unCaptcha: A Low-Resource Defeat of reCaptcha's Audio Challenge

Kevin Bock Daven Patel George Hughey Dave Levin
University of Maryland

Abstract

CAPTCHAs are the Internet's first line of defense against automated account creation and service abuse. Google's reCaptcha, one of the most popular captcha systems, is currently used by hundreds of thousands of websites to protect against automated attackers by testing whether a user is truly human. This paper presents unCaptcha, an automated system that can solve reCaptcha's most difficult auditory challenges with high success rate. We evaluate unCaptcha using over 450 reCaptcha challenges from live websites, and show that it can solve them with 85.15% accuracy in 5.42 seconds, on average. unCaptcha combines free, public, online speech-to-text engines with a novel phonetic mapping technique, demonstrating that it requires minimal resources to mount a large-scale successful attack on the reCaptcha system.

1 Introduction

CAPTCHAs (the Completely Automated Public Turing tests to tell Computers and Humans Apart) are defense systems designed to protect against automated account creation and service abuse by presenting users with a challenge that is easy for humans to solve but difficult for computers [27, 8].[1] Captchas are used extensively online as a defense against automated bots and Sybil attacks [4], as well as preventing spam. For instance, many online registration platforms, from social media services to email to ticketing systems [8], require the user to solve a captcha during registration to prevent automated creation of fake accounts. In a similar vein, some online servicets have recently begun requiring Tor clients to solve captchas before delivering web content [11, 16].

The security of captchas is paramount to protecting services on the Internet from these attacks. As the spread of news and information is increasingly driven by user content on sites like Twitter, YouTube, and Reddit, bots that could defeat the captcha system and register a disproportionate number of accounts could theoretically control the flow of information *en masse* [29]. It is therefore unsurprising that captchas have been the target of attack for researchers and attackers for years [27, 2, 25, 28, 21, 32, 13, 1, 22].

Until recently, captchas have featured distorted text that users must correctly type to pass. Bursztein et al. [1] showed these text-based captchas to be insecure by demonstrating a system with near-complete (98%) accuracy. As a result, text-based captchas have been largely phased out in favor of image captchas (discussed at greater length in Section 2).

However, visually impaired users are incapable of solving these visual captchas, prompting the creation of audio captchas [27]. Typical audio captchas consist of different speakers saying words or digits at randomly spaced intervals at a variable pitch or speed, often with an accent and distortion/noise [27]. To solve the captcha, a user must correctly identify the digits or words spoken in the audio clip.

Attacks have been demonstrated on these audio captchas with varying degrees of success in the past. This is usually done by training local machine-learning models to identify the spoken words [2, 25], a high-resource and time-consuming approach. Additionally, although researchers have explored using online speech recognition services, including Sphinx [2] or Google Speech Recognition [18], these services have not been accurate enough to compete with offline services or solve the captcha reliably.

We present *unCaptcha*, a low-resource, fully automated attack on Google's 2017 reCaptcha audio captcha. unCaptcha is viable for even a low-resource attacker; rather than perform its own analysis locally, unCaptcha leverages free, publicly-available speech recognition services, and performs a minimal phonetic mapping to boost

[1] For the remainder of the paper, we follow industry convention and write the acronym in lowercase, as "captcha", for readability.

accuracy. Using a free Amazon Elastic Compute Cluster (EC2) *t2.micro* instance, we demonstrate that unCaptcha can solve reCaptcha's audio challenges with 85.15% accuracy in 5.4 seconds, on average. As the audio captcha is sufficient for passing the entire reCaptcha system, this work presents a near complete defeat of the reCaptcha system, rendering the hundreds of thousands of sites[2] that rely on it vulnerable to abuse or automated attacks. We demonstrate that it takes far fewer resources to mount a high accuracy defeat of the reCaptcha system than previously thought, and make suggestions for more secure auditory captcha designs.

The rest of this paper is organized as follows. In the next section, we provide an overview of related work. Section 3 details more background into Google's reCaptcha system, and Section 4 describes unCaptcha's design. We analyze and evaluate unCaptcha in Section 5, and propose defenses and more secure audio captcha solutions in Section 6. Finally, we conclude in Section 7.

2 Related Work

Captchas have long been the target of automated attacks. Due to the breadth of literature, we will focus only on attacks concerning auditory captchas. Almost a decade ago, Kochanski et al. investigated the security of audio captchas and developed a synthetic benchmark for evaluating automatic solvers [12]. This work concluded that humans vastly outperform automated speech recognition systems (when audio noise/distortion is added to spoken digits), and has guided the design of modern audio captchas. Further independent studies deployed a two-phase segment-then-classify approach and successfully broke older versions of Google and Yahoo audio captchas [26, 21]. These two-phase solvers usually operate by first extracting portions of the captcha that contain the digit, and then running pre-trained machine learning algorithms to classify those individual digits, rather than classifying them all at once [24].

In 2012, a research team developed the Decaptcha system, a high performance auditory captcha solver to attack eBay's captcha system [2]. Decaptcha was a two-phase solver that first examined energy spikes in the audio file to segment by digit. Using a supervised learning algorithm and a large corpus of previously annotated eBay captchas, researchers developed a powerful model that could recognize digits. Decaptcha achieved 75% accuracy on tens of thousands of captchas on their largest machine learning model. Burzstein et al. also defined four threat models for attackers, and explored the potential damage each could do. Although the advanced and high-level attackers, defined to have tens to hundreds of thousands of IP addresses and tremendous local resources, could achieve high accuracy with their Decaptcha model, the defined low and even medium resource attackers achieved only 10% success [2]. Such a model discrepancy is common: many existing captcha solutions assume high attacker resources, including cores, bandwidth, IP addresses, and more, to achieve high accuracy solving [27, 2, 25, 28, 21, 32].

In 2012, a Defcon group demonstrated a successful attack on the original iteration of reCaptcha's auditory captcha [25]. Using a large, powerful, locally trained neural network, they achieved over 99% accuracy on the original audio captcha system. After Google updated the auditory captcha system to change it from words to digits, inject background noise, and vary number of digits to mitigate their vulnerability, the researchers rewrote their tool and still achieved 61% accuracy. After a third reCaptcha update (which was very similar to the original iteration, according to the researchers), they again defeated it with 60% accuracy using their large neural network. Their tremendous success was limited only by their need for a large corpus of training data; gigabytes of data were used for training in the original attacks [25]. A similar high-resource attack in 2013 using Hidden-Markov Models was also highly successful, solving these audio captchas with 52% accuracy[20].

In 2016, a proof of concept work was done to attack the newly updated audio captcha [18]. This work demonstrated that it was possible to submit the entire captcha audio to Google Speech Recognition and get a candidate solution. No accuracy measures were published, but the researcher noted that on the 4-5 digit captcha, it was successful. However, shortly after this work, the new 2017 captcha was released, introducing the longer audio captchas (10 digits) and background noise to each digit. Presumably, the background noise lowered the digit classification accuracy and the longer captcha required a higher accuracy (90%) to pass, and therefore the new captcha broke this methodology. Since this work only used Google's Speech Recognition API and did not segment the digits before analysis, ReCaptcha's protection against this attack was successful. Still, this work serves as a foundation for our approach to break the 2017 reCaptcha.

Threat Model

Prior work has generally assumed that attackers against captcha systems are well-resourced. In particular, the standard threat model involves an attacker who can attack the captcha tens or hundreds of thousands of times for a relatively small number of successes, and can scale this attack to abuse services.

