internet traffic through some anonymity networks like I2P [1] and Tor [20] in case of reCaptcha risk analysis engine or backend blacklists one of his IPs. **[Yazhou: The reader might be wondering how often would the engine usually blacklist the IP?]** Since the adversary has no direct access to the reCaptcha backend, we consider our attack to be a black-box approach. However, The adversary can study the front-end HTML and JavaScript source code and interact with the elements inside reCaptcha checkbox and challenge widgets.

4 System Overview

To validate the design flaw of the reCaptcha v2 and investigate whether the bot could do better than a human for image-based captcha solving, we build a real captcha solving system.

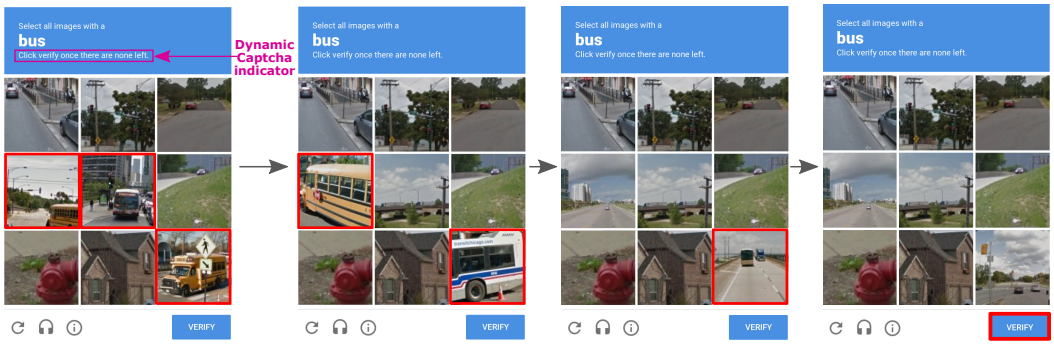

Figure 4: Dynamic captcha. Clicking on grids with red border causes dynamic loading of new images on them. The user is required to click on all potential grids holding the target object until no potential grids are any more. The dynamic captcha indicator is "Click verify once there are none left".

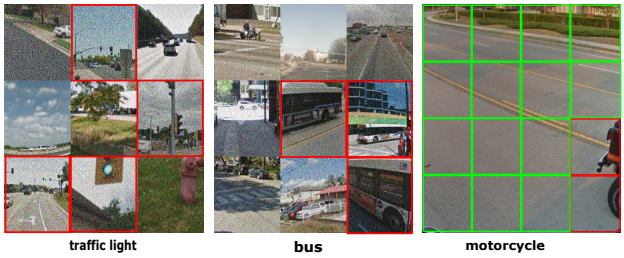

Figure 5: Challenge images with distorted backgrounds or unusual object shapes.

Our system consists of three modules: Information Gathering Module, Customized Object Detection Module, and Solver Module. The modules interact with each other, forming a completely online and fully automated system to break Google's image-based captcha service reCaptcha v2. Figure 6 shows the workflow of the system. In this section, we give a high-level overview of each module.

4.1 Information Gathering (IG) Module

The information gathering (IG) process starts once we are inside the challenge widget. The IG module extracts the target object name, determines the challenge type, and downloads challenge image(s). It also determines the number of rows R and cells per row C in HTML table holding the initial challenge image. It passes these data/information to the customized object recognition (COR) and Solver (S) modules.

4.2 Customized Object Recognition (COR) Module

This module detects and localizes objects in the challenge image and maps them to the potential grids holding them. It takes the initial challenge image I or the new image D as input. Besides, it obtains values of R and C from the IG module. First, the customized object recognition module calculates coordinates of each grid relative to (0, 0). Second, it sends the image to the base object detection system, which does the detection and returns detected objects' coordinates in the form of bounding boxes. It also returns the confidence scores for the detected objects. Finally, our bounding box to the grid mapping algorithm finds the potential grid numbers for each bounding box. The bounding box to the grid mapping algorithm yields a JSON array as output. The output includes object name, confidence score, and potential grid numbers for each detected object in the image.

4.3 Solver (S) Module

Two sub-modules, static solver and dynamic solver, constitute the solver module. If the solver determines a challenge to be of dynamic type, it delegates the tasks to the dynamic solver. Otherwise, static solver gets assigned to the task. Every time a captcha challenge gets submitted, they communicate back and forth with the checkbox widget to check reCaptcha verification status. The solver module terminates when the program passes the verification.

5 Implementation Details

There are several challenges in developing a system that can break captchas in real-time. First of all, we have to complete all the interactions within a limited time. Second, a standard object detection technique cannot fulfill the requirements of our system. Third, we need to develop an efficient way to handle dynamic captcha types. We discuss the challenges and our approaches in detail as we progress in this section.

5.1 Data and Information Gathering

This module is responsible for collecting information and data from the challenge widget. IG module is used extensively by other modules, and it assists them in making decisions. The following data/information is essential for the successful completion of the dependent modules.

5.1.1 Extract target object name

A short description of the challenge, which also includes the target object name, is available at the top of the challenge widget (Fig. 1) We call it as the challenge instruction text and locate the element holding this text by its' identifier `rc-imageselect-instructions`. We then extract the `innerText` inside this HTML element using JavaScript `getAttribute()` method. It is a multi-line text string, and the second line always refers to the target object name. We use the regular expression to split the text using newline delimiter

and extract the second line to get the target object name. It is important to note that the target object name may be in a plural form. We set up simple rules to convert a plural noun string (e.g., traffic lights) to a singular one (e.g., traffic light).

Identification of common objects. To determine what types of object names commonly appear in the reCaptcha challenges, we compile a list of web pages protected by reCaptcha for data collection purposes. We develop a script based on Google's puppeteer [3] library for driving a web browser in an automated manner. The script visits a randomly selected web page from the list and clicks on reCaptcha checkbox to trigger the reCaptcha challenge. Once it is inside reCaptcha challenge `iframe`, it extracts challenge instruction and downloads the image. In total, we make 6165 requests to 4 different web pages and show the results in Table 1. Only encountering 13 object names, we list the recurrence frequencies of them.

