Improving Auditory CAPTCHA Security

Sonja Bohr and Andrea Shome

August 6, 2008

Background

The notion of a machine imitating human intelligence was first addressed as early as 1950 by English mathematician and logician Alan Turing. Acknowledged as the father of modern computing, Turing recognized that computers might eventually be able to imitate human thought in very convincing ways. Therefore, he suggested what is now known as the Turing test, where a human converses with a computer without seeing it. If the human is convinced by the computer's answers that it is human, then the machine passes the test and is deemed to have some level of human-like intelligence. The idea of a reverse Turing test, where a computer attempts to differentiate between a human and a computer, arose during the late 1990s when computer programs began to imitate humans in order to misuse the resources of internet-based systems. Tests developed to differentiate these programs from real humans took the form of what would come to be known as CAPTCHAs.

The term CAPTCHA, which stands for Completely Automated Turing Test to Tell Computers and Humans Apart, was coined in 2000 by a group of professors at Carnegie Mellon University. These professors – Luis von Ahn, Manuel Blum, Nicholas Hopper and John Langford – were working on research for Yahoo developing the first of these programs when they first used the term. Later, Luis von Ahn and Manuel Blum would be credited with the invention of the first CAPTCHAs – ones that were soon adopted by Yahoo and later Google and Hotmail.

In a general sense, CAPTCHAs are used to prevent automated software ("bots") from abusing the resources of various systems. They arose from a need to reduce or eliminate mass spam on websites and in webmail, to prevent bots posing as humans from entering chat rooms, to eliminate voting fraud on online polls, and most importantly to protect sensitive data that could be accessed by a simulated user. A bot might use an email provider to create multiple fake email accounts in order to send spam. Similarly, it could use a social networking site to spam or otherwise annoy the other users. Or it might simply create fake accounts in order to use up system resources. CAPTCHAs can be used to prevent this misuse of resources by requiring that users prove they are human before using a system.

The first CAPTCHAs were visual, requiring users to enter a series of letters or numbers based on a given distorted image. The image was meant to create a barrier for simulated users because a program would presumably not be able to interpret the image in order to enter the correct string of characters. Visual CAPTCHAs, however, are an accessibility barrier to users who are visually impaired or have cognitive disabilities such as dyslexia. Auditory CAPTCHAs attempt to provide accessibility where visual ones cannot.

Since auditory CAPTCHAs are necessary for every website which must provide access to all users, it is important that they be at least as secure as their visual counterparts. A hacker wishing to gain access to a website will use the easiest means possible – so if an auditory CAPTCHA is easier to crack than the visual one, the website is as vulnerable as if the visual one did not exist. Besides this necessity for auditory CAPTCHAs to get up to par, they also have the potential to become much more difficult to break than visual ones (Balaji).

It is important to understand the advantages as well as the vulnerabilities of auditory CAPTCHAs. For example, a bot might crack an auditory CAPTCHA by filtering out the excess noise from the sound clip in order to identify the numbers present. But a computer identifies speech by examining the frequency of that speech at different points in time. It may be stumped, then, if a human voice is mixed with background noise that which shares similar frequencies; it can't tell which bits of frequency belong to the voice and which belong to the background noise. A human user may be able to understand the voice in that same situation, because the human brain has specialized ways to segment sound.

For instance, the "cocktail party" effect is a well-known example of the human mind;s surprising ability to separate sound from one source among many ("Cocktail Party Effect"). The analogy is that someone at a noisy party can understand what their friend is saying to them, and if spoken to from across the room, can respond to that as well. A computer would likely be unable to accomplish this since it would be confused by so many voices of similar frequency and volume. This is the idea behind auditory CAPTCHAs – to utilize the human brains ability to segregate sounds of similar frequencies from one another.

In our research, we looked at some of the techniques currently used in some of the most advanced CAPTCHAs. We attempted to duplicate some of these techniques and test how well they stand up to an automatic speech-recognizing program (ASR). We also formulated some of our own distortion methods to observe how well they fared against the ASR.