[2]Google does not publish reCaptcha usage statistics. We make this conservative estimate using a Censys [5] search for "recaptcha".

Tightly coupled with the captcha threat model is what level of accuracy is considered sufficient to "defeat" a given captcha system. An attacker with many resources can afford a lower success rate, and thus some have argued that even a success rate of 1/10,000 is sufficient to threaten the integrity of services against an advanced attacker [3]. Others assume attackers have fewer resources (limited training data and/or processing power) and define 1% success accuracy as sufficient to defeat a captcha system [2]. Still others raise the bar to only classifying success above a higher threshold, such as 50% or 70% [27].

In our work, we will assume an attacker with *limited* resources; unlike previous works attacking captchas, our threat model limits the attacker to one computer, one IP address, a small amount of RAM and limited training data (less than 100MB).[3] Therefore, we aim for accuracy benchmarks above 50%, as a low-resource attacker cannot afford a lower percentage of success. Other works have achieved low-resource attacks on captchas before: Yan et al. [32] successfully produced a low-resource attack on the Microsoft text-based captcha in 2008, prompting Microsoft to request the work to be held in confidence for months while a replacement system was built. To our knowledge, no similar low-resource attacks have been replicated for auditory captchas.

3 reCaptcha Background

The reCaptcha system relies on an advanced risk-analysis engine. As the user interacts with reCaptcha (clicking buttons and typing), the system determines a level of suspicion for that user. Today, many users will find that they simply need to click the checkbox and be verified without needing to solve a captcha. This occurs when the reCaptcha is fairly confident that the user is human and not an automated attacker (this is called the "noCaptcha reCaptcha"). If the system is unsure if the user is a human (but is not highly suspicious either), it will deliver a moderate challenge to the user (an easy image problem or a short audio string of numbers to transcribe). This often occurs when a user does not yet have a long enough history of interaction with Google. However, as the reCaptcha system becomes increasingly suspicious, it delivers harder challenges: 10 digits in the audio challenge, or prompting the user to solve multiple challenges. By default, a user with no past history with Google services will be automatically given the most difficult challenge. It is these most difficult challenges that unCaptcha attempts to solve. In the backend, a developer can enable different levels of security for reCaptcha on a 3 point scale between "Easiest for users" and "Most secure". Through our experimentation, we could discern no difference in the level of difficulty of the captcha itself across the three settings, suggesting that only auxiliary security checks are enabled in the higher security settings.

Although the new reCaptcha system was introduced in 2014 to replace the traditional "distorted text" captcha, not much is known about its inner workings. Google has protected the inner design of reCaptcha heavily, releasing few details about how their software works. The captcha system is run from an encrypted, isolated VM (Virtual Machine) in JavaScript with a unique bytecode language. To make reverse engineering even more difficult, the bytecode has direct access to JavaScript variables of its own interpreter, and changes its own decryption key and even its own opcodes numbers at many points during its own execution. A full working disassembler and decompiler for the system was released, and it was determined that the captcha system, in addition to confirming the actual captcha solving, checked for the presence of: valid plugins; a valid user-agent string; a valid screen resolution; execution time; computer timezone; number of click, keyboard, or touch actions in the `iframe` of the captcha; many browser-specific functions and CSS rules; canvas rendering properties; server side cookies; and likely more [15].

In 2016, a further analysis by Sivakorn et al. [23] of the reCaptcha system explored the weaknesses of the initial implementation of the image captcha. It is important to note that the image captcha has changed since that paper, and their methodology is no longer sufficient to defeat the captcha. However, their analysis of the captcha's risk analysis system lends insight into its inner workings. In particular, Silvakorn et al. found that Google's tracking cookies play an integral role in the captcha's defenses. The captcha system is made aware of every time a user interacts with a Google service (or a page with Google's tracking cookies, such as Google analytics). After just 9 days of automated browsing across different Google services, their bots' tracking cookie was sufficient to fool the risk analysis system into thinking they were human, and checking off the box. However, their experiments revealed each cookie could only immediately complete 8 captchas per day before needing to solve additional challenges. Their results also showed that the reCaptcha system attempts to fingerprint the browser, using canvas rendering techniques, comparing the user-agent to what the browser reports, and potentially more [23]. Despite these impressive efforts of the risk analysis engine to identify a bot before the captcha, as we will show next, reCaptcha still remains susceptible to low-resource attacks to its audio challenge.

[3] For ease of comparison between resource levels, all of our benchmark testing for the 'low-resource' attacker was done on a free-tier Amazon Elastic Computing *t2.micro* instance, with 1GB of RAM, 1 virtual CPU, and 8GB of storage.

4 unCaptcha Design

In this section, we present the design and methodology of unCaptcha, a novel, low-resource attack against the auditory challenges in Google's reCaptcha. unCaptcha is a fully automated, end-to-end attack, and comprises four key steps: (1) Obtaining the audio sample, (2) Segmenting the audio into per-digit sound bites, (3) Analyzing the sound bites, and finally (4) Entering the captcha solution. We present each of these in turn. See Figure 1 for a diagram of the design.

4.1 Acquiring Audio Captchas

Our primary technical contribution lies in our ability to accurately analyze audio challenges, but a full end-to-end attack must also be able to first obtain the audio challenge. This is nontrivial for a bot, as the reCaptcha system uses multiple heuristics to distinguish between humans and bots, including checks for fake browser agents, headless browsers, javascript checks, and more [15]. Important among these is the detection of keyboard and mouse inputs that appear "inorganic"—for instance, moving the mouse at superhuman speeds or in perfectly straight lines. We describe here how unCaptcha bypasses these; we use the popular Reddit website as a running example, which uses reCaptcha to defend against bot account creation. Note that some of the more aggressive security checks are disabled in reCaptcha's lower security levels, but at the time of writing, our attack was fully functional against reCaptcha on highest security level with all security features enabled. This methodology could be applied to any of the hundreds of thousands of services that use Google's reCaptcha.

Using the popular browser automation software Selenium[4], unCaptcha finds a functioning HTTP proxy to mask its connection from GatherProxy. It uses Firefox to first navigate to Reddit.com, and performs some minor page interaction. It clicks the link to create an account, which opens a "create new account" modal box. The bot then generates a random username, password, and email, clicks into each field, and types it as a human would, with random amounts of time between each keystroke so as to fool reCaptcha. This is just a proof of concept, since no additional processing is done to check if the username or email is valid; these fields are only filled out to initiate the captcha.

It is interesting to note here that our build of unCaptcha was tested on both Mac OS X and Linux (Ubuntu), but the attack only initially worked successfully on Linux, as reCaptcha could seemingly detect the use of Selenium on OS X. This broken detection of browser automation on Linux was a critical vulnerability that in part originally enabled this attack. However, we were later able to custom compile a Mac OS X Selenium ChromeDriver that evaded detection by simply renaming hardcoded DOM-accessible ChromeDriver-specific variables, suggesting that reCaptcha's Selenium detection system is still insufficient.

Once the username and password are filled, the reCaptcha appears. unCaptcha locates the "I'm not a robot" checkbox, and clicks it. Although we engineered the typing to be pseudo-organic, the mouse movements were left to Selenium's default, inorganic behavior. Across all captcha attacks, reCaptcha never seemed to pick up on these mouse movements; we hypothesize that reCaptcha does not actually examine mouse movement patterns, but just a set number of events generated from mouse usage (hover, unhover, etc), which are actually generated by browser automation software by default.

Next, the reCaptcha system generates a new iframe with the actual reCaptcha challenge. From here, the bot switches to an audio captcha, downloads the audio payload from the page, and clicks the "PLAY" button, to mimic a normal, listening user.