Table 1: The recurrence frequency of encounter object names

Object names	Recurrence number (RN)	Frequency
bus	1374	22.29%
traffic light	938	15.21%
crosswalk	907	14.71%
fire hydrant	708	11.48%
bicycle	672	10.90%
car	623	10.11%
store front	589	9.55%
motorcycle	169	2.74%
stair	126	2.04%
chimney	34	0.55%
taxi	17	0.28%
tractor	7	0.11%
parking meter	1	0.02%

5.1.2 Captcha type identification

As discussed in Section 2, the image-based reCaptcha has two variants: a) static captcha and b) dynamic captcha. We notice a consistent pattern in the challenge instruction that we utilize to distinguish them. Specifically, the presence of the phrase "click verify once there are none left" indicates a dynamic captcha challenge. Otherwise, we are dealing with a static captcha. Again, we use regular expression to see if the phrase is present in hint text or not and thus, determine the captcha type.

5.1.3 Get values of R and C

We need the number of rows (R) and the number of cells (C) per row of the HTML table holding the initial challenge image to represent it as a set of grids. There is only a single table element in the challenge widget. We locate this element by `getElementsByTagName()` JavaScript method. The actual values of R and C can be determined by finding the total number of `tr` tags and the number of `td` tags inside a `tr` tag, respectively. However, we use a different strategy, which we find to be more efficient, to determine the number of rows and columns. During our experiment, we observe the `class` attribute of the table element follows a particular pattern. Specifically, it follows the following pattern: "rc-imageselect-table-XY" and the "XY" part corresponds to two different (often same) digits. The first digit "X" determines the number of rows R in the table and "Y" indicates the number of cells C in a row. Like before, we use the regular expression to separate and find the "X" and "Y" from the string.

5.1.4 Download challenge image

We mentioned in Section 2 that when we trigger the challenge widget for the first time, a single image is split into $R \times C$ grids. In other words, the `src` attribute of the `img` tag inside each cell points to the same image regardless of captcha types. First, we find the cell elements in the table using the class name attribute `rc-imageselect-tile`. We then select one cell and find the `img` tag inside. Finally, we get the URL the challenge image by extracting the `src` attribute of the `img` tag. The IG module downloads the image from the URL and saves it to a predefined location in our storage device.

5.2 Customized Object Recognition

Object recognition systems identify and localize objects in images from pre-trained object classes. Since the identification and localization of objects in challenge image alone is not sufficient to map detected objects to grids, we need to develop a customized object detection system. Our main object detection task is delegated to a base detector. We also develop an algorithm to map detected objects locations returned by the base detector to the grids. The customized object recognition module executes some steps (Fig. 6) sequentially to complete the task. We now discuss each step in detail.

5.2.1 Get the dimensions of image I or D

We determine and record the width W and height H of the input image I from the IG module, or the image D from the dynamic solver sub-module.

5.2.2 Representing image I (or image D) as $R \times C$ rectangular shape grids

We use the dimensions ($W \times H$) of the challenge image I (or image D), and the values of R and C to get the $R \times C$ grid representation of the image as discussed in section 2 (Fig. 2).

5.2.3 Base object detector: YOLOv3

To determine the presence of objects and their locations in an image, an object detection model learns from a set of annotated training images (possibly thousands of them). The target locations of the detected objects are usually described in term of bounding boxes. The bounding box is a rectangular box and usually it is defined by x- and y-axis coordinates upper-left corner, and x- and y-axis coordinates of the bottom-right corner of the rectangle. Recent advancements in deep

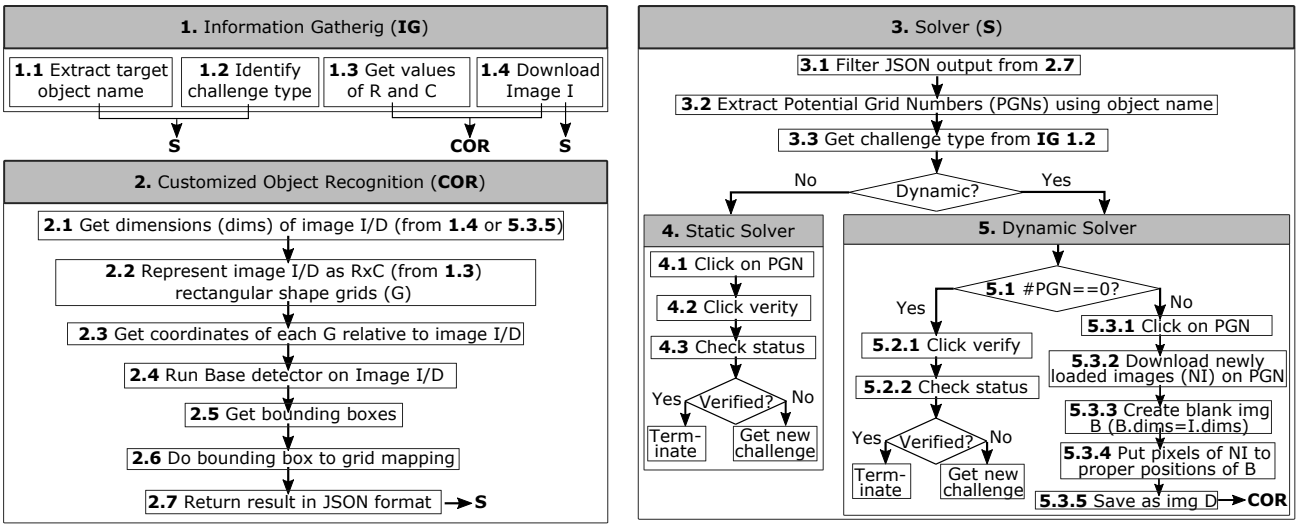

Figure 6: The process workflow of our ImageBreaker

learning and ubiquitousness of relatively low-cost computing devices such as Graphical Processing Units (GPUs), has resulted in some powerful object detection systems. We experimented with several advanced and state-of-art object recognition systems such as mask-RCNN [26], Faster-RCNN [40], and YOLOv3 [39]. We find most of the models, except for YOLOv3, to be computationally expensive even when we take advantage of a GPU. As our proposed system is fully online, and the captcha challenge is time sensitive, we decide to use YOLOv3. In our experiment, we find YOLOv3 extremely fast while maintaining almost similar detection performance when we compare it to the other detection systems. In some cases, it seems to outperform few the state-of-arts. For these reasons, we use YOLOv3 as a base object recognition and detection system in our work.