Hypotheses

The auditory CAPTCHAs used by the companies of Google[1] and Facebook[2] to protect their websites represent some of the most advanced distortion techniques used today. In listening to these auditory CAPTCHAS and ones from other popular websites, we observed a variety of techniques used to distort the spoken

[1] https://www.google.com/accounts/NewAccount

[2] http://www.facebook.com/r.php (click Try an Audio CAPTCHA)

numbers. For example, many of the numbers sounded as if a reverberation effect had been added- that is, the speaker sounded as if they were in an empty room. Other spoken digits sounded like a tempo or pitch change had been used. Both Google and Facebook intermixed both male and female voices in one CAPTCHA file, and added background noises derived from human speech. For instance, Facebook's CAPTCHA background noise was simply reversed human speech. Many CAPTCHAs also featured false leads - spikes of sound which were as loud as one of the spoken digits, but to the human ear were clearly reversed numbers or gibberish. Our work focused on the distortion to the numbers themselves but not the false numbers intended to fool an automatic speech recognizer.

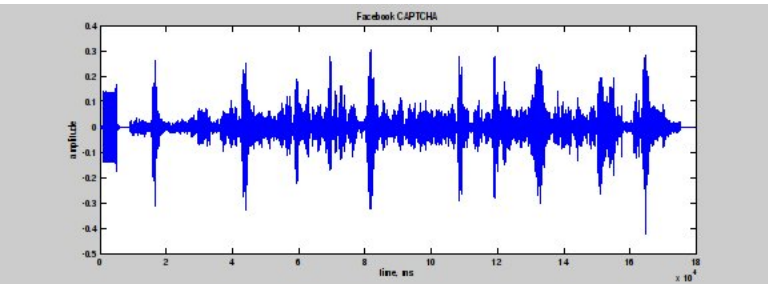

Figure 1: An example Facebook CAPTCHA in its entirety.

We set out to test our own versions of some of these observed distortion techniques, to determine which were most effective at making spoken digits unrecognizable by an ASR while still recognizable to a human. We decided to test a variety of background sounds derived from human speech, as well as other methods such as cutting out part of the speech signal of the digit. The specific techniques are discussed later in this paper. We expected each of our distortions to decrease the recognition rate of a speech recognition program when that program was trained on normal speech, although we did not know by how much. It was also our hope that combinations of these techniques would decrease the ASR's recognition rate to lower than those found with Facebook or Googles CAPTCHAs.

Setup and Methods

We used the speech recognizer HTK (Hidden Markov Model Toolkit)[3] to test the power of our various distortion techniques. This speech recognizer, developed by the Cambridge University Engineering Department, is a freeware program that can be trained on a number of sound files to recognize the difference between different words, or, in our case, digits. It can then be tested on new sound files, outputting which of these test words were identified correctly. In this experiment, it uses a six-step Markov modeling technique in order to create

[3]http://htk.eng.cam.ac.uk/

guesses and provide recognition statistics for the input files. In order to use this speech recognizer, we needed to obtain a large database of spoken digits to train and test on.

To reflect the wide range of voices used to create advanced CAPTCHAs, we chose as our training corpus a collection of spoken digits obtained from a Rutgers Universitys online database. There were approximately 2000 separate files in this database – almost 200 recordings of each digit zero through nine, spoken by different individuals. The people recorded were men and women, with normal, Southern and Midwestern accents, speaking at a variety of speeds and intonations. As a result, the digits were understandable but sometimes included accents such as dropping the "'r" of "four". Our hope was that by training on a wide variety of speaking styles, our speech recognizer would likewise be able to identify the different speaking styles featured in the online CAPTCHAs.

Because it is necessary for the HTK speech recognition program to train and test on files that have the same sampling frequency, we used MATLAB to re-sample all of our testing and training files to 8kHz. The sound files from the online CAPTCHAs were downloaded with an 8kHz sampling frequency so even though some of our other files were originally recorded at 16kHz, these were re-sampled to the lower frequency. We chose to re-sample everything to the lower frequency because in going from a lower to a higher sampling rate, the missing information would be interpolated; on the other hand, going from a higher to a lower sampling rate produces the same result as if the sound had originally been recorded at the lower rate.

The Rutgers database sound files were recorded so that each file featured a digit padded with silence at the beginning and the end. However, we realized that by training HTK on these files, we were falsely including the silence as part of the digit itself, causing recognition rates to drop when testing on other files that had different amounts of silence padding. In order to get the most accurate results it was therefore necessary to trim the silence from our database files. To accomplish this, MATLAB was used to automatically remove the silence from the sound files. This approach was imperfect, however, and many of the files had to be trimmed by hand by looking at a time-waveform plot in Audacity and visually identifying where the speech stopped or leveled off. Fig.2 shows an example of a file before trimming.

