Attacks as Defenses: Designing Robust Audio CAPTCHAs Using Attacks on Automatic Speech Recognition Systems

Hadi Abdullah[1*], Aditya Karleka[2], Saurabh Prasad[2], Muhammad Sajidur Rahman[2], Logan Blue[2], Luke A. Bauer[2], Vincent Bindschaedler[2], Patrick Traynor[2]

Visa Research, Atlanta, GA[1] University of Florida, Gainesville, FL[2]
Email: habdulla@visa.com, {akarlekar, sprasad1, rahmanm, bluel, lukedrebauer, vbindsch, traynor} @ufl.edu

***Abstract*—Audio CAPTCHAs are supposed to provide a strong defense for online resources; however, advances in speech-to-text mechanisms have rendered these defenses ineffective. Audio CAPTCHAs cannot simply be abandoned, as they are specifically named by the W3C as important enablers of accessibility. Accordingly, demonstrably more robust audio CAPTCHAs are important to the future of a secure and accessible Web. We look to recent literature on attacks on speech-to-text systems for inspiration for the construction of robust, principle-driven audio defenses. We begin by comparing 20 recent attack papers, classifying and measuring their suitability to serve as the basis of new "robust to transcription" but "easy for humans to understand" CAPTCHAs. After showing that none of these attacks alone are sufficient, we propose a new mechanism that is both comparatively intelligible (evaluated through a user study) and hard to automatically transcribe (i.e., $P(\text{transcription}) = 4 \times 10^{-5}$). We also demonstrate that our audio samples have a high probability of being detected as CAPTCHAs when given to speech-to-text systems ($P(\text{evasion}) = 1.77 \times 10^{-4}$). Finally, we show that our method can break WaveGuard, a mechanism designed to defend adversarial audio, with a 99% success rate. In so doing, we not only demonstrate a CAPTCHA that is approximately four orders of magnitude more difficult to crack, but that such systems can be designed based on the insights gained from attack papers using the differences between the ways that humans and computers process audio.**

I. Introduction

CAPTCHAs (Completely Automated Public Turing Test to tell Computers and Humans Apart) are a nearly ubiquitous security feature on the Web. These challenge-response puzzles attempt to regulate access to resources (e.g., account sign-up, service abuse, etc) through tasks that are simple for human beings to solve but extremely difficult for machines. CAPTCHAs now come in a wide array of formats, from the transcription of images of character strings [31], [45] and image recognition [55], [50], to noisy audio samples [57], [81]. This variety benefits inclusive computing, and ensures that users with access limitations (e.g., vision impairments/blindness) can still use the majority of the Web without being limited by security features [83].

Unfortunately, what constitutes a challenge for computers has changed drastically since the inception of CAPTCHAs. Nowhere has this shift been more obvious than in audio CAPTCHAs, where advances in the power of neural network-driven speech-to-text systems allow for the real-time, automated breaking[1] of audio CAPTCHAs [34]. However, audio CAPTCHAs cannot simply be discarded for other available methods, and remain the fallback/supplemental methods for many other techniques. This explains why Google's reCAPTCHA, one of the most CAPTCHA services on the Internet, offers audio CAPTCHA challenges (despite the availability of other methods such as math challenges, open-ended questions). Accordingly, *stronger* and more *principled* techniques are required for their construction.

We argue that audio CAPTCHAs are so easily breakable because of a fundamental misunderstanding of their adversary. That is, while adding noise to audio generally decreases comprehension by humans and unintelligent systems, it fails to model subtle differences between the two parties. The difference between how humans and computers perceive audio has been exploited repeatedly in recent literature attacking Automatic Speech Recognition (ASR) systems (e.g., Amazon's Alexa, Apple's Siri, etc) [25], which allows adversaries to create starkly different audio and transcriptions. Such approaches could potentially be used to exploit weaknesses in speech-to-text systems to make humans and computer observers "hear" dramatically different messages.

In this paper, we leverage the large body of literature on attacks against ASRs as a means of creating a *defense* in the form of a robust audio CAPTCHA generation algorithm. That is, instead of simply adding noise to audio, we modify existing audio samples in ways that significantly decrease machine transcription accuracy without impacting human in-

[*] This work was done while this author was at the University of Florida.

[1] Attack example from previous work: https://www.youtube.com/watch?v=wXrTQzskJLE

Network and Distributed System Security (NDSS) Symposium 2023
27 February - 3 March 2023, San Diego, CA, USA
ISBN 1-891562-83-5
https://dx.doi.org/10.14722/ndss.2023.24243
www.ndss-symposium.org

telligibility. However, these attacks were originally designed to force target ASRs to output specific transcriptions for perturbed "adversarial" audio samples. Therefore, their use as defenses prevent an adversary's unknown ASR from correctly transcribing — and therefore breaking — a CAPTCHA brings forth several fundamental questions, such as: *Can these attacks be successfully used as a defense?* If so, *what properties of such attacks are required for the defense to be effective? Which among the plethora of existing techniques are the most suitable? And what approaches (if any) are robust to adaptive adversaries who know that the defense is in place and can adjust their attacks accordingly?*

To answer these questions, we start our investigation by deriving requirements for an *attack that can be used as a defense*. We then turn to the literature and select 20 candidate attacks. Out of three promising candidates, we find that only one, the Kenansville attack [23], survives preliminary experiments because of its high transferability. Transferability — the ability of adversarial audio samples to fool models they are not directly crafted against — becomes an essential property. When an audio CAPTCHA is crafted, the defender does not know which target ASR an adversary may eventually use for transcription. Thus to be maximally robust, CAPTCHAs need to be mistranscribed by as many models as possible. Our design goal is to force ASR models to output an *empty string* when presented with our audio CAPTCHAs. This is ideal because it prevents the adversary from using phonetic mapping and statistical analysis techniques, in conjunction with incorrect or partial ASR outputs, to help break CAPTCHAs.

Further experiments reveal that the Kenansville attack is not robust against an adaptive adversary that simply adds noise to the CAPTCHA before transcribing it. Therefore, we propose Yeehaw Junction, a new algorithm that still achieves high transferability but is robust against an adaptive adversary. The Yeehaw Junction defense brings two key technical innovations: (1) the addition of Gaussian noise to the perturbed sample during CAPTCHA creation to increase robustness, and (2) a novel thresholding technique that clips large amplitudes of dominant frequencies in a way that confuses ASR models but still preserves the location of the frequency peaks so that the audio remains highly intelligible to the human ear.

We make the following contributions:

- **Principally-Designed CAPTCHA Construction:** The design of audio CAPTCHA has been largely ad hoc, relying on the experiential addition of cover noise to attempt to thwart bots. In contrast, our approach exploits weaknesses in modern ASR pipelines to intentionally target audio components that are ignored by our ears. We categorize 20 attacks, point to the desirable elements found in these works, and then demonstrate weaknesses in attempting to directly use any of these schemes for CAPTCHA creation in the face of an adaptive adversary.
- **Measure Robustness, Intelligibility, and Attack Detection:** We implement Yeehaw Junction and evaluate its suitability as a defense to produce audio CAPTCHAs. We find that an attacker has a low chance of correctly transcribing our audio CAPTCHAs ($P(\text{transcription}) = 4 \times 10^{-5}$). We then conduct a user study that shows that human transcription accuracy of Yeehaw Junction is larger than even Google's popular reCAPTCHA [13] audio CAPTCHA service. Finally, the distinctive profile of Yeehaw Junction audio CAPTCHAs provides ASR owners a 99.99% probability of detecting that their ASR is being used to break CAPTCHAs [34].
- **Robustness to Defenses:** We test our method against WaveGuard, which is one of the most effective defenses against adversarial examples in the audio domain. WaveGuard's goal is to undo the adversarial perturbations present in the audio sample, thereby allowing the ASR to output the original audio transcript instead of the adversarial one. We demonstrate that our adversarial CAPTCHAs can break all five defenses proposed in the WaveGuard paper with a probability greater than 99.99%. This means that there is no method in existing literature that bots can use to undo the perturbations introduced by our method. To our knowledge, ours work is the first to successfully break the WaveGuard defenses.

We note that the nature of adversarial systems means that other researchers may eventually break this new technique; however, the contribution of this work is deeper than a single defense. Rather, it is the *principle-driven* approach for designing transcription-resistant CAPTCHAs without making intelligibility worse. To assist other researchers in testing similar schemes, we will make all of our code and audio samples available at sites.google.com/view/attacksasdefenses/home

II. Background

A. Audio CAPTCHAs

A CAPTCHA protects resources against bots by generating and grading tests that humans can pass, but current computer programs cannot [3]. Audio CAPTCHAs are essential since they are one of the only ways for visually impaired users to complete a CAPTCHA. Without audio CAPTCHAs, a significant number of systems would either become vulnerable to adversaries or lockout visually impaired users from potentially critical services.

A good CAPTCHA must maintain accessibility to humans while protecting the resource from adversaries. However, not only can audio CAPTCHAs be broken by machines [37], [94], [34], [89] with high success rates (91%) [18], but the intended human users successfully pass them at a lower rate ($< 50\%$) [33] than the attack. These CAPTCHAs are especially difficult for their intended Visually impaired users since the audio playback can overlap with a screen reader software, the users have to remember the audio while navigating the page acoustically, and they are unable to compare the audio against the associated text CAPTCHA [33], [39], [49].

B. Automatic Speech Recognition systems (ASRs)

ASR systems take in speech samples and output transcriptions of the spoken content [26], [103]. They enable human-to-machine interaction, such as issuing commands to a digital voice assistant (e.g., Amazon Alexa [28]). This is not an easy task since the ASR system must be able to filter out

background noise and handle variations in accents and speech patterns. For this reason, ASR systems' processing generally consists of three stages: preprocessing, feature extraction, and decoding. Some models combine the feature extraction and the decoding steps into a single Machine Learning (ML) model, and others split the decoding step across several different ML models [29], but all still perform each of these steps in some manner.

Preprocessing: This phase in ASR systems consists of taking in the raw audio file and eliminating the background noise, interference, and other information that makes it more difficult to understand the speech. Generally, low pass filters which filters remove unwanted high-frequency noise from the signal that are not directly related to the speech [52].

Feature Extraction: Next, the clean signal is converted into overlapping frames, each of which is then passed through a feature extraction algorithm. This algorithm retains only the salient information from each frame. A variety of signal processing techniques [74] and ML extraction layers [98] have been used to establish which features to extract.

Decoding: Finally the extracted features are passed to a decoding function, usually a machine learning model. This model takes in the extracted features from the previous step and outputs the final transcription. There have been a wide variety of models are used for this purpose in ASR systems, including Convolutional Neural Networks (CNNs) [19], [76], Recurrent Neural Networks (RNNs) [56], [79], [80], [78], Hidden Markov Models[73], and Gaussian Mixture Models [65].

C. Phonemes

Human speech is made up of various component sounds known as phonemes. Each phoneme represents the smallest unit of sound in a word. The set of possible phonemes is fixed due to the anatomy that is used to create them, but different languages use different phonemes. For example, English is made up of 44 phonemes, while the International Phonetic Alphabet (IPA) contains 107 [30].

To perform phonetic translation, we use the Carnegie Melon University (CMU) Pronouncing Dictionary [101]. This is a machine-readable dictionary of over 130,000 words with information on their pronunciation. Namely, it maps the words to their phonemes. The CMU Pronouncing Dictionary outputs translations in the ARPAbet phoneme transcription code, which is composed of 39 unique phonemes. ARPAbet is one of several phoneme transcription codes related to the International Phonetic Alphabet (IPA), with ARPAbet specializing in English transcription. This dictionary greatly aids feature extraction and decoding, since it allows us to evaluate how translations are spoken, rather than how they are written for instance, "two bear sail" and "to bare sale" are spoken identically, yet written differently.