5.2.3.1 Darknet and customized bounding box detection

Darknet is the framework for building, training, and making predictions on YOLOv3-based models. The framework is written in C. Darknet, by default, provides class name and confidence score of the detected object when making a prediction. Since we also need coordinates of the bounding box for localization, we made modifications to the Darknet source to extract this information. In a YOLOv3 model, a bounding box is determined by the coordinates of the center, width, and height, respectively. However, we redefine the bounding box in terms of top-left and bottom-right corner coordinates for the ease of calculation. We express center coordinates, width, and height of a bounding box as (cx, cy), w and h, respectively. Now the top-left corner coordinates can be expressed as $(tx = (cx - bw)/2, ty = (cy - bh)/2)$. Similarly, the bottom-right corner coordinates will be $(bx = (cx + bw)/2, by = (cy + bh)/2)$.

5.2.3.2 Training YOLOv3 models

We use two datasets to train two separate YOLOv3 object detection models. We first briefly describe the datasets and then provide a short description of the training process.

MSCOCO. The Microsoft Common Objects in COntext, or popularly known as MSCOCO dataset [6] includes 330,000 images in 80 common object categories. It is proposed in [32] to provide a rich dataset for object recognition. Since MSCOCO already contains frequently appeared objects in reCaptcha challenge, we use this dataset to train our object recognition and detection system.

Customized dataset. To detect object categories that are not available in MSCOCO, we develop a customized dataset containing four object categories: crosswalk, storefront, stair, and chimney. We build a web crawler to scrape and fetch images from `images.google.com`. We also use some images from original reCaptcha challenges. Our final dataset includes 3286 manually annotated images.

Building the models. The network architecture of the YOLOv3 object detection model is defined in a configuration (cfg) file. The default cfg file for YOLOv3 has 106 layers. Most of these are convolutional layers, and other layers include shortcut, upsample, route, and yolo layers. We make two configuration files for the two models, one for MSCOCO dataset, and another one for the custom dataset. The MSCOCO YOLOv3 model has 80 object classes and our custom model has 4 object classes.

Training. We train the models using Darknet. Darknet reads models architecture and different parameters related to training from the configuration. It loads training images and their annotations from the data file. We train the model for MSCOCO dataset for roughly three weeks with batch size equals 256. The custom model is trained only for two days with

batch size equals 32. We stop the training when the losses converge.

The performance of an object detection model usually describes in term of mean average precision (mAP). During training, Darknet saves trained weights after every 1000 iterations. We evaluate the weights on the testing dataset and choose the weight file that provides the highest mAP. Our final models have mAP of 58.7 and 71.2 for MSCOCO dataset and the custom datasets, respectively.

Testing or predictions. We now make predictions on image *I* or image *D* (from dynamic solver) using our object detection models. We select the trained weights for MSCOCO if the target object name in captcha challenge matches with one of the object classes in MSCOCO dataset. Otherwise, we choose our custom model. We set the detection threshold to 0.2, e.g., the model discards any objects whose confidence scores are below 20% while making predictions. The output of predictions includes the object name, bounding box coordinates, and confidence score for each detected object.

5.2.4 Bounding box to grid mapping

Once we have the bounding boxes for each detected objects in the image, we need to map them to their corresponding grids. Our bounding box to the grid mapping algorithm uses the coordinates of the bounding box to identify potential grid numbers (PGNs) for each bounding box. Finally, it returns the result in JSON format. The result includes object name, confidence score (returned by the base detector), and potential grid numbers for each bounding box.

Bounding box to grid mapping algorithm. Algorithm 1 depicts the pseudocode for the bounding box to grid mapping algorithm. It takes bounding boxes and grids as input and returns the result in JSON format as output. In line 1 we initialize a JSON array object. There are two `for` loops (from line 2-6). The outer loop takes one bounding box (bbox) at a time, and the inner loop goes through all the grids to see whether the bounding box itself or part of it is present in any grids or not. We get the top-left (*tx, ty*) and bottom-right (*bx, by*) coordinates of a particular bounding box in line 3. The `bbox_belong_to_grid` function (line 5) takes bounding box *i*, and grid *j* as input and returns *True* if it can map the bounding box to the grid using their coordinates; otherwise it returns *False*. If line 5 returns true, it pushes a JSON object with three keys namely, *on* (for object name), *cs* (for confidence score) and *PGNs* (for potential grid numbers) respectively to the result (line 6). The *bbox[i].on* returns the object name, and *bbox[i].cs* returns confidence score for bounding box *i* , respectively. The key *PGNs* is an array since one bounding box may be mapped to multiple grids. Figure 7 shows a challenge image with detected bounding boxes and the JSON output (filtered for the target object "bus") returned by the algorithm.

Challenge and our strategy. The actual shape of an object may not be rectangular like the bounding boxes. Besides, an object may not always occupy the whole bounding box. These two factors can result in some false positives in the bounding box to grid mapping algorithm. Although it is difficult to completely eliminate the false positives, we employ some strategies in our actual implementation code to keep the impact as minimal as possible. For example, if a bounding box has only a tiny portion (say 5% of its total width or height) residing inside a grid, we discard the grid from potential grids.

Algorithm 1: Bounding box to Grid Mapping

Input: bounding boxes and grids
Output: object name (on), confidence score (cs), and PGNs in JSON format

```
1 result ← initialize JSON array
2 for i ← 0 to #bboxes − 1 do
      // get bounding box coordinates
3     tx, ty, bx, by ← get_bbox_coords(bbox_i)
4     for j ← 0 to #grids − 1 do
          // the bounding box (or part of it) is located
             inside the grid
5         if bbox_belong_to_grid(bbox_i, grid_j) = True
            then
              // push object name, confidence score and
                 PGNs to the result
6             result.push({"on" : "$bbox[i].on", "cs" :
                bbox[i].cs, "PGNs" : push(j)})
```

5.3 Solver

The solver module handles both static captchas and dynamic captchas. It gets the target object name and challenge type (static or dynamic) from the IG module. Since handling static and dynamic captchas requires different strategies, we develop two sub-modules to manage them separately.