Since HTK trains by recognizing differences between two different digits, we also did not want the loudness of the digits to be a possible difference on which HTK would mistakenly train. For this reason the sound files used for training and testing needed to be normalized. The sounds were linearly scaled such that the maximum amplitude of the sound file was set to .9, and the average value of all the samples was subtracted from each sample, so that the amplitude would fluctuate around (the new average of) zero. By normalizing each sound in this way, we ensured that HTKs training was not affected by variations in the loudness or average of the speech samples.

In order to define a human intelligibility level we also had to test all of our distorted files by listening to them individually. To accurately test our ability to recognize the numbers after distortion, we created a script to play randomly

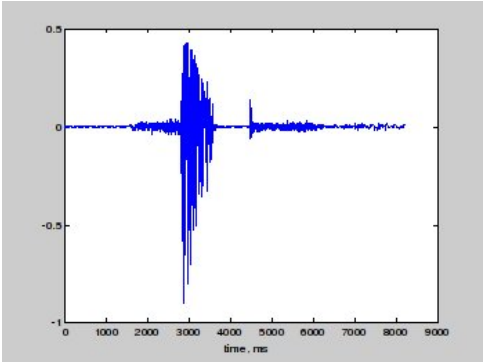

Figure 2: A typical digit before trimming.

chosen files from the distorted testing database. We then entered what number we thought we heard, and the output indicated whether or not our guess was correct. Based off this information we were able to identify intelligibility limits – when we started guessing incorrectly we knew the distortions were too strong. A batch of distorted digits was judged to be "intelligible: if we got no more than approximately 3% incorrect. However, this was an approximation – and some of the shorter numbers such as six were much more prone to becoming unintelligible than others.

This method for determining the point of ""intelligibility"" for these CAPTCHAs could be improved by testing with more, unbiased subjects. In this project we relied on only two test subjects (ourselves) to listen to the manipulated digits and determine their intelligibility. But it is possible that an average user who had not been working with the sound files as much would not be able to understand the distorted digits as well. In the future, more test subjects should be used to get an accurate measure of intelligibility, especially ones who have not had previous exposure to the numbers database or the distortion techniques. It would also increase precision to record the exact percentage of correct answers for each subject.

Distortion Techniques

As mentioned above, our distortion techniques consisted mainly of adding background sound derived from human voices and other sources, or manipulating the sound file itself. Some of the first ways we attempted to manipulate the sound files included changing the tempo and/or pitch. Speeding up or slowing down the file would increase or decrease the pitch, respectively, whereas changing the tempo would affect the duration of the number but not the pitch. It was found, however, that these changes did not change HTK's recognition significantly from the original, undistorted files, so the technique was not pursued further.

Another technique was to adjust a file by cutting out various parts of its

signal. Kochanski et. al. suggest that cutting out 60 ms of speech every 100 ms of a file should decrease an ASRs recognition rate to nearly zero percent, while not affecting human recognition. We tested this claim by cutting out the speech from digits in that ratio, and also by breaking up the digit into 8ms intervals and randomly removing half of the intervals. Besides removing parts of the signal in this way, we also attempted a similar technique where we replaced the deleted portions with white noise. We hoped that the human ear would be able to filter out this background noise and ignore it, but that the random signal would confuse HTK.

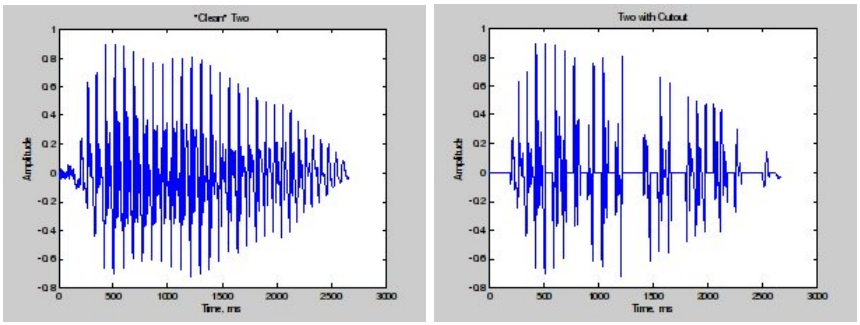

Figure 3: The number two, before and after the "cutout" technique is applied.