4.2 Audio Pre-processing

Following the methodologies in other works, the retrieved audio file is first segmented before recognition. Google's current audio reCaptcha does not have continuous background noise to interfere with segmentation, and relies entirely on distortion within the sound bites themselves for protection. Therefore, segmentation is surprisingly straightforward; unCaptcha simply splits the audio file by periods of silence using amplitude analysis to separate each spoken digit.

4.3 Analyzing the Audio

unCaptcha's primary contribution is in how it analyzes audio in an accurate and low-resource manner. Similar to ReBreakCaptcha [18], we make use of existing online speech-to-text services, but we apply two key additional steps: phonetic mapping and ensembling. We describe these in the following subsections.

4.3.1 Speech Recognition Services

After retrieving and segmenting the audio file, unCaptcha uploads each sound bite to a set of different free online speech recognition services. All of these services were registered within a day and required no cost. In our implementation, we use six: Google

[4] https://www.seleniumhq.org

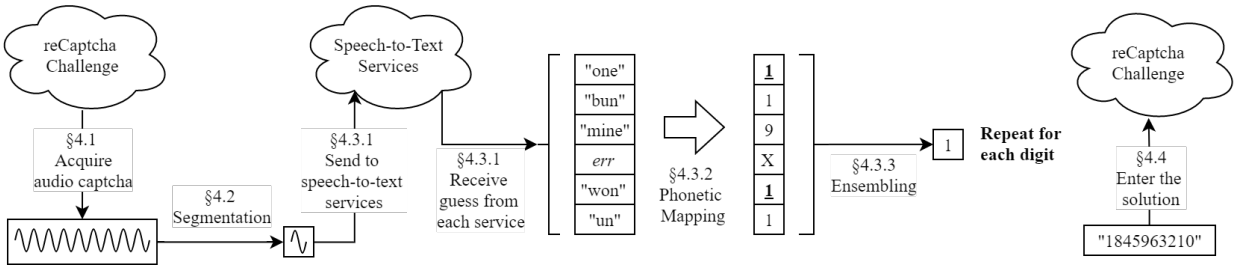


Figure 1: The overall design of unCaptcha. Note that speech-to-text analysis, phonetic mapping, and ensembling are applied to each digit separately.

Cloud[5], Bing Speech Recognition[6], IBM Bluemix[7], Google Speech API, Wit-AI[8], and Sphinx[9] (available through python's `speechrecognition` library[10]). We originally implemented a seventh service, Houndify, but it performed so poorly in comparison in the presence of distortion that we later removed it. By comparison, ReBreakCaptcha used only a single online service (Google Speech API). However, as we show in Section 5, using multiple speech recognition tools is critical to achieving high accuracy.

For each sound bite, each of these services returns a candidate text transcription. It is important to note that most of these services are designed for generic speech recognition, and unlike many prior approaches with specifically trained models to classify spoken digits, these multipurpose services frequently confuse digits with words. We address this issue with phonetic mapping (Section 4.3.2). Additionally, we have found some services to be significantly more accurate than others. We address this by borrowing a technique from machine learning: ensembling (Section 4.3.3).

4.3.2 Phonetic Mapping

Since these services support general speech recognition (not only digits), we employ two layers of phonetic mapping: (1) exact-homophone, and (2) near-homophone.

The preliminary layer of phonetic processing maps common mistakes in speech recognition back to corresponding numerical digits. We map homophones of each digit back to its numerical form (such as "to"/"too" $\mapsto$ "2," "for"/"fore" $\mapsto$ "4"). Frequently, services also spell out the digit instead of returning just the digit. For example, an API might return the word "four" instead of "4". Therefore, we mapped the spelled-out versions of all ten digits to their respective numbers as well in the preliminary layer.

In addition to the basic homophone mapping, a second layer of phonetic processing is needed. Minor distortion, heavily accented voices, or short sound bites negatively affected the service's ability to correctly identify the digit. The services would often return words that sounded phonetically similar to the correct digit. Therefore, we added a heuristics based mapping layer to correct for common mistakes made by the services. In our initial development, almost 100 captchas were solved by hand as a test set to curate the heuristic list of near-homophones. With just 36 near-homophones, unCaptcha solved captchas at a rate greater than 50%. While this list of near-homophones was manually created, it could be also done in an unsupervised manner with natural language processing. Some examples of near-homophones include "free" for "3", "sex" for "6", and "mine" for "9". Few of these near-homophones are actually needed for a basic level of success, but given the wide array of distortion, more mappings vastly improved accuracy overall.

We further extended our phonetic mapping to include partial matches for distinctive sounds of certain digit; for example, words ending in "icks" on average should be classified as a "6", responses containing "ee" would be classified as a "3", etc. In total, unCaptcha's heuristic contains 129 of these homophones, near-homophones, and partial matches which it uses to assemble a candidate solution string of digits. These homophones were curated manually and chosen systematically, first using a trial and error approach, making sure that common homophones were included. For fine-tuning, we employed a statistical analysis of common errors from the speech-to-text services, which identified other less intuitive mappings, such as "you know" for the number 0 (this alone increased our captcha success rate by 0.6%). A more comprehensive collection of such near-homophones and partial matches would only improve overall accuracy; however, our manually curated list was sufficient for the initial build of unCaptcha.

Finally, we note that we did not include any of the au-

[5]https://cloud.google.com/speech/
[6]https://microsoft.com/cognitive-services/en-us/speech-api
[7]https://ibm.com/watson/developercloud/doc/speech-to-text/
[8]https://wit.ai/
[9]http://cmusphinx.sourceforge.net/wiki/
[10]https://pypi.python.org/pypi/SpeechRecognition/

dio samples we used in constructing our phonetic mapping when evaluating unCaptcha (Section 5).

4.3.3 Ensembling

After performing phonetic mapping on each of the individual speech recognition services' predictions, we "ensemble" their responses to obtain a single answer. In essence, each candidate answer gets a weighted vote; the answer with the highest weight wins. We assign weights in two ways.

Throughout our evaluation, we observed that some online speech recognition services were more accurate than others at correctly identifying digits, and therefore the balance of our heuristic model was partially driven by these differences in accuracy. For example, during initial testing, we discovered that Google Cloud was much better than Sphinx at correctly identifying digits; therefore, results from Google Cloud later were given a higher weighting than from Sphinx.

An additional problem arises with this ensembling approach: multiple services returning different, potentially valid identifications. This was common with distorted digits, particularly with harder to distinguish digits. Some near-homophones began to fall under multiple categories, for example,"fine" for "5" and for "9", causing many false positives. To combat this, we assigned a weighting value based on the phonetic similarity of specific words and used a weighted majority vote to determine the digit. Exact-homophones (such as "too" or "for") were given a high weighting value, while more phonetically different near-homophone words that could be confused between categories (such as "fine") were given a low weighting value. This change dramatically improved accuracy (just over 65%). Phonetic analysis of additional common mistakes also showed more near-homophones, such as "brain" for 9, "we" for "3", and "they have" (they've) for "8". This ensembling approach and phonetic mapping layer sets this work apart from existing auditory attack methodologies, such as [18].

4.4 Entering the Solution

After a candidate string of digits has been assembled, unCaptcha organically (with uniform timing randomness between each character) types the solution into the field and clicks the "Verify" button. Continuing with our running example of Reddit: When successful, the bot could go forward to creating an account. Although we could have made Reddit accounts with every successful captcha solution, we chose to make only a small number (fewer than 10) of accounts as a proof of concept.