5.3.1 Static Solver

This sub-module solves static captcha challenges. First, it filters JSON output returned by the COR module to extract potential grid number (PGNs) for the target object in the challenge. Second, it locates potential grids using their CSS identifiers. Third, it performs mouse click actions on those grids. Finally, it finds `submit` (or `verify`) button which is defined by ID `recaptcha-verify-button` and does a mouse click on it. Once clicking on the verify button is done, three possible things can happen: a) it can get verified; b) reCaptcha may trigger errors; and c) a new challenge may get loaded. We check the reCaptcha verification status by switching to the checkbox widget and finding `recaptcha-accessible-status`. If our system passed the verification, we terminate the program. Otherwise, out system will get a new captcha challenge.

5.3.2 Dynamic Solver

When dynamic captcha gets triggered for the first time, we handle it the way similar to static captcha except for the fact


```
[
  {
    "object_name": "bus",
    "confidence_score": 99,
    "PGNs": [
      0
    ]
  },
  {
    "object_name": "bus",
    "confidence_score": 53,
    "PGNs": [
      3
    ]
  },
  {
    "object_name": "bus",
    "confidence_score": 27,
    "PGNs": [
      2
    ]
  }
]
```


Figure 7: Left: A challenge images with detected bounding boxes. Right: Filtered JSON output returned by the bounding box to grid mapping algorithm. The target object is a bus.

that we do not click `verify` button. After clicking on the potential grid, new images appear on the selected grids. The dynamic captcha handler extracts the `src` attribute of each `img` tag inside those grids and downloads the images. Now we can send each image individually to the base detector and see whether the target object is present in them or not. We can then click on the grids holding images with the target object. Although this approach is simple and can solve the challenge, we found it to be inefficient and time-consuming. It is not a practical approach since we need to run the base detector on each image separately, which is a computationally expensive process. We develop a strategy to address this problem efficiently. First, we create a blank image B, which has the same dimensions as initial challenge Image I. Second, we extract pixels from each image and put these pixels to the proper parts of the blank image B. We want to mention that the dynamically loaded images on selected grids have the same dimensions of the shape of the grids. Since our system keeps track of the potential grid numbers at each phase, and it knows the coordinates of the grids, it can place the pixels from new images correctly to the blank image. Third, it sends this image to the COR module to get the object detection results. The COR then sends the result back to the Solver, and dynamic Solver extracts potential grid numbers for the target object. Fourth, it clicks on potential grids. This process gets repeated until the PNGs become zero. Finally, we click `verify` button, check reCaptcha status, and check any errors.

6 Evaluation

In this section, we evaluate our system in an online setting against websites using reCaptcha v2. We also evaluate the offline performance of our model.

Implementation and Evaluation Platform. We use the puppeteer-firefox, a node library, to drive the Firefox web browser. The core functionalities of information gathering and solver modules are developed using JavaScript. We train and test the base detector with the darknet framework specifically modified to meet our need. Our bounding box to grid mapping algorithm is written in C to save time.

We use Selenium and Python in the initial prototype of our system. However, we find Selenium's Python interface to be slow and inflexible for our task. Then, we get rid of all of the Selenium and Python dependencies from our codebase and develop our final system using JavaScript and C for higher efficiency. We develop and test our system on a server with 6 Intel® Xeon® E5-2667 CPUs, an NVIDIA GeForce RTX 2070 GPU, and 96GB of RAM running the Arch Linux operating system. We compile the Darknet framework against the GPU with CUDA version 10.2 and cuDNN version 7.5.

Table 2: Static captcha (online) : the average response time and accuracy of different type of cases

Object name	RN	Freq.	Success rate	avg. RT (sec)
traffic light	98	32.45	94.90	13.98
car	40	13.25	95.00	14.40
bus	32	10.60	93.75	15.35
crosswalk	28	9.27	85.71	15.32
fire hydrant	25	8.28	100.00	14.20
bicycle	25	8.28	96.00	15.12
motorcycle	17	5.63	88.24	15.76
vehicle	13	4.30	84.62	14.73
parking meter	10	3.31	90.00	13.97
boat	6	1.99	83.33	13.95
stair	5	1.66	60.00	14.14
chimney	3	0.99	66.67	14.83
Weighted avg.	/	/	92.38	14.56

Online attack. Google has kept the internal working of reCaptcha backend hidden. The javascript code is also obfuscated to prevent reverse engineering by attackers. There is no direct way to evaluate the attacks in such a scenario. Besides, the risk analysis engine is adaptive, which often let users pass the verification even if they make some mistakes. While some times, it continues to ask the users to solve the captcha challenges when they select objects correctly, we keep the problem simple by defining a success metric.

We consider an attack to be successful when our system passes the verification. When our system needs to solve multiple challenges, we define an attack to be successful if the interactions between the system and the reCaptcha widget do not trigger any error. If our system makes a wrong grid selection, it will trigger the error for both static captcha and dynamic captcha. Additional, if it fails to detect an object, an error will be triggered as well. We regard these events as failures. Further, we find that there are at least two potential grids with the target object in challenge images for static captcha in most cases. Hence, if our system detects less than two potential girds, we set it as a failure. We need to solve a single dynamic captcha in more than one step (Fig. 4). We consider every step as a unique challenge to keep the evaluation process straightforward.

We collect and compile a list of sites defended by reCaptcha. Our system submitted 565 challenges in total. In every attempt, when it submits the challenge by clicking on the `verify` or `submit` button, it reads the current status of the challenge as described in Section 2. If the status indicates that it has passed the verification, the program terminates and starts the next request. Otherwise, it starts processing the next challenge. The system also logs captcha types, the time required to solve a challenge, the total time to pass the verification, and the number of failed attempts.

Table 2 shows evaluation results for static captchas. We encounter 321 challenges and submit 302 challenges. The system has broken the challenges with 92.38% weighted accuracy while taking 14.56 seconds per challenge on average. As we can see from the table, we encounter just 16 object categories in our experiment. It suggests that reCaptcha v2 has minimal object classes. A more concerning issue is that the top 5 categories make more than 73% of total challenges. We discard four objects that appear less frequently (5.92%) like a statue, bridge, mountain, and tractor. ImageBreaker reloads and gets a new captcha when it encounters any of these discarded object types in a challenge since our object detection models do not include these objects.