D. Breaking Audio CAPTCHAs

The adversary's goal is to use automated means to break an audio CAPTCHA (i.e. transcribe it correctly). Recent methods often involve the use of commercial ASRs for transcribing CAPTCHAs [89], [34]. This method is a multi-step process.

The audio CAPTCHA consists of sound bites or utterances of words, digits, characters, etc. The adversary splits these into individual sounds, each containing a single utterance. For example, the adversary splits the CAPTCHA containing the sequence "ABMKL2" into six individual audio files, each of which is then passed to the ASR individually for transcription. Passing individual utterances, instead of the entire CAPTCHA audio wholesale, improves accuracy. The adversary then parses the transcripts to ensure these are in the correct format and only contain numbers and digits.

In case the ASR outputs an incorrect transcript for an utterance, the adversary uses one of two techniques to recover the real one: phonetic mapping or statistical analysis. Phonetic mapping is employed when the ASR outputs a phonetically similar transcript to an expected digit or character (e.g., "too" for the digit "2"). In this case, the adversary can use phonetic mappings to identify the digit that sounds closest to "too" as the correct transcript ("2").

Statistical analysis is used when the ASR regularly outputs the same unique incorrect transcript for a sound bite. For example, the ASR might output "crown" for the CAPTCHA digit "two," even though they are phonetically dissimilar. An adversary with enough queries can perform simple statistical tests to detect this pattern. As a result, she transcribes "2" every time the ASR outputs "crown."

E. Datasets

We used several audio training datasets to create our models.

LibriSpeech [69]: is a corpus of English speech used to train ASR models. The audio is derived from audiobooks that are part of the LibriVox project. This dataset contains about 1000 hours of text-aligned, short utterances sampled at 16kHz. This data is segmented using Smith-Waterman alignment [88] and samples, where the audio and text do not match, are removed.

Google Speech Commands Dataset [77]: is composed of short single-word utterances. Specifically, the dataset consists of 65,000 one-second samples of thousands of different people saying one of 30 words. This dataset was designed to be used to train simple ASR models for words such as "Yes", "No", and "Up." While the vocabulary is small, the number of different voices, accents, and speech patterns in this dataset is useful in representing the many variations of speech that exist.

LDC: ISOLET Spoken Letter Database [48]: is used to train ASR models on the English Alphabet. The dataset contains two samples of each English letter being spoken by 75 males and 75 females of varying ages. If a letter was severely misspoken then the sample was removed from the dataset.

III. Problem Formulation & Threat Model

In this section, we formulate the goal of this work. Next, we describe the adversary's capabilities and techniques for breaking audio CAPTCHAs and detail the different requirements for a CAPTCHA generation algorithm to this adversary.

A. Problem Formulation

Our goal is to design a CAPTCHA generator that can perturb benign audio samples to produce audio CAPTCHAs that are intelligible to humans, but can force the attackers' ASR into outputting incorrect transcriptions. We also want to produce CAPTCHAs that are easily detectable by commercial ASR services, alerting when their APIs are being misused.

B. Adversary

We consider an adversary whose goal is to break audio CAPTCHAs (i.e., correctly transcribe their audio) in an automated way using an ASR. We assume that the adversary has the ability to query the CAPTCHA service (i.e., check if the proposed transcription is correct).

Query Access: CAPTCHA services enforce rate limits to prevent unfettered access from malicious users. However, we assume a strong adversary who has no such constraints and can make unlimited queries to the CAPTCHA service.

Success Rate: By combining statistical analysis and phonetic mappings, researchers have been able to break audio CAPTCHAs with success rates of up to 98.3% [89]. As in earlier works, we assume that the adversary is successful if it can correctly transcribe the entire CAPTCHA at a success rate of just 1% [38]. We show that our CAPTCHAs are robust to such a small upper bound. While the success rate of 1% is considered standard in the research community [38], improvements made by future attacks may cause the community to reduce the threshold of success in future years. As a result, our goal was to design an algorithm that is (at least partially) future proof. Our method has a success rate of less than 4×10^{-5}, which is orders of magnitude below the current threshold.

Adaptive Adversary: This adversary has perfect knowledge of the audio CAPTCHA generation algorithm. Having this knowledge allows the adversary to modify her behavior to overcome the algorithm, improving her chances of breaking the CAPTCHA. There are a number of ways the attacker can modify her behavior, the two most popular of which are vulnerability analysis [96] and adversarial training [66], [84], [63]. As the name suggests, vulnerability analysis involves finding weaknesses in the CAPTCHA algorithm. The adversary can then modify the CAPTCHA audio, before passing it to the ASR, so that the ASR will output the correct transcript. On the other hand, adversarial training involves training a local ASR to specifically label audio CAPTCHAs. The local model will be designed to be more robust to the CAPTCHA algorithm and will likely produce correct transcripts. We show in this paper that our CAPTCHAs are robust to an adaptive adversary with perfect knowledge of our algorithm.

C. Defender

Considering these adversarial capabilities, we now list the requirements of the defender (and her CAPTCHA generation algorithm) in order to be robust to such a strong adversary.

Intelligibility: The defender's audio CAPTCHAs must be intelligible to humans (i.e., human transcribable). Naively degrading audio CAPTCHA quality might help prevent ASRs from producing correct transcriptions, but will also negatively impact human intelligibility.

ASR Output: Even if the ASR mistranscribes an audio sample (e.g., "too" instead of "two"), an adversary can recover CAPTCHA text. To prevent this, our goal is to force the ASR to output an *empty string*. This is an incredibly strong threat model since ASRs can still output some text. However, by forcing an empty output, our attack makes it impractical for the adversary to recover the CAPTCHA text.

Transferability: An adversary can use any commercially available ASR, or even a locally trained one, to transcribe the CAPTCHA. This means that the defender will likely not have knowledge of or even query access to the adversary's ASR. Therefore, the CAPTCHA audio must be highly transferable to force any unknown ASR to output an empty string.

Detection: Adversaries leverage commercially available ASRs to conduct attacks against CAPTCHA services without the consent of the ASR owners. These ASRs are available via API calls and handle massive volumes of requests. Our final requirement is that the audio CAPTCHA be identifiable by the ASR owner. This will allow the ASR owner to flag, block, or delay the adversary's requests.

IV. Selecting Attacks as Defenses

One way to meet the requirements outlined above is to craft audio samples specifically designed to fool ASRs. Fortunately, there are a plethora of attacks in the space of adversarial machine learning that try to do exactly that [25]. We hypothesize that these attacks can be used as a defense mechanism to beat the ASRs used by CAPTCHA breaking adversaries. In this section, we give an overview of these attacks and evaluate them as potential defenses using our threat model.

A. Different Attacks on ASRs

The goal of the CAPTCHA generator is to craft audio samples that are intelligible to humans, but can force the adversary's remote ASRs into a phonetically dissimilar mistranscription (i.e., force the ASR to output random garbage e.g., "HadJSNm" for the audio containing "one two three", instead of "Juan too tree"). There are a number of attacks that can help achieve this goal (Table I):

White-Box Attacks: These attacks exploit knowledge of the model's decision boundaries to craft adversarial samples. The most popular of these are *optimization* attacks. However, it is unclear whether these attack audio can force phonetically dissimilar transferability. If so, these attacks will be a good candidate for crafting audio CAPTCHAs. There are a multitude of attacks, representative candidates are shown in Table I. Optimization attacks can be classified into two types: psychoacoustic and hard-clipping [24]. We choose one attack from each, CW [42] and P-PGD [24] as potential candidates. We choose these specific attacks because they have the highest rates of attack success and are architecture agnostic (i.e., can work against all ASR architectures). Other attacks in this category are too sensitive and can fail even due to minor modifications to the ASR architecture.

Black-Box Attacks: These attacks do not require *any* knowledge of the target ASR to craft adversarial samples. The two most popular types are *gradient-free* and *signal processing*

	Potential CAPTCHA Use	Audio Quality	Attack Type
Taori et al. [95]	✗	Intelligible	Grad Free
M. Azalnot et al. [27]	✗	Intelligible	Grad Free
HVC (2) [41]	✗	Inaudible	Misc
Cocaine Noodles [97]	✗	Inaudible	Misc
Dolphin Attack [105]	✗	Inaudible	Misc
Light Commands [92]	✗	Inaudible	Misc
Roy et al. [75]	✗	Inaudible	Misc
HVC (1) [41]	✗	Unintellgible	Opt
CW [42]	✓	Intelligible	Opt
Houdini [47]	✓	Intelligible	Opt
Schonherr et al. [82]	✓	Intelligible	Opt
Kreuk et al. [61]	✓	Intelligible	Opt
Qin et al. [72]	✓	Intelligible	Opt
Yakura et al. [102]	✓	Intelligible	Opt
Commander Song [104]	✓	Intelligible	Opt
Devil's Whisper [44]	✓	Intelligible	Opt
Abdoli et al. [20]	✓	Intelligible	Opt
P-PGD [24]	✓	Intelligible	Opt
Kenansville Attack [23]	✓	Intelligible	Sig Proc
Abdullah et al. [21]	✗	Unintellgible	Sig Proc

TABLE I: Overview of existing attacks. **Inaudible:** Audio will not be heard by the human ear. **Unintelligible:** Audio will sound noisy to the human ear. **Intelligible:** Audio will be clean and understandable by the human ear. **Grad Free:** Gradient Free, **Misc**: Miscellaneous, **Opt:** Optimization, **Sig-Proc:** Signal Processing

attacks. Gradient-free attacks are a constrained version of the optimization attack family because they estimate the decision boundaries of the target ASR. However, these produce lower quality adversarial audio and therefore, are not good candidates for audio CAPTCHA.

On the other hand, signal processing attacks exploit the imperfections in the signal processing algorithms used in the ASR pipeline that are designed to emulate the human ear [21]. These attacks can produce samples that can successfully transfer to and exploit remote models. This makes signal processing attacks good candidates for generating audio CAPTCHAs. Of the signal processing attacks, only Kenansville can produce intelligible audio [23]. As a result, we will use it as a candidate attack for our experiments.

B. Evaluation: Transferability

Having chosen the three attack candidates [24], [42], [23], we evaluate these as potential CAPTCHA algorithms. The goal is to quantify how well these attacks provide *phonetically dissimilar* transferability. Attack audio with higher transferability will have a higher probability of fooling the adversary's ASR.

ASRs: For this experiment, we chose a total of four models [16]: one surrogate (DeepSpeech [6]), and three remote (Wit.ai [9], Google [7], and IBM [8]). The surrogate model is a local model available to the CAPTCHA service owner. This is designed to emulate the adversary's ASR or the remote ASR. We chose the popular and open-source DeepSpeech ASR as the surrogate. In each case, the candidate algorithm constructs adversarial samples for the surrogate ASR and transfers them to the remaining remote ones.

Dataset: We followed the procedure laid out in previous works [25], [24]. We pooled a set of 1000 benign audio files by randomly sampling from the LibriSpeech [69] dataset. Each of these was transcribed using the surrogate DeepSpeech model to ensure an average Word Error Rate of less than 0.1.