At first, these techniques seemed to degrade HTKs recognition significantly. However, when identifying a file, HTK uses windows of a specified duration (typically 25 ms) to look at small intervals of the file, and it was found that as long as the window size was somewhat greater than the size of the removed portions, recognition did not decrease dramatically. Even with as large an interval as 60ms cut out, we were able to increase the window size such that the recognition rate did not degrade very much. Because HTK attempts to fill in the space where the signal is cut out (guessing what should be there), larger rather than smaller cutout intervals would have made HTK's recognition worse. Intervals larger than 60ms, however, would have degraded human recognition too much.

Another form of distortion that we attempted was to create an echo effect. An echo is created by taking a sample from a particular point in the sound file of a digit, then adding it back into the file a set number of samples later. This is done for every sample over the entire file. A double echo is created by performing this same process, but with two different intervals for samples, and a triple echo with three intervals, etc. We expected this distortion technique to achieve low recognition from HTK, since the numbers in the Facebook and Google CAPTCHAs sounded reverberated similar to echo. The results were poor, however – HTK had high recognition rates, within about 90%, for most all of the echo files we tested.

We also tried a similar reverberation technique where we applied two dif-

ferent pre-made MATLAB scripts[4] designed to add either normal or "natural" sounding reverberation. However, as with echo, after several trials it was found that these reverberation techniques did not have a significant impact on HTKs recognition rate so they ere not pursued further.

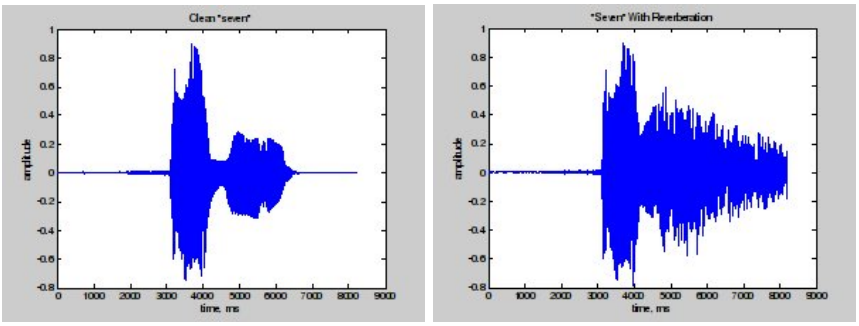

Figure 4: A digit before and after reverberation is applied.

As discussed earlier, the human brain has the ability to easily distinguish human speech from a cacophony of other voices, whereas this task is often more difficult for a computer. For this reason, we also decided to use background noises which themselves featured human speech, in attempt to mask each digit. Especially when the background noise shares a similar frequency range to the spoken digit, it can be harder for HTK to differentiate the two. Whereas the Facebook and Google CAPTCHAs use the reversed speech of one or two people speaking English, we decided to test backgrounds featuring more voices and speech of different languages. For each background noise, we downloaded the clips, analyzed them in Audacity[5] to pick out the most promising-sounding segments, and superimposed them on the clean speech files. Each segment was individually normalized and then the entire digit with distortion was again normalized.

Most of our background testing files were found from the audio file database Soundsnap[6], a "platform to find and share free sound effects and loops" (www.soundsnap.com). From this collection of original sounds we downloaded many recordings of "random" noises. One of these background files types we tested was "cheer", which consisted of recordings of people cheering at sporting events. This background type featured a wide range of frequencies, especially in the upper register, produced by the many people cheering. Another background type was "chatter", which sounds like a lot people talking in a crowded place like a restaurant. This background type has enough speech that no one individual can be understood, and featured most voices lower in pitch than those in the "cheer" files. We hypothesized that the broad spectrum of frequencies found in the cheer and chatter backgrounds would help prevent hackers from simply using a low pass

[4]http://www-ece.eng.uab.edu/DCallaha/courses/DSP/Projects%202001/T4/sound.htm
[5]http://audacity.sourceforge.net/
[6]http://www.soundsnap.com/

or high pass filter to extract the number.

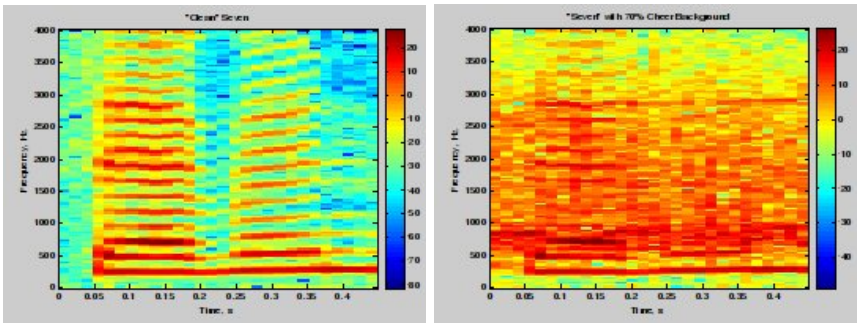

Figure 5: A digit before and after ““cheer”” is added.