Table 3: Dynamic captcha (online attacks): the average response time and accuracy of different type of cases

Object name	RN	Freq.	Success rate	avg. RT (sec)
car	71	26.89	92.96	15.06
bus	62	23.48	88.71	15.19
bicycle	47	17.80	93.62	15.18
fire hydrant	44	16.67	95.45	15.68
crosswalk	40	15.15	92.50	15.21
Weighted avg.	/	/	92.42	15.23

Table 3 depicts success rate and average response time for different object categories in dynamic captcha challenges. The weighted average success rate and response time per captcha are 92.42% and 15.23 seconds, respectively. We discover a surprising finding for dynamic captcha: we observe only five object categories while solving the 263 dynamic captchas. Moreover, all these objects are present in the static captcha challenges. Combining the results in Table 2 and 3, the weighted average success rate for static and dynamic captcha is $(302 \times 92.38\% + 263 \times 92.42\%)/565$=92.40%. Similarly, the weighted average response time is 14.86 seconds. If using captcha-solving services with thousands of human labors, within 14.86 seconds, the success rate range for Google ReCaptcha 2018 is [85%-92%] [52]. Our success rate 92.40% is higher than the highest success rate 92% of the six captcha-solving service using human labors. It shows that the motto of captcha – "Easy for humans, hard for bots" is not always valid any more because we find the bot can also solving the captcha with dynamic image **even better** than the human labors.

6.1 Offline attack evaluation

We randomly select 300 challenge images from 12 object categories for offline verification. The images are collected from real captcha challenges submitted by our system. We also make sure that the recurrence frequency for each object category reflects the actual recurrence frequency in the online evaluation.

We conduct offline experiments and compare the results with our manually verified results to evaluate the offline success rate of our model. We create a file to log information about the challenge images. The log file includes the full path to the challenge image in our file system, the target object name, manually verified potential grid numbers (PGNs) per line. We make a script that takes each line as input and extracts individual entries. First, it reads and sends the first two entries to the customized object recognition module. The COR module returns the bounding box to mapping result in JSON format as the final output. Second, it filters the result to extract PGNs for the target object. Finally, it cross-checks the PGNs in the second step to the manually verified PGN(s). If the program finds an exact match between these two values, it returns *true* and appends an extract entry with a value *success* in the line. Otherwise, it adds *failed* as the fourth entry to the line. The program goes over each line at a time and does the same repeatedly. The process stops once it reaches the end of the file.

Sivakorn et al. [46] reported that reCaptcha provides flexibility in its solution. The authors mentioned that one wrong selection, along with at least two correct ones, would be sufficient to pass the verification. They evaluated their results on the offline challenge database while considering this flexibility. However, in our online attack, we find none of the assumptions about solution flexibility reported in that paper hold. While those assumptions may be correct in some cases for human solvers, in most cases (more than 65%), any sense of perceived flexibility considerably diminishes when reCaptcha identifies bot-like activities in a client's request. Therefore, we set a hard limit while evaluating our results for offline verification. Particularly, **we consider a solution to be correct only when our system makes no wrong selection.**

The final results are shown in Table 4. Our program breaks the challenges with a weighted success rate of 95.00% while taking 5.27 seconds per challenge on average. It can be noticed that our custom object detection model does poorly for stair and chimney. The poor performance on both of these objects is due to a limited number of images (3000) in the training of our model. Overall, the accuracy of our offline attack model is much better than 79% in [52]. If using captcha-solving services with thousands of human labors, within 5.27 seconds, the success rate range for Google ReCaptcha 2018 is [88%-92%] [52]. The success rate of our offline attacks is 95.00%, much accurate than human labors within a short time. Without time limit, our success rate is the same as the

Table 4: Offline attacks: the average response time and accuracy of different types of cases

Object name	RN	Freq.	Success rate	Avg. RT (sec)
traffic light	70	23.33	98.57	4.85
car	50	16.67	98.00	4.96
bus	40	13.33	87.50	5.82
fire hydrant	35	11.67	97.14	5.70
bicycle	30	10.00	100.00	5.16
crosswalk	20	6.67	90.00	5.21
motorcycle	15	5.00	100.00	4.99
vehicle	15	5.00	93.33	5.68
parking meter	10	3.33	90.00	5.34
boat	8	2.67	100.00	6.19
stair	4	1.33	50.00	5.93
chimney	3	1.00	66.67	5.26
Weighted avg.	/	/	95.00	5.27

Table 5: Comparisons between our results and six captcha-solving services with human labors [52]

	Attack target	Success rate	avg. RT
ruokuai	ReCaptcha 2018	91	6.97
hyocr	ReCaptcha 2018	85	7.05
2captcha	ReCaptcha 2018	88	4.27
AntiCaptcha	ReCaptcha 2018	92	5.69
DeCaptcha	ReCaptcha 2018	62	31.12
imagetyperz	reCaptcha v2	95	41.68
Our offline attack	reCaptcha v2	95	5.27
Our online attack (with delay)	reCaptcha v2	92.40	14.86

service “imagetyperz”, however, “imagetyperz" needs 41.68 seconds to achieve its success rate 95.00%, much slower than our solver. Overall, it shows that the motto of captcha – “Easy for humans, hard for bots” has not been well conformed to because we find that the bot can also solve the captcha with dynamic image **even better** than human labors. Table 5 compares the success rate of our results to six captcha-solving services using human labors.

7 Countermeasures

Image-based captchas are based on the assumptions that there are certain tasks related to image recognition that humans are good at, but current computers are not. However, recent advances in computer vision show that machines are as good as, if not better than humans in image recognition domain [27,30]. Consequently, captchas relying on simple image recognition will eventually fail to generate robust Turing tests for bots. Since reCaptcha v2 falls into that category, it would require radical change to its current design to prevent kind of attacks we show in this paper. A robust captcha design would acknowledge the current state of AI and use the limitations to its advantage. To this end, we discuss the design flaws of reCaptcha v2 and propose a general design direction (7). However, it is outside of the paper to provide a formal security analysis for our potential design direction. We also recommend some countermeasures as a temporary deterrent to automated attacks on reCaptcha v2 and similar image-based captchas in this section.