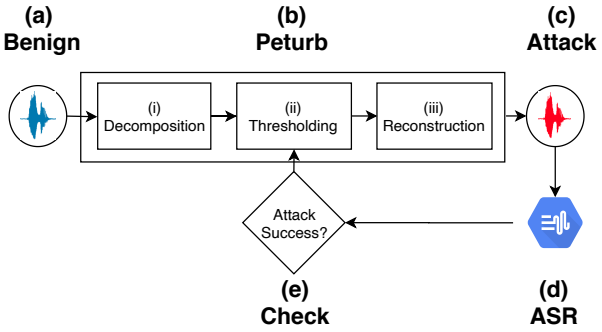

Fig. 1: The Kenansville attack pipeline. (a) We start with the audio sample we wanted to perturb. (b(i)) It is decomposed into frequency components using the DFT. (b(ii)) Next, all the frequency bins that have less power than the threshold are set to zero. (b(iii)) The remaining frequencies are reconstructed. (c) This produces a raw perturbed audio, (d) which is passed to the ASR. (e) If the attack succeeds, the threshold value is decreased, otherwise, the threshold is increased. Then the steps repeated again.

Adversarial Audio Generation: Each of the 1000 audio files was then perturbed using the three candidate attacks to work against the surrogate DeepSpeech model. Each audio sample was perturbed to random target transcripts using the three attacks using 1000 attack iterations. In the end, we have 1000 audio samples from each attack.

Transferability: Next, we pass each of the 1000 attack audio samples to the three remote ASRs and retrieve the corresponding adversarial transcriptions. We calculate the phonetic similarity using the Levenshtein distance scores between the phonetic representations (extracted using the CMU Pronouncing Dictionary [15]) of the original transcript and the one produced by the remote ASR.

Results: Table II shows the results of our experiment. Smaller distance scores mean that the adversarial transcript (produced by the remote ASR) is phonetically similar to the original label. We can observe that optimization attacks have lower distance scores compared to the signal processing attack. This demonstrates that optimization attack CAPTCHAs will produce transcripts similar to the original CAPTCHA label (e.g., "too" for the number "two"), allowing the adversary to easily reconstruct CAPTCHA text. Thus, optimization attacks are unfit for use as CAPTCHA generation algorithms. In contrast, the Kenansville attack resulted in the highest distance scores. This indicates that the Kenansville attack fulfills the transferability requirement for CAPTCHA generation mechanism.

C. Evaluation: Adaptive Adversary

As described in our threat model, any valid CAPTCHA generation method should succeed in the presence of an adaptive adversary. Having filtered the potential attack candidates to just Kenansville, we now evaluate its performance against an adaptive adversary. This adversary can either perform

Attack Name	Attack Type	Transcription Distance		
		Google [7]	**IBM** [8]	**Wit.ai** [9]
CW [42]	Optimization	2.56	5.94	23.76
P-PGD [24]	Optimization	13.64	24.5	19.82
Kenansville [23]	Signal Processing	49.31	49.76	48.39

TABLE II: Transferability results. The numbers represent the Levenshtein distances between the original and the adversarial transcript. Smaller distances imply high phonetic similarity to the original, benign label (e.g., "Juan too tree" for the original label "one two three"). Larger distances imply lower phonetic similarity to the original label (e.g., "HadJSNm" for the original label "one two three"). Optimization attacks have higher phonetic similarity (than signal processing attacks) and therefore, can not be used as CAPTCHA generators. This leaves the signal processing attack, which we further evaluate in Figure 2.

adversarial training or vulnerability analysis. While the original paper demonstrated that Kenansville is robust to adversarial training (reasons for which discuss in the next section), in this paper we will perform vulnerability analysis.

Attack Steps: We first discuss how the Kenansville attack works. The pipeline performs signal decomposition using the DFT on a given audio sample. Next, all the frequency bins with power less than the threshold are set to zero, reconstructed back into a raw signal, and passed to the ASR. If the ASR outputs the wrong transcript, we lower the threshold and start again. This is because smaller thresholds result in better audio quality. If the ASR outputs the correct transcript, the threshold is increased and the steps are repeated.The user can either manually pick a fixed threshold (as defined in the original Kenansville paper), or find one using binary search. In order to avoid over-fitting the adversarial samples to a single model, we manually pick a fixed threshold for our experiments. A diagram of this attack can be seen in Figure 1.

Why The Attack Works: This attack exploits the differences between how humans and ASR pipelines process audio. Specifically, the attack discards low power frequencies (by setting their intensities to zero). Due to their already low intensity, these frequencies are inaudible to the human ear. Therefore, removing them does not affect the audio quality for human listeners. However, passing these perturbed audio samples to the ASR forces an incorrect output. That is because audio samples with zero intensity frequencies do not occur in the natural world. As a consequence, such samples do not exist in the training datasets for ASRs and therefore force mistranscriptions.

Initial Hypotheses: Audio samples with empty frequency bins do not exist in the natural world, and therefore not in the ASR's training dataset. As a consequence, these samples force ASRs into producing a mistranscription. Adding power to the empty frequency bins will undo the effects of the attack and allow the ASR to produce the correct output.

Methodology: The adaptive adversary will use this knowledge to add small intensities of power *back* to the spectrum. One way to do this is to add Gaussian (white) noise. Using any other noise might bias some frequency bins over others. In contrast, Gaussian noise increases the power evenly across the previously empty frequency bins, thereby undoing the effects of the attack. To account for the noise, the attack algorithm will need to perturb the audio even more by setting a higher threshold. This will further degrade the quality of the attack audio but will make it robust to the Gaussian noise. In essence, the attack will balance robustness to Gaussian noise against the intelligibility of the attack audio sample.

In this experiment, we quantify the intelligibility of the Kenansville attack in the presence of this adaptive adversary. Specifically, we will use multiple magnitudes of noise to produce different noised versions of the adversarial audio. The attack succeeds *only* if all the resulting noised versions produce an empty string. We also run a similar baseline experiment, without the presence of the adaptive adversary. For a fair comparison, we maximize Kenansville's intelligibility and use binary search to find the optimal perturbation parameter (as recommended by the original authors).

Setup: For this experiment, we opt for datasets that are frequently used in real-world CAPTCHA services[2], consisting of words, letters, and digits. Our dataset included 26 English language characters from the LDC dataset [48], and 10 digits, and 20 words from the Google speech commands dataset [100]. We then randomly sample 10 audio files from each of these labels.[3] We only use files that the commercial ASRs transcribe correctly. This results in approximately 560 audio files for the adaptive adversary and the baseline experiments.

We use Gaussian noise with the mean-centered at zero, and variance selected on a logarithmic sweep between 0.001% to 20% of the maximum amplitude found in the attack audio, resulting in 46 amplitudes. We generate five different Gaussian noise samples for each of these 46, producing 230 noise samples. We repeat the trial five times since Gaussian noise is random, and could potentially impact our results. Each of the resulting 230 noise signals is added to the perturbed audio sample, resulting in 230 noise outputs. These are then passed to the model for transcription.

We pick four commercially available ASRs that the adversary can employ. These included Microsoft Azure, Google Speech, IBM, and Wit.ai.

Results: We evaluate the effectiveness of the adaptive adversary similar to the authors of the original Kenansville paper. Specifically, we compare the transcription success rate of several ASRs against the acoustic distortion measured using the Root Means Squared Error (RMSE) between the original and perturbed audio. Higher RMSE means greater audio distortion to successfully fool the model.

Figure 2 shows the results of these experiments. As the RMSE increases, the successful transcriptions decrease. And the faster the rate of decrease (greater steepness of line), the better the attack. We can observe that the steepness decreases

[2]We have placed these findings on current CAPTCHA services in the Appendix Table VII.

[3]We only chose 10 samples per label because we make 1,380 queries for each sample ballooning to a total of 900,000 queries. Increasing the total labels to greater than 10 would have made it difficult to use any of the commercial ASRs since these impose strict rate limiting.

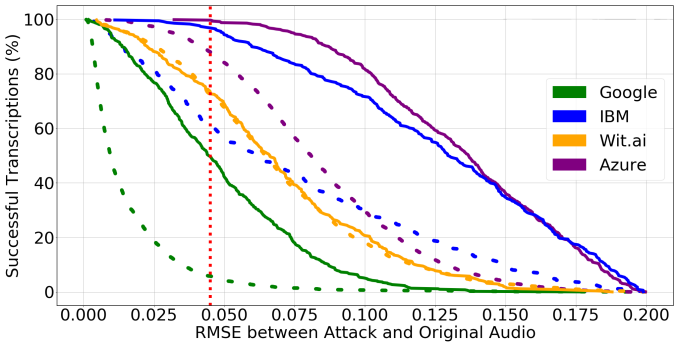

Fig. 2: We evaluate Kenansville against four popular ASRs. Each line shows the change in ASR accuracy with respect to Kenansville's distortion (measured using the RMSE). The vertical red line is the maximum distortion before which the audio starts to become unintelligible. The dotted lines represent the normal ASR, while the solid ones represent the ASR controlled by an adaptive adversary. Kenansville requires significantly more distortion to fool the adaptive adversary's ASR. Therefore, the solid lines have a dramatically higher RMSE than the corresponding dotted ones. The increased distortion makes the audio unintelligible. Therefore, Kenansville audio can not be used in CAPTCHAs.

markedly for the adaptive adversary (solid line), than without one (dotted line). For example, consider the results for the Google ASR (green). The percentage of successful transcripts degrades far more slowly for the adaptive adversary (solid green line) than the baseline (dotted green line). This indicates that the presence of the adaptive adversary (green solid line) results in higher audio distortion, significantly degrading human audio intelligibility.

To better visualize the intelligibility of the audio, we denote the GSM audio codec's average RMSE as the baseline for audio comprehension (the red vertical dashed line in Figure 2). The GSM audio codec is used during 2G cellular calls, which is the most common global means of telephony communication. We will assume any audio created that has an RMSE greater than GSM (to the right of the GSM line in Figure 2) is unintelligible, and thus unfit as an audio CAPTCHA.

Examining the Google results again, we can see that the presence of the adaptive adversary results in a significant degradation in the attack audio quality. In the baseline, Kenansville can force more than 90% of the audio samples to mistranscribe, before crossing the GSM line, and become unintelligible. However, in the presence of the adaptive adversary, Kenansville is only able to degrade the transcription success to 50% before the audio becomes too degraded for CAPTCHA use. We see these results consistently across all models (except for Wit.ai[4]). These plots, therefore, show that the Kenansville attack is insufficient for CAPTCHA generation as it is unable to produce intelligible audio in the presence of an adaptive adversary. *This means that the Kenansville attack is broken and should not be used for either CAPTCHA generation or for any of the applications proposed in the original paper.*[5].

[4]There are a number of reasons that might explain Wit.ai's performance. These range from model architecture, pre-processing to quality of the training data. However, it is difficult to make a concrete assessment as Wit.ai is a black-box system, with no publicly available information.

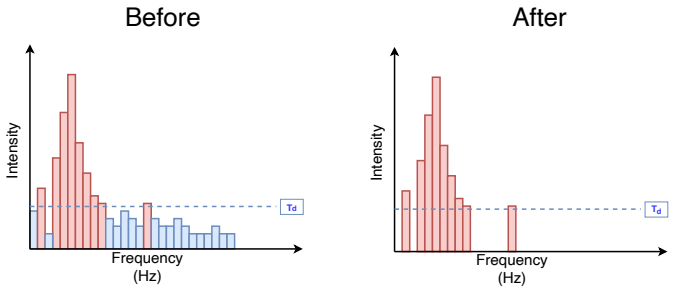

Fig. 3: From the Yeehaw Junction pipeline: Decimation is performed on the audio spectrum. We set all the frequency bins (blue) below T_d to zero.