Another type of background noise was created by combining ten random numbers from our speech database. The resulting sound was not recognizable as any single number, but was rather a confusing mix – we called this technique “babble”. Since this background noise was created using speech samples from the same pool used to create the testing files, this background was like cheer and chatter in that it featured many voices, but the frequency range was more similar to that of the digits themselves. We also tried the process of reversing the spoken digits before combining them, in an attempt to prevent any recognition of the numbers used to make the background noise.

We also used this “babble” background in another way. We realized that, for human recognition, hearing the end of a number is not as important as hearing the first part of the number. This is because the digits zero through nine all begin with different sounds, and humans can recognize a digit before it is even finished. Knowing that we could take advantage of this ability, the babble noise was added to the last 75% of the digits in an attempt to mask the end of the digit but leave the beginning with cleaner speech.

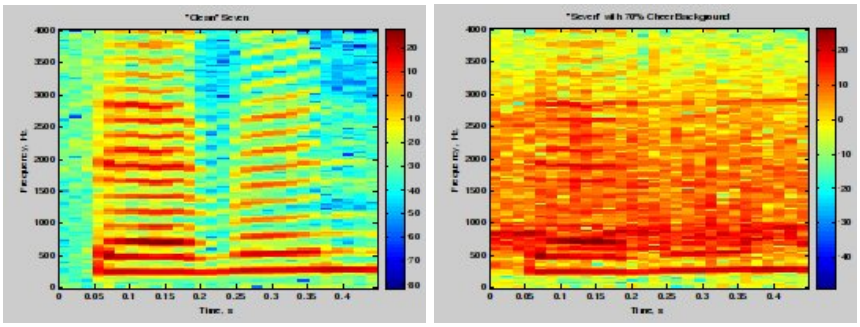

Figure 6: A number file before and after end-babble is applied.

Hypothesizing that background of another language would be less distract-

ing to a human user than an English background, we decided to add foreign speech to our digit files. We used the AT&T Text-to-Speech converter[7] to convert several paragraphs of French text into sound files and later added them on top of the digits. We used this computer-generated speech because there was a variation in speakers; we were able to choose between three voices, male and female, rather than just one persons recordings. This additional variation is important when working with HTK because it reduces the ability of the computer to recognize speech patterns. The French distortion was similar to babble in intelligibility levels for humans, but we hoped that the spikes in loudness at random times would confuse HTK.

Because Facebook's CAPTCHAs achieved significantly lower recognition rates than Googles, our goal was to create a distortion which could achieve even lower rates than Facebooks. We tried combining variations of the many distortions listed above in order to do this. We found at least one combination of background noises that achieved this goal – mixing cheer, chatter, and end-babble techniques. The figure below displays the spectrogram of a number file before and after this combination technique is applied.

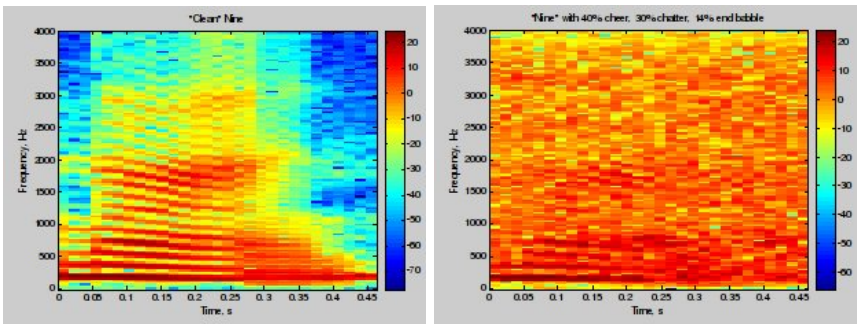

Figure 7: Our most effective distortion technique.