Design flaws. In [52], the authors point out the design flaws of selection-based captcha are: using a limit number of object categories, machine-encoded text hints, and easily recognizable candidate images. However, we have different visions. First, we argue that the essential flaw is the design of reCaptcha v2 change the normal object recognition problem to an object category verification problem. It reduces a hard problem to an easier problem. For example, it gives the object category and asks the bot to check whether the grids contain that object. The design reduces the difficulty level of the challenge for a bot to solve. Thus, we argue that enlarging the size of the object categories only makes the model training more time-consuming, and it could not eradicate the design flaws. The second argument we want to make is that adding noise to the candidate images and never reusing the candidate images again are not helpful as well. Through our experiments, we find that our system could solve the adaptive challenges; also, our ImageBreaker could solve the captchas without repeated images as well.

Web Driver fingerprinting and restriction. A secure captcha system should be able to detect and prevent the use of web driver and browser automation software which are used to drive automated programs. We notice reCaptcha can identify when our script is based on Selenium [5], a popular web driver that is also used by most prior works on reCaptcha. It also blocks our program from accessing reCaptcha challenges every time after we make four to five requests in a row. However, it has allowed our automated script based on the puppeteer framework [3] to solve the captchas. We also experiment with another relatively new browser automation framework name TestCafe [2], which is not detected by reCaptcha. It indicates the reCaptcha risk analysis engine cannot recognize and fingerprint web drivers and other browser automation frameworks universally. Since humans should only solve captchas, access to reCaptcha challenges by any third-party scripts should be restricted. It can be done by developing an advanced technique to detect web-driver and browser automation frameworks.

Adding constraints to the challenge instruction. Current reCaptcha v2 system asks users to select images (grids) containing some object in the instruction of the challenge. As we have seen in this paper, the bots can easily pass the verification by identifying and selecting grids with the target object. The captcha challenges could be hardened by adding more constraints to the challenge instruction or make them difficult

to been understood by a machine. These constraints should be intuitive for humans but hard for bots. For example instead of saying, "select all the images with traffic light", we can say "select all the images with traffic light in green arrow. A human can identify such an object without much effort. However, it is not a trivial task for the machine anymore since object detection models are not usually designed to identify objects with this kind of constrains. If the images have both normal traffic light and traffic light with a green arrow for that particular example, the computer will have a hard time distinguishing between these two. One can still develop and train an object detection model to identify objects with such restrictions. However, it will not be as straightforward as it is now.

Adding anti-bot resistance in obtaining challenge instruction. A bot cannot select the correct grids without extracting the target object's name from the challenge instruction. A robust captcha system must make sure that the target object's name is not easily obtained. For instance, the name of the object or the instruction itself could be embedded in an image to augment anti-bot resistance. Even though machine learning models can be trained to extract textual information from the image, it can still slow down the bots as a temporary solution.

Expanding the size of challenge object categories. Ideally, a secure image-based captcha should have an unlimited number of object categories. While it may not be possible to incorporate that many object types in a real-word captcha, it should include a relatively large number of object types to deter automated attacks. Besides, we find 8 of these objects are already present in MSCOCO dataset. If the object categories are not readily available in public image datasets with labels like MSCOCO [32], Open Images [31], and ImageNet [42], it will make the attacks more difficult because the adversary needs to manually collect, label, and annotate thousands of images with significant efforts. However, this method only makes the training time-consuming and cannot lower the success rate of our bot. Thus, it is a temporary solution as well.

Use geometry of the object to disable machine learning. In this paper, we reduce captcha solving to an object category verification problem. Although object recognition is still considered to be a hard AI problem, a captcha system based on mere object category verification does not guarantee significant robustness against attacks than an image classification based captcha. Our attack on reCaptcha v2 in this paper and the earlier successful attacks on captchas like ARTiFICIAL [41] and IMAGINATION [19], explain why object recognition alone cannot promise a more secure design. However, we can enhance the robustness of object recognition captchas against computer-based attacks by exploiting the shape, size, and orientation of the objects. For instance, if a rectangular bounding box is drawn around a tilted fire hydrant with relatively large size in an image, it will result in some

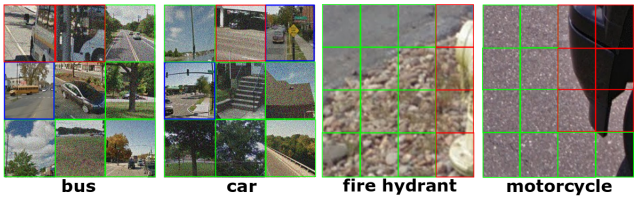

Figure 8: Some failed instances. Red rectangle indicates grids with undetected target object. Blue rectangle is a grid where the target object is present and also detected.

false positives. The false positives occur because we are essentially selecting grids based on bounding box's coordinates when part of the actual object may not present in all those grids. Likewise, a secure image recognition captcha should use objects with diverse shapes and sizes, and take advantage of the geometric properties of the objects to misdirect the computers.

Adversarial Machine Learning. Machine learning models, including deep neural networks (DNNs), have been reported to be vulnerable to adversarial examples [9, 11, 14, 25, 37]. By adding a small noise to a legitimate input, which is often imperceptible to human eyes, adversaries can trick a learned DNN model into misclassifying. This technique can be applied to reCaptcha challenge images as well. However, the attacker can use adversarial training [49] or similar methods [36, 53] to make the system robust against such attacks. Moreover, most of the current research work only explored the vulnerability of DNNs against adversarial examples in image classification domain. However, most real-world computer vision tasks are based on object detection systems. Eykholt et al. [22] in their 2018 WOOT paper showed that a stop sign could go undetected by YOLO v2 [38] and Faster-RCNN [40] detectors by creating an adversarial poster or physical stickers on it. It is a simple example, and to what degree object recognition models are susceptible to adversarial inputs is still an open research problem. In general, practical attacks against object detection models are more challenging than image classifiers because there can be multiple bounding boxes in a single image or scene.