Take Away: Despite the plethora of candidate attacks, none of them can be used as CAPTCHA generators. Attacks either fail to transfer between different ASRs or are vulnerable to adaptive adversaries. Thus, the current work does not suffice. Lastly, even though we use RMSE in this experiment, such metrics are a poor measure for human audio intelligibility [21]. Therefore, in the later subsections, we will evaluate the CAPTCHA quality under the adaptive adversary via a user study.

V. Yeehaw Junction Overview

So far, we have demonstrated that existing works from adversarial ML can not be used as CAPTCHA generation algorithms. The three potential candidate CAPTCHA generators do not meet the threat model requirements: The two optimization attacks failed to exploit remote ASRs and the Kenansville attack is vulnerable to adaptive adversaries. Based on the lessons learned from our experiments, we design a new strategy to overcome the limitations inherent to prior work.

We design our CAPTCHA algorithm, Yeehaw Junction, based on the signal processing family of attacks. This is primarily because this family of algorithms has high transferability against remote models (a necessary condition for CAPTCHA audio), as we saw in the Section IV. However, crucially, these attacks (including Kenansville) lack robustness to adaptive adversaries. Our new Yeehaw Junction defense overcomes this limitation by exploiting the differences in how humans and ASRs process audio, and produces CAPTCHA that are robust to bots.

A. Defense Steps

Our defense has three major steps: decimation, clipping, and noising. The Kenansville method has none of these steps, which explains its lack of robustness to adaptive attackers. Each of our steps is either motivated by the requirements of the threat model or the lessons learned during the evaluations in the previous section.

Step 1: Decimation: Decimation is the process of discarding frequencies below a predefined threshold T_d. This property

[5]Readers are invited to listen to and compare the audio samples here: https://sites.google.com/view/attacksasdefenses/home

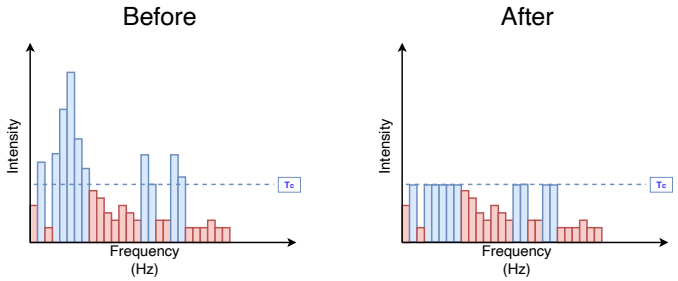

Fig. 4: From the Yeehaw Junction pipeline: Clipping is performed on the audio spectrum. We set all the frequency bins (blue) above the T_c to the value of T_c.

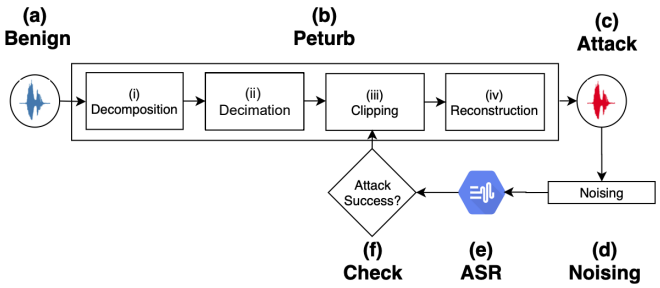

Fig. 5: Full Yeehaw Junction pipeline. (a) The audio sample is (bi) is decomposed into its frequency components. (b.ii) Decimated by discarding low-intensity frequencies. (b.iii) Clip the decimated frequencies based on a threshold. (b.iv) Reconstruct the modified spectrum into a (c) raw audio sample. (d) Add noise to the audio, to account for the adaptive adversary (e) and pass it to the ASR. (f) Check for success. If so, lower the clipping threshold and rerun.

is known to be robust against adversarial training [23], as it prevent the ASR from converging during training. As a result, models trained on decimated audio have lower accuracy than their baseline counterparts. Intuitively, it is because frequency bins are dissimilarly discarded across the spectrum (even for two utterances of the same word), which affects the model's ability to learn proper decision boundaries. We use this key observation to design the first step of our defense, shown in Figure 3. Here, we take the DFT of the audio sample, discard the frequencies below the threshold T_d (shown in blue). We then use the remaining frequencies (shown in red) to reconstruct the raw audio waveform.

Step 2: Clipping: We use spectral clipping to force the model to output an empty string. The human ear largely relies on small bands of dominant frequencies (e.g., formants in the case of vowels) to identify individual phonemes [91]. For example, the phoneme /æ/ (such as in the word "hat") has an average first formant of 585 Hz and the second formant of 1710 Hz. These frequency peaks vary between people and within the same speaker, thus making the raw frequency values less critical to intelligibility. Instead, humans rely on the ratio between these two frequency peaks. In the case of /æ/, the difference between the first two peaks is generally around 1.9 times the value of the first formant [35]. The human ear can still identify phonemes even if the maximum amplitude within these peaks have been clipped, as long as the location of the frequency bands with the spectrum remain unchanged and the frequency band still dominates its neighboring frequencies. This is because, the amplitude of the peak only determines the volume at which the listener hears the phoneme. As long as the clipped peaks do not change the overall structure of the waveform (i.e., the highest peaks are still at the same frequencies before and after clipping) the listener can still determine the intended phoneme. This is because the formant frequency values are the main determining factor in phoneme identification [91].

In the following procedure, we propose clipping the dominant frequencies of the spectrogram as a means of fooling the model. Clipping these frequencies will create phonetic structures that do not exist in the natural world, and will therefore fool the ASRs. However, since clipping does not change the location of the dominant frequencies, it will still sound the same to the human ear.

Additionally, clipping has the added benefit of being robust to the Gaussian noise-based adaptive adversary. By clipping, we are leveling out the structure of the peaks that the ASR needs for correct transcriptions. Adding random noise will not recreate the structure of the peaks that we clipped out and not result in the correct original transcript.

For this step, we will use a clip function to clip the values outside a specified interval [0, T_c] (Figure 4), calculated using:

$$T_c = \max\left(DFT(x)\right) - \alpha \tag{1}$$

where α is the clipping value between [0, $\max\left(DFT(x)\right)$]. All the frequencies that have more intensity than a threshold T_c (shown in blue) will be clipped to T_c. All the frequencies that lie below the T_c (shown in red) will remain unchanged. Therefore, higher values of T_c will clip out more components, producing lower-quality audio. Such clipping will impact the dominant frequencies (i.e., the ones with the highest power) while, maintaining the overall phonetic structure will. We will use a binary search to efficiently locate the minimum T_c to minimize the impact on audio quality. The goal here is to find the minimum threshold that *also* guarantees that the ASR will output an empty string. Instead of iteratively reducing the T_c value until we find the smallest value that can fool the ASR, we run the binary search algorithm.

Step 3: Noising: As seen in the previous section, an adaptive adversary can break Kenansville by adding small amounts of noise, enabling the ASR to transcribe the audio CAPTCHA correctly. Therefore, we account for the noising step as part of our defense pipeline, We will add noise to every sample produced with our defense before we pass it to the ASR. This will ensure that we take account of the adaptive adversary during audio CAPTCHA generation.

The full pipeline is shown in Figure 5. We perturb an audio sample (a), by first performing signal decomposition (bi). We then decimate (bii) the audio by some fixed threshold, T_d. Next, we clip the spectrum (biii) using a variable threshold, T_c, and then reconstruct the raw audio waveform. To account for the adaptive adversary, we noise the audio (d) by adding Gaussian noise, before passing it to the ASR for transcription. We then check whether the ASR output an empty string. If

	Yeehaw	Kenansville [23]	CW [42]	PPGD [24]
Intelligibility	✓	✓	✓	✗
Transferability	✓	✓	✗	✗
Adp-Adv (Adv-Training)	✓	✓	N/A	N/A
Adp-Adv (Vuln-Analysis)	✓	✗	N/A	N/A
Detection	✓	N/A	N/A	N/A

TABLE III: Comparison between our CAPTCHA algorithm and existing ones using the threat model. Our algorithm meets all requirements. ✓: Meets Requirement. ✗: Does not meet requirements. N/A: We did not test since other requirements not met.

so, we reduce T_c and repeat the steps until we find the lowest value for T_c that forces the ASR to produce an empty string. Lastly, lowering the value of T_c and T_d does improve attack success, but at the cost of audio quality. The lower the value, the lower the audio quality.

VI. Evaluation of Yeehaw Junction

We now evaluate Yeehaw Junction against the threat model (Table III) and compare it against other CAPTCHA algorithms from literature and industry.

A. Transferability

As discussed in Section III, audio CAPTCHAs should be able to transferable to unknown remote ASRs (i.e., force the ASR to output an empty string). This prevents the attacker from using phonetic mapping or statistical analyses to get the extract transcript. In this experiment, we will quantify our attack's transferability. We craft audio CAPTCHAs for each (surrogate) ASR using our CAPTCHA algorithm and then pass these to the remaining remote models to calculate the transferability rates.

Setup and Methodology: We use Yeehaw Junction to perturb each audio file. The CAPTCHA quality controlled using two parameters: the decimation (T_d) and clipping (T_c) thresholds. We use binary search to find the values of these parameters. *We will continually perturb the audio for different values until the surrogate model produces an empty transcript.* We target four models: Google, Wit.ai, IBM, and Azure.

The vast majority of our experiments were conducted on a 2015 Macbook Pro laptop. This is because our technique is designed to be fast. It takes a few milliseconds to generate an adversarial sample and requires O(logN) time (in the worst case). Moreover, it does not require any expensive hardware (such as GPUs).

Dataset: Similar to the previous experiments, our audio dataset consists of words, numbers, and characters. We will pool benign audio samples from across 56 labels (20 words, 10 digits, and 26 characters). We will select 10 audio samples for each file, resulting in 560 utterances.

Results: Table IV shows the results of the transferability experiments. Samples created for the Azure model are the most transferable. This means that CAPTCHAs designed to evade Azure (i.e., force empty string output) will also evade

		Remote			
		Azure	**Google**	**IBM**	**Wit.ai**
Surrogate	**Azure**	100%	**93%**	**97%**	**81%**
	Google	54%	100%	**96%**	76%
	IBM	31%	**82%**	100%	64%
	Wit.ai	43%	76%	**97%**	100%

TABLE IV: Transferability experiments for Yeehaw Junction show that it has high transferability. Numbers in **bold** show the highest levels. Models names have been arranged in descending order of their transferability rates.

remote models as well. In the worst case, the audio generated against Azure will have an evasion rate of at least 81% against remote models. Upon further inspection, we observed that Azure required the highest T_c of all the models. Therefore, crafting audio samples for Azure will guarantee success against a six-length CAPTCHA with at most probability of $P(\text{transcription}) = (1 - 0.81)^6 = 4 \times 10^{-5}$. This is orders of magnitude below the 1% success rate needed for the adversary to be considered successful. However, higher T_c values meant that adversarial audio that successfully evaded Azure had the lowest quality across all the models. This means that a sample that can fool the Azure model could also fool other models.