5 Evaluation

We tested unCaptcha on 459 hand-annotated captchas[11] on a free-tier Amazon *t2.micro* EC2 instance, with 1 virtual core, 8GB storage, and 1GB of RAM. All testing was done on reCaptcha's most difficult, 10-digit captchas.

5.1 Base, Per-digit Accuracy

We begin by investigating the base accuracy provided by each of the services used in unCaptcha. Table 1 shows the per-digit accuracy of each of the six services we use, sorted in decreasing order of base accuracy. As these results show, using a single service alone (such as ReBreakCaptcha, which used Google) would not lead to high accuracy for 10-digit challenges.

	Per-digit Accuracy		**Captcha**
Service	**Base**	**+ Mapping**	**Success**
Google Cloud	55.55%	55.81%	2.61%
Wit	48.01%	64.45%	8.06%
Bing	44.12%	66.65%	10.39%
IBM	31.62%	62.67%	6.50%
Google	21.25%	28.10%	0.01%
Sphinx	6.64%	31.88%	0.02%

Table 1: Per-digit accuracy of each free online speech recognition service used in unCaptcha, before ("Base") and after performing phonetic mapping. Also, overall success rate of phonetic mapping against 10-digit challenges.

5.2 Benefit of Phonetic Mapping

Next, we explore the effectiveness of unCaptcha's phonetic mapping (Section 4.3.2). Each service's per-digit accuracy after phonetic mapping is shown in the "+Mapping" column in Table 1. Across all the captchas, the phonetic processing increased the accuracy by an average of 17.04%, though some services saw far greater improvements than others. For example, Google Cloud needed almost no improvements—nearly all of its returned suggestions were either digits or exact-homophones of the digits. Wit AI performed similarly well, but faltered significantly with digits 0, 6, and 8. Bing's speech-to-text software performed moderately well across the board, again with the exception of the digit 6. Indeed, we see a preponderance of errors with the digit 6; we return to this point in Section 5.4.

[11] Although during the initial data collection phase 500 captchas were attacked and downloaded for analysis, 41 captchas were accidentally overwritten, leaving a test set of 459 captchas.

N	Best Ensemble of *N* Services	Per-digit Accuracy	Captcha Success
1	Google Cloud	55.81%	2.19%
2	Bing + IBM	82.87%	47.26%
3	+ Google Cloud	88.25%	66.96%
4	+ Wit	91.36%	78.12%
5	+ Sphinx	91.93%	80.09%
6	+ Google	91.99%	80.31%
	+ "X" $\mapsto$ "6"	**93.41%**	**85.15%**

Table 2: The marginal benefit of ensembling multiple services, after both phonetic mappings. After full ensembling, we replace each unknown ("X") with a guess for "6", yielding our highest overall accuracy.

Appendix A shows the benefit of the near-homophone mapping in a graphical manner. Note that many of the unidentified clips (in the "X" column) are correctly identified after the mapping, especially in Figures 7(a) and 7(b).

5.3 Benefit of Ensembling

Even if some services are unable to identify a sound bite, with ensembling (Section 4.3.3), it is likely that another one can. Table 2 shows the maximum level of success depending on the number of services used. This table shows that it multiple services before the success rate can reach the 50% success rate mark. After four services, the marginal benefit drops off steeply. At this point, the services are capable of correctly classifying 91.36% of the digits. Additional services have difficulty classifying the remaining 8.64% digits, so they barely improve the success rate.

5.4 unCaptcha Accuracy

Putting all of the techniques together, we now present unCaptcha's overall accuracy.

Per-digit accuracy As Table 2 shows, after the service aggregation and phonetic mapping, unCaptcha achieves a *per-digit* accuracy of 91.99%. Figure 2 shows the distribution of received digits across those 459 captchas, as well as the relative accuracy for each. The distribution of digits received is nearly uniform, suggesting that reCaptcha does a simple uniform sampling when generating the captchas.

Figure 3 presents confusion matrices that show how unCaptcha performs on a per-digit basis. The column labeled "X" in the confusion matrices represents a failure to produce a digit, meaning that unCaptcha could not even identify the sound bite as a numeral. As can be seen in Figure 3(a), the majority of errors came from misclassifying the digit "6".

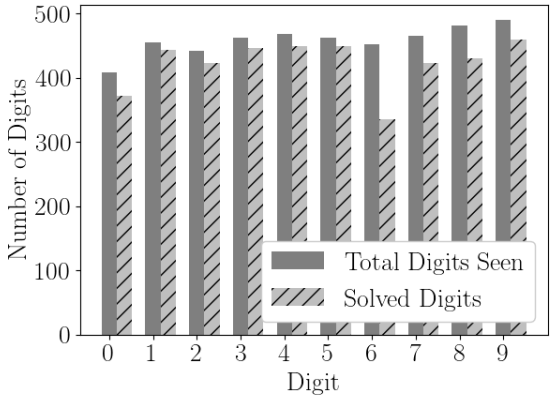
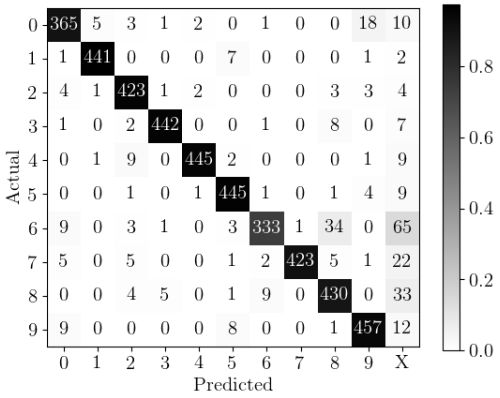


Figure 2: The distribution of digits sampled from reCaptcha, and the number unCaptcha successfully detected.

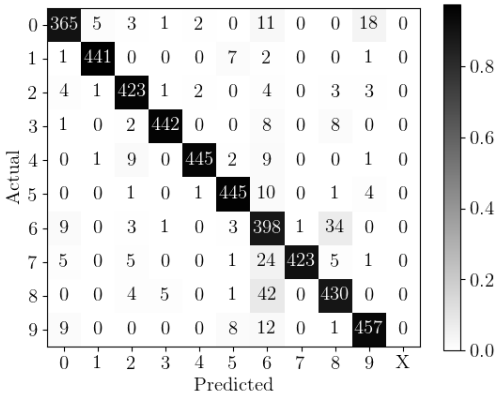
Improving the accuracy of "6" After noticing the low accuracy of "6", we imposed a crude (but surprisingly effective) countermeasure: we simply map all unknowns ("X") to "6", giving us the confusion matrix in Figure 3(b). This improved unCaptcha's per-digit accuracy from 91.99% to 93.41%, and overall captcha success rate from 80.31% to 85.15% (this includes all 6 services and both layers of phonetic mapping).

The success of unCaptcha's ensembling approach can be seen in comparing the overall confusion matrix in Figure 3(a) with the top performing services in Figures 5(b), 6(b), and 7(b); only the services working together were capable of defeating reCaptcha.

Captcha success rate unCaptcha successfully solved reCaptcha's auditory challenges with an overall success rate of 85.15%. This can be explained by two observations. First, reCaptcha requires only 9 of the 10 digits to be successfully identified in order to pass the captcha. We can thus view each digit as an independent trial. Second, as is shown in Figure 2, the distribution of digits is approximately uniform, and thus captcha success rate can be modeled with a simple binomial distribution, with a success probability equal to our per-digit accuracy of 93.41%. In testing, we also encountered no limitations on the number of times we could automatically request a new challenge. This weakness allows for even more accurate attacks; for example, an attacker could repeatedly request new audio challenges until it had higher confidence in its solution.