Potential design direction. To find a problem which is "easy for human, hard on bots", we propose to use a mapping from a short-story text to a descriptive image. The short-story text can be abstracted as a story of m relationships among n objects. For example, the short-story text challenge is "A cat is chasing a rat behind a green willow." We can generate many candidate descriptive images telling a similar but different short story from the challenge. The hard part of designing such a captcha is to generate the short-story texts and candidate descriptive images. However, it is easier than solving such a text-to-image challenge for bots. Also, the designers need to make sure the problem to solve could not be reduced into a verification problem easily.

8 Limitation and Future Work

During experiments, our object recognition system could not work well to identify some objects in the reCaptcha challenge images. After analyzing the failed cases, we noticed that most misdetections occurred when the test images were noisy and blurry. Our object detection models can identify objects from moderately distorted images — however, heavy distortions cause misdetections in many cases. Figure 8 shows some instances where it happened. It is apparent from the figure that identifying objects from such distorted images is challenging even for humans. Besides, adding too much noise and blurriness degrades the usability of the captcha system.

We think our ImageBreaker could be improved to address the issues mentioned above by adding noisy and blurry images in the training dataset. Image augmentation techniques also could be utilized to generate adversarial training samples from the existing datasets. We plan to do this in our future work. Also, we only tested Google reCaptcha v2. In the future work, we will test more other image-based captcha and make our online solver be a general security strength testing tool for image-based captchas.

9 Related Work

The text and audio-based captchas are now considered broken since they are highly prone to automated attacks, especially against machine learning-based attacks. Image-based captchas have gained popularity in recent years as they are more accessible and user-friendly to humans and relatively difficult for computers to solve. However, the assumption that machines are not good at recognizing semantic information from images is incorrect. It leads to various high profile attacks on image-based captchas deployed in real-world. In this section, we discuss related attacks on image-, text-, and audio-based captchas.

Image-based captcha. Golle et al. [24] designed a support vector machine classifier to break the Asirra CAPTCHA [21] where users were required to distinguish between images of cats and dogs. Zhu et al. [57] provided a systematic study of 13 existing image recognition captchas (IRCs). They assessed security and practicality of these captcha schemes in real-life applications. Their research showed that most of the IRCs studied did not meet real-world security requirements and thus prone to automated attacks. The authors also proposed the design of a novel IRC scheme called Cortcha based on their findings.

In 2016, Sivakorn et al. [46] revealed flaws in reCaptcha advanced risk analysis engine which would allow an attacker to influence the system to bypass captcha challenges in some instances. They also design an attack to break the image-based reCaptcha challenges. The authors used different deep learning technologies to perform the attack. It is worth mentioning that Google has changed the image-based reCaptcha in the same year. Their attack methodologies are not sufficient to solve recent reCaptcha challenges automatically. More recently, Zhao et al. [56] in 2018 evaluated the security of 10 real-world image captchas, including ReCaptcha 2018. They developed several convolutional neural network (CNN) based image classifiers to break captcha systems. Their attack achieved 79% accuracy in solving image-based ReCaptcha 2018. However, the authors only evaluated the attack on offline images with a limited number of object categories.

Text-based captcha. Ye et al. leverage Generative Adversarial Networks to solve text-based captchas automatically [55]. The authors first train a base solver to generate synthetic captchas and then perform fine-tuning on the basic model by applying transfer learning. As a result, their scheme does not require a large volume of captchas to train an effective solver. A major limitation of work is that it only works for fixed-length characters. Gao et al. built a captcha breaker based on 2D Log-Gabor filter [23]. They evaluated their attacks on a wide range of text-based captchas services.Bursztein et al. used machine-learning-based classifiers to build a generic attack for text-based captchas [12]. Their scheme solves text captcha in a single step by applying machine learning to attack the segmentation and the recognition problems simultaneously. Other notable attacks on text-based captchas include [15, 33, 54].

Audio-based captcha. Audio captchas are most vulnerable to different attacks and extensively studied in the literature. The scheme in [13] can break 75% of eBay audio captchas. Darnstädt et al. [18] utilized active and semi-supervised learning methods to build a low-cost system breaker for audio captchas. The authors of [48] analyzes the security of audio captcha using machine learning algorithms such as AdaBoost, SVM, and k-NN. Sano et al. [43] used hidden Markov (HMM) models to build a system to break audio captchas. Few other important works on audio captchas are [15, 35].

10 Conclusions

We designed and implemented a completely online system to break the image-based reCaptcha v2 in this paper. We utilized state-of-art object recognition and advanced browser automation technologies to build the system. Our system achieved a weighted success rate of 92.40% when breaking reCaptcha challenges. It took only 14.86 seconds on a weighted average per captcha while doing so. Our attack on reCaptcha v2 shows that it is highly vulnerable to machine learning-based attacks, especially to advanced object detection algorithms like YOLOv3.Based on our extensive analysis and findings, we also provided several countermeasures to defend captchas against such attacks. We hope our future design direction will shed light on secure captcha designs. We also believe it is the time to reconsider the assumption that most existing image-based captchas are made on: computers are not as good as human beings to solve visual captchas to pass the Turing test.

Ethics. During our experiment, we did not target any parts of the test web pages except the reCaptcha widgets. Specifically, we limit our interactions to two reCaptcha `iframe` objects only. Besides, we have reported our findings to Google.

References

[1] I2P Anonymous Network. https://geti2p.net/en/. Last accessed 21 July 2019.

[2] A node.js tool to automate end-to-end web testing | testcafe. https://devexpress.github.io/testcafe/. Last accessed 29 July 2019.

[3] Puppeteer | Tools for Web Developers | Google Developers. https://developers.google.com/web/tools/puppeteer/. Last accessed 22 July 2019.

[4] reCAPTCHA Usage Statistics. https://trends.builtwith.com/widgets/reCAPTCHA. Last accessed 21 July 2019.

[5] Selenium - Web Browser Automation. https://www.seleniumhq.org/. Last accessed 29 July 2019.

[6] COCO - Common Objects in Context. http://cocodataset.org/#overview, 2019. Last accessed 8 Aug 2019.