The Azure results make sense in the light of the following observation: lower quality audio generally has a higher probability of fooling the target ASR and higher probability of transferring to unknown models. Additionally, as the value of T_c increases the quality of the generate audio decreases, which consequently, makes it harder for the Azure model to understand the audio content. However, while the Azure model fails to produce *any* transcript, the human listener can manage to make out the contents of the audio samples (as we show later in the User Study).

B. Intelligibility

So far, we have demonstrated the robustness of Yeehaw Junction to remote ASRs. We want to ensure that our audio is intelligible to humans. As a result, we conduct a user study to examine intelligibility. We compare the human transcription rate of Yeehaw Junction against and reCAPTCHA (which is *the* most popular CAPTCHA service [11]). Our goal is to demonstrate that Yeehaw Junction is as intelligible as other services while providing robustness to ASRs. It is important to note that the goal of this paper is not to develop the most *usable* CAPTCHAs (like previous works [51]). Instead, we want to develop the most *intelligible* CAPTCHAs. We can improve overall usability with findings from other work [51].

We also include in our experiment Kenansville audio samples that can fool an adaptive adversary. While we have shown in earlier experiments that Kenansville produces audio samples with high distortion when trying to evade an ASR controlled by an adaptive adversary, this experiment will demonstrate that these audio samples are not intelligible to humans listeners. This will justify the need for Yeehaw Junction.

Setup: We conducted a single-session, within-subject user study (IRB reviewed and exempted), with participants recruited from Amazon's Mechanical Turk (MTurk) crowd-sourcing platform. Participants were located within the United States, had an approval rating for Human Intelligence Task completion of more than 95%, and had ages from 18 to 55 years.

During the study, each transcribed three randomly selected CAPTCHAs.

Four CAPTCHAs were sampled from each of Yeehaw Junction, Kenansville (Adaptive Adversary), and reCAPTCHA. Each audio samples from reCAPTCHA, Yeehaw Junction, and Kenansville contain either three or four-word phrases from the English language. Each participant was presented with three CAPTCHA audio samples from the three CAPTCHA generators in random order, and the participant was asked to transcribe. Transcription accuracy was evaluated using the word level edit distance. We treat each word as an individual token (instead of each character) as a misspelled word is considered a failure by the CAPTCHA service.

Moreover, as it is not feasible to test all the T_d and T_c values, we pick perturbed audio samples from our earlier experiments. However, we avoid using characters that were phonetically similar (e.g., m vs n), to avoid confusing listeners. This is a very common problem, which explains why militaries use the NATO phonetic alphabet [14], instead of the English alphabet, during audio communications.

In addition to transcribing, the participants were asked the number of times they played back the audio, a demographic questionnaire, and compensated with $1 for taking part in the study. Before running the experiment, we calculated the sample participant size to be 199, with an effect size of 0.5, a type-I error rate of 0.05, and statistical power of 0.8 and recruited 201 participants. On average, participants took three minutes to complete the study.

Lastly, we pass the CAPTCHA samples from three generators to the Google Speech API to see which ones transcribe correctly. If the model did not transcribe them correctly, we use the unCAPTCHA [34] strategy of passing one utterance at a time. If the model got a correct transcription in either scenarios, that CAPTCHA audio was considered broken.

Results: Table V shows the results of our experiment. There are three main takeaways here. First, Yeehaw Junction audio is more intelligible than reCAPTCHA. Smaller edit distance scores means smaller transcription errors. Table V shows that Yeehaw Junction has the smallest score. After running a repeated-measure t-test, we found that transcription errors (missing or misspelled words) for Yeehaw Junction are significantly lower than both reCAPTCHA ($t = 2.17, p < 0.05$) and Kenansville ($t = 11.63, p < 0.001$). Table V presents summary statistics (mean and standard deviation) of transcription evaluation based on word-level edit distance.

Second, the Kenansville produces highly unintelligible audio. We can see that Kenansville has the highest distance score of 3.09 across all the CAPTCHA samples. Upon further inspection, we noted that over 55% of the users had left the Kenansville audio transcriptions blank as they could not understand the audio at all. In stark contrast, no transcription was left blank for Yeehaw Junction. This proves that Kenansville audio is unintelligible and can not be used for audio CAPTCHAs. Additionally, participants, on average, had to playback Kenansville three times for transcription, compared to twice for for Yeehaw Junction and reCAPTCHA. This also supports the lower audio quality and incomprehensible nature of Kenansville. Even though Kenansville and Yeehaw Junction belong to the same family of signal processing attacks, they have widely different characteristics and audio quality.

CAPTCHA Algorithm	Broken	Edit Distance (M/SD)
reCAPTCHA [99]	✓	$1.50/1.49$
Yeehaw Junction		$1.21/1.69$
Kenansville [23] (Adaptive Adversary)		$3.09/1.45$

TABLE V: User study results. First, our Yeehaw Junction defense is robust to ASRs, while having the lowest edit distance scores (even lower than reCAPTCHA). Second, Kenansville has the highest scores, meaning its audio samples are least intelligible. Third, reCAPTCHA is broken (i.e., is unable to fool ASRs). **M:** Mean, **SD:** Standard Deviation.

Third, the table V shows that reCAPTCHA is broken as its samples were correctly transcribed by the ASR. This was possible even without using the stronger unCAPTCHA strategy. In contrast, the ASR did not correctly transcribe any of the audio samples from Yeehaw Junction or Kenansville, even when using the unCAPTCHA method. This means that even a lazy adversary can break reCAPTCHA by passing audio to the ASR wholesale. In contrast, such an adversary is unable to break Yeehaw Junction and Kenansville. These results confirm our hypothesis that Yeehaw Junction is a viable CAPTCHA generator. Our defense has both *higher* transcription accuracy and robustness than the popular reCAPTCHA.

C. Adversarial Training

So far, we have demonstrated that our defense can produce intelligible audio that can fool commercial ASRs with very high success rates. This effectiveness can be attributed to two reasons. First, commercial ASRs are not designed to break CAPTCHAs. Second, their training data does not account for our Yeehaw Junction audio. To overcome these limitations, an adaptive adversary can train a local model specifically designed to break our CAPTCHAs. This is known as adversarial training and has been shown to be the most robust means of tackling adversarial samples [96]. However, in this experiment, we show that even under ideal circumstances, adversarial training will be unable to break our CAPTCHAs.

As with any model, the model trained by the adversary will be more accurate on certain labels than others. We can assume, however, that the defender will also have this knowledge since she too can train a local model. The defender can use this information to decide how frequently certain labels should be used in CAPTCHAs. For example, if the defender knows that the adversarial model has poor accuracy on "2," she can have this label occur more frequently within the CAPTCHAs she produces. Therefore, to calculate the accuracy of the adversary's local model, we can not simply take the accuracy of the test set. We instead need to weigh the samples by how frequently a defender will select them for use in a CAPTCHA.

Methodology: We exactly follow the adversarial training methodology described in prior works [23]. We perturb the *entire* dataset (of words, letters, and digits) using our Yeehaw Junction. We use all the samples from the datasets to ensure high model accuracy, which is a total of 72,800 audio samples. Instead of merely augmenting the data, we perturb all the samples from both the train and test sets.

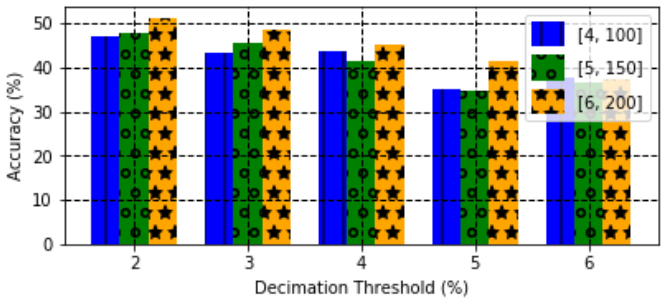

Fig. 6: Adversarial training experiments. Increasing T_d decreases the accuracy of the ASR. This is understandable since larger T_d results in poorer quality audio. **LEGEND:** [X,Y]: X = Number of layers. Y = Neurons per layer.

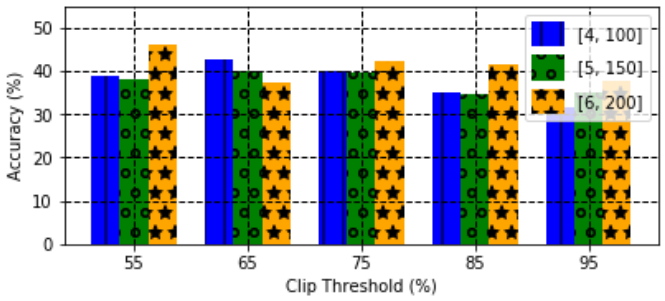

Fig. 7: Adversarial training experiments. Increasing the T_c decreases the accuracy of the ASR. This is understandable since larger T_c results in poorer quality audio. **LEGEND:** [X,Y]: X = Number of layers. Y = Neurons per layer.

This ASR will be specifically built for breaking CAPTCHA samples produced using Yeehaw Junction (instead of being used for general purpose speech recognition). This will ensure greater effectiveness at breaking CAPTCHAs as the decision boundaries will not be skewed towards some other recognition task. Additionally, since the attack is non-stochastic (i.e., does not *randomly* decimate or clips frequency bins), perturbing one sample for each attack parameter is enough [23]. We then conduct two sets of experiments to observe the effect of changing T_d and T_c on model accuracy.

Models: We specifically design our models for key-word spotting using the architecture from earlier work [23] which can achieve a baseline of 85% on benign audio samples. We improve on its robustness to adversarial samples by incorporating recent findings for robustness literature [22]. Recent work has shown that certain ASR architectures are particularly robust to adversarial audio [22]. Factors such as (1) presence of the RNN layer (e.g., LSTMS or GRUs), (2) multi-sequence output, and (3) large model complexity [22] can decrease robustness to adversarial samples, and therefore, prevent an ASR from breaking CAPTCHAs. This rules out the possibility of using any general-purpose ASR, such as DeepSpeech, for adversarial training (e.g., via fine-tuning, transfer learning, etc.). This is because DeepSpeech, and similar general-purpose ASRs, contain the above three listed factors. Therefore, we use an ASR architecture most closely resembles those used by key-wording spotting models [17]. Our ASR design does not contain the RNN, produces non-sequential outputs (i.e., a single output label per audio sample), and it does not include a large number of neurons/layers. As a consequence, our model architecture consists of a signal processing-based feature extraction layer, followed by multiple convolutional layers. We run experiments with three, four, and five layers containing 100, 150, and 200 hidden units, respectively to observe the impact of model complexity on model accuracy. We also assume a very powerful adversary that knows the *exact* timestamps of each label in the CAPTCHA. This will allow her to pass each label individually to the model, which is known to improve CAPTCHA recognition accuracy [34], [22].

Training: We train the model on a batch size of 32, for 50 epochs. We use early stopping if the cross-entropy loss on the validation set does not fall more by than 0.01 for three epochs.

Results: Figures 6 and 7 show the results of our experiments. Figure 6 shows the effect of keeping T_c constant (to a value of 4%), while increasing T_d. While Figure 7 shows the impact of increasing T_c while keeping T_d constant (to a value of 4%). In both cases, increasing the thresholds will reduce model accuracy. This is understandable as high thresholds reduce audio quality by discarding features that are necessary for learning valid decision boundaries.