(a) *Without* mapping unknowns to "6".



(b) *With* mapping unknowns to "6".

Figure 3: Confusion matrices of unCaptcha after ensembling all six services and performing phonetic mapping. As the digit "6" was the most consistently misclassified, we simply map all unknowns to "6".

5.5 unCaptcha Speed

Recall that unCaptcha issues 10 (single-digit) queries to each of 6 online speech recognition services. This is an embarrassingly parallelizable task: the services are independent of one another, and thus an attacker can query them in parallel. We implemented two variants of unCaptcha: one that solves each digit serially, and one that solves them in parallel.

Solving each digitally serially resulted in an average captcha completion time of 22.24 sec. Solving them in parallel, unCaptcha took 5.42 sec on average with a standard deviation of 1.25 sec, a significant improvement in speed.

The audio captchas we captured were on average 19.13 sec long, with a standard deviation of 2.39 sec. We hypothesized that we would have to wait for the duration of the entire audio clip before entering our solution for it to be valid, but surprisingly, reCaptcha allowed solutions to be submitted faster than the audio could play.

In short, the time it takes unCaptcha to *solve* reCaptcha's auditory challenges is on average 28.3% of the time it would take for a human to *listen* to them. As such, reCaptcha's primary bottleneck on attack speed is service response times.

5.6 Service Response Times

Across all six speech-to-text services unCaptcha uses, we observed an average response time of 2.95 sec, with a standard deviation of 1.99 sec. However, recall that unCaptcha uses the free tiers of these services; response times can go up considerably (or the services can become unavailable) after exhausting the free allotment.

Service	API Limits
Google	60 minutes/month
Google Cloud	60 minutes/month
Bing	5,000 queries/month
IBM	1,000 minutes/month
Sphinx	None
Wit	None

Table 3: API limits for the services unCaptcha uses (at their respective free tiers).

Although a few of the services are free to use, others limit the user based on number of queries or minutes of audio, as shown in Table 3. After this period, users have to decide either to sign up for a new account (to access new API keys) or pay for the service. Ironically, a reCaptcha captcha is the only defense against bot registration for a new account on many of these services, meaning that unCaptcha could theoretically be made self-sufficient by automatically signing up for a new account if it detects that a given account is close to its API limit.

5.7 Library Size Estimation

Throughout the process of manually solving each captcha, we noticed a significant number of the individual digits were repeated. We investigate this duplicate rate to see if we could estimate the library size of the reCaptcha's audio files. If the library size is too small, reCaptcha could be vulnerable to an attack where the attacker downloads all possible digits, classifies each of them, and automatically compares each digit from an un-

Digit	Duplication Rate (%)	
0	40 / 408	(9.80%)
1	56 / 456	(12.28%)
2	65 / 442	(14.71%)
3	49 / 463	(10.58%)
4	68 / 468	(14.53%)
5	66 / 463	(14.25%)
6	58 / 452	(12.83%)
7	67 / 466	(14.38%)
8	51 / 482	(10.58%)
9	62 / 490	(12.65%)
Total	582 / 4590	(12.68%)

Table 4: Per-digit duplication rate of the reCaptcha's audio clips we collected during our evaluation. There is considerable overlap in our moderately sized sample; we estimate ∼36,500 total variants.

solved captcha with the already-classified digits. Ideally, a captcha system would have none of these duplicates, so that such an attack would not work.

To determine the number of duplications, we classified each audio segment as its respective digit and analyzed the Mel-Frequency Cepstral Coefficients (MFCCs) of all digits. We found a non-negligible occurrence of these duplicates; Table 4 shows that, of the 4590 total per-digit audio samples we received, 12.68% of them were repeats.[12] Using the capture-recapture [10] method to estimate the overall population size, we estimate that Google stores approximately 3600 files per digit in its database, from which it randomly selects files to generate a captcha, implying an overall library size of about 36,500 digits. The entire audio captcha (10 digits) file size approximately 30KB, meaning the estimated library size constitutes ∼110 MB, well within the storage capacity of a low-resource attacker. Classification would require a comparison between the test digit and the stored digits, which could take a significant amount of time. However, other means of comparison, such as through the use of perceptual hashing [9], could drastically reduce the comparison time and make an attack feasible for a low-resource attacker.

5.8 Offline Attack

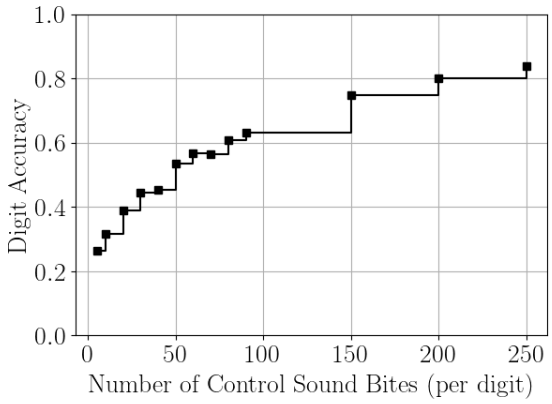
A possible defense against uploading the sound clip to online speech recognition services would be for services to recognize captcha audio clips and refuse to transcribe them or even purposefully transcribe incorrectly. Generally, this defense would only be easily feasible for Google to do (as they have access their own captcha audio library). As captcha sound bites are taken from normal speech, it would be difficult for the other speech recognition services to distinguish parts of speech taken from a captcha and regular legitimate sound bites to transcribe. Additionally, as the attacker has full control of the sound clip to upload, the attacker could add some small degrees of distortion, filtering, or other effects that would not hamper transcription, but would make it more difficult to identify as coming from a captcha. However, we entertain the notion that speech recognition services could refuse to transcribe captcha sound bites. With a defense like this in mind, or in cases where Internet connection is severely limited, we investigate to find if an equally low-resource offline solver is possible, taking advantage of the small library size used by reCaptcha (only 10 digits).

To perform crude, lightweight speech recognition, our offline solver relies on computing the Mel-Frequency Cepstral Coefficients (MFCCs) as means of comparison between sound bites of digits [7]. MFCCs capture features about the short-term power spectrum of a sound wave [14]. MFCCs are a feature widely used in automatic speech and speaker recognition, and were considered state-of-the-art until speech recognition switched largely to machine learning. As MFCCs were originally standardized in 2003 to run on mobile phones, low-resource implementations are available [6]. To properly compare MFCCs, Dynamic Time Warping (DTW) is commonly used to address temporal misalignment [14]. In this manner, to "recognize" a given sound bite, the MFCCs are computed for both the given sound and for a small library of "template" sounds. Using DTW, the given sound and each of the templates are compared, and the template class that most closely aligns with the given sound is considered correct. We follow this basic pattern of recognition in our offline solver.