[7] Elias Athanasopoulos and Spiros Antonatos. Enhanced captchas: Using animation to tell humans and computers apart. In Herbert Leitold and Evangelos P. Markatos, editors, *Communications and Multimedia Security*, pages 97–108, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[8] Paul Baecher, Niklas Büscher, Marc Fischlin, and Benjamin Milde. Breaking recaptcha: A holistic approach via shape recognition. In *SEC*, 2011.

[9] Battista Biggio, Igino Corona, Davide Maiorca, Blaine Nelson, Nedim Šrndić, Pavel Laskov, Giorgio Giacinto, and Fabio Roli. Evasion attacks against machine learning at test time. *Lecture Notes in Computer Science*, page 387–402, 2013.

[10] Andrew Braun. Captchas: Why we need them, how they're evolving, and how you can solve them more easily. https://www.maketecheasier.com/captchas-why-we-need-them/, 2018. Last accessed 12 July 2018.

[11] Wieland Brendel, Jonas Rauber, and Matthias Bethge. Decision-based adversarial attacks: Reliable attacks against black-box machine learning models. *ArXiv*, abs/1712.04248, 2017.

[12] 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.

[13] 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.

[14] Nicholas Carlini and David Wagner. Towards evaluating the robustness of neural networks. *2017 IEEE Symposium on Security and Privacy (SP)*, May 2017.

[15] 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.

[16] Francois Chollet. Xception: Deep learning with depthwise separable convolutions. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[17] Claudia Cruz Pérez, Oleg Starostenko, Fernando Uceda Ponga, Vicente Alarcon-Aquino, and Leobardo Reyes-Cabrera. Breaking recaptchas with unpredictable collapse: Heuristic character segmentation and recognition. pages 155–165, 06 2012.

[18] Malte Darnstädt, Hendrik Meutzner, and Dorothea Kolossa. Reducing the cost of breaking audio captchas by active and semi-supervised learning. In *Proceedings of the 2014 13th International Conference on Machine Learning and Applications*, ICMLA '14, pages 67–73, Washington, DC, USA, 2014. IEEE Computer Society.

[19] Ritendra Datta, Jia Li, and James Z. Wang. Imagination: A robust image-based captcha generation system. In *Proceedings of the 13th Annual ACM International Conference on Multimedia*, MULTIMEDIA '05, pages 331–334, New York, NY, USA, 2005. ACM.

[20] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. In *Proceedings of the 13th Conference on USENIX Security Symposium - Volume 13*, SSYM'04, Berkeley, CA, USA, 2004. USENIX Association.

[21] Jeremy Elson, John (JD) Douceur, Jon Howell, and Jared Saul. Asirra: A captcha that exploits interest-aligned manual image categorization. In *Proceedings of 14th ACM Conference on Computer and Communications Security (CCS)*. Association for Computing Machinery, Inc., October 2007.

[22] Kevin Eykholt, Ivan Evtimov, Earlence Fernandes, Bo Li, Amir Rahmati, Florian Tramèr, Atul Prakash, Tadayoshi Kohno, and Dawn Song. Physical adversarial examples for object detectors. In *Proceedings of the 12th USENIX Conference on Offensive Technologies*, WOOT'18, Berkeley, CA, USA, 2018. USENIX Association.

[23] 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.

[24] 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.

[25] Ian Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In *International Conference on Learning Representations*, 2015.

[26] Kaiming He, Georgia Gkioxari, Piotr Dollar, and Ross Girshick. Mask r-cnn. *2017 IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[27] 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.

[28] 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.

[29] Niloy J. Mitra, Hung-Kuo Chu, Tong-Yee Lee, Lior Wolf, Hezy Yeshurun, and Daniel Cohen-Or. Emerging images. *ACM Trans. Graph.*, 28(5):163, 2009.

[30] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[31] Alina Kuznetsova, Hassan Rom, Neil Alldrin, Jasper Uijlings, Ivan Krasin, Jordi Pont-Tuset, Shahab Kamali, Stefan Popov, Matteo Malloci, Tom Duerig, and Vittorio Ferrari. The open images dataset v4: Unified image classification, object detection, and visual relationship detection at scale, 2018.

[32] Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C. Lawrence Zitnick. Microsoft coco: Common objects in context. *Lecture Notes in Computer Science*, page 740–755, 2014.

[33] 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.

[34] 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.

[35] Jeffrey P. Bigham and Anna C. Cavender. Evaluating existing audio captchas and an interface optimized for non-visual use. pages 1829–1838, 04 2009.

[36] Nicolas Papernot, Patrick McDaniel, Xi Wu, Somesh Jha, and Ananthram Swami. Distillation as a defense to adversarial perturbations against deep neural networks. *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016.

[37] 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.

[38] Joseph Redmon and Ali Farhadi. Yolo9000: Better, faster, stronger. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017.

[39] Joseph Redmon and Ali Farhadi. Yolov3: An incremental improvement. *arXiv*, 2018.

[40] Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(6):1137–1149, Jun 2017.

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

[42] Olga Russakovsky, 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.

[43] 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.

[44] 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. Last accessed 21 July 2019.

[45] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition, 2014.

[46] Suphannee 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.

[47] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun 2016.

[48] Jennifer Tam, Jiri Simsa, 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.

[49] Florian Tramèr, Alexey Kurakin, Nicolas Papernot, Ian J. Goodfellow, Dan Boneh, and Patrick D. McDaniel. Ensemble adversarial training: Attacks and defenses. *ArXiv*, abs/1705.07204, 2017.

[50] Erkam Uzun, Simon Pak Ho Chung, Irfan Essa, and Wenke Lee. rtcaptcha: A real-time captcha based liveness detection system. In *NDSS*, 2018.

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

[52] Haiqin 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.

[53] Cihang Xie, Jianyu Wang, Zhishuai Zhang, Zhou Ren, and Alan L. Yuille. Mitigating adversarial effects through randomization. *ArXiv*, abs/1711.01991, 2017.

[54] 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.

[55] 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.

[56] Binbin Zhao, Haiqin Weng, Shouling Ji, Jianhai Chen, Ting Wang, Qinming He, and Reheem Beyah. Towards evaluating the security of real-world deployed image captchas. In *AISec@CCS*, 2018.

[57] Bin B. Zhu, Jeff Yan, Qiujie 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.