More importantly, the largest accuracy we get across all experiments is around 51% for a T_d factor of 2% (Figure 6). Since audio CAPTCHAs are on average six utterances long, the adversary can only break the entire audio CAPTCHA $0.51^6 = 1.7 * 10^{-2}$ of the time. This is a major improvement on the success rate of 94% observed in prior works [89]. A defender can easily reduce the accuracy rate even further by merely perturbing each utterance with a different T_d and T_c value. The attacker will then be forced to retrain their model for the highest expected threshold. For example, if any single utterance in the CAPTCHA has a T_d value of 5%, the attacker will need to train an ASR which has an accuracy of 41% in the best case. At this point, the success rate for breaking CAPTCHAs falls further to $0.41^6 = 0.0047$.

Lastly, our model architecture is specifically designed for keyword spotting and is based on a model from prior work [23]. It achieved a success rate of 85% on the clean dataset. We improved upon this architecture using recent work to increase accuracy and robustness [22]. However, despite these improvements, our adversarial perturbations prevented model accuracy from going barely over 50%.

D. Detection

Adversaries often abuse commercial ASR services to transcribe audio CAPTCHAs. For instance, Bock et al. showed that an adversary can defeat Google's reCaptcha [13] service by feeding its audio CAPTCHA to Google's own ASR [7]. If ASR owners can detect when an adversary is misusing their service, they can block the requests.

Yeehaw Junction is designed to force ASRs to confuse the CAPTCHAs for random noise by outputting only empty strings. Based purely on the empty transcript, it is not possible for the ASR owner to ascertain whether an input was real noise or an audio CAPTCHA. However, the key observation here is that random noise and our CAPTCHAs have different

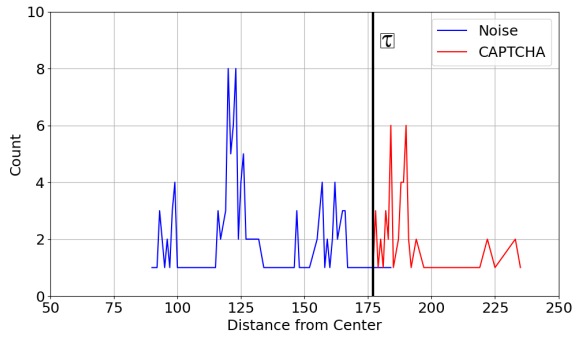

Fig. 8: Detection experiment for the word "Forward" for one layer of the ASR. We can see that CAPTCHA have higher distances than those for benign examples. Any sample that has an empty output and greater distance than the τ (solid black line), is a CAPTCHA.

levels of acoustic structure. Simply put, random noise lacks any form of structure, while Yeehaw Junction maintains the structure of the original audio sample. We hypothesize that due to the difference in these structures, both samples will produce different sets of activations [70].

We analyze the activations produced when passing random noise and Yeehaw Junction CAPTCHAs through an ASR. We start by extracting the activations from each layer of the ASR for random noise audio samples. These activations correspond to points in an d-dimensional space, where d is the size of the activation vector. For each layer, we find the center (i.e., the activation point that has the smallest L_2 distance from all the others). Next, we get the activations for CAPTCHA samples and calculate their distances from the center. If our hypothesis is correct, noise activations will be closer to the center than the adversarial one. This will enable us to detect whether an input sample was adversarial or just noise.

Setup and Methodology: We use the same dataset of 56 characters, words, and letters as before. Our data is split into train and test sets. The train and test set contained 778 and 600 files respectively. This defense method requires white-box access to the ASR since we are extracting sample activations. The operator of an ASR would realistically have this level of access. In our case, we use the DeepSpeech model, which is a general-purpose ASR and is locally available to us.

Results: Figure 8 shows the CDFs for distance values calculated for both the noise and the Yeehaw Junction sample sets for one layer of the ASR[6]. The figure shows that most of the noise samples (blue) are closer to the center (which exists at 0,0) than the CAPTCHA samples (red). This indicates that the activations for noise samples are clustered in a small section. In contrast, Yeehaw Junction activation vectors exist in a larger region. This means we can use the distance from the center as a simple differentiator between noise and CAPTCHA samples. For example, if the detection threshold, τ, was set at the value indicated by the black line, activations left of τ could be classified as noise, while those on the right could be classified as CAPTCHA. The ASR owner should therefore flag any samples that produce empty string outputs that have higher activations than τ as possibly malicious. Since values of τ are dependent on model activations, the *exact* value of τ will depend on the defender's ASR.

However, selecting the optimal τ is a balancing act and will depend on the ASR owner's goals. For instance, the aggressiveness of the ASR owner's mitigation strategy plays a major role in τ's selection. If the mitigation strategy is highly aggressive (e.g., account suspension or blocking), then τ should be selected to maximize the precision of the detector to ensure benign users are not being punished. However, if the mitigation strategy is more lenient (e.g., delayed response or rate-limiting), then the selected τ should balance the precision and recall. We select τ so that the precision and recall are both equal to 89%. Additionally, these values are calculated for transcribing a single label from our dataset. CAPTCHAs are typically comprised of up to six utterances (Table VII). An attacker would therefore need to evade our detector multiple times in order to successfully avoid detection. For instance, in a six label CAPTCHA where a single positive classification is sufficient to raise suspicious, then the attacker would only have an $P(\text{evasion}) = (1 - 0.89)^6 = 1.77 * 10^{-4}\%$ chance of evading detection. In comparison, the average benign sample will only have a 18.9% chance of being incorrectly classified as suspicious. However, all Yeehaw Junction audio samples only transcribe as an empty string. Therefore, samples that produce empty strings (an already rare event for real users) are at risk of being incorrectly labeled malicious.

Finally, an attacker could attempt to evade this detection mechanism by further perturbing the CAPTCHA audio. The adversary could perturb the audio randomly or by employing an optimization technique. In either case, the adversary is likely to fail since both would adversely affect the transcription of the underlying audio.

E. Evaluation against WaveGuard

WaveGuard paper [59] is one of the most popular defenses against adversarial audio examples. The paper proposes five transformations that can remove the perturbations from the adversarial audio, enabling the ASR to output the original audio transcript (instead of the adversarial one). While the WaveGuard method has been effective against the optimization family of attacks, we demonstrate that our signal processing inspired method is robust to these transformations. Specifically, these transformations are unable to undo the perturbations introduced by our Yeehaw Junction method. As a result, the ASR will still output the empty transcript even after the WaveGuard transformations are applied, which is the stated goal of the audio CAPTCHA algorithm.

Setup and Methodology: We randomly sample 1000 audio files from our dataset of words, digits, and letters. Next, we apply our Yeehaw Junction method to craft audio samples against a modified ASR that uses WaveGuard. We run our method until the ASR outputs the empty transcript. Next, we count the number of audio samples that resulted in a non-empty transcript despite the presence of the WaveGuard. A small empty transcript count means that the defense was able to defeat a small number of attack audio samples. Lastly, we performed manual listening tests to make sure the resulting audio had human intelligibility.

[6]We use all the layers of the ASR as part of the final detection algorithm. However, we are only showing the results for one layer for brevity.

	Single Character	Six Character CAPTCHA
Quantize	0.171	2.5E-10
Resample	0.002	6.4E-17
Filter	0.01	1E-12
LPC	0.001	1E-18
MFCC	0.004	4.096E-15

TABLE VI: The table shows the evaluation results of our method against the popular WaveGuard defense. The values in the columns correspond to the probability of audio samples that were successfully defeated by WaveGuard. Small values correspond to fewer successes by WaveGuard.

Results: Table VI shows the results of our experiments. The first column lists the names of all the transforms from WaveGuard. The second column shows the probability of the transformation breaking our method i.e., by resulting in a non-empty string transcript. Here we simply counted all the samples that were broken by WaveGuard and divided them by the total number of samples. The second row shows the projected probability of breaking our method a six-character CAPTCHA. Even in the worst-case (i.e., for the quantization transform, row 1), the probability of our method being successfully broken by WaveGuard is $0.17^6 = 0.000024$. Therefore, our method is robust to bots using WaveGuard to break and transcribe our audio CAPTCHAs.

Lastly, it is important to note the limitations of this methodology. Defeating WaveGuard does not imply our method is robust against all defenses. This is because WaveGuard is designed to be effective against the optimization family of attacks, while our method leverages signal processing-based methods to generate adversarial samples. WaveGuard is considered state-of-the-art for defending against adversarial audio samples and serves as a good proxy in the absence of defenses specifically designed against Signal Processing attacks. However, success against this single defense, does not imply success against all defenses.

F. Comparison to Existing Literature

Using the four metrics described in Section VI, we now compare our work against existing methods from current literature. The idea of using adversarial machine learning algorithms to build robust CAPTCHA is still new. Very little work has been done in this space. Therefore, we were only able to identify three papers [58], [86], [46] that attempt to build robust CAPTCHA algorithms using adversarial machine learning algorithms. While these works present good first steps in this space, they have significant limitations:

Transferability: Both these methods have very low transferability rates against black-box ASRs. This allows the attacker to easily reconstruct the CAPTCHA text, specifically, with rates of 36% and 48% (for [86] and [46] respectively) compared to our method which is $< 0.01\%$.

Audio Quality: It is not possible to assess their audio quality since the authors perform very small user studies. [86] ran the study with 15 participants, and [46] ran it with 10 participants. In contrast, our user study included around 200 participants across different age groups and backgrounds. Therefore, it is difficult to assess the quality of their audio.

Detection and Adaptive Adversary: Neither paper provides experiments regarding detection or adaptive adversaries. Our audio has distinctive characteristics that can help the ASR owner detect if their service is being abused, and is also robust to adaptive adversaries.

Defenses: None of the three papers were neither evaluated nor will be effective against the popular WaveGuard defense. This is because these CAPTCHA generation methods are based on optimization attacks. However, WaveGuard has been incredibly successfully at undoing the perturbations introduced by this class of attacks, thereby defeating the attack. On the other hand, we show that our Yeehaw Junction method can successfully break WaveGuard defenses with a probability greater than 99.99%.

VII. Related Work

Audio CAPTCHAs are an essential tool for protecting online resources. However, they can be easily broken using ASRs [34]. To prevent this, audio CAPTCHAs must be created to be robust to ASRs, while simultaneously ensuring human intelligibility. One way to do so would be via adversarial samples. These are imperceptibly modified inputs that can force ML models to produce a misclassification [54], [93], [63], [32], [36], [85], [71], [67], [106], [90]. One of the most popular ways of producing these samples against ASRs is via use of optimization attacks[82], [40], [53], [60], [104], [27], [61], [47], [62], [42], [102], [72]. However, these attacks require white-box access to the target ASR, which limits their use for CAPTCHA generation purposes.

To add to that, these attacks fail to demonstrate transferability [25] (i.e., surrogate and target ASRs produce different outputs). This alone does not rule out optimization attacks as viable CAPTCHA algorithms. This is because the attack audio might produce one of two transcriptions: phonetically similar or dissimilar. For CAPTCHA audio, we want a phonetically dissimilar output (e.g., "HadJSNm" for the audio containing "one two three"). As a result, the bot's ASR will be unable to reconstruct the original CAPTCHA text. However, if the attack audio output is phonetically similar to the original CAPTCHA command (e.g., "Juan too tree" for the audio containing "one two three"), then the bot can easily reconstruct the CAPTCHA. Prior work [25] failed to identify which one of the two was the case. We show that optimization attacks produce phonetically similar transcriptions, making them vulnerable to Bots.

Due to these aspects, the CAPTCHAalgorithms [86], [46] that build on these optimization attacks suffer from severe limitations, which include lack of effectiveness against black-box models, questionable audio quality, and robustness against adaptive adversaries. To overcome these, we propose a CAPTCHA generation that exploits the differences in how the human ear and ASRs process audio. As a result, our attack can successfully exploit *any* ASR, produce high-quality audio that is easy for human listeners to understand, while also being robust to adaptive adversaries.