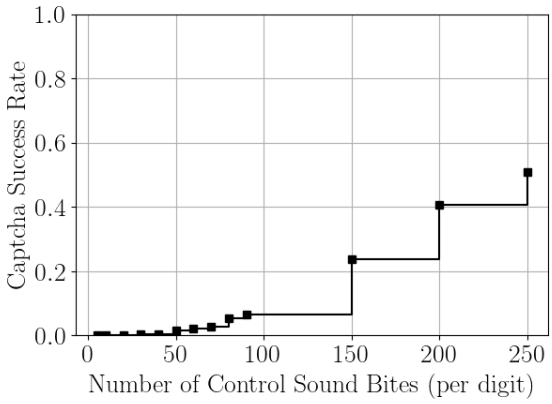
Offline solving proceeds as follows. First, a set of control digits are randomly selected from a list of hand-solved captchas. We tested the overall accuracy varying the size of this control set from 5 to 250 examples (per digit). MFCCs are computed and saved for each of these control digits. After the control set is created, the solver is ready to solve captchas. The test captcha is segmented into individual test digits (using the same method as the online unCaptcha solver), MFCCs are computed for each test digit of the captcha and compared using the DTW algorithm to each of the control digit clips. The class of control digits with the minimum distance to the test digit is chosen as the candidate solution. The candidate string is then assembled as usual, and can be submitted to reCaptcha.

It is important to note that like traditional MFCC speech recognition, the offline MFCC solver does not attempt to "transcribe" the spoken digits as typical speech

[12] Note that this duplicate rate is only a conservative lower-bound estimate, and the true duplication rate is likely higher than this.



(a) The accuracy of identifying a single digit, with varying control size.



(b) The percentage of passing the entire 10-digit captcha, with varying control size.

Figure 4: Digit accuracy and success rate of our MFCC-based offline captcha solver, as a function of the size of a ground truth dataset. We observe diminishing returns, but a surprisingly high success rate (50%) for even a modest ground truth (250 samples).

recognition services do; instead, it attempts to see which set of control spoken digits is most similar to a test sound bite. Even in the presence of noise, accents, or other minor distortion, as long as the waveform of a test digit more closely resembles other same digits, it will be classified correctly. Additionally, this solution intentionally relies on minimal training data and a low-resource algorithm to remain viable for a low-resource attacker. Even with 2500 comparisons per digit in the captcha, on Amazon's *t2.micro* instance with only one virtual CPU, captcha solutions could be produced in under 30 seconds. If more cores were available, this could also be effectively parallelized to be computed faster.

Figures 4(a) and 4(b) show the single-digit and overall captcha solving accuracy using the offline MFCC solver. Even at the highest of only 250 sample captchas, overall captcha solving accuracy reached 51% captchas. On average, with a sample control set of 250 captchas, captcha solving took 29.26 seconds, only 4 seconds slower than the online attack. Creating this set of captchas is similarly trivial; at 21 seconds per captcha, one person could build such a control set in under an hour and a half. Although our offline solver performed worse than the online solver, with 51% accuracy, it is still a sufficient success rate to constitute a defeat of the reCaptcha system.

6 Countermeasures

In this section, we discuss some potential countermeasures to make the reCaptcha system and similar auditory captchas more robust to withstand this attack.

6.1 Vocabulary Size

Improving the security of reCaptcha's audio captchas is critical since passing an auditory captcha bypasses reCaptcha entirely. Checking for simple impossibilities, such as moving the mouse faster than a human possibly could, clicking the precise center of an element, or solving captchas faster than they can be heard would help bolster security against less sophisticated bots. In the immediate term, reCaptcha resiliency can be improved by broadening the vocabulary of sound bites beyond just digits. Our system of phonetic mapping relies on this vocabulary of 10 elements; if a vocabulary of all spoken words was used instead, it would be much more difficult to detect or even correct errors from the speech to text services using a simple phonetic mapping.

The overall success of the offline MFCC captcha solver underscores the problem of using such a small vocabulary size of only digits; as a bot needs to decide only among 10 options, even relatively simplistic speech recognition algorithms designed to run on minimal hardware constraints (old mobile phones) are capable of working well on average. For the security of the auditory captcha to hold, a much more diverse vocabulary of sounds is needed. This conclusion is not new; in response to a similar conclusion drawn by Tam et al. in 2008, the reCaptcha team redesigned their audio captchas completely to include a full spoken sentence or a collection of spoken words, a far more difficult clip to defeat [27]. However, after migration to the new image/audio reCaptcha, this change seems to have been dropped.

6.2 Distortion

Adding background noise or background chatter between digits to would make segmentation more difficult. (For example, Amazon Web Services implements this for their audio captcha.) However, to make the actual digits to identify stand out to the user, this background noise often needs to be quieter than the actual digits to identify, making it susceptible for easy removal or identification with a low-pass filter. Similar analysis can also remove high-frequency noise, general distortion, or chatter from periods without digits, making segmentation still relatively easy. Recent studies support this, and have shown that background noise is largely ineffective at impeding automatic speech recognition [20]. Other audio based tasks that were previously thought to only be doable by humans, such as song identification, have also been solved with a very high degree of accuracy [30].

As speech recognition systems get stronger and more resilient to background noise and distortion, maintaining the security of the audio captcha in its current form will prove more difficult. In 2008, Tam et al. suggested taking advantage of the human ability to understand distorted audio through context clues. By replacing random isolated words or digits with familiar phrases or sentences, the idea was that humans could decipher distorted utterances through familiarity with the phrase or contextual clues in the sentence [26]. However, we believe this is not a viable longterm solution, particularly given the advancements in neural networks and other advanced speech recognition services taking advantage of contextual clues [31, 17, 19].

6.3 Auditory Instructions

We suggest a different approach: giving auditory instructions to the user to follow. For example, instead of challenging the user with a sound to transcribe (or identify), an automated series of tasks could be read to the user. Even simple tasks like "move your mouse upwards," "type a word," "type the following word but don't type this word," etc. could prove far more difficult for a computer to solve, as it adds an additional layer to the challenge. Beyond blind segmentation and transcription, understanding the challenge commands would require a host of custom processing, particularly since it would be relatively easy for the captcha system to build up a large vocabulary of similar commands. Such a challenge is also more truly akin to a Turing test. Although making audio captchas more difficult for machines has been traditionally associated with also making it harder for humans, we believe this higher level test would be easy for humans to solve while proving difficult for bots.

7 Conclusion

We demonstrate a low-resource, high accuracy defeat of the reCaptcha system. After successfully running against over 450 captchas, it can defeat reCaptcha with over 85% accuracy. By ensembling the results from free, non-specialized, online speech recognition services, unCaptcha demonstrates that it is far cheaper to mount a highly successful attack on reCaptcha than previously thought. This carries major consequences; reCaptcha is used by hundreds of thousands of sites as one of the foremost lines of defense against service abuse and automated account creation. We make a number of suggestions to mitigate our attack and reaffirm the security of the reCaptcha, providing a third line of computational defense against bots for auditory captcha systems. In the future, we look to explore the efficacy of our attack methodology on other auditory captcha systems, including Amazon's auditory captcha, eBay's, and more. We also seek to explore if this similar low-resource methodology of using free, online public services can be used to attacking reCaptcha's image captcha.

The code and data from this paper are publicly available at:

`https://uncaptcha.cs.umd.edu`

Responsible Disclosure

The day after we had a working prototype of unCaptcha (mid-March 2017), we emailed the Google BotGuard team to inform them of the weaknesses we discovered. They reported that they were aware of the issues and were working on a fix. As of mid-July 2017, minor updates to reCaptcha had been made tightening the security of the audio captcha specifically. Selenium detection now works across both Mac OS X and Linux on the higher two security levels for the audio captcha, but Selenium is still effective on the lowest security level. Interestingly, these changes were made specifically to the audio captcha: we can engage the image captcha without issue using Selenium across all security levels. The rest of our attack methodology still works, suggesting the only current barrier to attack is evading browser-automation detection. We have reported this to Google.