Results

The graph below (Fig.8) shows the trends of individual distortions as higher percentages of each distortion are applied to our testing database. As the "corruption percentage" increases, the graph shows which techniques better degrade HTKs ability to recognize the testing database. The second-to-last data point for all of the trend lines, corresponding to 100% corruption, shows the point which we deemed to be just-intelligible, based on our intelligibility testing. This means that the techniques with a lower recognition rate at this second-to-last point were overall more effective at masking the identity of the digits to the ASR while still maintaining human intelligibility.

[7] http://www.research.att.com/ ttsweb/tts/demo.php

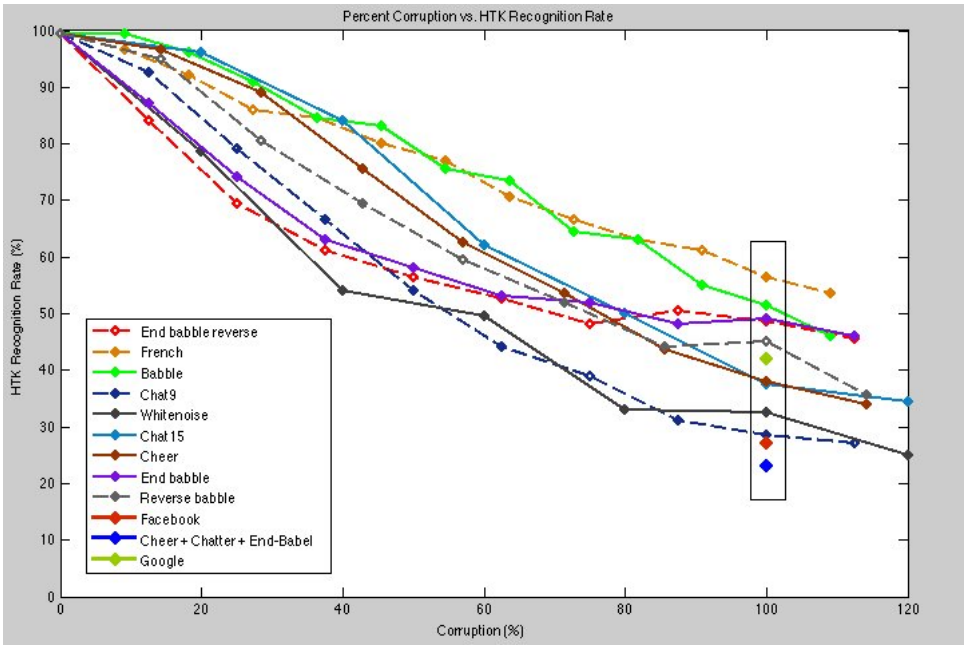

Figure 8: HTK recognition rates of our various distortion methods. The x-axis corresponds to an increasing corruption of the normal digits, where 100% corruption (the second-to-last data point) means that the digits are just still intelligible. Lower HTK recognition rates at this point indicate a more effective distortion.

The table below (Fig. 9) shows the recognition rates of plain speech, our successful distortion techniques, and digits from Google and Facebook CAPTCHAs. The second column lists recognition rates of the various distortions after HTK had been trained on the 2,000 clean files from the Rutgers database. The third column shows the recognition rates after HTK was trained on digits distorted using the same technique as those tested on. These percentages are important because a technique which is varied enough to withstand being trained on examples of itself is more resilient to attack than one which can be easily cracked by discovering or mimicking the distortion. Distortions with a higher percentage in the third column are more susceptible to this kind of attack than those with lower percentages.

This table shows that our most successful technique was a combination of cheer, chatter, and end-babble background noise. When trained on the Rutgers database of undistorted digits, this combination achieved a lower recognition rate (23%) than did the digits from Google (42%) or Facebook (27%) CAPTCHAs.

Type of Distortion	HTK Recognition (trained on database)	HTK Recognition (trained on self)
Cheer1 (40%) + chatter9 (30%) + end babble (14%)	23.0%	80.0%
Facebook CAPTCHA	27.0%	62.5%
Chat9 (70%) + end babble(14%)	28.0%	83.0%
Chat9 (80%)	28.5%	80.0%
Chat9 (60%) + end babble(14%)	33.5%	84.5%
Cheer (70%)	38.0%	73.0%
Google CAPTCHA	42.0%	98.0%
Babble (20%)	55.0%	90.0%
French (220%)	56.5%	88.0%
End Babble (20%)	74.0%	93.0%

Figure 9: HTK Recognition of our Distortion Techniques and CAPTCHAs. The second column shows the recognition percentages when HTK was trained on clean speech. The third column shows the recognition percentages when HTK was trained on digits distorted in an identical way as the testing files.