Additionally, we demonstrate how adversarial example attacks against ASR models can be used to create secure and usable audio CAPTCHAs. Using an adversarial example attack to generate CAPTCHAs has been previously proposed for text and image CAPTCHAs [87], [64], [68]. These CAPTCHAs were shown to be nearly impossible to transcribe by machine, but still usable by humans.

VIII. Conclusion

Breaking audio CAPTCHAs has become easier with the improvement in machine learning systems. Specifically, adversaries can now use ASRs to correctly transcribe CAPTCHAs. In response, CAPTCHA services have been forced to degrade the quality of their audio, which has adversely impacted human intelligibility. We design a new attack, Yeehaw Junction, that can produce intelligible audio CAPTCHAs that are also robust to ASRs. We evaluate it against a range of criteria, including intelligibility, robustness to adaptive adversaries, and transferability. We show that no attack in this space fulfills all of these requirements and that our method is the only viable audio CAPTCHA generator currently in existence.

Acknowledgements

This work is partly supported by the National Science Foundation under CNS-1933208 and CNS-2206950. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

References

[1] ASP.NET AJAX Captcha - RadControls for Web Forms: Telerik UI for ASP.NET AJAX.

[2] Authenticating users with sign in with apple.

[3] CAPTCHA: Telling humans and computers apart automatically.

[4] MTCaptcha.

[5] Securimage php captcha.

[6] DeepSpeech, Last accessed in 2019. Available at https://github.com/mozilla/DeepSpeech.

[7] Google Cloud Speech-to-Text API, Last accessed in 2019. Available at https://cloud.google.com/speech-to-text/.

[8] IBM Speech to Text, Last accessed in 2019. Available at https://ibm.co/2UGDcGc.

[9] Wit.ai Natural Language for Developers, Last accessed in 2019. Available at https://wit.ai/.

[10] Botdetect captcha generator. Available at https://captcha.com/, Last Accessed in 2021.

[11] CAPTCHA Usage Distribution in the Top 1 Million Sites, Last accessed in 2021. Available at https://trends.builtwith.com/widgets/captcha.

[12] Captchas.Net - Free CAPTCHA-Service. Available at http://captchas.net/, Last Accessed in 2021.

[13] Google reCAPTCHA. Available at https://www.google.com/recaptcha/about/, Last Accessed in 2021.

[14] Nato phonetic alphabet - wikipedia. Available at https://en.wikipedia.org/wiki/NATO_phonetic_alphabet, Last accessed in 2021.

[15] pronouncing 0.2.0, Last accessed in 2021. Available at https://pypi.org/project/pronouncing/.

[16] Uberi Speech Recognition Modules for Python, Last accessed in 2021. Available at https://github.com/Uberi/speech_recognition.

[17] Google reCAPTCHA. Available at https://www.tensorflow.org/tutorials/audio/simple_audio, Last Accessed in 2022.

[18] NikolaiT/uncaptcha3, Last accessed in 2022. Available at https://github.com/NikolaiT/uncaptcha3.

[19] Ossama Abdel-Hamid, Abdel-rahman Mohamed, Hui Jiang, and Gerald Penn. Applying convolutional neural networks concepts to hybrid nn-hmm model for speech recognition. In *2012 IEEE international conference on Acoustics, speech and signal processing (ICASSP)*, pages 4277–4280. IEEE, 2012.

[20] Sajjad Abdoli, Luiz G Hafemann, Jerome Rony, Ismail Ben Ayed, Patrick Cardinal, and Alessandro L Koerich. Universal Adversarial Audio Perturbations. *arXiv preprint arXiv:1908.03173*, 2019.

[21] Hadi Abdullah, Washington Garcia, Christian Peeters, Patrick Traynor, Kevin Butler, and Joseph Wilson. Practical Hidden Voice Attacks against Speech and Speaker Recognition Systems. *Proceedings of the 2019 Network and Distributed System Security Symposium (NDSS)*, 2019.

[22] Hadi Abdullah, Aditya Karlekar, Vincent Bindschaedler, and Patrick Traynor. Demystifying limited adversarial transferability in automatic speech recognition systems, 2021.

[23] Hadi Abdullah, Muhammad Sajidur Rahman, Washington Garcia, Logan Blue, Kevin Warren, Anurag Swarnim Yadav, Tom Shrimpton, and Patrick Traynor. Hear "no evil", see "Kenansville": Efficient and Transferable Black-box Attacks on Speech Recognition and Voice Identification Systems. *IEEE Security and Privacy*, 2021.

[24] Hadi Abdullah, Muhammad Sajidur Rahman, Christian Peeters, Cassidy Gibson, Vincent Bindschaedler, Washington Garcia, Tom Shrimpton, and Patrick Traynor. Beyond lp clipping: Equalization-based psychoacoustic attacks against asrs. 2020.

[25] Hadi Abdullah, Kevin Warren, Vincent Bindschaedler, Nicolas Papernot, and Patrick Traynor. SoK: The faults in our ASRs: An overview of attacks against automatic speech recognition and speaker identification systems, 2020.

[26] Alejandro Acero and Richard M Stern. Environmental robustness in automatic speech recognition. In *International Conference on Acoustics, Speech, and Signal Processing*, pages 849–852. IEEE, 1990.

[27] Moustafa Alzantot, Bharathan Balaji, and Mani B. Srivastava. Did you hear that? Adversarial Examples Against Automatic Speech Recognition. In *Neural Information Processing Systems Workshop on Machine Deception 2017*, 2017.

[28] Amazon. Amazon Alexa.

[29] Dario Amodei, Sundaram Ananthanarayanan, Rishita Anubhai, Jingliang Bai, Eric Battenberg, Carl Case, Jared Casper, Bryan Catanzaro, Qiang Cheng, Guoliang Chen, et al. Deep speech 2: End-to-end speech recognition in english and mandarin. In *International conference on machine learning*, pages 173–182. PMLR, 2016.

[30] International Phonetic Association. *Handbook of the International Phonetic Association: A guide to the use of the International Phonetic Alphabet*. Cambridge University Press, 1999.

[31] Henry S Baird, Allison L Coates, and Richard J Fateman. Pessimalprint: a reverse turing test. *International Journal on Document Analysis and Recognition*, 5(2):158–163, 2003.

[32] Shumeet Baluja and Ian Fischer. Adversarial transformation networks: Learning to generate adversarial examples. *arXiv preprint arXiv:1703.09387*, 2017.

[33] Jeffrey P Bigham and Anna C Cavender. Evaluating existing audio captchas and an interface optimized for non-visual use. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1829–1838, 2009.

[34] Kevin Bock, Daven Patel, George Hughey, and Dave Levin. uncaptcha: a low-resource defeat of recaptcha's audio challenge. In *11th USENIX Workshop on Offensive Technologies WOOT*, 2017.

[35] Ruth M Brend. A Practical Introduction to Phonetics. *Studies in Second Language Acquisition*, 12(3):352–353, 1990.

[36] Tom B Brown, Dandelion Mané, Aurko Roy, Martín Abadi, and Justin Gilmer. Adversarial patch. *arXiv preprint arXiv:1712.09665*, 2017.

[37] Elie Bursztein, Romain Beauxis, Hristo Paskov, Daniele Perito, Celine Fabry, and John Mitchell. The Failure of Noise-based Non-continuous Audio Captchas. In *IEEE symposium on security and privacy*, pages 19–31, 2011.

[38] Elie Bursztein and Steven Bethard. Decaptcha: breaking 75% of eBay audio CAPTCHAs. In *Proceedings of the 3rd USENIX conference on Offensive technologies*, volume 1, page 8. USENIX Association, 2009.

[39] Elie Bursztein, Steven Bethard, Celine Fabry, John C Mitchell, and Dan Jurafsky. How Good are Humans at Solving CAPTCHAs? A Large Scale Evaluation. In *IEEE Symposium on Security and Privacy*, pages 399–413, 2010.

[40] Wilson Cai, Anish Doshi, and Rafael Valle. Attacking speaker recognition with deep generative models. *arXiv preprint arXiv:1801.02384*, 2018.

[41] Nicholas Carlini, Pratyush Mishra, Tavish Vaidya, Yuankai Zhang, Micah Sherr, Clay Shields, David Wagner, and Wenchao Zhou. Hidden

Voice Commands. In *USENIX Security Symposium*, pages 513–530, 2016.

[42] Nicholas Carlini and David Wagner. Audio adversarial examples: Targeted attacks on speech-to-text. In *IEEE Security and Privacy Workshops (SPW)*, pages 1–7, 2018.

[43] Kumar Chellapilla, Kevin Larson, Patrice Simard, and Mary Czerwinski. Designing human friendly human interaction proofs (HIPs). In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 711–720, 2005.

[44] Yuxuan Chen, Xuejing Yuan, Jiangshan Zhang, Yue Zhao, Shengzhi Zhang, Kai Chen, and XiaoFeng Wang. Devil's whisper: A general approach for physical adversarial attacks against commercial black-box speech recognition devices. In *29th USENIX Security Symposium*, 2020.

[45] Monica Chew and Henry S Baird. Baffletext: A human interactive proof. In *Document Recognition and Retrieval X*, volume 5010, pages 305–316. International Society for Optics and Photonics, 2003.

[46] Jusop Choi, Taekkyung Oh, William Aiken, Simon S Woo, and Hyoungshick Kim. Poster: I can't hear this because i am human: A novel design of audio captcha system. In *Proceedings of the 2018 on Asia Conference on Computer and Communications Security*, pages 833–835, 2018.

[47] Moustapha Cissé, Yossi Adi, Natalia Neverova, and Joseph Keshet. Houdini: Fooling Deep Structured Visual and Speech Recognition Models with Adversarial Examples. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA*, pages 6980–6990, 2017.

[48] Ronald Allan Cole, Y Muthusamy, and Mark Fanty. CSLU: ISOLET Spoken Letter Database Version 1.3. *Advances in Neural Information Processing Systems*, 25:1097–1105, 2012.

[49] Josh Dzieza. Why captchas have gotten so difficult, Feb 2019.

[50] Jeremy Elson, John R Douceur, Jon Howell, and Jared Saul. Asirra: a captcha that exploits interest-aligned manual image categorization. In *ACM Conference on Computer and Communications Security*, volume 7, pages 366–374, 2007.

[51] Valerie Fanelle, Sepideh Karimi, Aditi Shah, Bharath Subramanian, and Sauvik Das. Blind and human: Exploring more usable audio CAPTCHA designs. In *16th Symposium on Usable Privacy and Security (SOUPS 2020)*, pages 111–125, 2020.

[52] Stanley A Gelfand. *Hearing: An Introduction to Psychological and Physiological Acoustics*. CRC Press, 2017.

[53] Yuan Gong and Christian Poellabauer. Crafting Adversarial Examples for Speech Paralinguistics Applications. *ICLR*, 2015.

[54] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. *arXiv preprint arXiv:1412.6572*, 2014.

[55] Rich Gossweiler, Maryam Kamvar, and Shumeet Baluja. What's up CAPTCHA? A CAPTCHA based on image orientation. In *Proceedings of the 18th International Conference on World Wide Web*, pages 841–850, 2009.

[56] Alex Graves, Abdel-rahman Mohamed, and Geoffrey Hinton. Speech Recognition with Deep Recurrent Neural Networks. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649, 2013.