Acknowledgments

unCaptcha was originally prototyped at Bitcamp, a hackathon at the University of Maryland; we thank the Bitcamp organizers. We also thank our shepherd, Christina Pöpper, and the anonymous WOOT reviewers for their helpful feedback. This work was supported in part by NSF grants CNS-1409249 and CNS-1564143.

References

[1] E. Bursztein, J. Aigrain, A. Moscicki, and J. C. Mitchell. The end is nigh: Generic solving of text-based captchas. In *WOOT*, 2014.

[2] E. Bursztein and S. Bethard. Decaptcha: Breaking 75% of eBay audio CAPTCHAs. In *WOOT*, 2009.

[3] K. Chellapilla, K. Larson, P. Y. Simard, and M. Czerwinski. Building segmentation based human-friendly human interaction proofs (HIPs). In *Human interactive proofs*, pages 1–26. Springer, 2005.

[4] J. R. Douceur. The Sybil attack. In *IPTPS*, 2002.

[5] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by internet-wide scanning. In *CCS*, 2015.

[6] ETSI Standard. Speech processing, transmission and quality aspects (STQ); distributed speech recognition; frontend feature extraction algorithm; compression algorithms. *ETSI ES*, 201(108):V1, 2003.

[7] T. Ganchev, N. Fakotakis, and G. Kokkinakis. Comparative evaluation of various MFCC implementations on the speaker verification task. In *SPECOM*, 2005.

[8] Google reCAPTCHA. https://www.google.com/ recaptcha/intro/.

[9] N. M. Hamza zer, Blent Sankur. Robust audio hashing for audio identification. *EUSIPCO*, 2004.

[10] S. Hjsgaard. *The Capture-Recapture Model for Estimating Population Size*. Department of Mathematical Sciences, Aalborg University, 2014.

[11] S. Khattak, D. Fifield, S. Afroz, M. Javed, S. Sundaresan, V. Paxson, S. J. Murdoch, and D. McCoy. Do you see what I see? Differential treatment of anonymous users. In *NDSS*, 2016.

[12] G. Kochanski, D. P. Lopresti, and C. Shih. A reverse turing test using speech. In *INTERSPEECH*, 2002.

[13] S. Li, S. A. H. Shah, M. A. U. Khan, S. A. Khayam, A.-R. Sadeghi, and R. Schmitz. Breaking e-banking CAPTCHAs. In *ACSAC*, 2010.

[14] L. Muda, M. Begam, and I. Elamvazuthi. Voice recognition algorithms using mel frequency cepstral coefficient (MFCC) and dynamic time warping (DTW) techniques. *arXiv preprint arXiv:1003.4083*, 2010.

[15] neuroradiology. InsideReCaptcha. https://github.com/neuroradiology/InsideReCaptcha, 2014.

[16] M. Prince. The trouble with Tor. CloudFlare Blog (https://blog.cloudflare.com/the-trouble-with-tor/), Mar. 2016.

[17] J. Rao, F. Ture, H. He, O. Jojic, and J. Lin. Talking to your TV: Context-aware voice search with hierarchical recurrent neural networks. *arXiv preprint arXiv:1705.04892*, 2017.

[18] ReBreakCaptcha: Breaking Googles ReCaptcha v2 using.. Google. https://east-ee.com/2017/02/28/rebreakcaptcha-breaking-googles-recaptcha-v2-using-google/, Feb. 2017.

[19] Y. Ren, Y. Zhang, M. Zhang, and D. Ji. Context-sensitive twitter sentiment classification using neural network. In *AAAI*, 2016.

[20] S. Sano, T. Otsuka, and H. G. Okuno. Solving Google's continuous audio CAPTCHA with HMM-based automatic speech recognition. In *IWSEC*, 2013.

[21] R. Santamarta. Breaking Gmails Audio Captcha. Archived: https://web.archive.org/web/20131214022419/ http://blog.wintercore.com/2008/03/05/breaking-gmails-audio-captcha/, 2008.

[22] Simplecaptcha. http://simplecaptcha.sourceforge.net.

[23] S. Sivakorn, J. Polakis, and A. D. Keromytis. I'm not a human: Breaking the Google reCAPTCHA. *Black Hat*, 2016.

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

[25] Stiltwalker: Nucaptcha, Paypal, SecurImage, Slashdot, Davids Summer Communication. http://www.dc949.org/projects/stiltwalker/, 2012.

[26] J. Tam, J. Simsa, D. Huggins-Daines, L. Von Ahn, and M. Blum. Improving audio captchas. In *SOUPS*, 2008.

[27] J. Tam, J. Simsa, S. Hyde, and L. Von Ahn. Breaking audio captchas. In *NIPS*, 2008.

[28] Using deep learning to break a captcha system. https://deepmlblog.wordpress.com/2016/01/03/ how-to-break-a-captcha-system/, Jan. 2016.

[29] O. Varol, E. Ferrara, C. A. Davis, F. Menczer, and A. Flammini. Online human-bot interactions: Detection, estimation, and characterization. *arXiv preprint arXiv:1703.03107*, 2017.

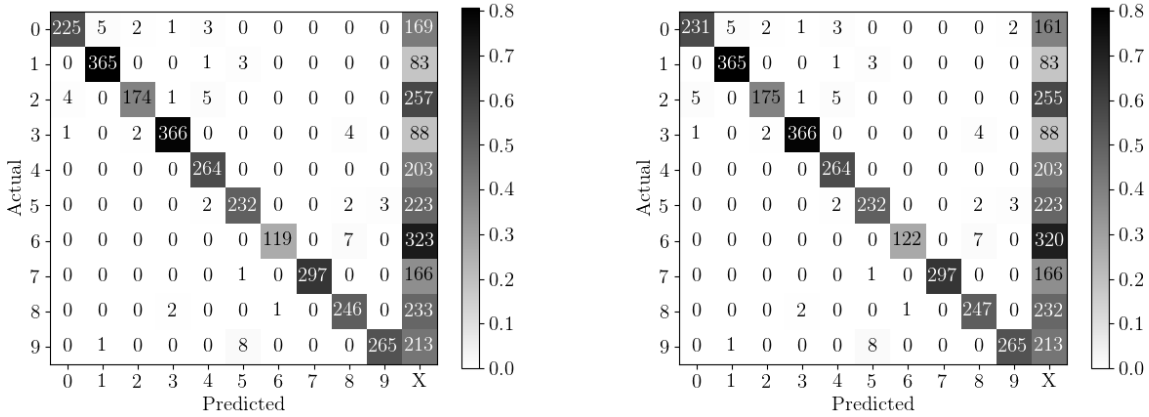
[30] A. Wang. The Shazam music recognition service. *Communications of the ACM*, 49(8):44–48, Aug. 2006.

[31] B. D. Whissel. Speaker-independent phoneme recognition in a continuous speech context using time-delay feed-forward neural networks. http://bretwhissel.net/portfolio/the_paper.pdf.

[32] J. Yan and A. S. El Ahmad. A low-cost attack on a Microsoft CAPTCHA. In *CCS*, 2008.