Conclusions

Our work was limited in several ways which could be improved on in the future. Even though our training database contained a variety of speech styles, pitches, and intonation, an even more extensive database would likely improve a speech recognizer's accuracy. The ASR could also be trained to recognize the silence before and after the spoken digits. This was something which proved too time-consuming for us, but training the recognizer to automatically distinguish between silence and the number would allow it to recognize digits which had not been hand-trimmed. Similarly, the recognizer could be trained to distinguish between a given background noise and the number.

Our work could probably be most improved by using more human testers to determine the intelligibility of different distortions. Although our judgment of intelligibility seemed to be about the same as that of others, more people should be used and their opinions averaged. There is certainly room for some bias on our parts, as we listened to these digits many times and became more familiar with them than the average person. Using more people, unfamiliar with the distortion techniques and digit database, should help get a more objective assessment of intelligibility.

We discovered that many of our results were very promising several combinations of distortions were able to achieve lower recognition rates than did Facebooks CAPTCHAs. One of these combinations, Cheer1 (40%) + chatter9

(30%) + end babble (14%), had only a 23% recognition rate by HTK (Fig.9). Variations of this combination such as increased babble and decreased cheer also achieved similar recognition rates. The variations most challenging to the human ear, though still intelligible, went as low as 20.5%.

When HTK tested on digits affected with solely white noise distortion it got only a 28% recognition rate. This may indicate that, at a level at which human intelligibility is unaffected, natural environment distortions (i. e. human-produced sounds) have a greater effect on confusing ASRs than do other noise distortions such as white noise.

Our "end-babble" technique was also surprisingly successful. Although on its own this technique did not degrade recognition as much as the CAPTCHAs did, end-babble was essential to create the low recognition rates obtained with our combination techniques.

Our testing not only showed how well these techniques fared against HTK when trained on clean speech, but how well they performed when HTK was trained on a database distorted in the same way. These figures are important because they indicate how difficult it would be for a hacker to crack a website's CAPTCHA by simply training using other examples the same website. Facebook's recognition rate when trained on itself was only 62.5%, whereas for Google the rate jumped to 98.0%. This shows that Google CAPTCHAs may be vulnerable to attackers which use samples derived from Google itself to train their ASRs. Our cheer + chatter + babble combos consistently got around 80.0% recognition rates when trained on themselves,though other distortions were somewhere in between – cheer, for example, got 73.0% against itself. None of our distortions achieved as low a recognition rate against themselves as did Facebook, showing that these techniques are weaker than Facebook's CAPTCHAs in this respect. However, using different techniques for different digits in the same CAPTCHA might alleviate this vulnerability.

The 4% discrepancy between the recognition rates of our combination distortions and those of Facebook CAPTCHAs is not great enough to show that our technique is necessarily superior to those currently used in advanced CAPTCHAs. However, the similarly low recognition rates suggest that our technique can also be used successfully. Perhaps in combination with existing methods, our techniques of background noise to mask the digits could be used to increase the security of auditory CAPTCHAs.

Balaji, V. “Defeating Bots with CAPTCHAs.” Palisade. December 2005 http://palisade.plynt.com/issues/2005Dec/captchas/.

“CAPTCHA.”” Wikipedia, The Free Encyclopedia. 8 Jun 2008, 12:53 UTC. Wikimedia Foundation, Inc. 9 Jun 2008.

”Cocktail party effect.” Wikipedia, The Free Encyclopedia. 21 May 2008, 14:40 UTC. Wikimedia Foundation, Inc. 9 Jun 2008.

Kochanski, Greg, Daniel Lopresti and Chilin Shih. “A reverse Turing Test Using Speech.” Bell Laboratories, Lucent Technologies, September 2002.

Daniel Lopresti,Henry Baird,“”Human/machine differentiation””, in AccessScience@McGraw-Hill, http://www.accessscience.com,DOI 10.1036/1097-8542.YB052140.

Raman, T.V. “Audio CAPTCHAs when visual images are unusable.” 28 Nov 2006. The Official Google Blog. 5

June 2008 http://googleblog.blogspot.com/2006/11/audio-captchas-when-visual-images-are.html. Santamarta, Rubn. “Breaking Gmail’s Audio Captcha.” Wintercore Labs. 5 March 2008. http://blog.wintercore.com/?p=11.