[57] Jonathan Holman, Jonathan Lazar, Jinjuan Heidi Feng, and John D'Arcy. Developing usable CAPTCHAs for blind users. In *Proceedings of the 9th International ACM SIGACCESS conference on Computers and Accessibility*, pages 245–246, 2007.

[58] Imran Hossen and Xiali Hei. aaecaptcha: The design and implementation of audio adversarial captcha. In *2022 IEEE 7th European Symposium on Security and Privacy (EuroS&P)*, pages 430–447. IEEE, 2022.

[59] Shehzeen Hussain, Paarth Neekhara, Shlomo Dubnov, Julian McAuley, and Farinaz Koushanfar. {WaveGuard}: Understanding and mitigating audio adversarial examples. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 2273–2290, 2021.

[60] Corey Kereliuk, Bob L Sturm, and Jan Larsen. Deep learning and music adversaries. *IEEE Transactions on Multimedia*, 17(11):2059–2071, 2015.

[61] Felix Kreuk, Yossi Adi, Moustapha Cisse, and Joseph Keshet. Fooling End-to-end Speaker Verification by Adversarial Examples. *The 43nd IEEE International Conference in Acoustic, Speech and Signal Processing (ICASSP)*, 2018.

[62] Deepak Kumar, Riccardo Paccagnella, Paul Murley, Eric Hennenfent, Joshua Mason, Adam Bates, and Michael Bailey. Skill squatting attacks on amazon alexa. In *27th USENIX Security Symposium*, pages 33–47, 2018.

[63] Alexey Kurakin, Ian Goodfellow, and Samy Bengio. Adversarial machine learning at scale. *ICLR*, 2017.

[64] Hyun Kwon, Hyunsoo Yoon, and Ki-Woong Park. Robust CAPTCHA Image Generation Enhanced with Adversarial Example Methods. *IEICE TRANSACTIONS on Information and Systems*, 103(4):879–882, 2020.

[65] Paul Lamere, Philip Kwok, William Walker, Evandro Gouvea, Rita Singh, Bhiksha Raj, and Peter Wolf. Design of the cmu sphinx-4 decoder. In *Eighth European Conference on Speech Communication and Technology*, 2003.

[66] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. Towards deep learning models resistant to adversarial attacks. *ICLR*, 2018.

[67] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 427–436, 2015.

[68] Margarita Osadchy, Julio Hernandez-Castro, Stuart Gibson, Orr Dunkelman, and Daniel Pérez-Cabo. No bot expects the deepcaptcha! introducing immutable adversarial examples, with applications to captcha generation. *IEEE Transactions on Information Forensics and Security*, 12(11):2640–2653, 2017.

[69] Vassil Panayotov, Guoguo Chen, Daniel Povey, and Sanjeev Khudanpur. Librispeech: an ASR corpus based on public domain audio books. In *IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5206–5210, 2015.

[70] Nicolas Papernot and Patrick McDaniel. Deep k-nearest neighbors: Towards confident, interpretable and robust deep learning. *arXiv preprint arXiv:1803.04765*, 2018.

[71] Nicolas Papernot, Patrick McDaniel, Somesh Jha, Matt Fredrikson, Z Berkay Celik, and Ananthram Swami. The limitations of deep learning in adversarial settings. In *IEEE European symposium on security and privacy (Euro S&P)*, pages 372–387. IEEE, 2016.

[72] Yao Qin, Nicholas Carlini, Ian Goodfellow, Garrison Cottrell, and Colin Raffel. Imperceptible, Robust, and Targeted Adversarial Examples for Automatic Speech Recognition. *36th International Conference on Machine Learning*, 2019.

[73] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[74] Lawrence R Rabiner, Ronald W Schafer, et al. *Digital processing of speech signals*. Prentice-hall, 1978.

[75] Nirupam Roy, Sheng Shen, Haitham Hassanieh, and Romit Roy Choudhury. Inaudible voice commands: The long-range attack and defense. In *15th USENIX Symposium on Networked Systems Design and Implementation*, pages 547–560, 2018.

[76] Tara N Sainath, Abdel-rahman Mohamed, Brian Kingsbury, and Bhuvana Ramabhadran. Deep convolutional neural networks for LVCSR. In *IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8614–8618, 2013.

[77] Tara N Sainath and Carolina Parada. Convolutional neural networks for small-footprint keyword spotting. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[78] Haşim Sak, Andrew Senior, Kanishka Rao, and Françoise Beaufays. Fast and accurate recurrent neural network acoustic models for speech recognition. *INTERSPEECH*, 2015.

[79] Hasim Sak, Andrew W Senior, and Françoise Beaufays. Long short-term memory recurrent neural network architectures for large scale acoustic modeling. 2014.

[80] Hasim Sak, Oriol Vinyals, Georg Heigold, Andrew Senior, Erik McDermott, Rajat Monga, and Mark Mao. Sequence discriminative

distributed training of long short-term memory recurrent neural networks. 2014.

[81] Graig Sauer, Harry Hochheiser, Jinjuan Feng, and Jonathan Lazar. Towards a universally usable captcha. In *Proceedings of the 4th Symposium on Usable Privacy and Security*, volume 6, page 1, 2008.

[82] Lea Schönherr, Katharina Kohls, Steffen Zeiler, Thorsten Holz, and Dorothea Kolossa. Adversarial Attacks Against Automatic Speech Recognition Systems via Psychoacoustic Hiding. 26th Annual Network and Distributed System Security Symposium, NDSS, 2019.

[83] Scott Hollier and Janina Sajka and Jason White and Michael Cooper. Inaccessibility of CAPTCHA – Alternatives to Visual Turing Tests on the Web. https://www.w3.org/TR/turingtest/, 2019. iW3C Working Group Note 09 December.

[84] Ali Shafahi, Mahyar Najibi, Amin Ghiasi, Zheng Xu, John Dickerson, Christoph Studer, Larry S Davis, Gavin Taylor, and Tom Goldstein. Adversarial training for free! *33rd Conference on Neural Information Processing Systems*, 2019.

[85] Mahmood Sharif, Sruti Bhagavatula, Lujo Bauer, and Michael K Reiter. Accessorize to a Crime: Real and stealthy attacks on state-of-the-art face recognition. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, pages 1528–1540, 2016.

[86] Heemany Shekhar, Melody Moh, and Teng-Sheng Moh. Exploring adversaries to defend audio captcha. In *2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA)*, pages 1155–1161. IEEE, 2019.

[87] Chenghui Shi, Xiaogang Xu, Shouling Ji, Kai Bu, Jianhai Chen, Raheem Beyah, and Ting Wang. Adversarial captchas. *arXiv preprint arXiv:1901.01107*, 2019.

[88] Temple F Smith, Michael S Waterman, et al. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[89] Saumya Solanki, Gautam Krishnan, Varshini Sampath, and Jason Polakis. In (cyber) space bots can hear you speak: Breaking audio captchas using ots speech recognition. In *Proceedings of the 10th ACM Workshop on Artificial Intelligence and Security*, pages 69–80, 2017.

[90] Yang Song, Rui Shu, Nate Kushman, and Stefano Ermon. Constructing unrestricted adversarial examples with generative models. In *Advances in Neural Information Processing Systems*, pages 8312–8323, 2018.

[91] Kenneth N Stevens. *Acoustic Phonetics*, volume 30. MIT press, 2000.

[92] Takeshi Sugawara, Benjamin Cyr, Sara Rampazzi, Daniel Genkin, and Kevin Fu. Light commands: laser-based audio injection attacks on voice-controllable systems. *29th USENIX Security Symposium*, 2020.

[93] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. Intriguing properties of neural networks. *ICLR*, 2014.

[94] Jennifer Tam, Jiri Simsa, Sean Hyde, and Luis V Ahn. Breaking audio captchas. In *Advances in Neural Information Processing Systems*, pages 1625–1632, 2008.

[95] Rohan Taori, Amog Kamsetty, Brenton Chu, and Nikita Vemuri. Targeted Adversarial Examples for Black Box Audio Systems. *IEEE Security and Privacy Workshops (SPW)*, 2019.

[96] Florian Tramer, Nicholas Carlini, Wieland Brendel, and Aleksander Madry. On adaptive attacks to adversarial example defenses. *Neural Information Processing Systems*, 2020.

[97] Tavish Vaidya, Yuankai Zhang, Micah Sherr, and Clay Shields. Cocaine Noodles: Exploiting the Gap between Human and Machine Speech Recognition. *WOOT*, 15:10–11, 2015.

[98] Subhashini Venugopalan, Huijuan Xu, Jeff Donahue, Marcus Rohrbach, Raymond Mooney, and Kate Saenko. Translating videos to natural language using deep recurrent neural networks. *Conference of the North American Chapter of the Association for Computational Linguistics – Human Language Technologies*, 2015.

[99] Luis Von Ahn, Benjamin Maurer, Colin McMillen, David Abraham, and Manuel Blum. recaptcha: Human-based character recognition via web security measures. *Science*, 321(5895):1465–1468, 2008.

[100] Pete Warden. Speech commands: A dataset for limited-vocabulary speech recognition. *arXiv preprint arXiv:1804.03209*, 2018.

[101] Robert Weide. The carnegie mellon pronouncing dictionary [cmudict. 0.6]. *Pittsburgh, PA: Carnegie Mellon University*, 2005.

[102] Hiromu Yakura and Jun Sakuma. Robust Audio Adversarial Example for a Physical Attack. *International Joint Conferences on Artificial Intelligence Organization*, 2019.

[103] Dong Yu and Li Deng. *AUTOMATIC SPEECH RECOGNITION*. Springer, 2016.

[104] Xuejing Yuan, Yuxuan Chen, Yue Zhao, Yunhui Long, Xiaokang Liu, Kai Chen, Shengzhi Zhang, Heqing Huang, Xiaofeng Wang, and Carl A Gunter. CommanderSong: A Systematic Approach for Practical Adversarial Voice Recognition. In *Proceedings of the USENIX Security Symposium*, 2018.

[105] Guoming Zhang, Chen Yan, Xiaoyu Ji, Tianchen Zhang, Taimin Zhang, and Wenyuan Xu. DolphinAttack: Inaudible voice commands. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 103–117. ACM, 2017.

[106] Zhengli Zhao, Dheeru Dua, and Sameer Singh. Generating natural adversarial examples. In *International Conference on Learning Representations*, 2018.

APPENDIX

Service	Potential Captcha Use	Min Len	Max Len
Apple [2]	Numbers (10-99)	3	5
BotDetect [10]	0-9 a-z	4	6
Captchas.net [12]	0-9 alpha-zulu	4	6
Microsoft [43]	Sound Identifcation	3	3
Securimage [5]	0-9 a-z	4	6
Telerik [1]	0-9 alpha-zulu	4	6
Google (Recaptcha v1) [99]	0-9	10	10
Google (Recaptcha v2) [99]	words	3	3
Google (Recaptcha v3) [99]	words	3	3
Ebay	Recaptcha	Recaptcha	Recaptcha
MTCaptcha [4]	0-9 a-z	4	4
Amazon	Recaptcha	Recaptcha	Recaptcha
	Sound Identifcation	3	3
Slashdot	Recaptcha	Recaptcha	Recaptcha

TABLE VII: An overview of the existing commercial CAPTCHA services. Since words, numbers, digits are commonly used by these services, we use them in our experimental setup. We also base our results on CAPTCHAs of length 6, since this is the most common size.