Appendix A Confusion Matrices

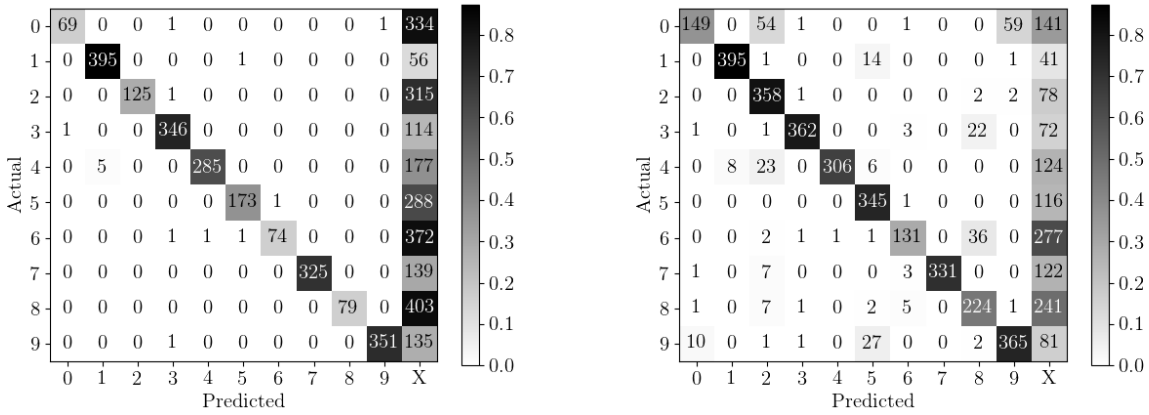
We present below the confusion matrices for each of the six services that unCaptcha uses. When the service is unable to predict a digit, we denote its predicted value as "X".



(a) Without phonetic mapping

(b) With phonetic mapping

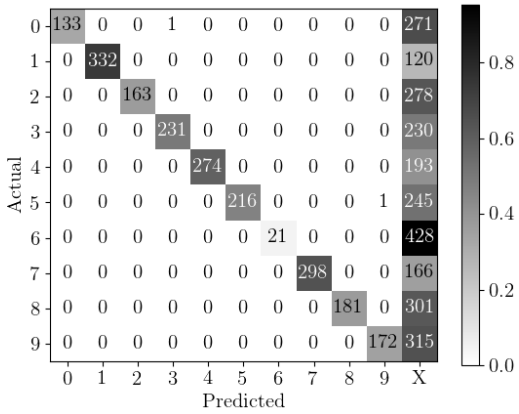
Figure 5: Google Cloud



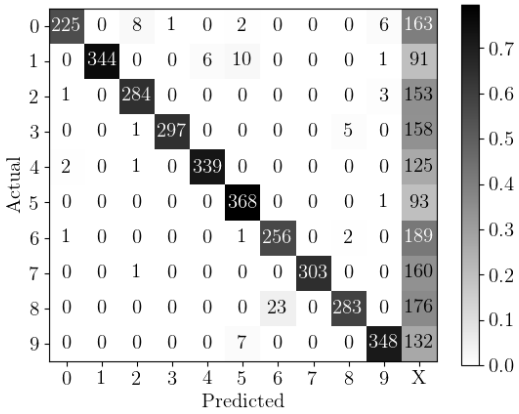
(a) Without phonetic mapping

(b) With phonetic mapping

Figure 6: WIT

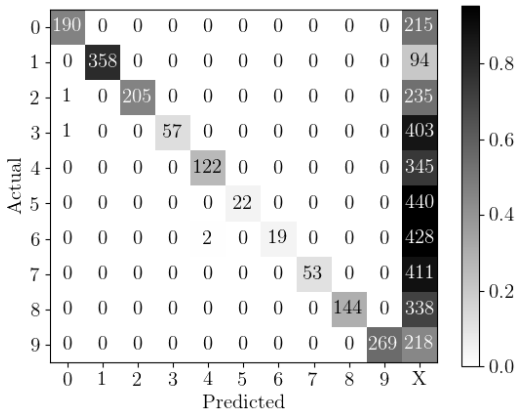


(a) Without phonetic mapping

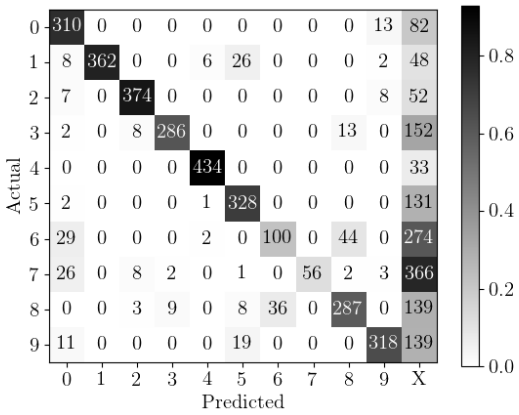


(b) With phonetic mapping

Figure 7: Bing

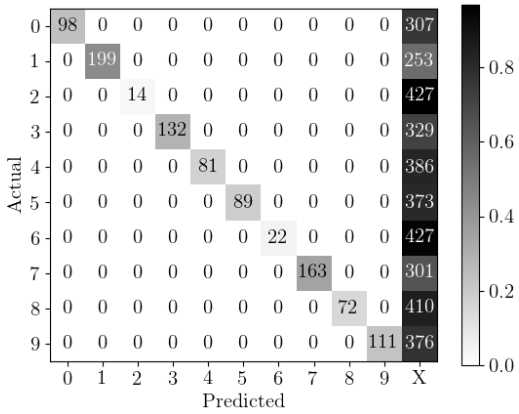


(a) Without phonetic mapping

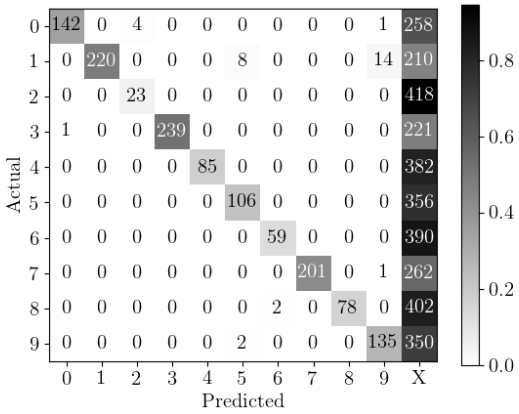


(b) With phonetic mapping

Figure 8: IBM

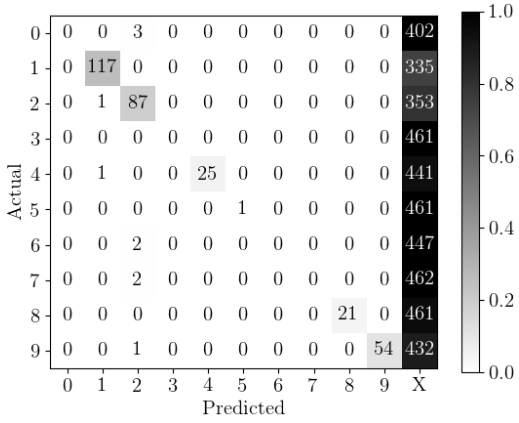


(a) Without phonetic mapping

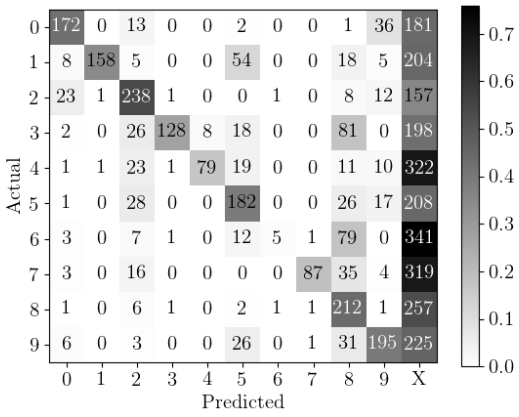


(b) With phonetic mapping

Figure 9: Google



(a) Without phonetic mapping



(b) With phonetic mapping

Figure 10: Sphinx